O'REILLY®

Выполняйте анализ данных с помощью R быстро и эффективно, используя свыше 275 практических рецептов!

Язык R предоставляет все, что вам нужно для работы со статистикой, но его структуру не всегда легко освоить. Эти ориентированные на задачи рецепты помогут вам сразу же приступить к работе. Решения варьируются от базовых задач до ввода и вывода, общей статистики, графики и линейной регрессии.

Каждый рецепт посвящен определенной задаче и сопровождается объяснением решения, показывая, как оно работает. Если вы только начинаете свое знакомство с R, эта книга поможет вам сориентироваться. Если вы промежуточный пользователь, она расширит ваши горизонты. Вы сможете работать быстрее, и узнаете больше о языке R в процессе прочтения.

«Из этой книги я узнал больше, чем из любой другой книги по программированию, которую я когда-либо читал».

Дэвид Кёрран, специалист по когнитивной инженерии, OpenJaw Technologies

Дж. Д. Лонг *– основатель Chicago R User Group. Заядлый пользователь Python, R, AWS , а также часто выступает с докладами на конференциях, посвященных языку R.*

Пол Титор *– специализируется на аналитике и разработке ПО для управления инвестициями, торговли ценными бумагами и управления рисками. Работает с хедж-фондами, маркетмейкерами и портфельными менеджерами в Большом Чикаго.*

Основные темы книги:

- создавайте векторы, обрабатывайте переменные и выполняйте основные функции;
- упростите ввод и вывод данных;
- изучите такие структуры данных, как матрицы, списки, факторы и фреймы данных;
- работайте с вероятностями, распределениями вероятностей и случайными величинами;
- рассчитайте статистику и доверительные интервалы и проведите статистические тесты;
- создавайте разнообразные графические представления;
- постройте статистические модели, используя линейные регрессии и дисперсионный анализ (ANOVA);
- изучите передовые статистические методы, такие как поиск кластеров в данных.

ISBN 978-5-97060-835-7

Интернет-магазин:
www.dmkpress.com

Оптовая продажа:
КТК «Галактика»
books@alians-kniga.ru

ДМК ИЗДАТЕЛЬСТВО
www.дмк.рф

R. Книга рецептов

O'REILLY®

R. Книга рецептов

Дж. Д. Лонг
Пол Титор

ДМК ИЗДАТЕЛЬСТВО

Дж. Д. Лонг и Пол Титор

R. Книга рецептов
Проверенные рецепты для статистики, анализа и визуализации

R Cookbook

Proven Recipes for Data Analysis, Statistics, and Graphics

J. D. Long and Paul Teetor

R. Книга рецептов
Проверенные рецепты для статистики, анализа и визуализации

Дж. Д. Лонг и Пол Титор

ДМК Пресс
издательство

Москва, 2020

УДК 004.438R
ББК 32.973.22
Л76

Дж. Д. Лонг и Пол Титор
Л76 R. Книга рецептов: Проверенные рецепты для статистики, анализа и визуализации данных / пер. с анг. Д. А. Беликова. – М.: ДМК Пресс, 2020. – 510 с.: ил.

ISBN 978-5-97060-835-7

Язык R – мощный инструмент статистического программирования, десятки тысяч людей ежедневно используют его для проведения серьезного статистического анализа. Но не все задачи, даже простые, удастся быстро решить с его помощью, если не знать определенных тонкостей.

Эта книга предлагает практические советы по решению разнообразных задач с подробным разбором каждой из них. От основных задач автор переходит к вводу и выводу, общей статистике, графике, линейной регрессии – любая значительная работа с R подразумевает знакомство с большинством этих областей или с ними всеми.

Издание пригодится для разработчиков на R с разным уровнем подготовки – от новичков до уверенных пользователей, желающих расширить свой кругозор.

УДК 004.438R
ББК 32.973.22

ISBN 978-1-492-04068-2 (анг.)
ISBN 978-5-97060-835-7 (рус.)

Оглавление

Предисловие редактора

Более двадцати лет назад, ещё в прошлом тысячелетии, в далёкой, далёкой, почти сказочной Новой Зеландии два статистика Джентельмен и Айхэка из Оклендского университета породили ещё один язык программирования. Немного подумали и назвали его R.

С тех пор, для того, чтобы вы могли легко и непринуждённо посчитать медиану, построить линейную модель, найти кластеры или, страшно подумать, размножить ваш набор данных для целей бутстрэпа, десятки, а возможно, и сотни тысяч квалифицированных и разных любителей статистической обработки данных положили на алтарь с буквой R самое ценное. Да, да – своё время! Пользуйтесь этой программой с уважением.

Когда мы (группа едва знакомых по интернету) чуть больше десяти лет назад начали цикл статей «Анализ данных с R» в ещё живой и бумажный журнал Linux Format, никакой печатной продукции, да и вообще хоть сколько-нибудь исчерпывающей информации на русскоязычном пространстве по этой теме просто не было. Хотя во всём остальном мире статистики R захватывал университет за университетом. Чуть позже благодаря усилиям лидера группы биолога Алексея Шипунова из этого цикла родилась первая значительная (мы честно на тот момент так думали) R-книга на русском языке «Наглядная статистика. Используем R!», выпущенная издательством «ДМК Пресс» в 2012 году и находящаяся сейчас в открытом доступе на CRAN для всех желающих.

Поэтому отрадно сейчас видеть, что мировая литература в области R-анализа уже достаточно регулярно, хоть и с задержкой, добирается до наших краёв. R – это просто. По сути, сборник рецептов является самым замечательным форматом для его практического применения. Другое дело – сам сборник рецептов – это только начало, потому что возможности R безграничны! В области обработки статистики, естественно.

Евгений Балдин, к. ф.-м. н.

Добро пожаловать в книгу рецептов R

Язык R представляет собой мощный инструмент для статистики, графики и статистического программирования. Он ежедневно используется десятками тысяч людей для проведения серьезного статистического анализа. Это бесплатная система с открытым исходным кодом, реализация которой является коллективным достижением большого числа умных, трудолюбивых людей. Для этого языка существует свыше 10 000 доступных пакетов дополнений. R является серьезным конкурентом всех коммерческих пакетов для статической обработки данных.

Но R может расстраивать, и может быть не ясно, как выполнить множество задач, даже простых. Простые задачи становятся легкими, когда вы знаете, как это сделать, но при этом понимаете, что это слово «как» может свести с ума.

Эта книга полна практических рецептов, каждый из которых решает определенную проблему. Каждый рецепт включает в себя краткое введение в решение, а затем обсуждение, целью которого является объяснить решение и дать вам представление относительно того, как оно работает. Мы знаем, что эти рецепты полезны, и знаем, что они работают, потому что сами используем их.

Диапазон рецептов обширен. Он начинается с основных задач, а затем переходит к вводу и выводу, общей статистике, графике и линейной регрессии. Любая значительная работа с R будет включать в себя большинство этих областей или их все.

Если вы новичок, то эта книга поможет вам быстрее сориентироваться. Если вы промежуточный пользователь, эта книга будет полезна, чтобы расширить ваш кругозор и покопаться в памяти («Как снова выполнить этот тест Колмогорова–Смирнова?»).

Данная книга не является учебником по языку R, хотя вы и узнаете что-то, изучая рецепты. Это не справочное руководство, но в ней содержится много полезной информации. Это не книга по программированию на R, хотя многие рецепты полезны в сценариях, написанных на этом языке.

Наконец, эта книга не является введением в статистику. Многие рецепты предполагают, что вы знакомы с базовой статистической процедурой, если таковая имеется, и просто хотите знать, как это делается в R.

Рецепты

В большинстве рецептов используется одна или две функции R для решения конкретной задачи. Важно помнить, что мы не описываем функции подробно; скорее, мы приводим достаточно описания, чтобы решить насущную проблему. Почти каждая такая функция имеет дополнительные возможности, помимо описанных здесь, а некоторые из них обладают удивительными возможностями.

Мы настоятельно рекомендуем вам прочитать страницы справки по функциям. Скорее всего, вы узнаете что-нибудь ценное.

Каждый рецепт предлагает один из способов решения конкретной задачи. Конечно, для каждой задачи существует несколько разумных решений. Когда нам было известно несколько вариантов решения, мы обычно выбирали самое простое. Для любой задачи вы, вероятно, сможете найти несколько альтернативных решений самостоятельно. Это книга рецептов, а не Библия.

В частности, R имеет буквально тысячи дополнительных пакетов, доступных для скачивания, многие из которых реализуют альтернативные алгоритмы и статистические методы. Эта книга концентрируется на основных функциях, доступных через базовый дистрибутив, в сочетании с несколькими важными пакетами, известными под общим названием *tidyverse*.

Наиболее краткое определение этого термина дает Хэдли Уикхем (https://blog.rstudio.com/2016/09/15/tidyverse-1-0-0/), его создатель и один из его основных разработчиков:

> *Tidyverse* – это набор пакетов, которые работают в гармонии, потому что они имеют общие представления данных и дизайн API. Пакет tidyverse разработан, чтобы упростить установку и загрузку основных пакетов из *tidyverse* одной командой. Лучшее место, чтобы познакомиться со всеми пакетами в tidyverse и их совместимостью, – это книга R for Data Science (https://r4ds.had.co.nz).

Примечание относительно терминологии

Цель каждого рецепта – решить задачу и сделать это быстро. Вместо того чтобы работать в утомительной прозе, мы иногда упорядочиваем описание с помощью терминологии, которая является правильной, но не точной. В качестве неплохого примера можно привести термин *обобщенная функция*. Мы называем `print(x)` и `plot(x)` обобщенными функциями, потому что они работают для многих видов `x`, соответственно обрабатывая каждый вид. Специалист, работающий в области теории вычислительных машин и систем, вздрогнет при виде нашей терминологии, потому что, строго говоря, это не просто «функции»; это полиморфные методы с динамической диспетчеризацией. Но если бы мы тщательно расшифровывали все эти подробности, основные решения были бы погребены под техническими деталями. Поэтому мы называем их просто функциями, что, как нам кажется, более читабельно.

Еще один пример, взятый из статистики, – это сложность, связанная с семантикой проверки статистических гипотез. Использование строгого языка теории вероятностей затруднит практическое применение некоторых тестов, поэтому мы используем более разговорный язык при описании всех статистических тестов. *См.* введение к главе 9 для получения дополнительной информации о том, как проверка гипотез представлена в рецептах.

Наша цель состоит в том, чтобы сделать мощь языка R доступной для широкой аудитории, сделав книгу читабельной, а не формальной. Мы надеемся, что эксперты в соответствующих областях поймут, является ли наша терминология неформальной.

Примечания по программному обеспечению и платформам

Базовый дистрибутив R имеет частые и запланированные выпуски, но определение языка и реализация ядра стабильны. Рецепты, приведенные в этой книге, должны работать с любым последним выпуском базового дистрибутива.

Некоторые рецепты имеют специфические для платформы соображения, и мы тщательно их отметили. Эти рецепты в основном касаются проблем, связанных с программным обеспечением, таких как установка и настройка. Насколько нам известно, все остальные рецепты будут работать на всех трех основных платформах для R: Windows, macOS и Linux/Unix.

Дополнительные источники

Ниже приводится ряд советов для получения дополнительной информации.

В сети

Своего рода «основа» для всего, что связано с R, – сайт проекта R (https://www.r-project.org). Отсюда вы можете скачать R для своей платформы, дополнительные пакеты, документацию и исходный код, а также множество других ресурсов.

Помимо сайта проекта R, мы рекомендуем использовать R-поисковик, например RSeek (https://rseek.org), созданный Сашей Гудманом. Вы можете использовать общую поисковую систему, такую как Google, но при применении «R» в качестве поискового запроса выдается слишком много посторонних вещей. *См.* рецепт 1.11 для получения дополнительной информации о поиске в интернете.

Чтение блогов – отличный способ узнать о R и быть в курсе самых передовых разработок. Таких блогов на удивление много, поэтому мы рекомендуем два из них: R-bloggers (https://www.r-bloggers.com), созданный Талом Галили, и PlanetR (http://planetr.stderr.org). Подписавшись на их RSS-каналы, вы будете получать уведомления об интересных и полезных статьях с десятков сайтов.

Книги

Есть много, много книг об обучении и использовании R. Перечисленные здесь несколько изданий мы нашли полезными. Обратите внимание, что сайт проекта R содержит обширную библиографию книг, связанных с R (https://www.r-project.org/doc/bib/R-books.html).

R for Data Science (http://shop.oreilly.com/product/0636920034407.do) Хэдли Уикхэма и Гарретта Гроулмунда (издательство *O'Reilly*) – отличное введение в пакеты *tidyverse*, особенно в том, что касается их использования в анализе данных и статистике. Данное издание также доступно онлайн (https://r4ds.had.co.nz).

Мы полагаем, что книга *R Graphics Cookbook* (http://shop.oreilly.com/product/0636920063704.do) Уинстона Чанга (издательство *O'Reilly*) незаменима для создания графики. Книга *ggplot2: Elegant Graphics for Data Analysis* Хэдли Уикхэма (издательство *Springer*) представляет собой наиболее полный справочник по графическому пакету ggplot2, который мы используем в этой книге. Любому, кто занимается серьезной графикой в R, понадобится книга *R Graphics* Пола Мёррелла (издательство *Chapman & Hall / CRC*).

R in a Nutshell (http://shop.oreilly.com/product/9780596801717.do) Джозефа Адлера

(издательство *O'Reilly*) – это краткое руководство и справочник, который всегда будет рядом с вами. Оно охватывает гораздо больше тем, нежели эта книга рецептов. Новые книги по программированию на языке R появляются регулярно. Мы предлагаем *Hands On Programming with R* (https://www.oreilly.com/library/view/hands-on-programming-with/9781449359089/) Гарретта Гроулмунда (издательство *O'Reilly*) для введения или *The Art of R Programming* Нормала Мэтлоффа (издательство *No Starch Press*). *Advanced R* Хэдли Уикхэма (издательство *Chapman & Hall / CRC*) доступна в печатном виде, а также бесплатно на странице http://adv-r.had.co.nz. В ней еще более подробно представлены продвинутые темы R. *Efficient R Programming* (http://shop.oreilly.com/product/0636920047995.do) Колина Гиллеспи и Робина Ловеласа (издательство *O'Reilly*) – еще одно хорошее руководство для изучения более глубоких концепций программирования на языке R.

Modern Applied Statistics with S, 4-е изд., Уильяма Венейблса и Брайана Рипли (Springer) использует R для иллюстрации множества передовых статистических методов. Функции и наборы данных книги доступны в пакете MASS, который входит в стандартный дистрибутив R.

Серьезные фанаты могут скачать *R Language Definition* (https://cran.r-project.org/doc/manuals/R-lang.pdf) от R Core Team. *R Language Definition* находится в стадии разработки, но может ответить на многие ваши подробные вопросы о R как о языке программирования.

Книги по статистике

Для изучения статистики прекрасно подойдет *Using R for Introductory Statistics* Джона Верзани (издательство *Chapman & Hall / CRC*). Эта книга обучает статистике и R, давая необходимые навыки работы с компьютером для применения статистических методов.

Вам понадобится хороший учебник по статистике или справочник, чтобы точно интерпретировать статистические тесты, выполненные в R. Подобных хороших книг множество – их слишком много, чтобы можно было рекомендовать какую-либо одну.

Авторы книг по статистике все чаще используют R для иллюстрации своих методов. Если вы работаете в специализированной области, то, вероятно, найдете полезную и актуальную книгу в библиографии проекта R (https://www.r-project.org/doc/bib/R-books.html).

Условные обозначения и соглашения, принятые в книге

В книге используются следующие типографские соглашения.

Курсив используется для смыслового выделения новых терминов, адресов электронной почты, а также имен и расширений файлов.

`Моноширинный шрифт` применяется для листингов программ, а также в обычном тексте для обозначения имен переменных, функций, типов, объектов, баз данных, переменных среды, операторов и ключевых слов.

`Моноширинный полужирный шрифт` используется для обозначения команд или фрагментов текста, которые пользователь должен ввести дословно без изменений.

`Моноширинный курсив` – для обозначения в исходном коде или в командах шаблонных меток-заполнителей, которые должны быть заменены соответствующими контексту реальными значениями.

Этот элемент означает подсказку или предложение.

Этот элемент означает общее примечание.

Этот элемент указывает на предупреждение или предостережение.

Использование примеров кода

Дополнительный материал (примеры кода, исходный код, упражнения и т. д.) доступен для скачивания на странице http://rc2e.com. Аккаунт в Twitter, посвященный этой книге, – @R_cookbook (https://twitter.com/R_cookbook).

Данная книга призвана помочь вам выполнить свою работу. В общем, вы можете использовать код из этой книги в своих программах и документации. Вам не нужно обращаться к нам за разрешением, если вы не воспроизводите значительную часть кода. Например, для написания программы, в которой используется несколько фрагментов кода из этой книги, оно не требуется. Продажа или распространение CD-ROM с примерами из книг O'Reilly требует разрешения. Чтобы ответить на вопрос, сославшись на эту книгу и приведя пример кода, разрешение не требуется. Включение значительного количества примеров кода из этой книги в документацию вашего продукта требует разрешения.

Атрибуция желательна, но не является обязательной. Обычно она включает в себя название книги, автора, издателя и ISBN. Например: «Книга рецептов R, 2-е изд., Дж. Д. Лонг и Пол Титор. Copyright 2019 Дж. Лонг и Пол Титор, 978-1-492-04068-2».

Если вы считаете, что применение примеров кода выходит за рамки добросовестного использования или только что описанного разрешения, свяжитесь с нами по адресу permissions@oreilly.com.

Обучение в режиме онлайн

На протяжении почти 40 лет *O'Reilly Media* (https://www.oreilly.com) предоставляет технологии и бизнес-тренинги, знания и анализ, чтобы помочь компаниям добиваться успеха.

Наша уникальная сеть экспертов и новаторов делится своими знаниями и опытом через книги, статьи, конференции и нашу онлайн-платформу обучения. Платформа онлайн-обучения O'Reilly предоставляет доступ по требованию к курсам обучения в режиме реального времени, углубленным способам обучения, интерактивным средам кодирования и обширной коллекции текстов и видео от O'Reilly и свыше 200 других издательств. Для получения дополнительной информации, пожалуйста, посетите сайт http://oreilly.com.

Как с нами связаться

Пожалуйста, направляйте комментарии и вопросы относительно этой книги издателю:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (в США или Канаде)
707-829-0515 (международный или местный)
707-829-0104 (факс)

У нас есть страница этой книги в сети, где перечислены ошибки, примеры и вся дополнительная информация. Вы можете получить доступ к этой странице по адресу http://bit.ly/RCookbook_2e.

Чтобы оставить комментарий или задать технические вопросы об этой книге, отправьте электронное письмо по адресу bookquestions@oreilly.com.

Для получения дополнительной информации о наших книгах, курсах, конференциях и новостях посетите наш сайт по адресу http://www.oreilly.com.

Ищите нас на Facebook: http://facebook.com/oreilly.

Подпишитесь на нас в Twitter: http://twitter.com/oreillymedia.

Смотрите нас в YouTube: http://www.youtube.com/oreillymedia.

Благодарности

Выражаем благодарность сообществу R в целом и R Core Team в частности. Их бескорыстный вклад огромен. Мир статистики извлекает огромную выгоду из их работы. Участники дискуссии сообщества R Studio очень помогли в разработке идей касательно того, как объяснить многие вещи. А сотрудники и руководство R Studio оказывали поддержку во многих отношениях. Мы в долгу перед ними за все, что они дали сообществу R.

Мы хотели бы поблагодарить технических рецензентов книги: Дэвида Кёррана, Джастина Ши и майора Дасти Тернера. Их отзывы были очень важны для улучшения качества, точности и полезности этой книги. Наши редакторы, Мелисса Поттер и Рэйчел Монаган, оказали невообразимую помощь, часто не давая нам публично демонстрировать свое невежество. Нашему редактору по производству, Кристен Браун, завидуют все технические авторы благодаря ее скорости и умению работать с Markdown и Git.

Павел хотел бы поблагодарить свою семью за поддержку и терпение при создании этой книги.

Дж. Д. Лонг хотел бы поблагодарить свою жену Мэри Бет и дочь Аду за их терпение в течение всех ранних утренних часов и выходных, которые он провел с ноутбуком, работая над этой книгой.

Глава 1

Начало работы и получение справочной информации

Эта глава закладывает основу для других глав. В ней объяснено, как скачать, установить и запустить R.

Что еще более важно, в ней также объясняется, как получить ответы на ваши вопросы. Сообщество R предоставляет большое количество документации и помощи. Вы не одиноки. Вот несколько распространенных источников справочной информации.

Локальная, установленная документация

Когда вы устанавливаете R на свой компьютер, вместе с ним устанавливается огромное количество документации. Вы можете просматривать локальную документацию (рецепт 1.7) и искать ее (рецепт 1.9). Поразительно, насколько часто ищем в интернете ответ на тот или иной вопрос только для того, чтобы обнаружить, что он уже доступен в установленной документации.

Представления задач

Представление задач (https://cran.r-project.org/web/views/) описывает пакеты, которые относятся к одной из областей статистической работы, такой как эконометрика, медицинская визуализация, психометрия или пространственная статистика. Каждое представление написано и поддерживается экспертом в данной области. Существует свыше 35 таких представлений, поэтому, вероятно, среди них будет один или несколько, имеющих отношение к вашей области интересов. Мы рекомендуем каждому новичку найти и прочитать хотя бы одно из представлений, чтобы познакомиться с возможностями R (рецепт 1.12).

Документация пакетов

Большинство пакетов содержат полезную документацию. Документация хранится вместе с пакетами в репозиториях пакетов, таких как CRAN (https://cran.r-project.org), и автоматически устанавливается на ваш компьютер при установке пакета.

Сайты с вопросами и ответами на них

На сайте, содержащем вопросы и ответы на них, каждый может задать вопрос, на который могут ответить знающие люди. Читатели голосуют за ответы, поэтому со временем будут показаны лучшие ответы. Вся эта информация помечается и архивируется для поиска. Эти сайты представляют собой нечто среднее между списком рассылки и социальной сетью; канонический пример – сайт Stack Overflow (https://stackoverflow.com).

Сеть

В сети много информации о R, и есть специальные инструменты для ее поиска (рецепт 1.11). Информация, содержащаяся в сети, постоянно обновляется, поэтому ищите новые, улучшенные способы организации и поиска информации о языке R.

Списки рассылки

Волонтеры щедро пожертвовали много часов, чтобы ответить на вопросы новичков, которые публикуются в списках рассылок R. Списки архивируются, поэтому вы можете искать в архивах ответы на свои вопросы (рецепт 1.13).

1.1. Загрузка и установка R

Задача

Вам необходимо установить R на свой компьютер.

Решение

Пользователи Windows и macOS могут скачать R на сайте CRAN (Comprehensive R Archive Network). Пользователи Linux и Unix могут установить пакеты R с помощью своего инструмента управления пакетами.

Windows

1. Перейдите на страницу http://www.r-project.org/ в своем браузере.
2. Нажмите **CRAN**. Вы увидите список зеркал сайта, упорядоченных по странам.
3. Выберите ближайший к вам сайт или самый первый указанный в верхней части списка (0-Cloud), который, как правило, хорошо работает для большинства местоположений (https://cloud.r-project.org/).
4. Нажмите **Download R for Windows** (Скачать R для Windows) в разделе **Download and Install R** (Скачать и установить R).
5. Нажмите **base**.
6. Нажмите на ссылку, чтобы скачать последнюю версию R (файл .exe).
7. По завершении загрузки дважды щелкните файл *.exe* и ответьте на привычные вопросы.

macOS

1. Перейдите на страницу http://www.r-project.org/ в своем браузере.
2. Нажмите **CRAN**. Вы увидите список зеркал сайта, упорядоченных по странам.

3. Выберите ближайший к вам сайт или самый первый указанный в верхней части списка (0-Cloud), который, как правило, хорошо работает для большинства местоположений.
4. Нажмите **Download R for (Mac) OS X** (Скачать R для (Mac) OS X).
5. Нажмите на файл *.pkg* для последней версии R в разделе **Latest release** (Последний выпуск), чтобы скачать его.
6. По завершении загрузки дважды щелкните файл *.pkg* и ответьте на привычные вопросы.

Linux или Unix

В основных дистрибутивах Linux есть пакеты для установки R. В табл. 1-1 приведены примеры.

Таблица 1-1. Дистрибутивы Linux

Дистрибутив	Название пакета
Ubuntu или Debian	r-base
Red Hat или Fedora	R.i386
SUSE	R-base

Используйте системный менеджер пакетов для скачивания и установки пакета. Вам понадобится пароль администратора или привилегии sudo; в противном случае попросите системного администратора выполнить установку.

Обсуждение

Установка R в Windows или macOS проста, потому что для этих платформ существуют готовые двоичные файлы (скомпилированные программы). Вам нужно всего лишь следовать предыдущим инструкциям. Страницы CRAN также содержат ссылки на ресурсы, связанные с установкой, например задаваемые вопросы (FAQ) и советы для особых ситуаций («Работает ли R на Windows Vista / 7/8 / Server 2008?»), что может оказаться полезным.

Наиболее подходящий способ установить R в Linux или Unix – использовать диспетчер пакетов дистрибутивов Linux для установки R в виде пакета. Дистрибутивные пакеты значительно упрощают как первоначальную установку, так и последующие обновления.

В Ubuntu или Debian используйте команду `apt-get` для загрузки и установки R. Воспользуйтесь утилитой `sudo`, чтобы получить необходимые привилегии:

```
$ sudo apt-get install r-base
```

В Red Hat или Fedora используйте `yum`:

```
$ sudo yum install R.i386
```

Большинство платформ Linux также имеют графические менеджеры пакетов, которые могут оказаться более удобными.

Помимо базовых пакетов, мы рекомендуем также установить пакеты документации. Нам нравится `r-base-html` (потому что мы любим просматривать документацию с гиперссылками), а также `r-doc-html`, который устанавливает важные руководства по R локально:

```
$ sudo apt-get install r-base-html r-doc-html
```

Некоторые репозитории Linux включают в себя готовые копии пакетов R, доступных в CRAN. Мы не используем их, потому что предпочитаем получать программное обеспечение непосредственно от самого CRAN, где обычно имеются самые свежие версии.

В редких случаях вам может понадобиться выполнить сборку R с нуля. У вас может быть неподдерживаемая версия Unix, или у вас могут быть особые соображения относительно производительности или конфигурации. Процедура сборки в Linux или Unix вполне стандартна. Скачайте архив с домашней страницы вашего зеркала CRAN; он будет называться как-нибудь вроде *R-3.5.1.tar.gz*, за исключением того, что вместо 3.5.1 будет указана последняя версия. Распакуйте архив, найдите файл с именем *INSTALL* и следуйте инструкциям.

См. также

В книге *R in a Nutshell* Джозефа Адлера (издательство *O'Reilly*) содержится более подробная информация о загрузке и установке R, включая инструкции по сборке версий для Windows и macOS. Возможно, самым полным является руководство по установке и администрированию R, доступное в CRAN, где описывается сборка и установка R на различных платформах.

Это рецепт установки базового пакета. *См.* рецепт 3.10 для установки пакетов дополнений из CRAN.

1.2. Установка RStudio

Задача

Вам нужна более комплексная интегрированная среда разработки (IDE), чем та, что есть в R по умолчанию. Другими словами, вы хотите установить RStudio Desktop.

Решение

За последние несколько лет RStudio стал наиболее широко используемой интегрированной средой разработки для R. Мы считаем, что почти вся работа с R должна выполняться в среде разработки RStudio Desktop, если нет веской причины поступить иначе. RStudio производит ряд продуктов, включая RStudio Desktop, RStudio Server и RStudio Shiny Server среди прочих. В этой книге мы будем использовать термин RStudio для обозначения RStudio Desktop, хотя большинство концепций применимо и к RStudio Server.

Чтобы установить RStudio, загрузите последнюю версию установщика для вашей платформы с сайта RStudio (https://rstudio.com/products/rstudio/download/).

Версию RStudio Desktop с открытым исходным кодом можно скачать и использовать бесплатно.

Обсуждение

В этой книге используются RStudio версии 1.2.x и R версии 3.5.x. Новые версии RStudio выпускаются каждые несколько месяцев, поэтому регулярно обновляй-

те их. Обратите внимание, что RStudio работает с любой версией R, которую вы установили, поэтому обновление до последней версии RStudio *не* обновляет вашу версию R. R необходимо обновлять отдельно.

Взаимодействие с R в RStudio немного отличается от встроенного пользовательского интерфейса R. Для этой книги мы решили использовать RStudio для всех примеров.

1.3. Запуск RStudio

Задача

Вы хотите запустить RStudio на своем компьютере.

Решение

Распространенная ошибка, которую допускают новые пользователи R и RStudio, – это случайный запуск R, когда они намеревались запустить RStudio. Самый простой способ убедиться в том, что вы действительно запускаете RStudio, – это найти RStudio на своем рабочем столе, а затем использовать любой метод, который предоставляет ваша ОС для закрепления иконки там, где ее легко можно будет найти позже:

Windows

Нажмите на меню **Пуск** в левом нижнем углу экрана. В поле поиска введите `RStudio`.

macOs

Найдите в панели запуска приложение RStudio или нажмите **Cmd-space** (**Cmd** – это команда или клавиша ⌘)) и введите `RStudio`, чтобы выполнить поиск с помощью Spotlight Search.

Ubuntu

Нажмите сочетание клавиш **Alt-F1** и введите `RStudio` для поиска RStudio.

Обсуждение

На рис. 1-1, видно, что иконки R и RStudio похожи , что может привести вас в замешательство

Рис. 1-1. Иконки R и RStudio в macOS

Если вы нажмете на значок R, то увидите нечто похожее на то, что изображено на рис. 1-2. Это базовый интерфейс R в Mac, но, конечно же, не RStudio.

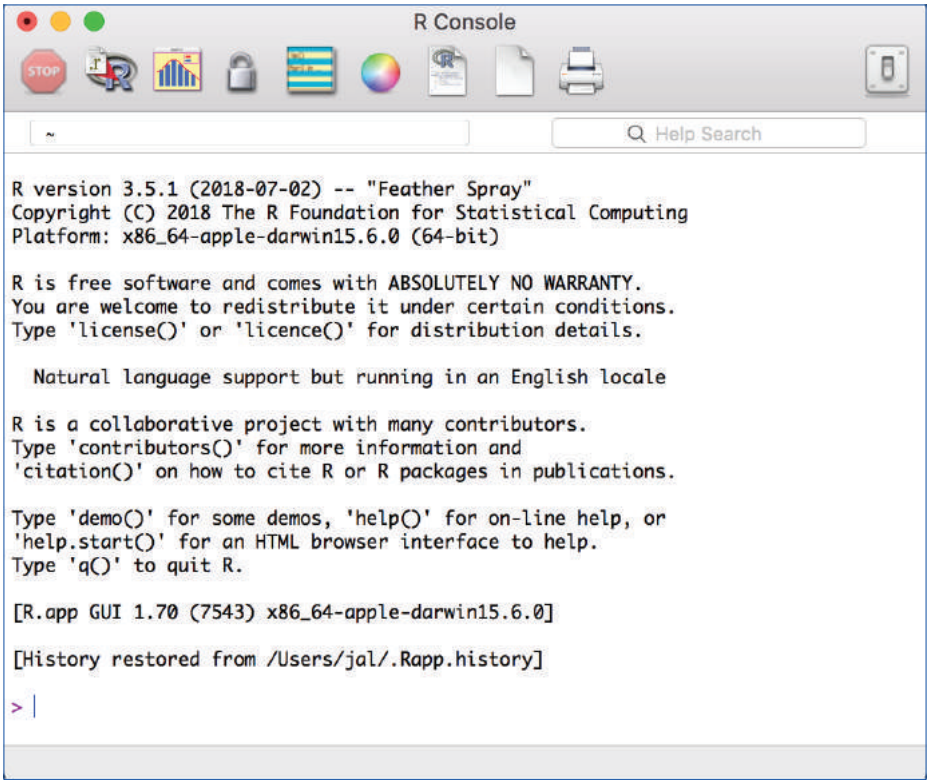


Рис. 1-2. Консоль R в macOS

Когда вы запускаете RStudio, по умолчанию он снова откроет последний проект, над которым вы работали в RStudio.

1.4. Ввод команд

Задача

Вы запустили RStudio. Что теперь?

Решение

Когда вы запускаете RStudio, главное окно в левой части экрана – это сеанс R. Оттуда вы можете вводить команды в интерактивном режиме непосредственно в R.

Обсуждение

Приглашение командной строки в R выглядит так: >. Для начала просто воспринимайте R как большой калькулятор: введите выражение, а R вычислит его и выведет результат:

```
> 1+1
[1] 2
>
```

Компьютер складывает 1 и 1 и отображает результат, 2.

[1] перед 2 может сбивать с толку. Для R результат – это вектор, хотя у него имеется только один элемент. R помечает значение с помощью [1], чтобы показать, что это первый элемент вектора... что неудивительно, поскольку это *единственный* элемент вектора.

R будет запрашивать ввод, пока вы не введете полное выражение. Выражение `max(1,3,5)` является полным, поэтому R прекращает чтение ввода и вычисляет то, что получил:

```
> max(1, 3,
+5)
[1] 5
>
```

`max (1,3`, напротив, представляет собой неполное выражение, поэтому R запрашивает дополнительный ввод. Вместо знака «больше, чем» (>) появится «плюс» (+), сообщая, что R ожидает большего:

```
> max(1, 3, +5)
[1] 5
>
```

Команды легко набирать неправильно, а вводить их снова утомительно и неприятно. Таким образом, R включает редактирование командной строки, чтобы сделать жизнь проще. Он определяет одиночные нажатия клавиш, которые позволяют легко вызывать, исправлять и повторно выполнять ваши команды. Типичное взаимодействие с командной строкой выглядит так:

1. Вы вводите выражение с опечаткой.
2. R жалуется на вашу ошибку.
3. Вы нажимаете клавишу со стрелкой вверх, чтобы вызвать на экран строку, в которой есть ошибка.
4. Вы используете клавиши со стрелками влево и вправо, чтобы переместить курсор назад к ошибке.
5. Вы используете клавишу **Delete**, чтобы удалить неправильные символы.
6. Вы вводите исправленные символы, которые вставляются в командную строку.
7. Вы нажимаете **Enter**, чтобы повторно выполнить исправленную команду.

Это лишь основы. R поддерживает обычные нажатия клавиш для вызова и редактирования командных строк, как показано в табл. 1-2.

В большинстве операционных систем также можно использовать мышь, чтобы выделить команды, а затем использовать обычные команды копирования и вставки, чтобы вставить текст в новую командную строку.

Таблица 1-2. Оперативные клавиши выбора в R

Маркированная клавиша	Сочетание клавиш	Результат
Up arrow	**Ctrl**-**P**	Вспомнить предыдущую команду, перемещаясь назад по истории команд
Down arrow	**Ctrl**-**N**	Движение вперед по истории команд
Backspace	**Ctrl**-**H**	Удалить символ слева от курсора
Delete (**Del**)	**Ctrl**-**D**	Удалить символ справа от курсора
Home	**Ctrl**-**A**	Переместить курсор в начало строки
End	**Ctrl**-**E**	Переместить курсор в конец строки
Right arrow	**Ctrl**-**F**	Переместить курсор вправо (вперед) на один символ
Left arrow	**Ctrl**-**B**	Переместить курсор влево (назад) на один символ
	Ctrl-**K**	Удалить все начиная от положения курсора до конца строки
	Ctrl-**U**	Очистить всю чертову строку и начать все сначала
Tab		Автодополнение имени (на некоторых платформах)

См. также

См. рецепт 2.12. В главном меню Windows выберите **Справка** → **Консоль** для получения полного списка нажатий клавиш, полезных для редактирования в командной строке.

1.5. Выход из RStudio

Задача

Вам нужно выйти из RStudio.

Решение

Windows и большинство дистрибутивов Linux

Выберите **Select File** → **Quit Session** (Файл → Выйти из сеанса) в главном меню или нажмите **X** в верхнем правом углу окна.

macOS

Выберите **Select File** → **Quit Session** (Файл → Выйти из сеанса) в главном меню, или нажмите **Cmd**-**Q**, либо щелкните красный кружок в верхнем левом углу окна.

На всех платформах вы также можете использовать функцию `q` (как в *q*uit) для завершения сеанса R и RStudio:

```
q()
```

Обратите внимание на пустые скобки, которые необходимы для вызова функции.

Обсуждение

Каждый раз, когда вы выходите, R обычно спрашивает, хотите ли вы сохранить рабочее пространство. У вас есть три варианта:

- сохранить рабочее пространство и выйти;
- не сохранять свое рабочее пространство и все равно выйти;
- выполнить отмену, вернувшись в командную строку, вместо того чтобы выйти.

Если вы сохраняете свое рабочее пространство, R записывает его в файл с именем *.RData* в текущем рабочем каталоге. При сохранении рабочей области сохраняются любые объекты R, которые вы создали. В следующий раз, когда вы запустите R в том же каталоге, рабочая область будет загружена автоматически. При сохранении вашей рабочей области ранее сохраненная рабочая область будет перезаписана, если таковая имеется, поэтому не сохраняйте ее, если вам не нравятся ваши изменения (например, если вы случайно удалили важные данные из своего рабочего пространства).

Мы рекомендуем никогда не сохранять рабочее пространство при выходе и вместо этого всегда в явном виде сохранять проект, сценарии и данные. Мы также рекомендуем отключить приглашение к вводу команды, чтобы сохранить и автоматически восстановить рабочее пространство в RStudio, используя глобальные параметры, которые находятся в меню **Tools → Global Options** (Инструменты → Глобальные параметры) и показаны на рис. 1-3. Таким образом, когда вы выходите из R и RStudio, вам не будет предложено сохранить рабочее пространство. Но имейте в виду, что любые объекты, созданные, но не сохраненные на диск, будут потеряны!

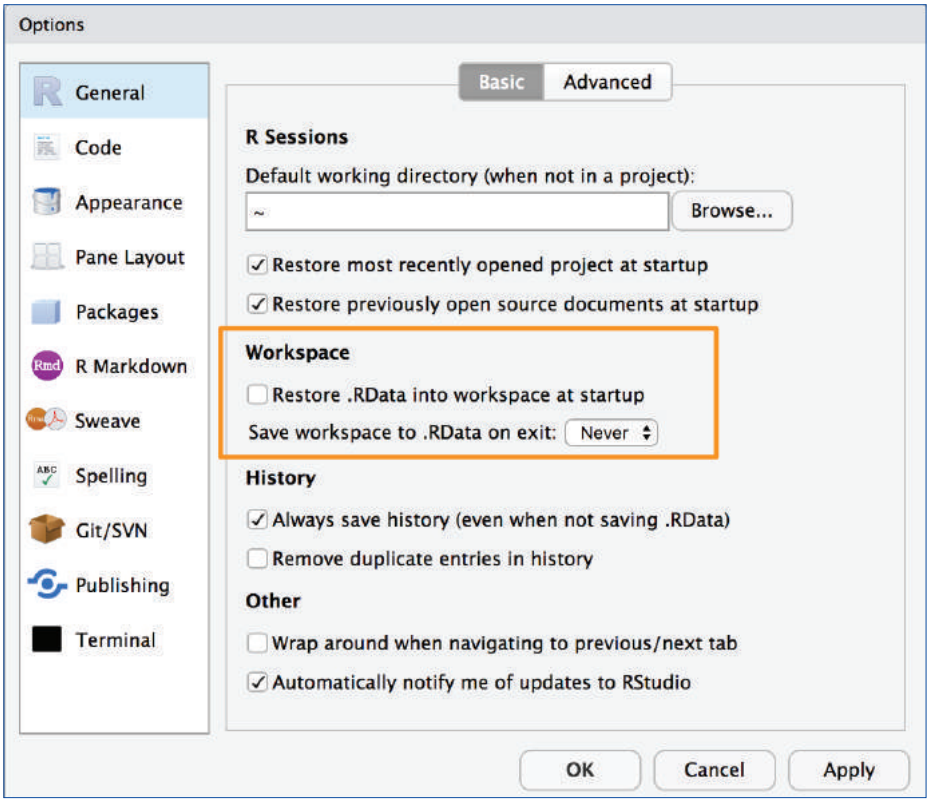


Рис. 1-3. Параметры сохранения рабочего пространства

См. также

См. рецепт 3.1 для получения дополнительной информации о текущем рабочем каталоге и рецепт 3.3 для получения дополнительной информации о сохранении рабочего пространства. Также *см.* вторую главу книги *R in a Nutshell*.

1.6. Прерывание R

Задача

Вам нужно прервать длительные вычисления и вернуться в командную строку, не выходя из RStudio.

Решение

Нажмите клавишу **Esc** на клавиатуре или щелкните на меню **Session** (Сеанс) в RStudio и выберите **Interrupt R** (Прервать R). Вы также можете щелкнуть значок знака остановки в окне консоли кода.

Обсуждение

Прерывание R означает указание R прекратить выполнение текущей команды, но без удаления переменных из памяти или полного закрытия RStudio. Тем не менее прерывание R может оставить ваши переменные в неопределенном состоянии, в зависимости от того, насколько далеко продвинулись вычисления, поэтому проверьте свое рабочее пространство после прерывания.

См. также

См. рецепт 1.5.

1.7. Просмотр прилагаемой документации

Задача

Вы хотите прочитать документацию, поставляемую с R.

Решение

Используйте функцию `help.start`, чтобы увидеть оглавление документации:

```
help.start()
```

Отсюда доступны ссылки для всей установленной документации. В RStudio справка будет отображаться на панели справки, которая по умолчанию находится в правой части экрана.

В RStudio также можно нажать **Help** → **R Help** (Справка → Справка R), чтобы получить список с опциями справки как для R, так и для RStudio.

Обсуждение

Базовый дистрибутив R включает в себя множество документации – буквально тысячи страниц. При установке дополнительных пакетов эти пакеты содержат документацию, которая также установлена на вашем компьютере.

Эту документацию легко просмотреть с помощью функции `help.start`, которая открывает оглавление верхнего уровня. На рис. 1-4 показано, как `help.start` отображается на панели справки в RStudio.

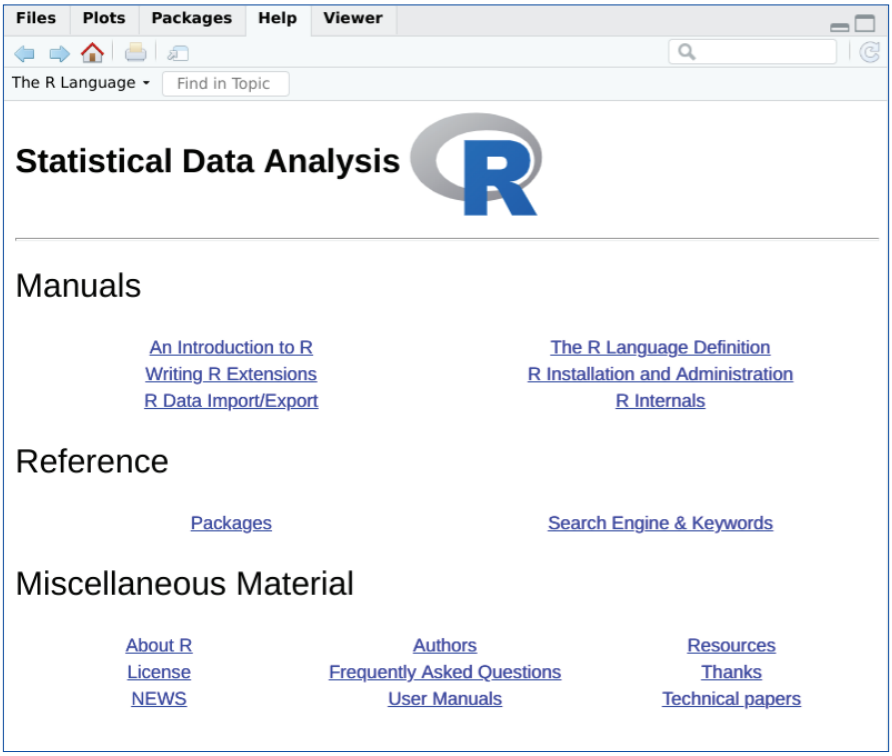


Рис. 1-4. help.start в RStudio

Две ссылки в разделе «Базовый справочник R» особенно полезны:

Пакеты (Packages)

Нажмите здесь, чтобы увидеть список всех установленных пакетов – как базовых, так и установленных дополнительно. Нажмите на имя пакета, чтобы увидеть список его функций и наборов данных.

Поисковая система и ключевые слова (Search Engine & Keywords)

Нажмите здесь, чтобы получить доступ к простой поисковой системе, которая позволяет выполнять поиск в документации по ключевому слову или фразе. Существует также список распространенных ключевых слов, организованных по темам; щелкните на одно из них, чтобы увидеть связанные страницы.

Базовая документация R, доступ к которой осуществляется через `help.start`, загружается на ваш компьютер при установке R. Справка RStudio, доступ к которой вы получаете с помощью опции меню **Help** → **R Help** (Справка → Справка R), представляет страницу с ссылками на сайт RStudio. Таким образом, вам понадобится доступ в интернет для доступа к справочным ссылкам RStudio.

См. также

Локальная документация скопирована с сайта R Project (https://www.r-project.org), где могут быть обновленные документы.

1.8. Получение справки по функции

Задача

Вы хотите больше узнать о функции, которая установлена на вашем компьютере.

Решение

Используйте функцию `help` для отображения документации по функции:

```
help(имяфункции)
```

Используйте функцию `args` для быстрого напоминания аргументов функции:

```
args(имяфункции)
```

Используйте функцию `example`, чтобы увидеть примеры использования функции:

```
example(имяфункции)
```

Обсуждение

В этой книге мы представляем много функций R. Каждая функция имеет больше дополнительных свойств, чем мы можем описать. Если функция заинтересовала вас, мы настоятельно рекомендуем прочитать страницу справки для этой функции. Одно из таких свойств может оказаться очень полезным для вас.

Предположим, вы хотите узнать больше о функции `mean`. Используйте функцию `help`:

```
help mean
```

Откроется страница справки для функции `mean` на панели справки в RStudio. Для быстрого вызова команды `help` просто наберите `?`, за которым следует имя функции:

```
?mean
```

Иногда вам просто нужно быстрое напоминание об аргументах функции: каковы они и в каком порядке встречаются? Для этого случая используйте функцию `args`:

```
args(mean)
#> function (x, ...)
#> NULL

args(sd)
#> function (x, na.rm = FALSE)
#> NULL
```

Первая строка вывода представляет собой краткий обзор вызова функции. В случае с `mean` в синопсисе показан один аргумент, `x`, который представляет собой вектор чисел. В случае с `sd` синопсис показывает тот же вектор, `x` и необязательный аргумент `na.rm`. (Вторую строку вывода, которая часто представляет собой просто NULL,

можно игнорировать.) В RStudio вы будете видеть вывод `args` в виде всплывающей подсказки над курсором при вводе имени функции, как показано на рис. 1-5.

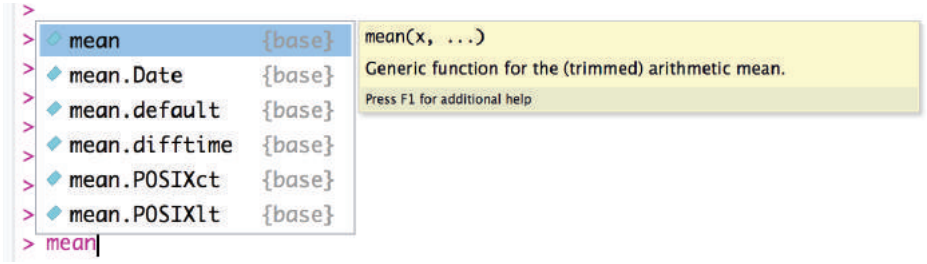


Рис. 1-5. Подсказка RStudio

Большая часть документации по функциям содержит пример кода в конце документа. Отличная особенность R заключается в том, что вы можете запросить выполнение примеров, в результате чего увидите небольшую демонстрацию возможностей функции. Например, в документации функции `mean` содержатся примеры, но вам не нужно вводить их самостоятельно: просто используйте функцию `example`, чтобы увидеть их выполнение:

```
example(mean)
#>
#> mean> x <- c(0:10, 50)
#>
#> mean> xm <- mean(x)
#>
#> mean> c(xm, mean(x, trim = 0.10))
#> [1] 8.75 5.50
```

Все, что вы видите после `example(mean)`, было создано R, который выполнил примеры со страницы справки и отобразил результаты.

См. также

См. рецепт 1.9 для поиска функций и рецепт 3.6 для получения дополнительной информации о пути поиска.

1.9. Поиск в прилагаемой документации

Задача

Вы хотите узнать больше о функции, которая установлена на вашем компьютере, но функция `help` сообщает, что не может найти подходящую документацию.

Или же вы хотите найти в установленной документации ключевое слово.

Решение

Используйте функцию `help.search` для поиска в документации R на своем компьютере:

```
help.search("шаблон")
```

Типичным шаблоном является имя функции или ключевое слово. Обратите внимание, что он должен быть заключен в кавычки.

Для своего удобства вы также можете вызвать поиск, используя два вопросительных знака (в этом случае кавычки не требуются). Обратите внимание, что для поиска функции по имени используется один знак вопроса, а для поиска текстового шаблона – два:

```
> ??шаблон
```

Обсуждение

Время от времени вы можете запрашивать справку по функции, только чтобы получить сообщение, в котором говорится, что R ничего о ней неизвестно:

```
help(adf.test)
#> No documentation for 'adf.test' in specified packages and libraries:
#> you could try '??adf.test'
```

Если вы знаете, что функция установлена на вашем компьютере, это может быть неприятно. Проблема здесь состоит в том, что пакет функции в данный момент не загружен, и вы не знаете, в каком пакете содержится эта функция. Это своего рода уловка-22 (сообщение об ошибке указывает на то, что пакет в данный момент отсутствует в вашем пути поиска, поэтому R не может найти файл справки; см. рецепт 3.6 для получения более подробной информации).

Решение заключается в поиске всех установленных пакетов для этой функции. Просто используйте функцию `help.search`, как указано в сообщении об ошибке:

```
help.search("adf.test")
```

Поиск выдаст список всех пакетов, которые содержат эту функцию:

```
Help files with alias or concept or title matching 'adf.test' using
regular expression matching:

tseries::adf.test    Augmented Dickey-Fuller Test
Type '?PKG::FOO' to inspect entry 'PKG::FOO TITLE'.
```

Предыдущий вывод показывает, что пакет `tseries` содержит функцию `adf.test`. Можно увидеть ее документацию, явно указав справке, какой пакет содержит функцию:

```
help(adf.test, package = "tseries")
```

или можно использовать оператор двойного двоеточия, чтобы указать R выполнить поиск в определенном пакете:

```
?tseries::adf.test
```

Вы можете расширить свой поиск, используя ключевые слова. Затем R найдет любую установленную документацию, которая содержит ключевые слова. Предположим, вы хотите найти все функции, в которых упоминается расширенный тест Дики–Фуллера (ADF). Можно искать по вероятному шаблону:

```
help.search("dickey-fuller")
```

См. также

Вы также можете получить доступ к локальной поисковой системе через браузер документации; *см.* рецепт 1.7, чтобы узнать, как это сделать. *См.* рецепт 3.6 для получения дополнительной информации о пути поиска и рецепт 1.8 для получения справки по функциям.

1.10. Получение справки по пакету

Задача

Вы хотите узнать больше о пакете, установленном на вашем компьютере.

Решение

Используйте функцию `help` и укажите имя пакета (без имени функции):

```
help(package = "имяпакета")
```

Обсуждение

Иногда вам нужно знать содержимое пакета (функции и наборы данных).

Это особенно верно, например, после загрузки и установки нового пакета. Функция `help` может предоставить содержимое плюс другую информацию, как только вы укажете имя пакета.

В результате этого вызова функции `help` будет показана информация для пакета `tseries`, стандартного пакета в базовом дистрибутиве:

```
help(package = "tseries")
```

Эта информация начинается с описания и продолжается указателем функций и наборов данных. В RStudio страница справки в формате HTML откроется в окне справки интегрированной среды разработки.

Некоторые пакеты также содержат сопроводительную документацию, это введения, учебные пособия или справочные карточки. Они устанавливаются на ваш компьютер как часть документации пакета при его инсталляции. Страница справки для пакета содержит список сопроводительной документации в нижней части.

Вы можете просмотреть этот список на своем компьютере, используя функцию `vignette`:

```
vignette()
```

В RStudio откроется новая вкладка со списком всех пакетов, установленных на вашем компьютере, которые содержат виньетки, а также их названия и описания.

Можно увидеть документацию для конкретного пакета, включив его название:

```
vignette(package = "имяпакета")
```

У каждой виньетки есть имя, которое используется для ее просмотра:

```
vignette("имявиньетки")
```

См. также

См. рецепт 1.8 для получения справки по конкретной функции в пакете.

1.11. Поиск справки в интернете

Задача

Вам нужно найти в интернете информацию и ответы, касающиеся R.

Решение

Внутри R используйте функцию `RSiteSearch` для поиска по ключевому слову или фразе:

```
RSiteSearch("ключевая фраза")
```

В своем браузере попробуйте использовать приведенные ниже сайты для поиска:

RSeek (https://rseek.org)
: Это пользовательская поисковая система Google, ориентированная на сайты, посвященные R.

Stack Overflow (https://stackoverflow.com)
: Stack Overflow – сайт с вопросами и ответами с возможностью поиска от сети Stack Exchange, который ориентирован на такие проблемы программирования, как структуры данных, кодирование и графика. Stack Overflow – отличная «первая остановка» для всех ваших вопросов, касающихся синтаксиса.

Cross Validated (https://stats.stackexchange.com)
: Cross Validated – это сайт Stack Exchange, нацеленный на статистику, машинное обучение и анализ данных, а не на программирование. Это хорошее место, чтобы задать вопрос по поводу того, какой статистический метод использовать.

*Сообщество RStudio (*https://community.rstudio.com*)*
: Сайт сообщества RStudio – это дискуссионный форум, организованный RStudio. Темы включают в себя R, RStudio и связанные с ними технологии. Поскольку этот форум посвящен RStudio, его часто посещают сотрудники компании RStudio Inc и те, кто нередко использует это программное обеспечение. Это неплохое место для вопросов общего характера и вопросов, которые, возможно, не вписываются в формат, ориентированный на синтаксис Stack Overflow.

Обсуждение

Функция `RSiteSearch` откроет окно браузера и направит его в поисковую систему на сайте R Project (http://search.r-project.org). Здесь вы можете уточнить параметры поиска. Например, так мы ищем термин «каноническая корреляция»:

```
RSiteSearch("canonical correlation")
```

Это очень удобно. Таким образом можно выполнять быстрый поиск, не выходя из R. Однако область поиска ограничена документацией R и архивами списков рассылки.

RSeek (https://rseek.org) обеспечивает более широкий поиск. Его достоинство заключается в том, что он использует всю мощь поисковой системы Google, концентрируясь на сайтах, имеющих отношение к R, и исключает посторонние результаты общего поиска Google. Прелесть RSeek в том, что он организует результаты полезным способом.

На рис. 1-6 показаны результаты посещения сайта RSeek и поиска термина «корреляция». Обратите внимание, что вкладки в верхней части позволяют переходить к различным типам контента:

- **All results** (Все результаты);
- **Packages** (Пакеты);
- **Books** (Книги);
- **Support** (Служба поддержки);
- **Articles** (Статьи);
- **For Beginners** (Для начинающих).

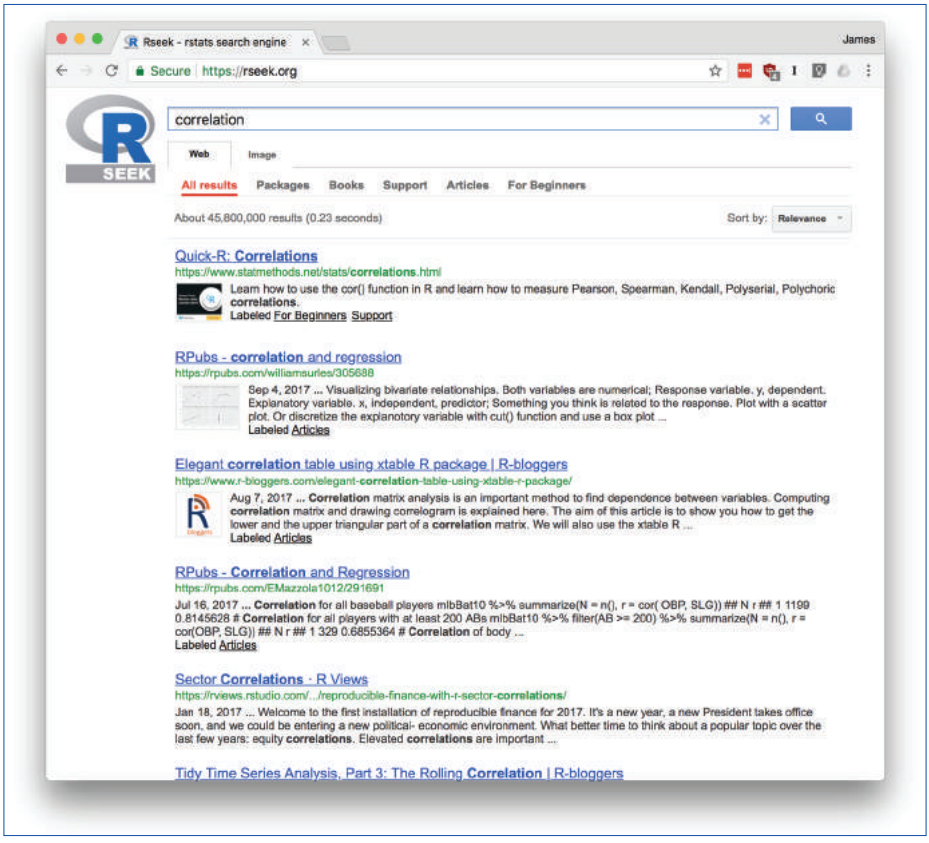


Рис. 1-6. RSeek

Stack Overflow (https://stackoverflow.com) – это сайт с вопросами и ответами. Это означает, что любой может отправить вопрос, а опытные пользователи дадут на него ответы – часто на каждый вопрос есть несколько ответов. Читатели голосуют за ответы, поэтому хорошие ответы обычно поднимаются наверх, что созда-

ет богатую базу данных диалогов, состоящих из вопросов и ответов, по которым можно выполнять поиск. Stack Overflow ориентирован главным образом на решение проблем, и темы прежде всего касаются программирования на языке R.

Stack Overflow содержит вопросы по многим языкам программирования; поэтому при вводе термина в поле поиска добавьте к нему «[r]», чтобы сфокусировать поиск на вопросах, касающихся R. Например, при поиске «стандартная ошибка [r]» будут выбраны только вопросы, помеченные как касающиеся языка R, и вам не будут попадаться вопросы по Python и C ++.

Stack Overflow также включает в себя статью о языке R (https://stackoverflow.com/tags/r/info), которая предоставляет превосходный список онлайн-ресурсов для сообщества.

У Stack Exchange (материнская компания Stack Overflow) есть сайт для статистического анализа под названием Cross Validated (https://stats.stackexchange.com). Этот сайт больше сфокусирован на статистике, нежели на программировании, поэтому используйте его при поиске ответов, которые больше касаются статистики в целом, а не R в частности.

У RStudio также есть собственный форум (https://community.rstudio.com). Это отличное место, чтобы задавать общие и более концептуальные вопросы, которые могут не дать нужного результата на Stack Overflow.

См. также

Если в результате поиска вы обнаружите полезный пакет, используйте рецепт 3.10, чтобы установить его на свой компьютер.

1.12. Поиск соответствующих функций и пакетов

Задача

Из 10 000+ пакетов для R вы не знаете, какие из них будут вам полезны.

Решение

Чтобы найти пакеты, относящиеся к определенной области, посетите список представлений задач CRAN (https://cran.r-project.org/web/views/). Выберите представление задачи для вашей области, которая даст вам описания соответствующих пакетов и ссылки на них. Или посетите сайт RSeek (https://rseek.org), выполните поиск по ключевому слову, перейдите на вкладку **Представления задач** и выберите подходящее представление.

Посетите сайт https://crantastic.org и поищите пакеты по ключевому слову.

- Чтобы найти соответствующие функции, посетите сайт RSeek, выполните поиск по имени или ключевому слову и нажмите на вкладку **Функции**.

Обсуждение

Эта проблема особенно неприятна для начинающих. Вы думаете, что R может решить ваши проблемы, но не знаете, какие пакеты и функции были бы полезны. Распространенный вопрос, встречающийся в списках рассылки: «Существует ли какой-нибудь пакет для решения проблемы X?» Это тихий крик того, кто тонет в R.

На момент написания этих строк доступно свыше 10 000 пакетов для бесплатной загрузки с CRAN. У каждого пакета имеется страница с кратким описанием и ссылками на документацию. Найдя потенциально интересный пакет, вы обычно нажимаете ссылку **Reference manual** (Справочное руководство), чтобы просмотреть документацию в формате PDF со всеми подробностями. (Сводная страница также содержит ссылки на скачивание для установки пакета, но вы редко устанавливаете пакет таким образом; *см.* рецепт 3.10.)

Иногда у вас просто есть общий интерес – например, байесовский анализ, эконометрика, оптимизация или графика. CRAN содержит набор страниц с представлениями задач, описывающих пакеты, которые могут быть полезны. Представление задач – отличное место для начала, поскольку вы получаете обзор того, что доступно. Вы можете просмотреть список страниц с представлениями задач по адресу https://cran.r-project.org/web/views/ или выполнить их поиск, как описано в решении. Представления задач CRAN перечисляют ряд широких сфер деятельности и показывают пакеты, которые используются в каждой области. Например, существуют представления задач для высокопроизводительных вычислений, генетики, временных рядов и социальных наук, и это лишь некоторые из них.

Предположим, вы знаете название полезного пакета, скажем, увидев его в интернете. Полный список пакетов в алфавитном порядке доступен по адресу https://cran.r-project.org/web/packages/.

См. также

Вы можете скачать и установить пакет R под названием `sos`, который предоставляет другие мощные способы поиска пакетов; *см.* https://cran.r-project.org/web/packages/sos/vignettes/sos.pdf.

1.13. Поиск в списках рассылки

Задача

У вас есть вопрос, и вы хотите найти его в архивах списков рассылки, чтобы узнать, был ли ранее дан ответ на ваш вопрос.

Решение

Перейдите на страницу https://r.789695.n4.nabble.com. Выполните поиск по ключевому слову или другому поисковому запросу из вашего вопроса. Вы увидите результаты из списков рассылки поддержки.

Обсуждение

Данный рецепт в действительности является всего лишь приложением рецепта 1.11. Но это важное приложение, потому что вы должны выполнить поиск в архивах списка рассылки, прежде чем отправлять новый вопрос в список. На ваш вопрос, вероятно, уже есть ответ.

См. также

В CRAN есть список дополнительных ресурсов для поиска в сети; *см.* https://cran.r-project.org/search.html.

1.14. Отправка вопросов в Stack Overflow или в другое место в сообществе

Задача

У вас есть вопрос, на который вы не можете найти ответ в интернете, поэтому вы хотите отправить вопрос сообществу R.

Решение

Первое, что необходимо сделать, чтобы задать вопрос онлайн, – создать воспроизводимый пример. Наличие примера кода, который можно выполнить и четко увидеть вашу проблему, является наиболее важной частью обращения за помощью онлайн. Вопрос с хорошим воспроизводимым примером состоит из трех компонентов:

Пример данных

Это могут быть смоделированные данные или какие-либо реальные данные, которые вы предоставляете.

Пример кода

Этот код показывает, что вы пытались сделать что-то, или ошибку, которую вы получаете.

Письменное описание

Здесь вы объясняете, что у вас есть, что бы вы хотели и что вы пробовали сделать, но это не сработало.

Подробности написания воспроизводимого примера приведены в разделе «Обсуждение». Получив воспроизводимый пример, вы можете опубликовать свой вопрос на сайте Stack Overflow (https://stackoverflow.com/users/login?ssrc=anon_ask&returnurl=https%3a%2f%2fstackoverflow.com%2fquestions%2fask). Обязательно включите тег `r` в разделе «Теги» на странице вопросов.

Если ваш вопрос носит более общий характер или связан с концепциями, а не с конкретным синтаксисом, RStudio организует дискуссионный форум сообщества RStudio (https://community.rstudio.com). Обратите внимание, что этот сайт разбит на несколько тем, поэтому выберите категорию тем, которая наилучшим образом соответствует вашему вопросу.

Вы также можете отправить свой вопрос в списки рассылки R (но не отправляйте его на несколько сайтов, в списки рассылки и Stack Overflow, поскольку это воспринимается как грубый кросспостинг).

Страница списков рассылки (https://www.r-project.org/mail.html) содержит общую информацию и инструкции по использованию списка рассылки Rhelp. Вот как выглядит этот процесс в общих чертах:

1. Подпишитесь на основной список рассылки R, R-help (https://stat.ethz.ch/mailman/listinfo/r-help).
2. Тщательно и правильно напишите свой вопрос и включите туда свой воспроизводимый пример.
3. Отправьте свой вопрос по адресу r-help@r-project.org.

Обсуждение

Список рассылки R-help, Stack Overflow и сайт сообщества RStudio – отличные ресурсы, но, пожалуйста, относитесь к ним как к последнему средству. Прочтите страницы справки, документацию, выполните поиск в архивах и в интернете. Скорее всего, на ваш вопрос уже есть ответ. Не обманывайте себя: очень немногие вопросы уникальны. Если вы исчерпали все другие варианты, возможно, пришло время создать хороший вопрос.

Воспроизводимый пример – основа хорошего запроса о помощи. Первый компонент – пример данных. Хороший способ получить его – смоделировать данные, используя несколько функций R.

В приведенном ниже примере мы создали таблицу данных example_df с тремя столбцами, каждый из которых имеет свой тип данных:

```
set.seed(42)
n <- 4
example_df <- data.frame(
   some_reals = rnorm(n),
   some_letters = sample(LETTERS, n, replace = TRUE),
   some_ints = sample(1:10, n, replace = TRUE)
)
example_df
#>    some_reals   some_letters   some_ints
#> 1    1.371          R            10
#> 2    -0.565         S            3
#> 3    0.363          L            5
#> 4    0.633          S            10
```

Обратите внимание, что в этом примере вначале используется команда set.seed. Это гарантирует, что при каждом запуске этого кода ответы будут одинаковыми. Значение n – это число строк примеров данных, которые вы хотели бы создать. Сделайте ваш пример данных максимально простым, чтобы проиллюстрировать свой вопрос.

Альтернативой созданию смоделированных данных является использование примера данных, поставляемых с R. Например, набор данных mtcars содержит таблицу данных с 32 записями о различных моделях автомобилей:

```
data(mtcars)
head(mtcars)
#>                   mpg   cyl disp   hp   drat   wt    qsec   vs   am   gear carb
#> Mazda RX4         21.0   6   160    110 3.90   2.62   16.5   0    1    4      4
#> Mazda RX4 Wag     21.0   6   160    110 3.90   2.88   17.0   0    1    4      4
#> Datsun 710        22.8   4   108    93  3.85   2.32   18.6   1    1    4      1
#> Hornet 4 Drive    21.4   6   258    110 3.08   3.21   19.4   1    0    3      1
#> Hornet Sportabout 18.7   8   360    175 3.15   3.44   17.0   0    0    3      2
#> Valiant           18.1   6   225    105 2.76   3.46   20.2   1    0    3      1
```

Если ваш пример воспроизводим только с вашими собственными данными, вы можете использовать dput, чтобы поместить часть собственных данных в строку, которую можно использовать в своем примере. Мы проиллюстрируем этот подход, используя две строки из набора данных mtcars:

```
dput(head(mtcars, 2))
#> structure(list(mpg = c(21, 21), cyl = c(6, 6), disp = c(160,
```

```
#> 160), hp = c(110, 110), drat = c(3.9, 3.9), wt = c(2.62, 2.875
#> ), qsec = c(16.46, 17.02), vs = c(0, 0), am = c(1, 1), gear = c(4,
#> 4), carb = c(4, 4)), row.names = c("Mazda RX4", "Mazda RX4 Wag"
#> ), class = "data.frame")
```

Вы можете поместить полученную структуру прямо в ваш вопрос:

```
example_df <- structure(list(mpg = c(21, 21), cyl = c(6, 6), disp = c(160,
160), hp = c(110, 110), drat = c(3.9, 3.9), wt = c(2.62, 2.875
), qsec = c(16.46, 17.02), vs = c(0, 0), am = c(1, 1), gear = c(4,
4), carb = c(4, 4)), row.names = c("Mazda RX4", "Mazda RX4 Wag"
), class = "data.frame")

example_df
#>                mpg   cyl disp   hp    drat  wt    qsec  vs  am gear carb
#> Mazda RX4      21    6   160    110   3.9   2.62  16.5  0   1   4    4
#> Mazda RX4 Wag  21    6   160    110   3.9   2.88  17.0  0   1   4    4
```

Вторая часть хорошего воспроизводимого примера – это пример кода. Пример кода должен быть максимально простым и иллюстрировать то, что вы пытаетесь сделать или уже пробовали сделать. Это *не* должен быть большой блок кода со множеством разных вещей. Сделайте так, чтобы пример содержал минимальное количество необходимого кода. Если вы используете какие-либо пакеты, обязательно включите вызов `library` в начале своего кода. Кроме того, не включайте в свой вопрос ничего, что могло бы нанести вред человеку, выполняющему ваш код, например `rm(list = ls())`, что могло бы удалить все объекты R в памяти. Имейте сочувствие к человеку, пытающемуся помочь вам, и поймите, что он добровольно жертвует свое время, чтобы помочь вам, и может выполнять ваш код на той же машине, которую использует для выполнения своей работы.

Чтобы проверить свой пример, откройте новый сеанс R и попробуйте запустить его. После того как вы отредактировали свой код, пришло время дать немного больше информации потенциальным респондентам. В текстовом виде опишите, что вы пытаетесь сделать, что пытались сделать и свой вопрос. Будьте максимально лаконичны. Как и в примере кода, ваша цель – максимально эффективно общаться с человеком, читающим ваш вопрос. Может быть полезно указать в своем описании, какую версию R вы используете и какую платформу (Windows, Mac, Linux). Эту информацию можно легко получить с помощью команды `sessionInfo`.

Если вы собираетесь отправить свой вопрос в список рассылки R, вы должны знать, что на самом деле существует несколько таких списков. R-help – это основной список для общих вопросов. Существует также множество списков рассылки специальных групп интересов (SIG), посвященных конкретным областям, таким как генетика, финансы, разработка на языке R и даже вакансии для разработчиков. Полный список можно увидеть на странице https://stat.ethz.ch/mailman/listinfo. Если ваш вопрос относится к какой-то области, вы получите более подходящий ответ, выбрав соответствующий список. Однако, как и в случае с R-help, внимательно выполняйте поиск в архивах списков SIG, прежде чем задавать свой вопрос.

См. также

Мы предлагаем вам прочитать превосходное эссе Эрика Рэймонда и Рика Моена под названием «Как правильно задавать вопросы» (http://www.catb.org/~esr/faqs/smart-questions.html), перед тем как отправлять какие-либо вопросы. Серьезно. Прочтите его.

На сайте Stack Overflow есть отличный пост, в котором содержатся подробные сведения о создании воспроизводимого примера. Его можно найти на странице https://stackoverflow.com/q/5963269/37751.

У Дженни Брайан есть замечательный пакет под названием `reprex`, который помогает создать хороший воспроизводимый пример и предоставляет вспомогательные функции для написания текста на языке разметки Markdown для таких сайтов, как Stack Overflow. Этот пакет можно найти на ее странице в GitHub (https://github.com/tidyverse/reprex).

Глава 2

Немного основ

Рецепты, приведенные в этой главе, лежат где-то между идеями для решения конкретных задач и учебными пособиями. Да, они решают распространенные проблемы, но решения демонстрируют общепринятые методы и шаблоны, используемые в большей части кода, написанного на языке R, включая код, который содержится в этой книге рецептов. Если вы новичок в R, мы рекомендуем прочитать данную главу, чтобы познакомиться с этими рецептами.

2.1. Вывод на экран

Задача

Вам нужно отобразить значение переменной или выражения.

Решение

Если вы просто введете имя переменной или выражения в командной строке, R выведет их значение. Используйте функцию `print` для стандартного вывода любого объекта. Используйте функцию `cat` для создания вывода в произвольном формате.

Обсуждение

Попросить R вывести что-нибудь на экран очень просто – просто наберите это в командной строке:

```
pi
#> [1] 3.14
sqrt(2)
#> [1] 1.41
```

Когда вы вводите выражения, подобные этим, R вычисляет выражение и затем неявно вызывает функцию `print`. Поэтому предыдущий пример идентичен этому:

```
print(pi)
#> [1] 3.14
print(sqrt(2))
#> [1] 1.41
```

Прелесть функции `print` состоит в том, что она умеет форматировать любое значение R для вывода, включая такие сложные структуры, как матрицы и списки:

```
print(matrix(c(1, 2, 3, 4), 2, 2))
#>      [,1] [,2]
#> [1,]  1    3
#> [2,]  2    4
print(list("a", "b", "c"))
#> [[1]]
#> [1] "a"
#>
#> [[2]]
#> [1] "b"
#>
#> [[3]]
#> [1] "c"
```

Это полезно, потому что вы всегда можете просмотреть свои данные: просто напечатайте (`print`) их. Не нужно программировать логику вывода даже для сложных структур данных.

Однако у функции `print` есть существенное ограничение: она выводит только один объект за раз. При попытке вывести несколько элементов выдается следующее сообщение об ошибке:

```
print("The zero occurs at", 2 * pi, "radians.")
#> Error in print.default("The zero occurs at", 2 * pi, "radians."):
#> invalid 'quote' argument
```

Единственный способ вывести несколько элементов – выводить их по одному, однако, вероятно, это не то, что вам нужно:

```
print("The zero occurs at")
#> [1] "The zero occurs at"
print(2 * pi)
#> [1] 6.28
print("radians")
#> [1] "radians"
```

Функция `cat` является альтернативой `print`. Она позволяет выводить несколько элементов одновременно:

```
cat("The zero occurs at", 2 * pi, "radians.", "\n")
#> The zero occurs at 6.28 radians.
```

Обратите внимание, что по умолчанию функция `cat` ставит пробел элементами списка. Для перевода строки необходимо явно добавить символ новой строки (\n).

Функция `cat` также может выводить простые векторы:

```
fib <- c(0, 1, 1, 2, 3, 5, 8, 13, 21, 34)
cat("The first few Fibonacci numbers are:", fib, "...\n")
#> The first few Fibonacci numbers are: 0 1 1 2 3 5 8 13 21 34 ...
```

Использование функции `cat` дает вам больше контроля над выводом, что делает ее особенно полезной в программных сценариях, написанных на языке R, генерирующих текстовый вывод, который используется внешними обработчиками. Однако серьезным ограничением является то, что она не может выводить составные структуры данных, такие как матрицы и списки. Попытка использовать для них функцию `cat` приводит к появлению еще одного невнятного сообщения:

```
cat(list("a", "b", "c"))
#> Error in cat(list("a", "b", "c")): argument 1 (type 'list') cannot
#>    be handled by 'cat'
```

См. также

См. рецепт 4.2, чтобы узнать, как управлять форматом вывода.

2.2. Установка переменных

Задача

Вы хотите сохранить значение в переменной.

Решение

Используйте оператор присваивания (<-). Объявлять переменную нет необходимости:

```
x <- 3
```

Обсуждение

Использование R в «режиме калькулятора» быстро надоедает. Вскоре вам понадобится определить переменные и сохранить в них значения. Это избавляет вас от лишних нажатий клавиш, экономит время и упрощает вам работу.

Нет необходимости объявлять или явно создавать переменные в R. Просто присвойте имени значение, и R создаст переменную:

```
x <- 3
y <- 4
z <- sqrt(x^2 + y^2)
print(z)
#> [1] 5
```

Обратите внимание, что оператор присваивания состоит из символа «меньше, чем» (<) и дефиса (-) без пробела между ними.

Когда вы определяете переменную в командной строке таким образом, она сохраняется в вашей рабочей области. Рабочая область хранится в основной памяти компьютера, но ее можно сохранить на диск. Определение переменной остается в рабочей области, пока вы ее не удалите.

R – это *динамически типизированный язык*. Это означает, что мы можем изменить тип данных переменной по желанию. Мы можем присвоить числовое значение, как было показано только что, а затем передумать и сразу же перезаписать его, скажем, используя вектор символьных строк. R не будет жаловаться:

```
x <- 3
print(x)
#> [1] 3

x <- c("fee", "fie", "foe", "fum")
print(x)
#> [1] "fee" "fie" "foe" "fum"
```

В некоторых функциях R вы увидите операторы присваивания, которые используют оператор присваивания странного вида << -:

```
x <<- 3
```

Это вызывает присвоение глобальной переменной, а не локальной. Понятие области видимости переменных немного выходит за рамки данного обсуждения.

R также поддерживает две другие формы операторов присваивания. В качестве оператора присваивания можно использовать знак равенства (=). Оператор правостороннего присваивания (->) можно использовать везде, где может использоваться оператор левостороннего присваивания (<-) (но с перевернутыми аргументами):

```
foo <- 3
print(foo)
#> [1] 3
5 -> fum
print(fum)
#> [1] 5
```

Мы не рекомендуем использовать эти альтернативы. Оператор присваивания в виде знака равенства легко спутать с тестом на равенство. Правостороннее присваивание может быть полезным в определенных контекстах, но может ввести в заблуждение тех, кто не привык к нему.

См. также

См. рецепты 2.4, 2.14 и 3.3. Также *см.* страницу справки для функции `assign`.

2.3. Перечисление переменных

Задача

Вы хотите знать, какие переменные и функции определены в вашем рабочем пространстве.

Решение

Используйте функцию `ls`. Используйте функцию `ls.str` для получения более подробной информации о каждой переменной. Вы также можете увидеть свои переменные и функции на панели **Environment** в RStudio, как показано в следующем рецепте на рис. 2-1.

Обсуждение

Функция `ls` отображает имена объектов в вашем рабочем пространстве:

```
x <- 10
y <- 50
z <- c("three", "blind", "mice")
f <- function(n, p) sqrt(p * (1 - p) / n)
ls()
#> [1] "f" "x" "y" "z"
```

Обратите внимание, что функция `ls` возвращает вектор символьных строк, где каждая строка является именем одной из переменных или функций. Когда ваше рабочее пространство пусто, `ls` возвращает пустой вектор, что приводит к странному сообщению:

```
ls()
#> character(0)
```

R использует это сообщение, чтобы сказать, что `ls` вернула вектор строк нулевой длины; то есть она вернула пустой вектор, потому что в вашей рабочей области ничего не определено.

Если вам нужно нечто большее, чем просто список имен, попробуйте функцию `ls.str`; она также расскажет вам кое-что о каждой переменной:

```
x <- 10
y <- 50
z <- c("three", "blind", "mice")
f <- function(n, p) sqrt(p * (1 - p) / n)
ls.str()
#> f : function (n, p)
#> x : num 10
#> y : num 5
#> z : chr [1:3] "three" "blind" "mice"
```

Эта функция носит название `ls.str`, потому что она перечисляет ваши переменные, одновременно с этим применяя к ним функцию `str`, показывая их структуру (см. рецепт 12.13).

Обычно функция `ls` не возвращает имя, начинающееся с точки (.). Такие имена считаются скрытыми и обычно не представляют интереса для пользователей. (Это соответствует общепринятой в Unix практике об игнорировании файлов, имена которых начинаются с точки.) Можно заставить `ls` перечислить все переменные, установив для аргумента `all.names` значение `TRUE`:

```
ls()
#> [1] "f" "x" "y" "z"
ls(all.names = TRUE)
#> [1] ".Random.seed" "f"    "x"    "y"
#> [5] "z"
```

Панель **Environment** в RStudio также скрывает объекты с именами, которые начинаются с точки.

См. также

См. рецепт 2.4, чтобы узнать, как удалять переменные, и рецепт 12.13, чтобы узнать, как проверить свои переменные.

2.4. Удаление переменных

Задача

Вам нужно удалить ненужные переменные или функции из вашего рабочего пространства либо полностью удалить его содержимое.

Решение

Используйте функцию `rm`.

Обсуждение

Ваше рабочее пространство может быстро загромождаться. Функция `rm` удаляет один или несколько объектов из рабочей области без возможности восстановления:

```
x <- 2 * pi
x
#> [1] 6.28
rm(x)
x
#> Error in eval(expr, envir, enclos): object 'x' not found
```

Здесь нет опции **Отменить**; как только переменная исчезла, это навсегда. Можно удалить несколько переменных одновременно:

```
rm(x, y, z)
```

Можно даже стереть всю свою рабочую область сразу. У функции `rm` есть аргумент `list`, состоящий из вектора имен переменных, которые нужно удалить. Напомним, что функция `ls` возвращает вектор имен переменных; следовательно, вы можете комбинировать функции `rm` и `ls`, чтобы стереть все:

```
ls()
#> [1] "f" "x" "y" "z"
rm(list = ls())
ls()
#> character(0)
```

Кроме того, можно щелкнуть значок метлы в верхней части панели **Environment** в RStudio, как показано на рис. 2-1.

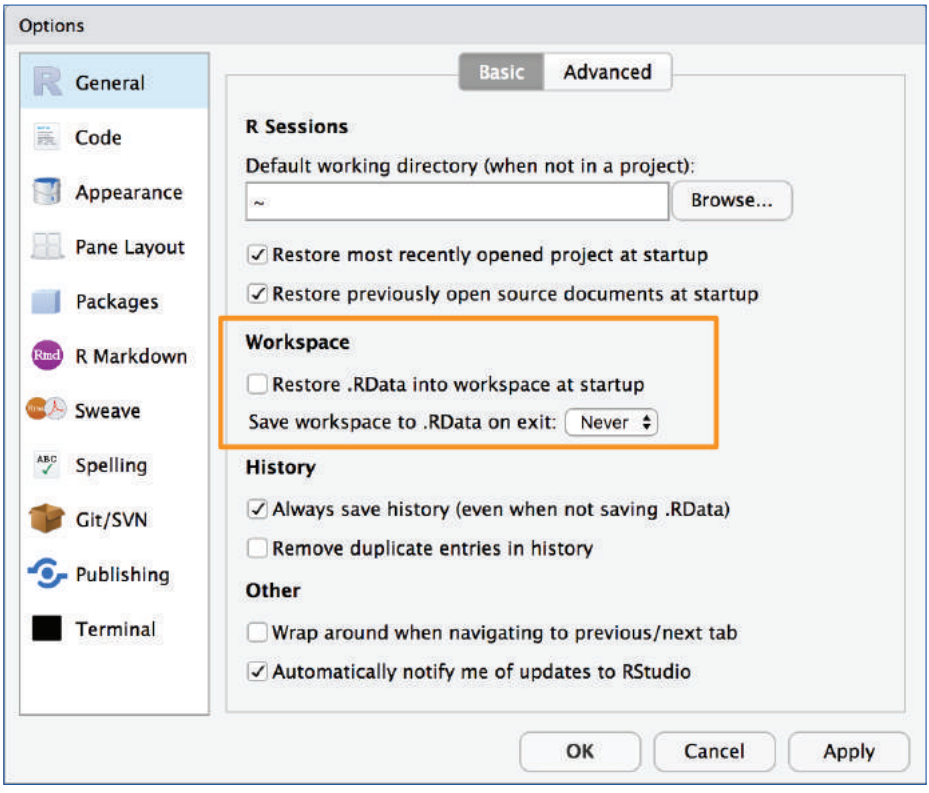


Рис. 2-1. Панель **Environment** в RStudio

Никогда не добавляйте `rm(list=ls())` в код, которым вы делитесь с другими, например в функцию библиотеки или пример кода, который вы отправляете в список рассылки или на сайт Stack Overflow. Удаление всех переменных в чужом рабочем пространстве хуже, чем просто грубость. Это сделает вас крайне непопулярным.

См. также

См. рецепт 2.3.

2.5. Создание вектора

Задача

Вам нужно создать вектор.

Решение

Используйте оператор `c(...)` для построения вектора из заданных значений.

Обсуждение

Векторы являются основными объектами R, а не просто еще одной структурой данных. Вектор может содержать числа, строки или логические значения, но не их смесь.

Оператор `c(...)` может построить вектор из простых элементов:

```
c(1, 1, 2, 3, 5, 8, 13, 21)
#> [1] 1 1 2 3 5 8 13 21
c(1 * pi, 2 * pi, 3 * pi, 4 * pi)
#> [1] 3.14 6.28 9.42 12.57
c("My", "twitter", "handle", "is", "@cmastication")
#> [1] "My"         "twitter"          "handle"         "is"
#> [5] "@cmastication"
c(TRUE, TRUE, FALSE, TRUE)
#> [1] TRUE TRUE FALSE TRUE
```

Если аргументы для `c(...)` сами по себе являются векторами, он унифицирует их и объединяет в один вектор:

```
v1 <- c(1, 2, 3)
v2 <- c(4, 5, 6)
c(v1, v2)
#> [1] 1 2 3 4 5 6
```

Векторы не могут совмещать разные типы данных, таких как числа и строки. Если вы создадите вектор из смешанных элементов, R попытается угодить вам, преобразовав один из них:

```
v1 <- c(1, 2, 3)
v3 <- c("A", "B", "C")
c(v1, v3)
#> [1] "1" "2" "3" "A" "B" "C"
```

Здесь мы попытались создать вектор из чисел и строк. Перед созданием вектора R преобразовал все числа в строки, что сделало элементы данных совмести-

мыми. Обратите внимание, что R делает это без какого-либо предупреждения или жалобы.

Говоря технически, два элемента данных могут сосуществовать в векторе, только если у них одинаковый *тип*. Типы `3.1415` и `"foo"` – это `numeric` и `character` соответственно:

```
mode(3.1415)
#> [1] "numeric"
mode("foo")
#> [1] "character"
```

Эти типы несовместимы. Чтобы сделать из них вектор, R преобразует `3.1415` в тип `character`, чтобы он был совместим с `"foo"`:

```
c(3.1415, "foo")
#> [1] "3.1415" "foo"
mode(c(3.1415, "foo"))
#> [1] "character"
```

`c` – это универсальный оператор, а это означает, что он работает со многими типами данных, а не только с векторами. Тем не менее он может делать не совсем то, что вы ожидаете, поэтому проверьте его поведение, прежде чем применять его к другим типам данных и объектам.

См. также

См. введение к главе 5 для получения дополнительной информации о векторах и других структурах данных.

2.6. Базовая статистика

Задача

Вам нужно рассчитать базовую статистику: среднее значение, медиану, стандартное отклонение, дисперсию, корреляцию или ковариацию.

Решение

Используйте одну из этих функций, предполагая, что `x` и `y` – это векторы:

- `mean(x)`
- `median(x)`
- `sd(x)`
- `var(x)`
- `cor(x, y)`
- `cov(x, y)`

Обсуждение

Когда вы впервые используете R, вы, вероятно, открываете документацию и начинаете поиск материала, озаглавленного «Расчет стандартного отклонения». Похоже, что для такой важной темы, скорее всего, потребуется целая глава.

Но не всё так сложно.

Стандартное отклонение и другие основные статистические данные рассчитываются с помощью простых функций. Обычно аргумент функции представляет собой вектор чисел, и функция возвращает вычисленную статистику:

```
x <- c(0, 1, 1, 2, 3, 5, 8, 13, 21, 34)
mean(x)
#> [1] 8.8
median(x)
#> [1] 4
sd(x)
#> [1] 11
var(x)
#> [1] 122
```

Функция `sd` вычисляет стандартное отклонение выборки, а функция `var` вычисляет дисперсию.

Соответственно, функции `cor` и `cov` могут вычислять корреляцию и ковариацию между двумя векторами:

```
x <- c(0, 1, 1, 2, 3, 5, 8, 13, 21, 34)
y <- log(x + 1)
cor(x, y)
#> [1] 0.907
cov(x, y)
#> [1] 11.5
```

Все эти функции не терпимы к значениям, которые недоступны или пропущены (NA). Даже одно значение NA в аргументе вектора заставляет любую из этих функций возвращать NA или вообще прекращать работу с загадочной диагностикой:

```
x <- c(0, 1, 1, 2, 3, NA)
mean(x)
#> [1] NA
sd(x)
#> [1] NA
```

Когда R настолько осторожен, это раздражает, но это уместно. Тщательно обдумайте данную ситуацию. Делает ли присутствие в ваших данных значений NA вашу статистику бессмысленной? Если да, то R поступает правильно. Если нет, вы можете переопределить это поведение, установив для `na.rm` значение `TRUE`, что говорит R игнорировать значения NA:

```
x <- c(0, 1, 1, 2, 3, NA)
sd(x, na.rm = TRUE)
#> [1] 1.14
```

В более старых версиях R функции `mean` и `sd` вели себя по-умному в отношении таблиц данных. Они понимали, что каждый столбец таблицы данных – это отдельная переменная, поэтому рассчитывали свою статистику для каждого столбца в отдельности. Теперь это уже не так, и, как следствие, вы можете прочитать запутанные комментарии в интернете или в старых книгах (например, в первом издании этой книги). Чтобы применить функции к каждому столбцу таблицы данных, теперь нам нужно использовать вспомогательную функцию. Семейство вспомогательных функций *tidyverse* для такого рода процедуры находится в паке-

те purrr. Как и в случае с другими пакетами *tidyverse*, он загружается при запуске library(tidyverse). Функция, которую мы будем использовать для применения функции к каждому столбцу таблицы данных, – map_dbl:

```
data(cars)

map_dbl(cars, mean)
#> speed dist
#> 15.4 43.0
map_dbl(cars, sd)
#> speed dist
#> 5.29 25.77
map_dbl(cars, median)
#> speed  dist
#>    15    36
```

Обратите внимание, что в этом примере функции mean и sd возвращают два значения, по одному для каждого столбца, определенного таблицей данных. (Технически они возвращают двухэлементный вектор, чей атрибут name взят из столбцов таблицы данных.)

Функция var понимает таблицы данных без помощи функции отображения. Она вычисляет ковариацию между столбцами таблицы данных и возвращает ковариационную матрицу:

```
var(cars)
#>        speed  dist
#> speed    28    110
#> dist    110    664
```

Аналогично, если x – это таблица данных либо матрица, то cor(x) возвращает матрицу корреляции, а cov(x) возвращает ковариационную матрицу:

```
cor(cars)
#>        speed  dist
#> speed  1.000 0.807
#> dist   0.807 1.000
cov(cars)
#>        speed dist
#> speed  28    110
#> dist   110   664
```

См. также

См. рецепты 2.14, 5.27 и 9.17.

2.7. Создание последовательностей

Задача

Вам нужно создать последовательность чисел.

Решение

Используйте выражение n:m для создания простой последовательности $n, n + 1, n + 2, \ldots, m$:

```
1:5
#> [1] 1 2 3 4 5
```

Используйте функцию seq для последовательностей с шагом, отличным от 1:

```
seq(from = 1, to = 5, by = 2)
#> [1] 1 3 5
```

Используйте функцию rep, чтобы создать серию повторяющихся значений:

```
rep(1, times = 5)
#> [1] 1 1 1 1 1
```

Обсуждение

Оператор двоеточия (n:m) создает вектор, содержащий последовательность *n, n + 1, n + 2, , m*:

```
0:9
#> [1] 0 1 2 3 4 5 6 7 8 9
10:19
#> [1] 10 11 12 13 14 15 16 17 18 19
9:0
#> [1] 9 8 7 6 5 4 3 2 1 0
```

R повел себя хитро с последним выражением (9: 0). Поскольку 9 больше 0, то последовательность отсчитывается в обратном порядке. Вы также можете использовать оператор двоеточия напрямую с каналом для передачи данных в другую функцию:

```
10:20 %>% mean()
```

Оператор двоеточия работает для последовательностей, которые увеличиваются только на единицу. Функция seq также создает последовательности, но поддерживает необязательный третий аргумент – инкремент или приращение:

```
seq(from = 0, to = 20)
#> [1] 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
seq(from = 0, to = 20, by = 2)
#> [1] 0 2 4 6 8 10 12 14 16 18 20
seq(from = 0, to = 20, by = 5)
#> [1] 0 5 10 15 20
```

В качестве альтернативы можно указать длину для выходной последовательности, а затем R рассчитает необходимый прирост:

```
seq(from = 0, to = 20, length.out = 5)
#> [1] 0 5 10 15 20
seq(from = 0, to = 100, length.out = 5)
#> [1] 0 25 50 75 100
```

Приращение не обязательно должно быть целым числом. R также может создавать последовательности с дробными приращениями:

```
seq(from = 1.0, to = 2.0, length.out = 5)
#> [1] 1.00 1.25 1.50 1.75 2.00
```

Для особого случая «последовательности», которая является просто повторяющимся значением, следует использовать функцию rep, которая повторяет свой первый аргумент:

```
rep(pi, times = 5)
#> [1] 3.14 3.14 3.14 3.14 3.14
```

См. также

См. рецепт 7.13, чтобы узнать, как создавать последовательности объектов `Date`.

2.8. Сравнение векторов

Задача

Вам нужно сравнить два вектора или сравнить целый вектор со скаляром.

Решение

Операторы сравнения (==,! =, <,>, <=,> =) могут выполнять поэлементное сравнение двух векторов. Они также могут сравнивать элемент вектора со скаляром. Результатом является вектор логических значений, в котором каждое значение – это результат одного поэлементного сравнения.

Обсуждение

В R есть два логических значения, `TRUE` и `FALSE`. В других языках программирования их часто называют *булевыми* значениями.

Операторы сравнения сравнивают два значения и возвращают `TRUE` или `FALSE`, в зависимости от результата сравнения:

```
a <- 3
a == pi # Проверка на наличие равенства.
#> [1] FALSE
a != pi # Проверка на наличие неравенства.
#> [1] TRUE
a < pi
#> [1] TRUE
a > pi
#> [1] FALSE
a <= pi
#> [1] TRUE
a >= pi
#> [1] FALSE
```

Вы можете испытать мощь R, сравнивая целые векторы одновременно. R выполнит поэлементное сравнение и вернет вектор логических значений, по одному для каждого сравнения:

```
v <- c(3, pi, 4)
w <- c(pi, pi, pi)
v == w # Сравниваем два 3-элементных вектора.
#> [1] FALSE TRUE FALSE
v != w
#> [1] TRUE FALSE TRUE
v < w
#> [1] TRUE FALSE FALSE
v <= w
#> [1] TRUE TRUE FALSE
v > w
#> [1] FALSE FALSE TRUE
v >= w
#> [1] FALSE TRUE TRUE
```

Вы также можете сравнить вектор с одним скаляром, и в этом случае R расширит скаляр до длины вектора, а затем выполнит поэлементное сравнение. Предыдущий пример можно упростить:

```
v <- c(3, pi, 4)
v == pi # Сравниваем 3-элементный вектор с одним числом.
#> [1] FALSE TRUE FALSE
v != pi
#> [1] TRUE FALSE TRUE
```

Это применение правила повторного использования, описанного в рецепте 5.3.

После сравнения двух векторов вам часто нужно узнать, было ли *какое-либо* сравнение истинным или *все* сравнения были истинными. Функции `any` и `all` выполняют эти тесты. Они проверяют логический вектор. Функция `any` возвращает `TRUE`, если какой-либо элемент вектора равен `TRUE`. Функция `all` возвращает `TRUE`, если все элементы вектора равны `TRUE`:

```
v <- c(3, pi, 4)
any(v == pi)    # Возвращаем TRUE, если любой элемент v равен pi.
#> [1] TRUE
all(v == 0)     # Возвращаем TRUE, если все элементы v равны нулю.
#> [1] FALSE
```

См. также

См. рецепт 2.9.

2.9. Выбор элементов вектора

Задача

Вы хотите извлечь один или несколько элементов из вектора.

Решение

Выберите метод индексации, подходящий для вашей задачи:

- используйте квадратные скобки, чтобы выбрать элементы вектора по их положению, например v [3] для третьего элемента v;
- применяйте отрицательные индексы для исключения элементов;
- используйте вектор индексов для выбора нескольких значений;
- применяйте логический вектор для выбора элементов на основе условия;
- используйте имена для доступа к именованным элементам.

Обсуждение

Выбор элементов из векторов – это еще одна мощная особенность R. Основной выбор выполняется так же, как и во многих других языках программирования, – используйте квадратные скобки и простой индекс:

```
fib <- c(0, 1, 1, 2, 3, 5, 8, 13, 21, 34)
fib
#> [1] 0 1 1 2 3 5 8 13 21 34
fib[1]
#> [1] 0
```

```
fib[2]
#> [1] 1
fib[3]
#> [1] 1
fib[4]
#> [1] 2
fib[5]
#> [1] 3
```

Обратите внимание, что первый элемент имеет индекс 1, а не 0, как в некоторых других языках программирования.

Отличная особенность векторной индексации состоит в том, что вы можете выбрать несколько элементов одновременно. Сам индекс может быть вектором, и каждый элемент этого вектора индекса выбирает элемент из вектора данных:

```
fib[1:3]      # Выбираем элементы с 1 по 3.
#> [1] 0 1 1
fib[4:9]      # Выбираем элементы с 4 по 9.
#> [1] 2 3 5 8 13 21
```

Индекс 1: 3 означает выбор элементов 1, 2 и 3, как было показано выше. Однако индексный вектор не должен быть простой последовательностью. Вы можете выбрать элементы в любом месте вектора данных – как в этом примере, где выбираются элементы 1, 2, 4 и 8:

```
fib[c(1, 2, 4, 8)]
#> [1] 0 1 2 13
```

R использует отрицательные индексы для исключения данных. Например, индекс –1 означает исключить первое значение и вывести все остальные:

```
fib[-1] # Исключить первый элемент вектора
#> [1] 1 1 2 3 5 8 13 21 34
```

Можно расширить этот метод, чтобы исключить целые фрагменты, используя вектор индексации отрицательных индексов:

```
fib[1:3]               # Как и прежде.
#> [1] 0 1 1
fib[-(1:3)]            # Инвертируем знак индекса, чтобы исключить, а не выбирать.
#> [1] 2 3 5 8 13 21 34
```

Еще один метод индексации использует логический вектор для выбора элементов из вектора данных. Везде, где логический вектор равен `TRUE`, выбирается соответствующий элемент:

```
fib < 10            # Этот вектор имеет значение TRUE везде, где fib меньше 10.
#> [1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE FALSE FALSE FALSE
fib[fib < 10]       # Используем этот вектор для выбора элементов меньше 10.
#> [1] 0 1 1 2 3 5 8
fib %% 2 == 0       # Этот вектор имеет значение TRUE везде, где fib четный.
#> [1] TRUE FALSE FALSE TRUE FALSE FALSE TRUE FALSE FALSE TRUE
fib[fib %% 2 == 0]  # Используем этот вектор для выбора четных элементов.
#> [1] 0 2 8 34
```

Обычно логический вектор должен иметь ту же длину, что и вектор данных, чтобы вы ясно включали или исключали каждый элемент. (Если длины раз-

личаются, тогда вам необходимо изучить правило повторного использования, обсуждаемое в рецепте 5.3.)

Комбинируя векторные сравнения, логические операторы и векторную индексацию, можно реализовывать сложные процедуры отбора с помощью малого количества кода.

Например, можно выбрать все элементы больше медианы:

```
v <- c(3, 6, 1, 9, 11, 16, 0, 3, 1, 45, 2, 8, 9, 6, -4)
v[ v > median(v)]
#> [1] 9 11 16 45 8 9
```

или выбрать все элементы в нижних и верхних 5 %:

```
v[ (v < quantile(v, 0.05)) | (v > quantile(v, 0.95)) ]
#> [1] 45 -4
```

В предыдущем примере используется оператор |. Это означает «или» при индексации. Если вам нужно «и», используйте оператор &.

Вы также можете выбрать все элементы, которые превышают ±1 стандартное отклонение от среднего значения:

```
v[ abs(v - mean(v)) > sd(v)]
#> [1] 45 -4
```

или выберите все элементы, которые не являются ни NA, ни NULL:

```
v <- c(1, 2, 3, NA, 5)
v[!is.na(v) & !is.null(v)]
#> [1] 1 2 3 5
```

Одна последняя функция индексации позволяет выбирать элементы по имени. Предполагается, что у вектора есть атрибут names, определяющий имя каждого элемента. Вы можете определить имена, присвоив атрибуту вектор символьных строк:

```
years <- c(1960, 1964, 1976, 1994)
names(years) <- c("Kennedy", "Johnson", "Carter", "Clinton")
years
#> Kennedy Johnson Carter Clinton
#>  1960    1964    1976    1994
```

Как только имена определены, вы можете ссылаться на отдельные элементы по имени:

```
years["Carter"]
#> Carter
#> 1976
years["Clinton"]
#> Clinton
#> 1994
```

Это также разрешает индексацию по векторам имен; R возвращает каждый элемент, указанный в индексе:

```
years[c("Carter", "Clinton")]
#> Carter Clinton
#>  1976    1994
```

См. также

См. рецепт 5.3 для получения дополнительной информации о правиле повторного использования .

2.10. Векторная арифметика

Задача

Вы хотите работать с целым вектором одновременно.

Решение

Обычные арифметические операторы могут выполнять поэлементные операции над целыми векторами. Многие функции также работают с целыми векторами и возвращают векторный результат.

Обсуждение

Векторные операции являются одной из сильных сторон R. Все основные арифметические операторы могут быть применены к векторным парам. Они действуют поэлементно; то есть оператор применяется к соответствующим элементам из обоих векторов:

```
v <- c(11, 12, 13, 14, 15)
w <- c(1, 2, 3, 4, 5)
v + w
#> [1] 12 14 16 18 20
v - w
#> [1] 10 10 10 10 10
v * w
#> [1] 11 24 39 56 75
v / w
#> [1] 11.00 6.00 4.33 3.50 3.00
w^v
#> [1] 1.00e+00 4.10e+03 1.59e+06 2.68e+08 3.05e+10
```

Заметьте, что длина результата здесь равна длине исходных векторов. Причина состоит в том, что каждый элемент происходит из пары соответствующих значений во входных векторах.

Если один из операндов – это вектор, а другой – скаляр, то операция выполняется между каждым элементом вектора и скаляром:

```
w
#> [1] 1 2 3 4 5
w + 2
#> [1] 3 4 5 6 7
w - 2
#> [1] -1 0 1 2 3
w * 2
#> [1] 2 4 6 8 10
w / 2
#> [1] 0.5 1.0 1.5 2.0 2.5
2^w
#> [1] 2 4 8 16 32
```

Например, можно повторно центрировать весь вектор в одном выражении, просто вычтя среднее его содержимого:

```
w
#> [1] 1 2 3 4 5
mean(w)
#> [1] 3
w - mean(w)
#> [1] -2 -1 0 1 2
```

Аналогичным образом можно вычислить стандартизованную *z*-оценку вектора в одном выражении – вычесть среднее и разделить на стандартное отклонение:

```
w
#> [1] 1 2 3 4 5
sd(w)
#> [1] 1.58
(w - mean(w)) / sd(w)
#> [1] -1.265 -0.632 0.000 0.632 1.265
```

И все же реализация операций на векторном уровне выходит далеко за рамки элементарной арифметики. Она пронизывает язык, а многие функции работают с целыми векторами. Например, функции `sqrt` и `log` применяются к каждому элементу вектора и возвращают вектор результатов:

```
w <- 1:5
w
#> [1] 1 2 3 4 5
sqrt(w)
#> [1] 1.00 1.41 1.73 2.00 2.24
log(w)
#> [1] 0.000 0.693 1.099 1.386 1.609
sin(w)
#> [1] 0.841 0.909 0.141 -0.757 -0.959
```

У векторизованных операций есть два больших преимущества. Первое и самое очевидное – это удобство. Операции, которые требуют организации циклов в других языках, в R являются однострочными сценариями.

Второе – это скорость. Большинство векторизованных операций реализуются непосредственно в коде C, поэтому они значительно быстрее, чем эквивалентный код на языке R, который вы могли бы написать.

См. также

Выполнение операции между вектором и скаляром на самом деле является частным случаем правила повторного использования; *см.* рецепт 5.3.

2.11. Разбираемся с приоритетом оператора

Задача

Ваше выражение на языке R дает любопытный результат, и вы задаетесь вопросом, не является ли приоритет оператора причиной проблем.

Решение

Полный список операторов приводится в табл. 2-1, в порядке приоритета от высшего к низшему. Операторы с одинаковым приоритетом оцениваются слева направо, кроме случаев, где это указано.

Таблица 2-1. Приоритет оператора

Оператор	Значение	См. также
[[[	Индексирование	Рецепт 2.9
:: :::	Обращение к переменным в пространстве имен (окружение)	
$ @	Извлечение элементов, извлечение атрибутов	
^	Возведение в степень (справа налево)	
- +	Одинарный минус и плюс	
:	Создание последовательности	Рецепты 2.7, 7.13
%any% (включая %>%)	Специальные операторы	Обсуждение (в этом рецепте)
* /	Умножение, деление	Обсуждение (в этом рецепте)
+ –	Сложение, вычитание	
== != < > <= >=	Сравнение	Рецепт 2.8
!	Логическое отрицание	
& &&	Логическое «и», сокращенное «и»	
| ||	Логическое «или», сокращенное «или»	
~	Формула	Рецепт 11.1
-> ->>	Правостороннее присваивание	Рецепт 2.2
=	Присваивание (справа налево)	Рецепт 2.2
<- <<-	Присваивание (справа налево)	Рецепт 2.2
?	Справка	Рецепт 1.8

Не важно, знаете ли вы, что делает каждый из этих операторов или что они значат. Приведенный здесь список предназначен просто для того, чтобы вы поняли, что разные операторы имеют разный приоритет.

Обсуждение

Неправильное определение приоритета оператора в R является распространенной ошибкой. Такое, конечно же, часто происходит с нами. Мы бездумно ожидаем, что выражение `0:n-1` создаст последовательность целых чисел от 0 до n – 1, но это не так:

```
n <- 10
0:n - 1
#>   [1] -1 0 1 2 3 4 5 6 7 8 9
```

Оно создает последовательность от –1 до n – 1, потому что R интерпретирует его как `(0:n)-1`.

Вы можете не распознать нотацию `%any%`, приведенную в таблице. R интерпретирует любой текст между знаками процента (%...%) как бинарный оператор. Несколько таких операторов имеют предопределенные значения:

%%

Оператор модуля.

% /%

Целочисленное деление.

% *%

Перемножение матриц.

%in%

Возвращает значение TRUE, если левый операнд встречается в его правом операнде; в противном случае – FALSE.

%>%

Конвейер, который передает результаты слева в функцию справа.

Вы также можете определить новые двоичные операторы, используя обозначение %...%; *см.* рецепт 12.17. Дело в том, что все эти операторы имеют одинаковый приоритет.

См. также

См. рецепт 2.10 для получения дополнительной информации об операциях с векторами, рецепт 5.15 для получения дополнительной информации об операциях с матрицами и рецепт 12.17, чтобы узнать, как определить собственные операторы. *См.* также разделы «Арифметика» и «Синтаксис» на страницах справки R, главы 5 и 6 книги *R in a Nutshell* (http://shop.oreilly.com/product/9780596801717.do).

2.12. Меньше печатать и больше делать

Задача

Вы устали набирать длинные последовательности команд, и особенно вам надоело вводить одно и то же снова и снова.

Решение

Откройте окно редактора и добавьте туда свои повторно используемые блоки команд R. Затем выполните эти блоки непосредственно из этого окна. Зарезервируйте окно консоли для ввода кратких или одноразовых команд.

Когда вы закончите, то можете сохранить накопленные блоки кода в файле сценария для дальнейшего использования.

Обсуждение

Типичный новичок в R набирает выражение в окне консоли и видит, что происходит. По мере того как он осваивается, он набирает все более сложные выражения. Затем начинает печатать многострочные выражения. Вскоре он набирает одни и те же многострочные выражения снова и снова, возможно, с небольшими вариациями, чтобы выполнять свои все более сложные вычисления.

Опытный пользователь R редко набирает сложное выражение. Он может напечатать одно и то же выражение один или два раза, но когда поймет, что оно полезно и его можно использовать повторно, он вырезает его и вставляет в окно редактора. Чтобы выполнить позже этот фрагмент кода, он выбирает фрагмент в окне редактора и говорит R выполнить его, вместо того чтобы набирать его снова. Эта техника особенно мощна, по мере того как фрагменты превращаются в длинные блоки кода.

В RStudio есть несколько сочетаний клавиш в интегрированной среде разработки, которые облегчают такой стиль работы. На компьютерах с Windows и Linux клавиши несколько отличаются от тех, что имеются на компьютерах с Mac: в Windows и Linux используются модификаторы **Ctrl** и **Alt**, тогда как в Mac применяются **Cmd** и **Opt**.

Чтобы открыть окно редактора

В главном меню выберите **File → New File** (Файл → Новый файл), затем выберите тип файла, который вы хотите создать, – в данном случае сценарий на языке R. Или если вы знаете, что вам нужен сценарий на языке R, то можете нажать сочетание клавиш **Shift-Ctrl-N** (в Windows) либо **Shift-Cmd-N** (в Mac).

Выполнить одну из строк окна редактора

Поместите курсор на строку и нажмите сочетание клавиш **Ctrl-Enter** (в Windows) или **Cmd-Enter** (в Mac), чтобы выполнить ее.

Выполнить несколько строк окна редактора

Выделите строки с помощью мыши; затем нажмите сочетание клавиш **Ctrl-Enter** (в Windows) или **Cmd-Enter** (в Mac), чтобы выполнить их.

Выполнить все содержимое окна редактора

Нажмите сочетание клавиш **Ctrl-Alt-R** (Windows) или **Cmd-Opt-R** (Mac), чтобы запустить все окно редактора. Или выберите в меню **Code → Run Region → Run All** (Код → Выполнить регион → Выполнить все).

Вы можете найти эти и многие другие сочетания клавиш в RStudio, выбрав пункт меню **Tools ⊠ Keyboard Shortcuts Help** (Инструменты → Сочетания клавиш. Справка).

Воспроизведение строк из окна консоли в окне редактора – это просто копирование и вставка. Когда вы будете выходить из RStudio, он спросит, хотите ли вы сохранить новый сценарий. Вы можете сохранить его для повторного использования в будущем или отказаться от него.

2.13. Создание конвейера вызовов функций

Задача

Создание множества промежуточных переменных в вашем коде утомительно и слишком многословно, в то время как вложение функций R делает код практически нечитаемым.

Решение

Используйте оператор конвейера (%>%), чтобы облегчить чтение и запись выражений. Оператор конвейера, созданный Стефаном Бахом и который можно найти в пакете `magrittr`, также широко используется во многих функциях *tidyverse*.

Используйте этот оператор для объединения нескольких функций в «конвейер» функций без промежуточных переменных:

```
library(tidyverse)
data(mpg)

mpg %>%
   filter(cty > 21) %>%
   head(3) %>%
   print()
#> # Tibble: 3 x 11
#> manufacturer model   displ year  cyl   trans   drv   cty   hwy   fl    class
#> <chr>        <chr>   <dbl> <int> <int> <chr>   <chr> <int> <int> <chr> <chr>
#> 1 chevrolet  malibu  2.4   2008  4     auto~   f     22    30    r     mids~
#> 2 honda      civic   1.6   1999  4     manu~   f     28    33    r     subc~
#> 3 honda      civic   1.6   1999  4     auto~   f     24    32    r     subc~
```

Использование конвейера намного чище и проще для чтения, нежели использование промежуточных временных переменных:

```
temp1 <- filter(mpg, cty > 21)
temp2 <- head(temp1, 3)
print(temp2)
#> # Tibble: 3 x 11
#> manufacturer model   displ year  cyl   trans   drv   cty   hwy   fl    class
#> <chr>        <chr>   <dbl> <int> <int> <chr>   <chr> <int> <int> <chr> <chr>
#> 1 chevrolet  malibu  2.4   2008  4     auto~   f     22    30    r     mids~
#> 2 honda      civic   1.6   1999  4     manu~   f     28    33    r     subc~
#> 3 honda      civic   1.6   1999  4     auto~   f     24    32    r     subc~
```

Обсуждение

Оператор конвейера не предоставляет R никаких новых функциональных возможностей, но может значительно улучшить читаемость кода. Он принимает вывод функции или объекта слева от оператора и передает их в качестве первого аргумента функции справа.

Это:

```
x %>% head()
```

с функциональной точки зрения то же самое, что и:

```
head(x)
```

В обоих случаях `x` является аргументом `head`. Мы можем предоставить дополнительные аргументы, но `x` всегда является *первым* аргументом. Эти две строки с функциональной точки зрения также идентичны:

```
x %>% head(n = 10)
head(x, n = 10)
```

Подобная разница может показаться незначительной, но на более сложном примере выгоды начинают накапливаться. Если бы у нас был рабочий процесс, в ко-

тором нам нужно было бы использовать функцию `filter`, чтобы ограничить свои данные значениями, а затем использовать команду `select`, чтобы сохранить только определенные переменные, а после этого использовать команду `ggplot` для создания простого графика, мы могли бы воспользоваться промежуточными переменными:

```
library(tidyverse)
filtered_mpg <- filter(mpg, cty > 21)
selected_mpg <- select(filtered_mpg, cty, hwy)
ggplot(selected_mpg, aes(cty, hwy)) + geom_point()
```

Этот инкрементальный подход довольно читабелен, но создает ряд промежуточных таблиц данных и требует, чтобы пользователь отслеживал состояние множества объектов, что может добавить когнитивную нагрузку. Но этот код действительно дает желаемый график.

В качестве альтернативы используется вложение функций:

```
ggplot(select(filter(mpg, cty > 21), cty, hwy), aes(cty, hwy)) + geom_point()
```

Хотя это очень лаконичный вариант, поскольку в нем всего одна строка, этот код требует гораздо большего внимания для чтения и понимания происходящего. Код, который пользователю сложно анализировать мысленно, может привести к ошибкам, а также его может быть сложнее поддерживать в будущем. Вместо этого мы можем использовать конвейеры:

```
mpg %>%
  filter(cty > 21) %>%
  select(cty, hwy) %>%
  ggplot(aes(cty, hwy)) + geom_point()
```

Предыдущий код начинается с набора данных `mpg` и передает его в функцию `filter`, которая хранит только записи, где значение города (`cty`) больше 21. Эти результаты передаются в команду `select`, в которой хранятся только перечисленные переменные `cty` и `hwy`, которые, в свою очередь, передаются в команду `ggplot`, создающую точечный график, показанный на рис. 2-2.

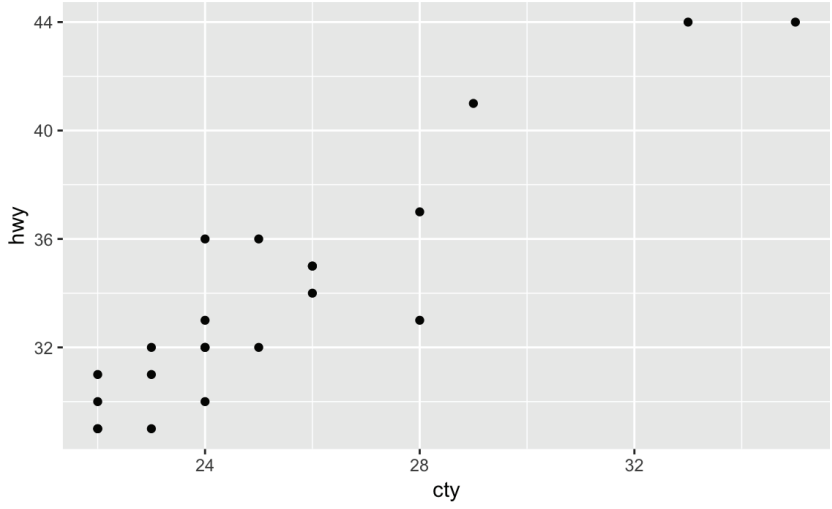


Рис. 2-2. Пример построения графика с помощью конвейера

Если вы хотите, чтобы аргумент, входящий в вашу целевую функцию (справа), отличался от первого аргумента, используйте оператор точки (.). Итак:

```
iris %>% head(3)
```

то же самое, что и:

```
iris %>% head(3, x = .)
```

Однако во втором примере мы передали таблицу данных `iris` во второй именованный аргумент, используя оператор точки. Это может быть удобно для функций, в которых таблица входных данных находится в позиции, отличной от первого аргумента.

В этой книге мы используем конвейеры, чтобы объединить преобразования данных с помощью нескольких шагов. Обычно мы форматируем код с разрывом строки после каждого оператора конвейера, а затем делаем отступ кода в следующих строках. Это делает код легко идентифицируемым как части одного и того же конвейера данных.

2.14. Как избежать некоторых распространенных ошибок

Задача

Вы хотите избежать некоторых распространенных ошибок, совершаемых начинающими и опытными пользователями.

Обсуждение

Вот несколько простых способов создать себе проблемы.

Забыть поставить скобки после вызова функции

Вы вызываете функцию, помещая скобки после имени. Например, эта строка вызывает функцию `ls`:

```
ls()
```

Однако если опустить скобки, R не выполняет эту функцию. Вместо этого он показывает определение функции, что почти никогда не соответствует тому, что вам нужно:

```
ls
```

```
# > function (name, pos = -1L, envir = as.environment(pos), all.names = FALSE,
# >     pattern, sorted = TRUE)
# > {
# >     if (!missing(name)) {
# >       pos <- tryCatch(name, error = function(e) e)
# >       if (inherits(pos, "error")) {
# >         name <- substitute(name)
# >         if (!is.character(name))
# >           name <- deparse(name)
# > # etc.
```

Опечатка: «<(пробел) -» вместо «<-»

Оператор присваивания – это <-, без пробела между < и -:

```
x <- pi # Присваиваем x значение 3.1415926...
```

Если вы случайно вставите пробел между < и -, значение полностью изменится:

```
x < -pi # Ой! Мы выполняем сравнение x, вместо того чтобы присваивать ему значение!
#> [1] FALSE
```

Теперь это сравнение (<) между x и -pi (отрицательным π). Это не меняет x. Если вам повезет, x не будет определен, и R будет жаловаться, предупреждая вас, что тут что-то не так:

```
x < -pi
#> Error in eval(expr, envir, enclos): object 'x' not found
```

Если x определен, R выполнит сравнение и выведет логическое значение TRUE или FALSE. Это должно предупредить вас о том, что что-то не так, поскольку присваивание обычно ничего не выводит:

```
x <- 0 # Присваиваем x значение 0.
x < -pi # Ой!
#> [1] FALSE
```

Некорректное продолжение выражения в следующей строке

R читает ваш ввод, пока вы не закончите выражение, независимо от того, сколько строк ввода требуется. Он запрашивает дополнительный ввод, используя приглашение +, пока это не будет сделано. В этом примере мы разбиваем выражение на две строки:

```
total <- 1 + 2 + 3 + # Продолжение на следующей строке.
   4 + 5
print(total)
#> [1] 15
```

Проблемы начинаются, когда вы случайно закончите выражение преждевременно, что может легко произойти:

```
total <- 1 + 2 + 3 # Ой! R видит завершенное выражение.
+ 4 + 5 # Это новое выражение; R выводит его значение.
#> [1] 9
print(total)
#> [1] 6
```

Есть два признака того, что что-то не так: R запросил у вас обычное приглашение (>), а не приглашение продолжения (+), и вывел значение 4 + 5.

Эта распространенная ошибка – головная боль для эпизодического пользователя. Однако для программистов это кошмар, поскольку может послужить причиной появления трудно обнаруживаемых ошибок в сценариях.

Использование = вместо ==

Используйте оператор двойного равенства (==) для сравнения. Если вы случайно используете оператор одинарного равенства (=), вы необратимо перезапишите свою переменную:

```
v <- 1 # Присваиваем 1 v.
v == 0 # Сравниваем v с 0.
#> [1] FALSE
v = 0 # Присваиваем 0 v, переписывая предыдущее содержимое.
print(v)
#> [1] 0
```

Пишем 1: n + 1, когда имеется в виду 1: (n + 1)

Вы можете подумать, что `1:n+1` – это последовательность чисел 1, 2, ..., *n, n* + 1. Это не так. Это последовательность 1, 2, ..., *n* с 1, которая добавляется к каждому элементу, в результате чего получается 2, 3, ..., *n, n* + 1. Это происходит потому, что R интерпретирует `1:n+1` как `(1:n)+1`. Используйте скобки, чтобы получить именно то, что вам нужно:

```
n <- 5
1:n + 1
#> [1] 2 3 4 5 6
1:(n + 1)
#> [1] 1 2 3 4 5 6
```

Пострадать от правила повторного использования

Векторная арифметика и сравнение векторов работают хорошо, когда оба вектора имеют одинаковую длину. Тем не менее результаты могут сбивать с толку, когда операнды являются векторами различной длины. Остерегайтесь такой возможности, понимая и помня правило повторного использования (см. рецепт 5.3).

Установить пакет, но не загрузить его с библиотекой или требовать

Установка пакета – первый шаг к его использованию, но требуется еще один шаг. Используйте `library` или `require`, чтобы загрузить пакет. Пока вы не сделаете этого, R не будет распознавать функции или наборы данных из пакета (*см.* рецепт 3.8):

```
x <- rnorm(100)
n <- 5
truehist(x, n)
#> Error in truehist(x, n): could not find function "truehist"
```

Однако если вы сначала загрузите библиотеку, то код запустится, и вы получите диаграмму, показанную на рис. 2-3:

```
library(MASS) # Загружаем пакет MASS в R.
truehist(x, n)
```

Обычно мы используем `library` вместо `require`. Причина состоит в том, что если вы создаете сценарий, где используется `library`, а нужный пакет еще не установлен, R вернет ошибку. Напротив, `require` просто вернет `FALSE`, если пакет не установлен.

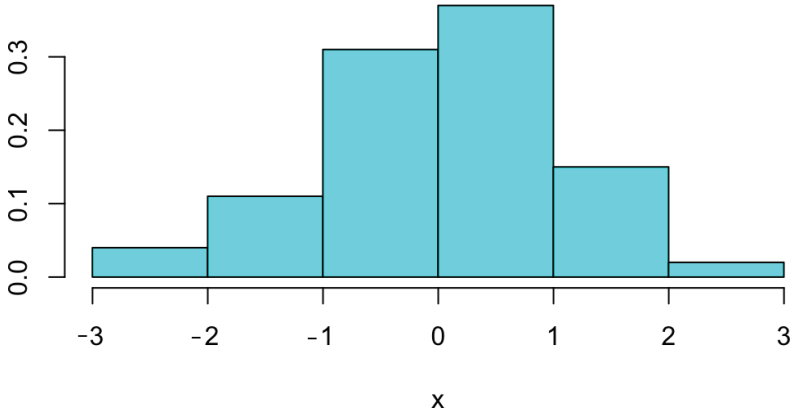


Рис. 2-3. Пример использования функции truehist

Пишем lst [n], когда имеется в виду lst [[n]], или наоборот

Если переменная `lst` содержит список, его можно проиндексировать двумя способами: `lst[[n]]` – это *n*-й элемент списка, тогда как `lst[n]` – это список, единственным элементом которого является *n*-й элемент из `lst`. Это большая разница. *См.* рецепт 5.7.

Использование & вместо &&, или наоборот; то же самое для | и ||

Используйте & и | в логических выражениях, включающих логические значения `TRUE` и `FALSE`. *См.* рецепт 2.9.

Используйте && и || для условного выражения внутри операторов `if` и `while`. Программисты, привыкшие к другим языкам программирования, могут рефлексивно везде использовать && и ||, потому что «они быстрее». Но эти операторы дают особые результаты, если их применять к векторам логических значений, поэтому избегайте их, если вы не уверены, что они делают то, что вам нужно.

Передача нескольких аргументов в функцию с одним аргументом

Как вы думаете, каково значение `mean(9,10,11)`? Нет, это не 10. Это 9. Функция `mean` вычисляет среднее значение первого аргумента. Второй и третий аргументы интерпретируются как другие позиционные аргументы. Чтобы передать несколько элементов в один аргумент, мы помещаем их в вектор с помощью оператора `c`. `mean(c(9,10,11))` вернет 10, как вы могли ожидать.

Некоторые функции, такие как `mean`, принимают один аргумент. Другие функции, такие как `max` и `min`, принимают несколько аргументов и применяются ко всем аргументам. Убедитесь, что вы знаете, кто есть кто.

Мнение, что max ведет себя как pmax, или что min ведет себя как pmin

Функции `max` и `min` имеют несколько аргументов и возвращают одно значение: максимум или минимум всех своих аргументов.

Функции `pmax` и `pmin` имеют несколько аргументов, но возвращают вектор со значениями, взятыми поэлементно из аргументов. Для получения дополнительной информации *см.* рецепт 12.8.

Неправильное использование функции, которая не понимает таблицы данных

Некоторые функции ведут себя довольно хитро в отношении таблиц данных. Они применяются к отдельным столбцам таблицы данных, вычисляя их результат для каждого отдельного столбца. К сожалению, не все функции такие умные. Речь идет о функциях `mean`, `median`, `max` и `min`. Они будут объединять все элементы из каждого столбца вместе и вычислять итоговый результат по всей выборке или, возможно, просто возвращать ошибку. Помните о том, какие функции умеют работать с таблицами данных, а какие нет. В случае сомнений прочитайте документацию по рассматриваемой функции.

Использование одиночной обратной косой черты (\) в полных именах файлов Windows

Обычно полные имена файлов копируют и вставляют в сценарии, но если вы используете R в Windows, вам нужно быть внимательными. Проводник Windows может показать вам, что ваше полное имя файла – *C:* \ *temp* \ *my_file.csv*, но если вы попытаетесь дать R указание прочитать этот файл, то получите загадочное сообщение:

```
Error: '\m' is an unrecognized escape in character string starting "'.\temp\m"
```

Это связано с тем, что R воспринимает обратную косую черту как специальный символ. Это можно обойти, используя прямую косую черту (/) или двойную обратную косую черту (\\):

```
read_csv('./temp/my_file.csv')
read_csv('.\\temp\\my_file.csv')
```

Подобная проблема возникает только в Windows, потому что и Mac, и Linux используют прямую косую черту в качестве разделителей.

Отправка вопроса на сайт Stack Overflow, или Список рассылки перед поиском ответа

Не тратьте свое время. Не тратьте время других людей. Перед тем как опубликовать вопрос в списке рассылки или на сайте Stack Overflow, сделайте домашнюю работу и выполните поиск в архивах. Скорее всего, кто-то уже ответил на ваш вопрос. Если это так, вы увидите ответ в ветке обсуждения, посвященной этому вопросу. *См.* рецепт 1.13.

См. также

См. рецепты 1.13, 2.9, 3.8, 5.3, 5.7 и 12.8.

Глава 3

Навигация по программному обеспечению

И R, и RStudio – это прежде всего сложное программное обеспечение. Вы неизбежно будете тратить время, делая то, что обычно делают с любой большой программой: настраивать его, обновлять и устанавливать его в своей вычислительной среде. Эта глава поможет вам выполнить данные задачи. В ней нет ничего о цифрах, статистике или графике. Речь тут идет о работе с R и RStudio в качестве программного обеспечения.

3.1. Получение и настройка рабочего каталога

Задача

Вам нужно изменить свой рабочий каталог, или вы просто хотите узнать, что это такое.

Решение

RStudio

Перейдите в каталог на панели **Files** (Файлы). Затем выберите **More** → **Set as Working Directory** (Дополнительно → Установить в качестве рабочего каталога), как показано на рис. 3-1.

Консоль

Используйте функцию `getwd`, чтобы вывести информацию о рабочем каталоге, и функцию `setwd`, чтобы изменить его:

```
getwd()
#> [1] "/Volumes/SecondDrive/jal/DocumentsPersonal/R-Cookbook"
setwd("~/Documents/MyDirectory")
```

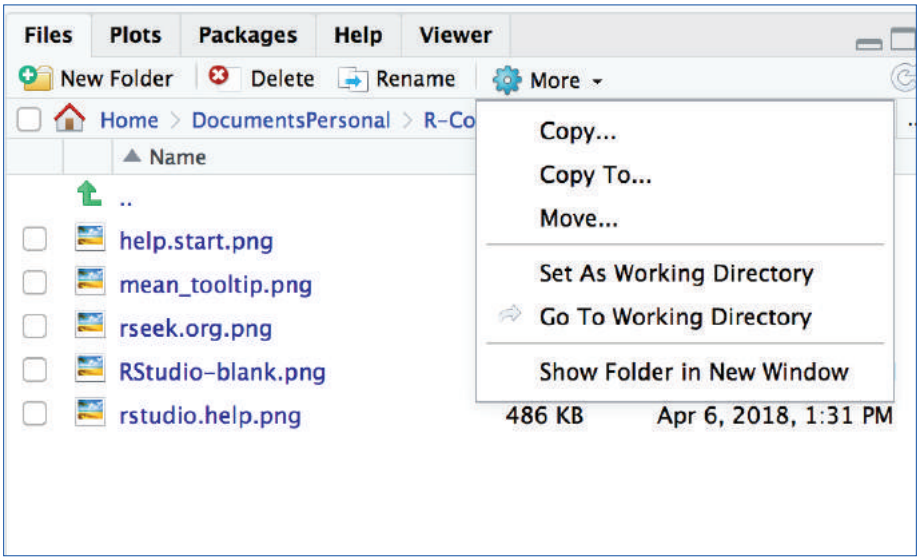


Рис. 3-1. RStudio: установка в качестве рабочего каталога

Обсуждение

Ваш рабочий каталог важен, потому что он является местоположением по умолчанию для файлового ввода и вывода, включая чтение и запись файлов данных, открытие и сохранение файлов сценариев, а также сохранение образа рабочей области. Если вы открываете файл и не указываете абсолютный путь, R будет считать, что файл находится в вашем рабочем каталоге.

Если вы используете проекты RStudio, рабочим каталогом по умолчанию будет домашний каталог проекта. *См.* рецепт 3.2 для получения дополнительной информации о создании проектов RStudio.

См. также

См. рецепт 4.5, чтобы узнать, как работать с именами файлов в Windows.

3.2. Создание нового проекта RStudio

Задача

Вы хотите создать новый проект RStudio, чтобы все ваши файлы были связаны с конкретным проектом.

Решение

Нажмите **File** → **New Project** (Файл → Новый проект), как показано на рис. 3-2.

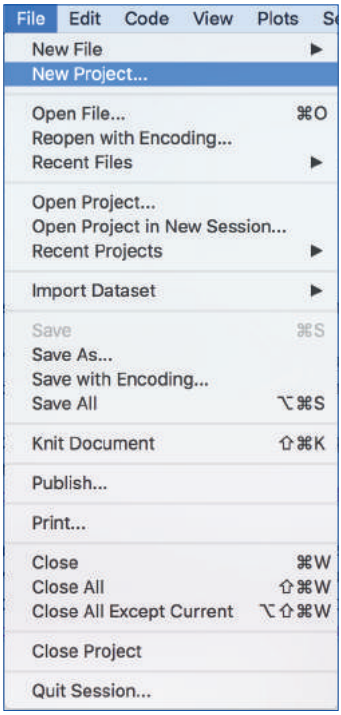


Рис. 3-2. Создание нового проекта

Откроется диалоговое окно **New Project** (Новый проект), и вы сможете выбрать тип проекта, который хотите создать, как показано на рис. 3-3.

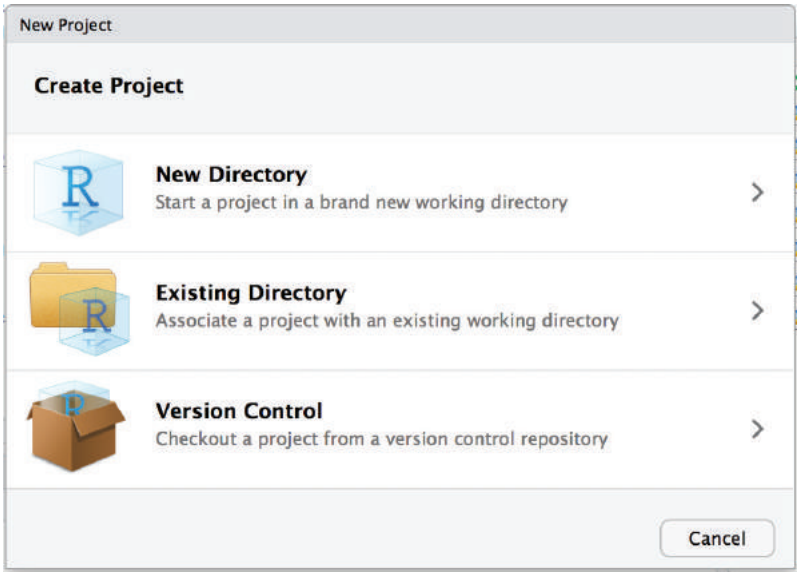


Рис. 3-3. Диалоговое окно **New Project**

Обсуждение

Проекты – это мощная концепция, специфичная для RStudio. Они помогают вам, делая следующее:

- устанавливая ваш рабочий каталог в каталог проекта;
- сохраняя состояние окна в RStudio, чтобы при возврате в проект все окна были такими же, какими вы их оставили. Это включает в себя открытие любых файлов, которые вы открывали, когда в последний раз сохраняли свой проект;
- сохраняя настройки проекта RStudio.

Чтобы сохранить настройки проекта, RStudio создает файл проекта с расширением *.Rproj* в каталоге проекта. Если вы открываете файл проекта в RStudio, он работает как ярлык для открытия проекта. Кроме того, RStudio создает скрытый каталог с именем *.Rproj.user* для размещения временных файлов, связанных с вашим проектом.

Каждый раз, когда вы работаете над чем-то нетривиальным в R, мы рекомендуем создавать проект RStudio. Проекты помогают оставаться организованным и облегчают рабочий процесс вашего проекта.

3.3. Сохранение своего рабочего пространства

Задача

Вам нужно сохранить свое рабочее пространство и все переменные и функции, которые у вас есть в памяти.

Решение

Вызовите функцию `save.image`:

```
save.image()
```

Обсуждение

Ваше рабочее пространство содержит ваши переменные и функции R и создается при запуске R. Рабочее пространство хранится в основной памяти вашего компьютера и действует до тех пор, пока вы не выйдете из R. Вы можете легко просмотреть содержимое своей рабочей области в RStudio на вкладке **Environment**, как показано на рис. 3-4.

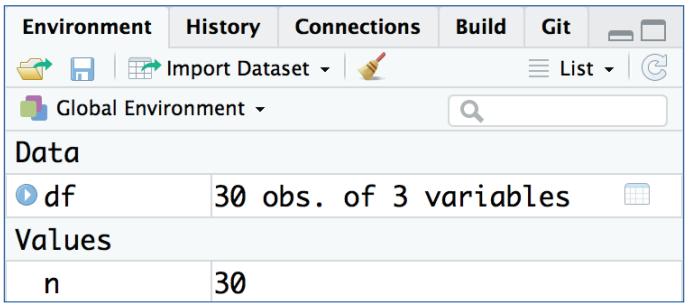


Рис. 3-4. Панель **Environment** RStudio

Однако у вас может возникнуть желание сохранить свое рабочее пространство, не выходя из R, потому что вы знаете, что когда закрываете свой ноутбук, чтобы отнести его домой, таинственным образом происходят странные вещи. В этом случае используйте функцию `save.image`.

Рабочее пространство записывается в файл с именем *.RData* в рабочем каталоге. Когда R запускается, он ищет этот файл и, если находит, инициализирует из него рабочую область.

К сожалению, рабочее пространство не включает в себя ваши открытые графики: например, этот крутой график на вашем экране исчезнет, когда вы выходите из R. Рабочее пространство также не содержит расположения ваших окон или настроек RStudio. Вот почему мы рекомендуем использовать проекты RStudio и писать сценарии, чтобы вы могли воспроизвести все, что создали.

См. также

См. рецепт 3.1 для настройки рабочего каталога.

3.4. Просмотр истории команд

Задача

Вы хотите увидеть свою недавнюю последовательность команд.

Решение

В зависимости от того, что вы пытаетесь сделать, вы можете использовать несколько различных методов для доступа к предыдущей истории команд. Если вы находитесь на панели консоли RStudio, то можете нажать стрелку, указывающую наверх, чтобы в интерактивном режиме прокручивать прошедшие команды.

Если вы хотите просмотреть список прошлых команд, можете выполнить функцию `history`, либо открыть панель **History** (История) в RStudio, чтобы просмотреть свой последний ввод:

```
history()
```

В RStudio при вводе `history()` в консоль вы просто активируете панель **History** (рис. 3-5). Также можно сделать эту панель видимой, кликнув на ней курсором.

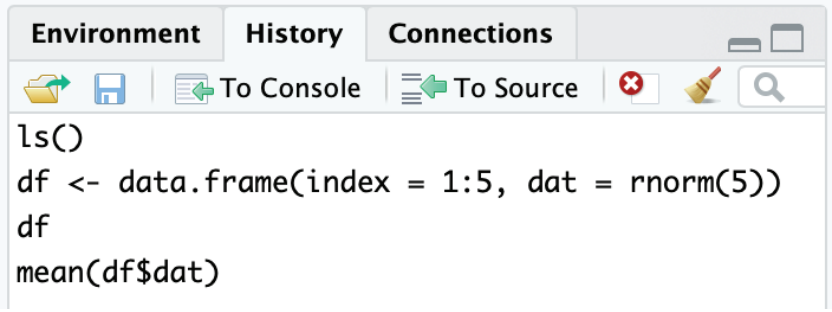


Рис. 3-5. Панель **History**

Обсуждение

Функция `history` отображает ваши последние команды. В RStudio команда `history` активирует панель **History**. Если вы работаете с R за пределами RStudio, `history` показывает последние 25 строк, но можно запросить и больше:

```
history(100) # Показывает 100 последних строк истории.
history(Inf) # Показывает всю сохраненную историю.
```

В RStudio на вкладке **History** показан исчерпывающий список прошлых команд в хронологическом порядке, причем самые последние находятся внизу списка. Вы можете выделить прошлые команды курсором, затем нажать **To Console** или **To Source**, чтобы скопировать прошлые команды в консоль или редактор исходного кода соответственно. Это может быть очень удобно, когда вы выполнили интерактивный анализ данных, а затем решили, что хотите сохранить какие-либо предыдущие шаги в исходный файл для последующего использования.

В консоли вы можете просмотреть свою историю, просто нажав стрелку, указывающую наверх, чтобы прокрутить назад введенные данные, в результате этого то, что вы ввели ранее, появится снова, по одной строке за раз.

Если вы вышли из R или RStudio, вы все равно можете просмотреть историю команд. R сохраняет историю в файле с именем *.Rhistory* в рабочем каталоге. Откройте файл в текстовом редакторе, а затем прокрутите вниз; вы увидите то, что набирали, в последний раз.

3.5. Сохранение результата предыдущей команды

Задача

Вы ввели в R выражение, которое вычислило значение, но забыли сохранить результат в переменной.

Решение

Специальная переменная `.Last.value` сохраняет значение последнего вычисленного выражения. Сохраните его в переменной, прежде чем вводить что-либо еще.

Обсуждение

Очень неприятно набирать длинное выражение или вызывать долго выполняющуюся функцию, а потом забыть сохранить результат. К счастью, вам не нужно повторно вводить выражение и не нужно снова вызывать функцию – результат был сохранен в переменной `.Last.value`:

```
aVeryLongRunningFunction() # Ой! Мы забыли сохранить результат!
x <- .Last.value # Фиксируем результат.
```

Небольшое предостережение: содержимое переменной `.Last.value` перезаписывается каждый раз, когда вы вводите другое выражение, поэтому сразу фиксируйте значение. Если вы забудете сделать это, до того как другое выражение не будет вычислено, будет слишком поздно!

См. также

См. рецепт 3.4, чтобы узнать, как вспомнить историю своих команд.

3.6. Отображение загруженных пакетов через путь поиска

Задача

Вы хотите увидеть список пакетов, загруженных в R.

Решение

Используйте функцию `search` без аргументов:

```
search()
```

Обсуждение

Путь поиска – это список пакетов, которые в данный момент загружены в память и доступны для использования. Хотя многие пакеты могут быть установлены на вашем компьютере, только некоторые из них фактически загружаются в интерпретатор R в любой момент. Вам может быть интересно, какие пакеты загружены прямо в данный момент.

Без аргументов функция `search` возвращает список загруженных пакетов, например:

```
search()
#>  [1]  ".GlobalEnv"         "package:knitr"       "package:forcats"
#>  [4]  "package:stringr"    "package:dplyr"       "package:purrr"
#>  [7]  "package:readr"      "package:tidyr"       "package:tibble"
#> [10]  "package:ggplot2"    "package:tidyverse"   "package:stats"
#> [13]  "package:graphics"   "package:grDevices"   "package:utils"
#> [16]  "package:datasets"   "package:methods"     "Autoloads"
#> [19]  "package:base"
```

Ваш компьютер может выдать иной результат в зависимости от того, что на нем установлено. Возвращаемое значение `search` – вектор строк. Первая строка – `".GlobalEnv"`, которая относится к вашей рабочей области. Большинство строк имеют форму `"package:packagename"`, которая указывает на то, что пакет с именем `packagename` в настоящее время загружен в R. В предыдущем примере видно множество установленных пакетов из коллекции *tidyverse*, включая `purrr`, `ggplot2` и `tibble`.

R использует пути, перечисленные в выводе search, чтобы найти функции. Когда вы вводите имя функции, R ищет путь в указанном порядке, пока не найдет функцию в загруженном пакете. Если функция найдена, R выполняет ее. В противном случае он выводит сообщение об ошибке и останавливается. (На самом деле тут есть еще кое-что: пути search или поиска могут содержать содержать среды, а не только пакеты, и алгоритм поиска отличается, когда инициируется объектом в пакете; *см.* подробности на странице https://cran.r-project.org/doc/manuals/R-lang.pdf.)

Поскольку ваше рабочее пространство (`.GlobalEnv`) является первым в списке, R ищет функции в вашем рабочем пространстве, перед тем как искать в пакетах. Если ваше рабочее пространство и пакет содержат функцию с одинаковым именем, ваше рабочее пространство будет «маскировать» функцию; это означает, что

R прекращает поиск после того, как находит вашу функцию, и поэтому так и не увидит функцию из пакета. Это благословение, если вы хотите переопределить функцию пакета… и проклятие, если вы по-прежнему хотите получить к ней доступ. Если вы чувствуете себя так, как будто вас прокляли из-за того, что вы (или какой-либо загруженный вами пакет) переопределили функцию (или другой объект) из существующего загруженного пакета, можете использовать полную форму `environment::name` для вызова объекта из среды загруженного пакета. Например, если вы хотите вызвать функцию `count` из пакета `dplyr`, это можно сделать с помощью `dplyr::count`. Использование полного явного имени для вызова функции сработает, даже если вы не загрузили пакет, поэтому если `dplyr` установлен, но не загружен, вы все равно можете вызвать `dplyr::count`.

В онлайн-примерах становится все более распространенным показывать полное `packagename:: function`. Хотя это и устраняет двусмысленность в отношении того, откуда взялась функция, пример кода становится гораздо более многословным.

Обратите внимание, что R будет включать в пути поиска только *загруженные* пакеты. Таким образом, если вы установили пакет, но не загрузили его с помощью `library(имя_пакета)`, R не станет добавлять этот пакет в путь поиска.

R также использует путь поиска, чтобы найти наборы данных R (не файлы) или любой другой объект с помощью аналогичной процедуры.

Пользователи Unix и Mac: не путайте путь поиска в R с путем поиска в Unix (переменная среды `PATH`). Концептуально они похожи, но это две разные вещи. Путь поиска в R является внутренним по отношению к R и используется только для поиска функций и наборов данных, тогда как путь поиска в Unix используется ОС для поиска исполняемых программ.

См. также

См. рецепт 3.8, чтобы узнать, как загружать пакеты в R, и рецепт 3.7, чтобы узнать, как осуществлять просмотр списка установленных пакетов (а не только загруженных).

3.7. Просмотр списка установленных пакетов

Задача

Вы хотите знать, какие пакеты установлены на вашем компьютере.

Решение

Используйте функцию `library` без аргументов для основного списка. Используйте функцию `installed.packages`, чтобы увидеть более подробную информацию о пакетах.

Обсуждение

Функция `library` без аргументов выводит список установленных пакетов:

```
library()
```

Список может быть довольно длинным. В RStudio он отображается на новой вкладке в окне редактора.

Вы можете получить более подробную информацию с помощью функции `installed.packages`, которая возвращает матрицу информации касательно пакетов на вашем компьютере. Каждый ряд соответствует одному установленному пакету. Столбцы содержат такую информацию, как имя пакета, путь к библиотеке и версия. Эта информация берется из внутренней базы данных установленных пакетов R.

Чтобы извлечь полезную информацию из этой матрицы, используйте обычные методы индексации. В приведенном ниже фрагменте кода мы вызываем функцию `installed.packages` и извлекаем столбцы `Package` и `Version` для первых пяти пакетов, что позволяет нам увидеть, какая версия пакета установлена:

```
installed.packages()[1:5, c("Package", "Version")]
#>             Package        Version
#> abind       "abind"        "1.4-5"
#> ade4        "ade4"         "1.7-13"
#> adegenet    "adegenet"     "2.1.1"
#> analogsea   "analogsea"    "0.6.6.9110"
#> ape         "ape"          "5.3"
```

См. также

См. рецепт 3.8, чтобы узнать, как загрузить пакет в память.

3.8. Доступ к функциям в пакете

Задача

Пакет, установленный на вашем компьютере, является либо стандартным пакетом, либо пакетом, который вы скачали. Однако когда вы пытаетесь использовать функции из этого пакета, R не может их найти.

Решение

Используйте функцию `library` либо функцию `require` для загрузки пакета в R:

```
library(имяпакета)
```

Обсуждение

R поставляется с несколькими стандартными пакетами, но не все из них автоматически загружаются при запуске R. Также вы можете скачать и установить множество полезных пакетов из CRAN или GitHub, но они не загружаются автоматически при запуске R. Например, пакет `MASS` идет в стандартной поставке с R, но при использовании функции `lda` из этого пакета вы можете получить вот такое сообщение:

```
lda(x)
#> Error in lda(x): could not find function "lda"
```

R жалуется, что не может найти функцию `lda` среди пакетов, в данный момент загруженных в память.

Когда вы используете функцию `library` или `require`, R загружает пакет в память, и его содержимое становится сразу доступным для вас:

```
my_model <-
   lda(cty ~ displ + year, data = mpg)
#> Error in lda(cty ~ displ + year, data = mpg): could not find function "lda"

 library(MASS)                # Загружаем библиотеку MASS в память.
#>
#> Attaching package: 'MASS'
#> The following object is masked from 'package:dplyr':
#>
#>    select
my_model <-
   lda(cty ~ displ + year, data = mpg) # Теперь R может найти функцию.
```

Перед вызовом функции `library` R не распознает имя функции. После этого содержимое пакета становится доступным, и вызов функции `lda` проходит успешно.

Обратите внимание, что вам не нужно заключать имя пакета в кавычки.

Функция `require` почти идентична функции `library`, но у нее есть два свойства, которые полезны при написании сценариев. Она возвращает `TRUE`, если пакет был успешно загружен, и в противном случае `FALSE`, а также генерирует простое предупреждение в случае сбоя загрузки – в отличие от функции `library`, которая генерирует ошибку.

Обе эти функции имеют ключевую особенность: они не перезагружают пакеты, которые уже были загружены, поэтому двойной вызов одного и того же пакета безопасен, что особенно хорошо для написания сценариев. Сценарий может загружать необходимые пакеты, зная, что загруженные пакеты не будут перезагружены.

Функция `detach` выгрузит пакет, который загружен в данный момент:

```
detach(package:MASS)
```

Обратите внимание, что нужно уточнить имя пакета, как здесь: `package:MASS`.

Одна из причин выгрузки пакета состоит в том, что он содержит функцию, имя которой конфликтует с функцией с тем же именем, расположенной ниже в списке поиска. Когда возникает такой конфликт, мы говорим, что высшая функция *маскирует* низшую функцию. Вы больше не «видите» нижнюю функцию, потому что R прекращает поиск, когда находит более высокую функцию. Следовательно, выгрузка более высокого пакета демаскирует более низкое имя.

См. также

См. рецепт 3.6.

3.9. Доступ к встроенным наборам данных

Задача

Вы хотите использовать один из встроенных наборов данных R или получить доступ к одному из наборов данных, который поставляется с другим пакетом.

Решение

Стандартные наборы данных, распространяемые вместе с R, уже доступны для вас, поскольку пакет `dataset` находится в вашем пути поиска. Если вы загрузили любые другие пакеты, наборы данных, которые поставляются с этими загруженными пакетами, также будут доступны в вашем пути поиска.

Чтобы получить доступ к наборам данных в других пакетах, используйте функцию `data`, указав имя набора данных и имя пакета:

```
data(dsname, package = "pkgname")
```

Обсуждение

R поставляется со множеством встроенных наборов данных. Другие пакеты, такие как `dplyr` и `ggplot2`, также поставляются с примерами данных, которые используются в примерах из файлов справки. Эти наборы данных полезны, когда вы изучаете R, поскольку они предоставляют данные для экспериментов.

Многие наборы данных хранятся в пакете под названием (и это вполне естественно) `datasets`, который распространяется вместе с R. Этот пакет находится в вашем пути поиска, поэтому у вас есть мгновенный доступ к его содержимому. Например, можно использовать встроенный набор данных под названием `pressure`:

```
head(pressure)
#>  temperature pressure
#> 1       0     0.0002
#> 2      20     0.0012
#> 3      40     0.0060
#> 4      60     0.0300
#> 5      80     0.0900
#> 6     100     0.2700
```

Если вы хотите узнать больше о `pressure`, используйте функцию `help`:

```
help(pressure) # Открывает страницу справки для набора данных pressure.
```

Вы можете просмотреть список доступных наборов данных, вызвав функцию `data` без аргументов:

```
data() # Открывает список наборов данных.
```

Любой пакет R может включать в свой состав наборы данных, которые дополняют наборы, поставляемые в `datasets`. Например, пакет `MASS` включает в себя множество интересных наборов данных. Используйте функцию `data` с аргументом `package`, чтобы загрузить набор данных из определенного пакета. `MASS` включает в себя набор данных `Cars93`, который можно загрузить в память следующим образом:

```
data(Cars93, package = "MASS")
```

После этого вызова функции `data` вы получите доступ к набору данных `Cars93`, после чего можете выполнить команды `summary(Cars93)`, `head(Cars93)` и т. д.

При добавлении пакета в список поиска (например, с помощью `library(MASS)`) не нужно вызывать функцию `data`. Наборы данных становятся доступными автоматически, когда вы присоединяете его.

Вы можете увидеть список доступных наборов данных в MASS или любом другом пакете, используя функцию `data` с аргументом `package` и без имени набора данных:

```
data(package = "pkgname")
```

См. также

См. рецепт 3.6 для получения дополнительной информации о пути поиска и рецепт 3.8 для получения дополнительной информации о пакетах и функции `library`.

3.10. Установка пакетов из CRAN

Задача

Вы нашли пакет в CRAN и теперь хотите установить его на свой компьютер.

Решение

Код R

Используйте функцию `install.packages`, поместив имя пакета в кавычки:

```
install.packages("packagename")
```

RStudio

Панель **Packages** в RStudio помогает упростить установку новых пакетов R. В этой панели перечислены все пакеты, которые установлены на вашем компьютере наряду с описанием и информацией о версии. Чтобы загрузить новый пакет из CRAN, нажмите кнопку **Install** (Установить) в верхней части панели **Packages**, как показано на рис. 3-6.

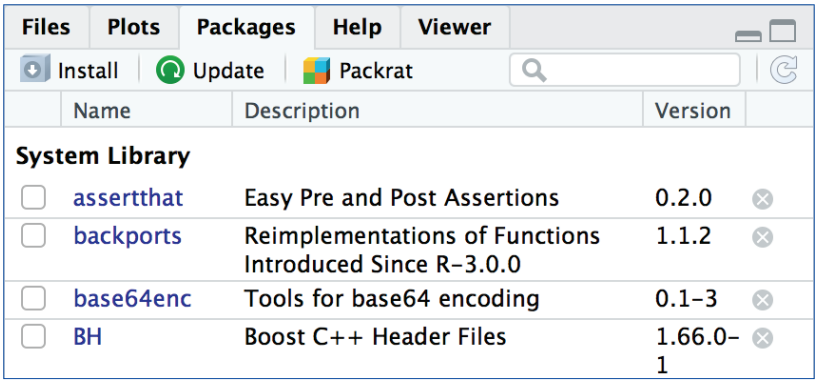


Рис. 3-6. Панель **Packages**

Обсуждение

Локальная установка пакета – первый шаг к его использованию. Если вы устанавливаете пакеты за пределами RStudio, установщик может запросить у вас сайт-

зеркало, с которого он может скачать файлы пакетов. Затем отобразится список зеркал CRAN. Самое верхнее зеркало – это 0-Cloud. Обычно это самый подходящий вариант, поскольку оно предоставляет автоматическое перенаправление на сервер по всему миру и спонсируется RStudio. Если вы хотите выбрать другое зеркало, выберите то, что географически ближе к вам.

Официальный сервер CRAN – это относительно скромный компьютер, размещенный на территории Департамента статистики и математики Венского экономического университета в Австрии. Если бы каждый пользователь R выполнял скачивание с официального сервера, сервер бы завис, поэтому по всему миру существует множество сайтов-зеркал. В RStudio сервер CRAN по умолчанию настроен как зеркало RStudio CRAN. Оно доступно для всех пользователей R, а не только для тех, кто использует интегрированную среду разработки RStudio.

Если новый пакет зависит от других пакетов, которые еще не установлены локально, установщик R автоматически загрузит и установит необходимые пакеты. Это огромное преимущество, которое освобождает вас от утомительной задачи по выявлению и устранению этих зависимостей.

При выполнении установки в Linux или Unix есть особый момент. Вы можете установить пакет либо в общесистемной библиотеке, либо в своей личной библиотеке. Пакеты в общесистемной библиотеке доступны каждому; пакеты в вашей личной библиотеке (обычно) используются только вами. Таким образом, популярный, протестированный пакет, скорее всего, попадет в общесистемную библиотеку, тогда как неизвестный или непроверенный пакет попадет в вашу личную библиотеку.

По умолчанию функция `install.packages` предполагает, что вы выполняете общесистемную установку. Если у вас недостаточно пользовательских прав для выполнения установки в общесистемной библиотеке, R спросит, хотите ли вы установить пакет в пользовательской библиотеке. Значение по умолчанию, которое предлагает R, обычно является хорошим вариантом. Однако если вы хотите контролировать путь к своей библиотеке, то можете использовать аргумент `lib` функции `install.packages`:

```
install.packages("packagename", lib = "~/lib/R")
```

Вы также можете изменить сервер CRAN по умолчанию, как описано в рецепте 3.12.

См. также

См. рецепт 1.12, чтобы узнать о способах поиска соответствующих пакетов, и рецепт 3.8, чтобы узнать, как использовать пакет после его установки.

Также *см.* рецепт 3.12.

3.11. Установка пакета из GitHub

Задача

Вы нашли интересный пакет, который хотели бы опробовать. Тем не менее автор еще не опубликовал его на CRAN, но он есть на GitHub. Вы хотите установить пакет напрямую из GitHub.

Решение

Убедитесь, что у вас установлен и загружен пакет `devtools`:

```
install.packages("devtools")
library(devtools)
```

Затем используйте функцию `install_github` и имя репозитория в GitHub для установки непосредственно из GitHub. Например, чтобы установить пакет `tidygraph` Томаса Лин Педерсона, вы должны выполнить следующее:

```
install_github("thomasp85/tidygraph")
```

Обсуждение

Пакет `devtools` содержит вспомогательные функции для установки пакетов R из удаленных репозиториев, таких как GitHub. Если пакет был создан как пакет для R, а затем размещен на GitHub, вы можете установить его с помощью функции `install_github`, передав имя пользователя GitHub и имя репозитория в качестве строкового параметра. Можно определить имя пользователя и имя репозитория GitHub на основании URL-адреса GitHub или посмотрев в верхней части страницы GitHub, как в примере, показанном на рис. 3-7.

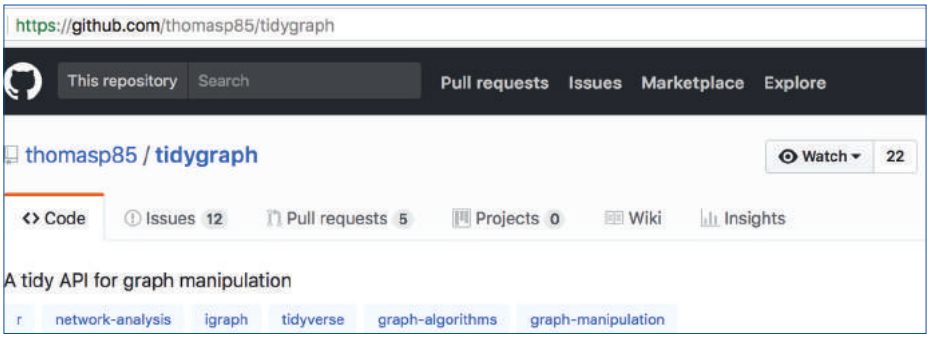


Рис. 3-7. Пример страницы проекта в GitHub

3.12. Установка или изменение зеркала CRAN по умолчанию

Задача

Вы скачиваете пакеты. Вам нужно установить или изменить зеркало CRAN по умолчанию.

Решение

В RStudio можно изменить зеркало CRAN по умолчанию из меню настроек, как показано на рис. 3-8.

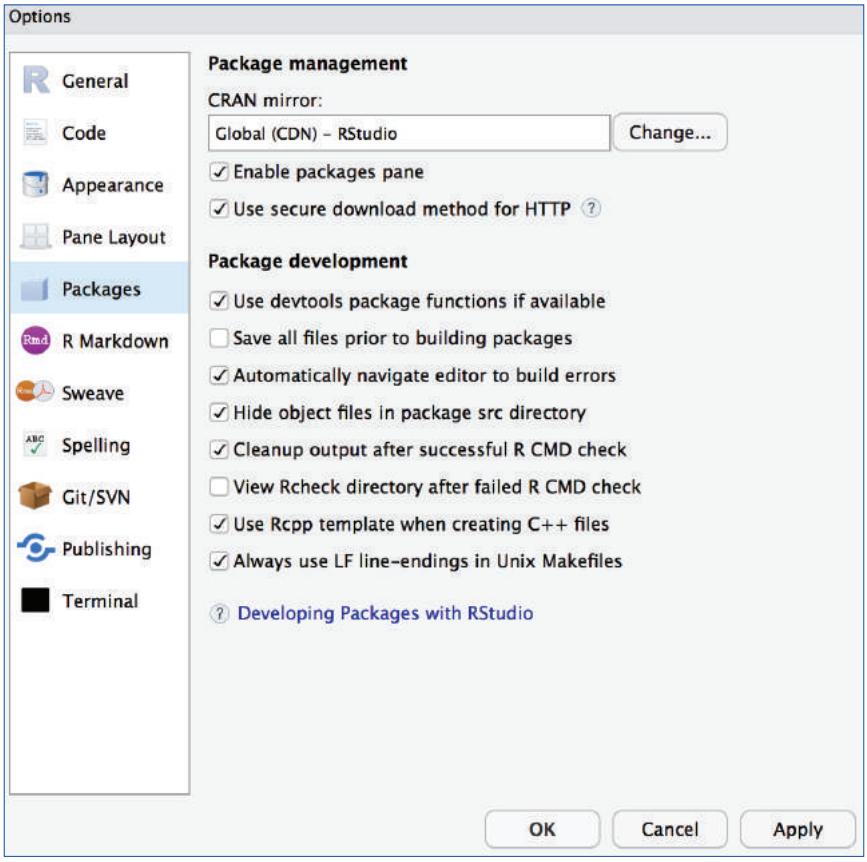


Рис. 3-8. Настройки пакета в RStudio

Если вы запускаете R без RStudio, можете изменить свое зеркало CRAN, используя приведенное ниже решение. Оно предполагает, что у вас есть файл *.Rprofile*, как было описано в рецепте 3.16.

1. Вызовите функцию `chooseCRANmirror`:

```
chooseCRANmirror()
```

R представит список зеркал CRAN.

2. Выберите зеркало CRAN из списка и нажмите **OK**.
3. Чтобы получить URL-адрес зеркала, посмотрите на первый элемент опции `repos`:

```
options("repos")[[1]][1]
```

4. Добавьте эту строку в ваш файл *.Rprofile*. Если вам нужно зеркало RStudio CRAN, вы должны сделать следующее:

```
options(repos = c(CRAN = "http://cran.rstudio.com"))
```

Или вы можете использовать URL-адрес другого зеркала CRAN.

Обсуждение

Когда вы устанавливаете пакеты, то, вероятно, каждый раз используете одно и то же зеркало CRAN (а именно ближайшее к вам зеркало или зеркало RStudio), потому что RStudio не будет спрашивать вас каждый раз, когда вы загружаете пакет; он просто использует настройки из меню настроек. Вы можете изменить это зеркало, чтобы использовать другое зеркало, которое ближе к вам или контролируется вашим работодателем. Используйте это решение, чтобы изменить свой репозиторий, чтобы каждый раз при запуске R или RStudio применять желаемый репозиторий.

Опция `repos` – имя вашего зеркала по умолчанию. Функция `chooseCRANmirror` обладает важным побочным эффектом установки этой опции в соответствии с вашим выбором. Проблема состоит в том, что R забывает настройку при выходе, не оставляя постоянной настройки по умолчанию. Установив `repos` в своем файле *.Rprofile*, вы восстанавливаете настройки каждый раз при запуске R.

См. также

См. рецепт 3.16 для получения дополнительной информации о файле *.Rprofile* и функции `options`.

3.13. Запуск сценария

Задача

Вы записали серию команд R в текстовом файле. Теперь вы хотите их выполнить.

Решение

Функция `source` инструктирует R прочитать текстовый файл и выполнить его содержимое:

```
source("myScript.R")
```

Обсуждение

Если у вас есть длинный или часто используемый фрагмент кода R, запишите его в текстовый файл. Это позволит вам легко перезапустить код без необходимости его повторного ввода. Используйте функцию `source` для чтения и выполнения этого кода, как если бы вы ввели его в консоль R.

Предположим, что файл *hello.R* содержит одно знакомое приветствие:

```
print("Hello, World!")
```

Затем будет выполнено содержимое файла:

```
source("hello.R")
#> [1] "Hello, World!"
```

Когда `echo = TRUE`, будут отображены строки сценария, прежде чем они будут выполнены, а перед каждой строкой будет отображаться приглашение R:

```
source("hello.R", echo = TRUE)
#>
#> > print("Hello, World!")
#> [1] "Hello, World!"
```

См. также

См. рецепт 2.12, чтобы узнать, как выполнить запуск блоков кода R внутри графического интерфейса пользователя.

3.14. Запуск пакетного сценария

Задача

Вы пишете командный сценарий, например сценарий оболочки в Unix или macOS или сценарий BAT в Windows. Внутри своего сценария вы хотите выполнить сценарий R.

Решение

Запустите программу R с подкомандой `CMD BATCH`, указав имя сценария и имя выходного файла:

```
R CMD BATCH файлсценария выходнойфайл
```

Если вы хотите, чтобы вывод отправлялся в `stdout`, или если вам нужно передать аргументы командной строки в сценарий, используйте команду `Rscript`:

```
Rscript файлсценария арг1 арг2 арг3
```

Обсуждение

Обычно R – это интерактивная программа, которая запрашивает ввод данных пользователем и затем отображает результаты. Иногда вам нужно запустить R в пакетном режиме, читая команды из сценария. Это особенно полезно внутри сценариев командной оболочки, таких как сценарии, которые включают в себя статистический анализ.

Подкоманда `CMD BATCH` переводит R в пакетный режим, выполняя чтение из файла сценария и записывая в выходной файл. Она не взаимодействует с пользователем.

Скорее всего, вы будете использовать опции командной строки, чтобы настроить пакетное поведение R в соответствии с вашими обстоятельствами. Например, использование опции `--quiet` подавляет сообщения, выводимые при начале работы сценария, которые в противном случае загромоздили бы вывод:

```
R CMD BATCH --quiet myScript.R results.out
```

Другие полезные опции в пакетном режиме включают в себя:

```
--slave
```

Похожа `--quiet`, но делает R еще тише, подавляя эхо ввода.

```
--no-restore
```

Используется при запуске, чтобы не восстанавливать рабочее пространство R. Это важно, если ваш сценарий ожидает, что R начнется с пустой рабочей области.

```
--no-save
```

Используется при выходе, чтобы не сохранять рабочее пространство R. В противном случае R сохранит свое рабочее пространство и перезапишет файл *.RData* в рабочем каталоге.

```
--no-init-file
```

Используется, чтобы не читать ни файл *.Rprofile*, ни файл *~/.Rprofile*.

Подкоманда `CMD BATCH` обычно вызывает функцию `proc.time`, когда ваш сценарий завершается, показывая время выполнения. Если это вас раздражает, завершите свой сценарий, вызвав функцию `q` с определенным параметром `runLast = FALSE`, что предотвратит вызов `proc.time`.

Подкоманда `CMD BATCH` имеет два ограничения: вывод всегда идет в файл, и нельзя просто передать аргументы командной строки в свой сценарий. Если какое-либо ограничение является проблемой, рассмотрите возможность использования программы `Rscript`, поставляемой с R. Первым аргументом командной строки является имя сценария, а остальные аргументы передаются сценарию:

```
Rscript файлсценария.R арг1 арг2 арг3
```

Внутри сценария вы можете получить доступ к аргументам командной строки, вызвав функцию `commandArgs`, которая возвращает аргументы в качестве вектора строк:

```
argv <- commandArgs(TRUE)
```

Программа `Rscript` использует те же параметры командной строки, что и `CMD BATCH`, которые были описаны только что.

Вывод записывается в `stdout`, который, конечно, R наследует от вызывающего сценария оболочки. Можно перенаправить вывод в файл, используя обычное перенаправление:

```
Rscript --slave файлсценария.R арг1 арг2 арг3 >results.out
```

Вот небольшой сценарий, *arith.R*, который принимает два аргумента командной строки и выполняет с ними четыре арифметические операции:

```
argv <- commandArgs(TRUE)
x <- as.numeric(argv[1])
y <- as.numeric(argv[2])

cat("x =", x, "\n")
cat("y =", y, "\n")
cat("x + y = ", x + y, "\n")
cat("x - y = ", x - y, "\n")
cat("x * y = ", x * y, "\n")
cat("x / y = ", x / y, "\n")
```

Сценарий вызывается так:

```
Rscript arith.R 2 3.1415
```

что дает следующий вывод:

```
x = 2
y = 3.1415
x + y = 5.1415
x - y = -1.1415
x * y = 6.283
x / y = 0.6366385
```

В Linux, Unix или Mac можно сделать сценарий полностью автономным, поместив строку с `#!` в начало. В ней будет прописан путь к программе `Rscript`. Предположим, что `Rscript` установлена в вашей системе в */usr/bin/Rscript*. Добавьте эту строку в сценарий arith.R, и он станет автономным:

```
#!/usr/bin/Rscript --slave

argv <- commandArgs(TRUE)
x <- as.numeric(argv[1])
.
. (etc.)
.
```

В приглашении оболочки мы помечаем этот сценарий как исполняемый:

```
chmod +x arith.R
```

Теперь мы можем вызвать его напрямую без префикса `Rscript`:

```
arith.R 2 3.1415
```

См. также

См. рецепт 3.13, чтобы узнать, как запускать сценарий из R.

3.15. Поиск домашнего каталога R

Задача

Вам нужно знать домашний каталог R, в котором хранятся файлы конфигурации и установки.

Решение

R создает переменную среды с именем `R_HOME`, к которой можно получить доступ, используя функцию `Sys.getenv`:

```
Sys.getenv("R_HOME")
#> [1] "/Library/Frameworks/R.framework/Resources"
```

Обсуждение

Большинству пользователей никогда не потребуется знать что-либо про домашний каталог R. Но системные администраторы или опытные пользователи должны знать это, чтобы иметь возможность проверить или изменить установочные файлы R.

Когда R запускается, он определяет системную переменную *среды* (не переменную R) с именем `R_HOME`, которая представляет собой путь к домашнему каталогу R. Функция `Sys.getenv` может получить значение системной переменной среды. Ниже приводятся примеры в зависимости от платформы. Точное значение почти наверняка будет отличаться от того, что есть на вашем компьютере:

- в Windows:

```
> Sys.getenv("R_HOME")
[1] "C:/PROGRA~1/R/R-34~1.4"
```

- в macOS:

```
> Sys.getenv("R_HOME")
[1] "/Library/Frameworks/R.framework/Resources"
```

- в Linux или Unix:

```
> Sys.getenv("R_HOME")
[1] "/usr/lib/R"
```

Результат для Windows выглядит странно, потому что R сообщает старое сжатое имя пути в стиле DOS. Полный, удобный для пользователя путь в этом случае будет выглядеть так: `C:\Program Files \R\R-3.4.4`.

В Unix и macOS вы также можете запустить программу R из оболочки и использовать подкоманду `RHOME` для отображения домашнего каталога:

```
R RHOME
# /usr/lib/R
```

Обратите внимание, что домашний каталог R в Unix и macOS содержит установочные файлы, но не обязательно исполняемый файл R. Исполняемый файл может находиться в */usr/bin*, а домашний каталог R, например, может быть здесь: */usr/lib/R*.

3.16. Настройка запуска R

Задача

Вы хотите настроить сеансы R, например, изменив параметры конфигурации или предварительно загрузив пакеты.

Решение

Создайте сценарий с именем *.Rprofile*, который настраивает ваш R-сеанс. R выполнит его при запуске. Размещение сценария *.Rprofile* зависит от вашей платформы:

macOS, Linux или Unix

Сохраните файл в своем домашнем каталоге (*~/.Rprofile*).

Windows

Сохраните файл в своем каталоге *Documents*.

Обсуждение

R запускает сценарии из файла настроек при запуске, позволяя настроить параметры конфигурации R.

Вы можете создать конфигурационный файл с именем *.Rprofile* и поместить его в свой домашний каталог (macOS, Linux, Unix) или в каталог *Documents* (Windows). Этот сценарий может вызывать функции для настройки ваших сеансов, как, например, следующий простой сценарий, который устанавливает две переменные среды и устанавливает для поля ввода консоли значение `R>`:

```
Sys.setenv(DB_USERID = "my_id")
Sys.setenv(DB_PASSWORD = "My_Password!")
options(prompt = "R> ")
```

Профильный сценарий выполняется в пустой среде, поэтому существуют ограничения на то, что он может делать. Попытка открыть графическое окно не удастся, например потому, что пакет `graphics` еще не загружен. Кроме того, вы не должны пытаться выполнять длительные вычисления.

Вы можете настроить конкретный проект, поместив файл *.Rprofile* в каталог, содержащий файлы проекта. Когда R запускается в этом каталоге, он читает локальный файл *.Rprofile*; это позволяет вам выполнять настройки, относящиеся к конкретному проекту (например, задавать в качестве значения для поля ввода консоли определенное имя проекта). Однако если R находит локальный файл настроек, он *не* читает глобальный конфигурационный файл. Это может раздражать, но это легко исправить: просто используйте функцию `source`. Например, в Unix этот локальный файл настроек сначала загружает глобальный конфигурационный файл, а затем выполняет локальные инструкции:

```
source("~/.Rprofile")
#
# ... Напоминание о локальном файле .Rprofile ...
#
```

Настройка параметров

Некоторые настройки обрабатываются посредством вызовов функции `options`, которая устанавливает параметры конфигурации R. Есть много таких опций, и на странице справки по функции `options` они все перечислены:

```
help(options)
```

Вот несколько примеров:

browser="путь"

Путь к HTML-браузеру по умолчанию.

digits=n

Предлагаемое количество цифр для вывода при выводе числовых значений.

editor="путь"

Текстовый редактор по умолчанию.

prompt="строка"

Приглашение к вводу.

repos="url"

URL-адрес для хранилища пакетов по умолчанию.

warn=n

Управляет отображением предупреждающих сообщений.

Воспроизводимость

Многие из нас снова и снова используют определенные пакеты в своих сценариях (например, пакеты *tidyverse*). Очень заманчиво загрузить эти пакеты в свой сценарий *.Rprofile*, чтобы они всегда были доступны без необходимости набирать что-либо на клавиатуре. Действительно, мы давали такой совет в первом издании этой книги.

Однако у загрузки пакетов в ваш сценарий *.Rprofile* есть один недостаток. Это воспроизводимость. Если кто-то (или вы на другом компьютере) попытается запустить ваш сценарий, он может не знать, какие пакеты вы загружаете до его исполнения. Ваш сценарий может не подойти ему, в зависимости от того, какие пакеты загружает уже он сам. Поэтому, хотя загружать пакеты в *.Rprofile* может быть и удобно, у вас будет меньше проблем с коллегами (и самим собой в будущем), если будете явно вызывать функцию `library(имяпакета)` в своих сценариях.

Еще одна проблема, связанная с воспроизводимостью, – когда пользователи изменяют поведение R по умолчанию в своем сценарии *.Rprofile*. Примером этого может быть установка `options (stringsAsFactors =FALSE)`. Это удобно, поскольку многие пользователи предпочли бы данный вариант по умолчанию. Однако если кто-то выполнит сценарий без установки этой опции, он получит другие результаты или не сумеет запустить сценарий вообще, что может привести к сильному разочарованию.

В качестве руководства вы должны в первую очередь поместить в *.Rprofile* то, что:

- изменяет внешний вид R (например, `digits`);
- специфично для вашей локальной среды (например, `browser`);
- то, что должно находиться за пределами ваших сценариев (например, пароли базы данных);
- не изменяет результатов вашего анализа.

Последовательность запуска

Ниже приводится упрощенный обзор того, что происходит, когда запускается R (введите `help (Startup)`, чтобы увидеть полную информацию).

1. **R выполняет сценарий *Rprofile.site*.** Это сценарий уровня сайта, который позволяет системным администраторам переопределять параметры по умолчанию с помощью локализаций. Полный путь к сценарию: *R_HOME/etc/Rprofile.site*. (*R_HOME* – домашний каталог R; *см.* рецепт 3.15.)
2. В состав дистрибутива R не входит файл *Rprofile.site*. Скорее всего, системный администратор создаст его, если это необходимо.
3. **R выполняет сценарий *.Rprofile* в рабочем каталоге, или, если этот файл не существует, выполняет сценарий *.Rprofile* в вашем домашнем каталоге.** Это возможность пользователя настроить R для своих собственных целей. Сценарий *.Rprofile* в домашнем каталоге используется для глобальных настроек. Сценарий *.Rprofile* в каталоге более низкого уровня может выполнять определенные настройки при запуске там R – например, настраивая R при запуске в специфичном для проекта каталоге.
4. **R загружает рабочее пространство, сохраненное в *.RData*, если этот файл существует в рабочем каталоге.**
5. R сохраняет ваше рабочее пространство в файле с именем *.RData*, если он существует. Он перезагружает ваше рабочее пространство из этого файла, восстанавливая доступ к вашим локальным переменным и функциям. В RStudio это поведение можно отключить с помощью **Tools → Global Options** (Сервис → Глобальные параметры). Мы рекомендуем отключить эту опцию и всегда явно сохранять и загружать свою работу.
6. **R выполняет функцию `.First`, если она определена**. Функция `.First` – полезное место для пользователей или проектов для определения кода

инициализации при запуске. Вы можете определить ее в своем сценарии *.Rprofile* или в своей рабочей области.

7. **R выполняет функцию `.First.sys`**. На этом этапе загружаются пакеты по умолчанию. Эта функция является внутренней по отношению к R и обычно не изменяется ни пользователями, ни администраторами.

Обратите внимание, что R не загружает пакеты по умолчанию до последнего шага, когда он выполняет функцию `.First.sys`. До этого был загружен только базовый пакет. Это ключевой момент, потому что это означает, что предыдущие шаги не могут предполагать, что доступны пакеты, отличные от базового. Это также объясняет, почему попытка открыть графическое окно в сценарии *.Rprofile* оканчивается неудачей: графические пакеты еще не загружены.

См. также

См. страницы справки по `Startup(help(Startup))` и `options(help(options))`. *См.* рецепт 3.8 для получения дополнительной информации о загрузке пакетов.

3.17. Использование R и RStudio в облаке

Задача

Вы хотите запустить R и RStudio в облачной среде.

Решение

Самый простой способ использовать R в облаке – это применить веб-сервис RStudio.cloud. Чтобы воспользоваться этим сервисом, перейдите на страницу http://rstudio.cloud и настройте учетную запись, или войдите в систему, используя свои учетные данные в Google или GitHub.

Обсуждение

После входа в систему нажмите **New Project** (Новый проект), чтобы начать новый сеанс RStudio в новой рабочей области. Вас поприветствует знакомый интерфейс RStudio, показанный на рис. 3-9.

Помните, что на момент написания этих строк служба RStudio.cloud находится в режиме альфа-тестирования и может быть нестабильна на все 100 %. Ваша работа будет сохранена после выхода из системы. Однако, как и в случае с любой другой системой, рекомендуется обеспечить резервное копирование всей работы, которую вы делаете. Обычная схема работы – это подключение вашего проекта в RStudio.cloud к репозиторию GitHub и частая отправка своих изменений из Rstudio.cloud в GitHub. Данный рабочий процесс использовался при написании этой книги.

Использование Git и GitHub выходит за рамки данной книги, но если вы заинтересованы в получении дополнительной информации, мы настоятельно рекомендуем веб-книгу Дженни Брайан *Happy Git and GitHub for the useR* (https://happygitwithr.com).

В своем текущем альфа-состоянии RStudio.cloud ограничивает каждую сессию 1 ГБ ОЗУ и 3 ГБ дискового пространства – поэтому это отличная платформа для обучения и преподавания, но возможно (пока еще) это не та платформа, на кото-

рой вы захотите создать коммерческую научно-исследовательскую лабораторию обработки и анализа данных. RStudio выразил намерение предложить большую вычислительную мощность и объем хранилища как часть платного уровня обслуживания по мере развития платформы.

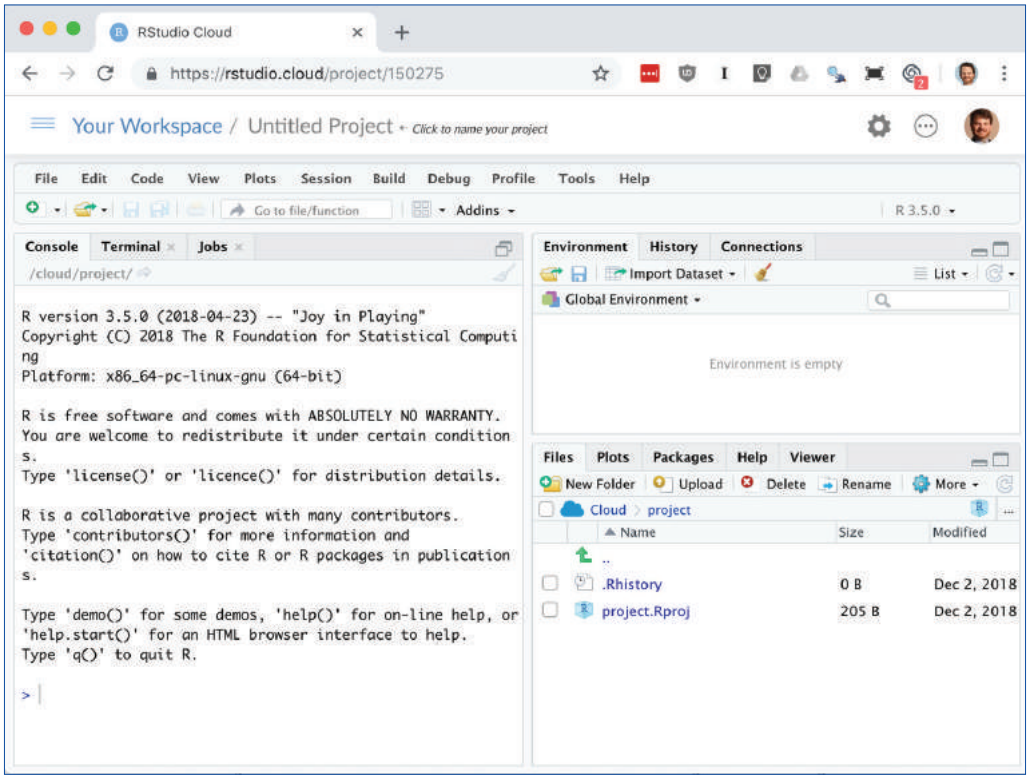


Рис. 3-9. RStudio.cloud

Если вам требуется больше вычислительной мощности, чем предлагает RStudio.cloud, и вы готовы платить за услуги, Amazon Web Services (AWS) (https://aws.amazon.com/marketplace/pp/B06W2G9PRY) и Google Cloud Platform (https://console.cloud.google.com/marketplace/details/rstudio-launcher-public/rstudio-server-pro-for-gcp?pli=1) предлагают облачные решения для RStudio. Другие облачные платформы, поддерживающие Docker, такие как Digital Ocean (https://www.digitalocean.com/community/tutorials/how-to-set-up-rstudio-on-an-ubuntu-cloud-server), также являются разумным вариантом для облачной RStudio.

Глава 4

Ввод и вывод

Вся работа со статистикой начинается с данных, а большая часть данных хранится в файлах и базах данных. Работа с исходными данными, вероятно, является первым шагом к реализации любого значимого проекта в области статистики.

Вся работа со статистикой заканчивается сообщением числовых данных клиенту, даже если вы и являетесь этим клиентом. Форматирование и создание вывода, вероятно, являются кульминацией вашего проекта.

Обычные пользователи R могут решить свои проблемы ввода, используя базовые функции пакета `readr`, такие как `read_csv` для чтения CSV-файлов и `read_delim` для чтения более сложных табличных данных, а также могут использовать функции `print`, `cat` и `format` для создания простых отчетов.

Пользователям с более серьезными требованиями к вводу/выводу (I/O) настоятельно рекомендуется прочитать руководство по импорту/экспорту данных (https://cran.r-project.org/doc/manuals/R-data.pdf). Это руководство содержит важную информацию о чтении данных из таких источников, как электронные таблицы, двоичные файлы, другие статистические системы и реляционные базы данных.

4.1. Ввод данных с клавиатуры

Задача

У вас небольшой объем данных – слишком маленький, чтобы оправдать затраты на создание входного файла. Вы просто хотите ввести данные непосредственно в свое рабочее пространство.

Решение

В случае с очень маленькими наборами данных введите данные в качестве литералов, используя конструктор `c` для векторов:

```
scores <- c(61, 66, 90, 88, 100)
```

Обсуждение

При работе над простой проблемой вы можете не захотеть создавать, а затем читать файл данных за пределами R. Возможно, вы просто захотите ввести данные в R. Самый простой способ сделать это – использовать конструктор `c` для векторов, как было показано выше.

Вы также можете применять этот подход при работе с таблицами данных, вводя каждую переменную (столбец) как вектор:

```
points <- data.frame(
   label = c("Low", "Mid", "High"),
   lbound = c(0, 0.67, 1.64),
   ubound = c(0.67, 1.64, 2.33)
)
```

См. также

Чтобы вырезать и вставить данные из другого приложения в R, обязательно ознакомьтесь с `datapasta` (https://github.com/MilesMcBain/datapasta), пакетом, который предоставляет надстройки для RStudio, упрощающие вставку данных в ваши сценарии.

4.2. Вывод меньшего числа цифр (или большего)

Задача

Ваш вывод содержит слишком много цифр или слишком мало. Вы хотите выводить меньше цифр или больше.

Решение

В случае с `print` параметр `digits` может контролировать количество выводимых цифр.

В случае с `cat` используйте функцию `format` (которая также имеет параметр `digits`), чтобы изменить форматирование чисел.

Обсуждение

R обычно форматирует вывод с плавающей точкой, чтобы цифр было семь. В большинстве случаев это работает хорошо, но может начать раздражать, если вам нужно вывести большое количество цифр в небольшом пространстве. Когда в ваших числах всего несколько значащих цифр, а R по-прежнему выводит семь, это вводит в заблуждение.

Функция `print` позволяет изменять количество выводимых цифр, используя параметр `digits`:

```
print(pi, digits = 4)
#> [1] 3.142
print(100 * pi, digits = 4)
#> [1] 314.2
```

Функция `cat` не дает вам прямого контроля над форматированием. Вместо этого используйте функцию `format` для форматирования своих чисел, прежде чем вызвать `cat`:

```
cat(pi, "\n")
#> 3.14
cat(format(pi, digits = 4), "\n")
#> 3.142
```

Это R, поэтому и `print`, и `format` сразу же будут форматировать целые векторы:

```
print(pnorm(-3:3), digits = 2)
#> [1] 0.0013 0.0228 0.1587 0.5000 0.8413 0.9772 0.9987
format(pnorm(-3:3), digits = 2)
#> [1] "0.0013" "0.0228" "0.1587" "0.5000" "0.8413" "0.9772" "0.9987"
```

Обратите внимание, что и `print`, и `format` форматируют векторные элементы последовательно, находя количество значащих цифр, необходимое для форматирования наименьшего числа, а затем форматируя все числа, чтобы у них была одинаковая ширина (хотя не обязательно количество цифр должно быть одинаковым). Это чрезвычайно полезно для форматирования всей таблицы:

```
q <- seq(from = 0, to = 3, by = 0.5)
tbl <- data.frame(Quant = q,
                  Lower = pnorm(-q),
                  Upper = pnorm(q))
tbl                             # Форматирование отсутствует.
#>   Quant   Lower   Upper
#> 1  0.0   0.50000   0.500
#> 2  0.5   0.30854   0.691
#> 3  1.0   0.15866   0.841
#> 4  1.5   0.06681   0.933
#> 5  2.0   0.02275   0.977
#> 6  2.5   0.00621   0.994
#> 7  3.0   0.00135   0.999
print(tbl, digits = 2)          # Форматирование присутствует: цифр меньше.
#>   Quant   Lower   Upper
#> 1 0.0    0.5000    0.50
#> 2 0.5    0.3085    0.69
#> 3 1.0    0.1587    0.84
#> 4 1.5    0.0668    0.93
#> 5 2.0    0.0228    0.98
#> 6 2.5    0.0062    0.99
#> 7 3.0    0.0013    1.00
```

Как видите, при форматировании всего вектора или столбца каждый элемент в векторе или столбце форматируется одинаково.

Вы также можете изменить формат *всех* выходных данных, используя функцию `options`, чтобы изменить значение по умолчанию для `digits`:

```
pi
#> [1] 3.14
options(digits = 15)
pi
#> [1] 3.14159265358979
```

Но, по нашему опыту, это плохой выбор, поскольку он также изменяет вывод встроенных функций R, и это изменение, вероятно, будет неприятным.

См. также

Другие функции для форматирования чисел включают в себя `sprintf` и `formatC`; для получения дополнительных сведений *см.* страницы со справочной информацией по ним.

4.3. Перенаправление вывода в файл

Задача

Вам нужно перенаправить вывод из R в файл вместо вашей консоли.

Решение

Можно перенаправить вывод функции cat, используя ее аргумент file:

```
cat("The answer is", answer, "\n", file = "filename.txt")
```

Используйте функцию sink, чтобы перенаправить *весь* вывод из print и cat. Вызовите функцию sink с аргументом имени файла, чтобы начать перенаправление вывода консоли в этот файл. Когда вы закончите, используйте функцию sink без аргументов, чтобы закрыть файл и возобновить запись вывода в консоль:

```
sink("filename")          # Начинаем писать вывод в файл.
# ... другой сеанс ...
sink()
```

Обсуждение

Функции print и cat обычно записывают свой вывод в вашу консоль. Функция cat пишет в файл, если вы предоставите аргумент file, который может быть как именем файла, так и соединением. Функция print не может перенаправить свой вывод, но функция sink может принудительно направить весь вывод в файл. Обычно эта функция используется, чтобы зафиксировать вывод сценария:

```
sink("script_output.txt")  # Перенаправляем вывод в файл.
source("script.R")         # Выполняем сценарий, фиксируя вывод.
sink()                     # Возобновляем запись вывода в консоль.
```

Если вы постоянно используете функцию cat при работе с одним файлом, обязательно установите для append значение TRUE. В противном случае при каждом вызове cat содержимое файла будет просто перезаписываться:

```
cat(data, file = "analysisReport.out")
cat(results, file = "analysisRepart.out", append = TRUE)
cat(conclusion, file = "analysisReport.out", append = TRUE)
```

Жесткое кодирование имен файлов, подобных этому, представляет собой утомительный и подверженный ошибкам процесс. Вы заметили, что во второй строке имя файла написано с ошибкой? Вместо того чтобы многократно кодировать имя файла, мы предлагаем открыть соединение с файлом и записать свой вывод в это соединение:

```
con <- file("analysisReport.out", "w")
cat(data, file = con)
cat(results, file = con)
cat(conclusion, file = con)
close(con)
```

(При записи в соединение не нужно устанавливать для append значение TRUE, потому что в данном случае append является значением по умолчанию.) Этот метод особенно полезен в сценариях R, поскольку он делает ваш код более надежным и понятным.

4.4. Список файлов

Задача

Вам нужен вектор R, представляющий собой список файлов в вашем рабочем каталоге.

Решение

Функция `list.files` показывает содержимое вашего рабочего каталога:

```
list.files()
#> [1] "_book"                    "_bookdown_files"
#> [3] "_bookdown.yml"            "_common.R"
#> [5] "_main.log"                "_main.rds"
#> [7] "_output.yml"              "01_GettingStarted_cache"
#> [9] "01_GettingStarted.md"     "01_GettingStarted.Rmd"
#> # etc.
```

Обсуждение

Эта функция очень удобна для получения имен всех файлов в подкаталоге. Вы можете использовать ее, чтобы освежить память касательно имен файлов или, что более вероятно, в качестве ввода в другой процесс, например импорта файлов данных.

Вы можете передать функции `list.files` путь и шаблон, чтобы показать файлы в определенном пути и соответствовать определенному шаблону регулярного выражения:

```
list.files(path = 'data/') # show files in a directory
#> [1] "ac.rdata"                 "adf.rdata"
#> [3] "anova.rdata"              "anova2.rdata"
#> [5] "bad.rdata"                "batches.rdata"
#> [7] "bnd_cmty.Rdata"           "compositePerf-2010.csv"
#> [9] "conf.rdata"               "daily.prod.rdata"
#> [11] "data1.csv"               "data2.csv"
#> [13] "datafile_missing.tsv"    "datafile.csv"
#> [15] "datafile.fwf"            "datafile.qsv"
#> [17] "datafile.ssv"            "datafile.tsv"
#> [19] "datafile1.ssv"           "df_decay.rdata"
#> [21] "df_squared.rdata"        "diffs.rdata"
#> [23] "example1_headless.csv"   "example1.csv"
#> [25] "excel_table_data.xlsx"   "get_USDA_NASS_data.R"
#> [27] "ibm.rdata"               "iris_excel.xlsx"
#> [29] "lab_df.rdata"            "movies.sas7bdat"
#> [31] "nacho_data.csv"          "NearestPoint.R"
#> [33] "not_a_csv.txt"           "opt.rdata"
#> [35] "outcome.rdata"           "pca.rdata"
#> [37] "pred.rdata"              "pred2.rdata"
#> [39] "sat.rdata"               "singles.txt"
#> [41] "state_corn_yield.rds"    "student_data.rdata"
#> [43] "suburbs.txt"             "tab1.csv"
#> [45] "tls.rdata"               "triples.txt"
#> [47] "ts_acf.rdata"            "workers.rdata"
#> [49] "world_series.csv"        "xy.rdata"
#> [51] "yield.Rdata"             "z.RData"
```

```
list.files(path = 'data/', pattern = '\\.csv')
#> [1] "compositePerf-2010.csv"    "data1.csv"
#> [3] "data2.csv"                 "datafile.csv"
#> [5] "example1_headless.csv"     "example1.csv"
#> [7] "nacho_data.csv"            "tab1.csv"
#> [9] "world_series.csv"
```

Чтобы увидеть все файлы в своих подкаталогах, также используйте:

```
list.files(recursive = T)
```

Возможная «ловушка» функции `list.files` заключается в том, что она игнорирует скрытые файлы – как правило, это любой файл, имя которого начинается с точки. Если вы не видите файл, который ожидали увидеть, попробуйте установить для `all.files` значение `TRUE`:

```
list.files(path = 'data/', all.files = TRUE)
#> [1] "."                         ".."
#> [3] ".DS_Store"                 ".hidden_file.txt"
#> [5] "ac.rdata"                  "adf.rdata"
#> [7] "anova.rdata"               "anova2.rdata"
#> [9] "bad.rdata"                 "batches.rdata"
#> [11] "bnd_cmty.Rdata"           "compositePerf-2010.csv"
#> [13] "conf.rdata"               "daily.prod.rdata"
#> [15] "data1.csv"                "data2.csv"
#> [17] "datafile_missing.tsv"     "datafile.csv"
#> [19] "datafile.fwf"             "datafile.qsv"
#> [21] "datafile.ssv"             "datafile.tsv"
#> [23] "datafile1.ssv"            "df_decay.rdata"
#> [25] "df_squared.rdata"         "diffs.rdata"
#> [27] "example1_headless.csv"    "example1.csv"
#> [29] "excel_table_data.xlsx"    "get_USDA_NASS_data.R"
#> [31] "ibm.rdata"                "iris_excel.xlsx"
#> [33] "lab_df.rdata"             "movies.sas7bdat"
#> [35] "nacho_data.csv"           "NearestPoint.R"
#> [37] "not_a_csv.txt"            "opt.rdata"
#> [39] "outcome.rdata"            "pca.rdata"
#> [41] "pred.rdata"               "pred2.rdata"
#> [43] "sat.rdata"                "singles.txt"
#> [45] "state_corn_yield.rds"     "student_data.rdata"
#> [47] "suburbs.txt"              "tab1.csv"
#> [49] "tls.rdata"                "triples.txt"
#> [51] "ts_acf.rdata"             "workers.rdata"
#> [53] "world_series.csv"         "xy.rdata"
#> [55] "yield.Rdata"              "z.RData"
```

Если вы просто хотите увидеть, какие файлы находятся в каталоге, а не использовать имена файлов в процедуре, самый простой способ – открыть панель **Files** (Файлы) в нижнем правом углу RStudio. Но имейте в виду, что эта панель скрывает файлы, которые начинаются с точки, как видно на рис. 4-1.

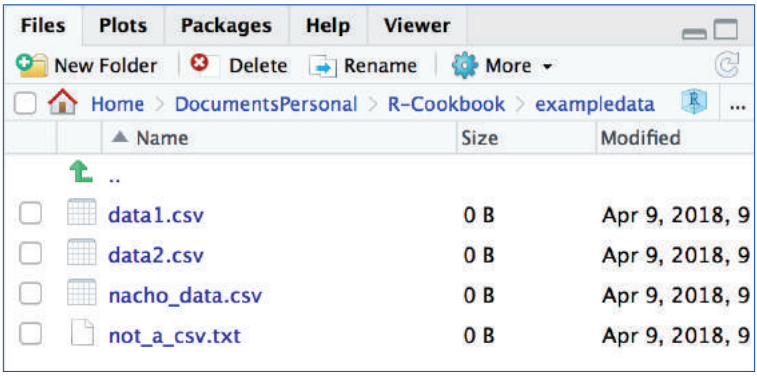


Рис. 4-1. Панель **Files**

См. также

В R есть и другие удобные функции для работы с файлами; *см.* `help(files)`.

4.5. Не удается открыть файл в Windows – что делать?

Задача

Вы запускаете R в Windows и используете такие имена файлов, как *C:\data\sample.txt*. R говорит, что не может открыть файл, но вы знаете, что этот файл существует.

Решение

Обратные слеши в пути к файлам вызывают проблемы. Эту проблему можно решить одним из двух способов:

- измените обратную косую черту на прямую: «`C:/data/sample.txt`»;
- сделайте обратную косую черту двойной: «`C:\\data\\sample.txt`».

Обсуждение

Когда вы открываете файл в R, то даете имя файла в виде строки символов. Проблемы возникают, когда имя содержит обратную косую черту (\), потому что эти символы содержат особое значение внутри строк. Вы, вероятно, получите что-то вроде этого:

```
samp <- read_csv("C:\Data\sample-data.csv")
#> Error: '\D' is an unrecognized escape in character string starting ""C:\D"
```

R избегает всех символов, которые следуют за обратной косой чертой, а затем удаляет эти косые. В результате этого остается бессмысленный путь к файлу, такой как `C:Datasample-data.csv` в этом примере.

Простое решение – использовать прямые косые вместо обратных. R оставляет прямые косые, а Windows воспринимает их как обратную косую черту. Задача решена:

```
samp <- read_csv("C:/Data/sample-data.csv")
```

Альтернативным решением является удвоение обратной косой черты, поскольку R заменяет две последовательные обратные косые черты одной обратной косой:

```
samp <- read_csv("C:\\Data\\sample-data.csv")
```

4.6. Чтение записей фиксированной ширины

Задача

Вы читаете данные из файла записей фиксированной ширины: записи, элементы данных которых находятся на фиксированных границах.

Решение

Используйте функцию `read_fwf` из пакета `readr` (который является частью *tidyverse*). Основными аргументами являются имя файла и описание полей:

```
library(tidyverse)
records <- read_fwf("myfile.txt",
                    fwf_cols(col1 = 10,
                             col2 = 7))
records
```

Эта форма использует параметр `fwf_cols` для передачи имен и ширины столбцов в функцию. Параметры столбца также можно передавать другими способами, которые будут описаны далее.

Обсуждение

Для чтения данных в R мы настоятельно рекомендуем пакет `readr`. Существуют функции базовой версии R для чтения в текстовых файлах, но `readr` улучшает эти базовые функции с помощью большей производительности, более подходящих значений по умолчанию и большей гибкости.

Предположим, что мы хотим прочитать весь файл записей фиксированной ширины, такой как *fixed-width.txt*, показанный здесь:

```
Fisher R.A. 1890 1962
Pearson Karl 1857 1936
Cox Gertrude 1900 1978
Yates Frank 1902 1994
Smith Kirstine 1878 1939
```

Нам нужно знать ширину столбцов. В этом случае столбцы выглядят так:

- фамилия, 10 символов;
- имя, 10 символов;
- год рождения, 5 знаков;
- год смерти, 5 символов.

Существует пять различных способов определения столбцов с использованием `read_fwf`. Выберите тот, который проще всего использовать (или запомнить) в вашей ситуации:

- `read_fwf` может попытаться угадать ширину вашего столбца, если между столбцами есть свободное пространство, с помощью опции `fwf_empty`:

```
file <- "./data/datafile.fwf"
t1 <- read_fwf(file,
          fwf_empty(file,
          col_names = c("last", "first", "birth", "death")))
#> Parsed with column specification:
#> cols(
#> last = col_character(),
#> first = col_character(),
#> birth = col_double(),
#> death = col_double()
#> )
```

- можно определить каждый столбец вектором ширины, за которым следует вектор имен, с помощью `fwf_widths`:

```
t2 <- read_fwf(file, fwf_widths(c(10, 10, 5, 4),
                                c("last", "first", "birth", "death")))
#> Parsed with column specification:
#> cols(
#>    last = col_character(),
#>    first = col_character(),
#>    birth = col_double(),
#>    death = col_double()
#> )
```

- столбцы можно определить с помощью функции `fwf_cols`, которая принимает серию имен столбцов, за которыми следуют ширины столбцов:

```
t3 <-
   read_fwf("./data/datafile.fwf",
            fwf_cols(
              last = 10,
              first = 10,
              birth = 5,
              death = 5
            ))
#> Parsed with column specification:
#> cols(
#>    last = col_character(),
#>    first = col_character(),
#>    birth = col_double(),
#>    death = col_double()
#> )
```

- каждый столбец можно определить начальной и конечной позициями с помощью `fwf_cols`:

```
t4 <- read_fwf(file, fwf_cols(
   last = c(1, 10),
   first = c(11, 20),
   birth = c(21, 25),
   death = c(26, 30)
))
#> Parsed with column specification:
#> cols(
#>    last = col_character(),
#>    first = col_character(),
#>    birth = col_double(),
#>    death = col_double()
#> )
```

- вы также можете определить столбцы с вектором начальных позиций, вектором конечных позиций и вектором имен столбцов с помощью `fwf_positions`:

```
t5 <- read_fwf(file, fwf_positions(
   c(1, 11, 21, 26),
   c(10, 20, 25, 30),
   c("first", "last", "birth", "death")
))
#> Parsed with column specification:
#> cols(
#>   first = col_character(),
#>   last = col_character(),
#>   birth = col_double(),
#>   death = col_double()
#> )
```

Функция `read_fwf` возвращает *tibble*, который представляет собой разновидность таблицы данных, предоставляемой коллекцией tidyverse. Как и в случае с пакетами tidyverse, `read_fwf` имеет хороший набор значений по умолчанию, которые делают ее менее сложной в использовании, по сравнению с некоторыми функциями базовой версии R для импорта данных.

Например, `read_fwf` по умолчанию импортирует символьные поля как символы, а не как факторы, что избавляет пользователей от неудобств.

См. также

См. рецепт 4.7, где более подробно обсуждается чтение текстовых файлов.

4.7. Чтение файлов табличных данных

Задача

Вы хотите прочитать текстовый файл, который содержит таблицу данных, разделенных пробелами.

Решение

Используйте функцию `read_table2` из пакета `readr`, которая возвращает tibble:

```
library(tidyverse)

tab1 <- read_table2("./data/datafile.tsv")
#> Parsed with column specification:
#> cols(
#>   last = col_character(),
#>   first = col_character(),
#>   birth = col_double(),
#>   death = col_double()
#> )
tab1
#> # Tibble: 5 x 4
#>   last    first   birth   death
#>   <chr>   <chr>   <dbl>   <dbl>
#> 1 Fisher  R.A.    1890    1962
#> 2 Pearson Karl    1857    1936
```

```
#> 3 Cox    Gertrude   1900    1978
#> 4 Yates    Frank    1902    1994
#> 5 Smith Kirstine    1878    1939
```

Обсуждение

Файлы табличных данных довольно распространены. Это текстовые файлы в простом формате:

- каждая строка содержит одну запись;
- внутри каждой записи поля (элементы) разделены пробелами, такими как пропуск или табуляция;
- каждая запись содержит одинаковое количество полей.

Этот формат имеет более свободную форму, нежели формат с фиксированной шириной, поскольку поля не нужно выравнивать по позиции. Ниже приводится файл данных из рецепта 4.6 в табличном формате с использованием символа табуляции между полями:

```
last    first   birth death
Fisher  R.A.    1890  1962
Pearson Karl    1857  1936
Cox   Gertrude  1900  1978
Yates   Frank   1902  1994
Smith Kirstine  1878  1939
```

Функция `read_table2` предназначена для того, чтобы строить подходящие догадки относительно ваших данных. Она предполагает, что у ваших данных есть имена столбцов в первой строке. Она угадывает ваш разделитель и приписывает типы столбцов на основе первых 1000 записей в вашем наборе данных. Далее приводится пример с данными, разделенными пробелами.

Исходный файл выглядит так:

```
last first birth death
Fisher R.A. 1890 1962
Pearson Karl 1857 1936
Cox Gertrude 1900 1978
Yates Frank 1902 1994
Smith Kirstine 1878 1939
```

`read_table2` делает рациональные предположения:

```
t <- read_table2("./data/datafile1.ssv")
#> Parsed with column specification:
#> cols(
#>   last = col_character(),
#>   first = col_character(),
#>   birth = col_double(),
#>   death = col_double()
#> )
print(t)
#> # Tibble: 5 x 4
#>   last    first   birth death
#>   <chr>   <chr>   <dbl> <dbl>
#> 1 Fisher  R.A.    1890  1962
#> 2 Pearson Karl    1857  1936
#> 3 Cox   Gertrude  1900  1978
```

```
#> 4  Yates  Frank     1902  1994
#> 5  Smith  Kirstine  1878  1939
```

`read_table2` часто угадывает правильно. Но, как и в случае с другими функциями импорта `readr`, вы можете перезаписать значения по умолчанию явными параметрами:

```
t <-
 read_table2(
   "./data/datafile1.ssv",
   col_types = c(
     col_character(),
     col_character(),
     col_integer(),
     col_integer()
   )
)
```

Если какое-либо поле содержит строку `"NA"`, `read_table2` предполагает, что значение отсутствует, и преобразует его в `NA`. Ваш файл данных может использовать другую строку для обозначения пропущенных значений, в этом случае используйте параметр `na`. Например, в соглашении SAS сказано, что пропущенные значения обозначаются одной точкой (.). Мы можем читать такие текстовые файлы, используя опцию `na="."`. Если у нас есть файл с именем *datafile_missing.tsv* с пропущенным значением, обозначенным знаком `.` в последнем ряду:

```
last    first     birth death
Fisher  R.A.      1890  1962
Pearson Karl      1857  1936
Cox     Gertrude  1900  1978
Yates   Frank     1902  1994
Smith   Kirstine  1878  1939
Cox     David     1924  .
```

мы можем импортировать его так:

```
t <- read_table2("./data/datafile_missing.tsv", na = ".")
#> Parsed with column specification:
#> cols(
#>   last = col_character(),
#>   first = col_character(),
#>   birth = col_double(),
#>   death = col_double()
#> )
t
#> # Tibble: 6 x 4
#>   last    first     birth death
#>   <chr>   <chr>     <dbl> <dbl>
#> 1 Fisher  R.A.       1890  1962
#> 2 Pearson Karl       1857  1936
#> 3 Cox     Gertrude   1900  1978
#> 4 Yates   Frank      1902  1994
#> 5 Smith   Kirstine   1878  1939
#> 6 Cox     David      924   NA
```

Мы большие поклонники данных *с самоописанием*: файлов данных, которые описывают собственное содержимое. (Специалист в области информатики сказал

бы, что этот файл содержит собственные метаданные.) Функция `read_table2` делает допущение по умолчанию, что первая строка вашего файла содержит строку заголовка с именами столбцов. Если у вашего файла нет имен столбцов, можно отключить его с помощью параметра `col_names = FALSE`.

Дополнительный тип метаданных, поддерживаемый `read_table2`, – это строки комментариев. Используя параметр `comment`, можно сообщить `read_table2`, какой символ различает строки комментариев. Приведенный ниже файл содержит строку комментария в верхней части, которая начинается с `#`:

```
# Ниже приводится список статистиков.
last first birth death
Fisher R.A. 1890 1962
Pearson Karl 1857 1936
Cox Gertrude 1900 1978
Yates Frank 1902 1994
Smith Kirstine 1878 1939
```

поэтому можно импортировать этот файл следующим образом:

```
#> Parsed with column specification:
#> cols(
#>   last = col_character(),
#>   first = col_character(),
#>   birth = col_double(),
#>   death = col_double()
#> )
t
#> # Tibble: 5 x 4
#>   last    first     birth death
#>   <chr>   <chr>     <dbl> <dbl>
#> 1 Fisher  R.A.       1890  1962
#> 2 Pearson Karl       1857  1936
#> 3 Cox     Gertrude   1900  1978
#> 4 Yates   Frank      1902  1994
#> 5 Smith   Kirstine   1878  1939
```

У функции `read_table2` есть множество параметров для управления чтением и интерпретацией входного файла. *См.* страницу справки (`?read_table2`) или сопроводительную документацию (`vignette("readr")`) для получения дополнительной информации. Если вас интересует, в чем состоит разница между `read_table` и `read_table2`, эти сведения есть в файле справки.., но краткий ответ таков: функция `read_table` несколько менее снисходительна в том, что касается структуры файла и длины строки.

См. также

Если ваши элементы данных разделены запятыми, *см.* рецепт 4.8, чтобы узнать, как читать файл CSV.

4.8. Чтение из файлов CSV

Задача

Вы хотите прочитать данные из файла значений, разделенных запятыми (CSV).

Решение

Функция read_csv из пакета readr – это быстрый (и, согласно документации, занятный) способ чтения файлов CSV. Если в вашем CSV-файле есть строка заголовка, используйте это:

```
library(tidyverse)

tbl <- read_csv("datafile.csv")
```

Если ваш CSV-файл не содержит строку заголовка, установите для опции col_names значение FALSE:

```
tbl <- read_csv("datafile.csv", col_names = FALSE)
```

Обсуждение

Формат файлов CSV популярен, потому что многие программы могут импортировать и экспортировать данные в этом формате. К ним относятся R, Excel, другие программы для работы с электронными таблицами, большое количество менеджеров баз данных и большинство статистических пакетов. Файл CSV представляет собой плоский файл табличных данных, где каждая строка в файле – это строка данных, а каждая строка содержит элементы данных, разделенные запятыми. Вот очень простой CSV-файл с тремя строками и тремя столбцами. Первая строка – это строка заголовка, которая содержит имена столбцов, также разделенных запятыми:

```
label,lbound,ubound
low,0,0.674
mid,0.674,1.64
high,1.64,2.33
```

Функция read_csv читает данные и создает tibble. Она предполагает, что в вашем файле есть строка заголовка, если не указано иное:

```
tbl <- read_csv("./data/example1.csv")
#> Parsed with column specification:
#> cols(
#>   label = col_character(),
#>   lbound = col_double(),
#>   ubound = col_double()
#> )
tbl
#> # Tibble: 3 x 3
#> label   lbound  ubound
#> <chr>   <dbl>   <dbl>
#> 1 low   0       0.674
#> 2 mid   0.674   1.64
#> 3 high  1.64    2.33
```

Обратите внимание, что функция read_csv взяла имена столбцов из строки заголовка для tibble. Если бы в файле не было заголовка, мы бы указали для опции col_names значение FALSE, и R синтезировал бы имена столбцов за нас (в этом случае X1, X2 и X3):

```
tbl <- read_csv("./data/example1.csv", col_names = FALSE)
#> Parsed with column specification:
```

```
#> cols(
#>   X1 = col_character(),
#>   X2 = col_character(),
#>   X3 = col_character()
#> )
tbl
#> # Tibble: 4 x 3
#> X1 X2 X3
#>   <chr> <chr>   <chr>
#> 1 label lbound  ubound
#> 2 low     0     0.674
#> 3 mid   0.674   1.64
#> 4 high  1.64    2.33
```

Иногда удобно помещать метаданные в файлы. Если эти метаданные начинаются с распространенного символа, такого как знак решетки (#), мы можем использовать параметр `comment=FALSE`, чтобы игнорировать строки метаданных.

У функции `read_csv` есть много полезных «наворотов». Некоторые из этих параметров и их значения по умолчанию включают в себя:

`na = c("", "NA")`
: Указывает, какие значения представляют пропущенные или недоступные значения.

`comment = ""`
: Указывает, какие строки игнорировать как комментарии или метаданные.

`trim_ws = TRUE`
: Указывает, следует ли удалять пробелы в начале и/или конце полей.

`skip = 0`
: Указывает количество пропускаемых строк в начале файла.

`guess_max = min(1000, n_max)`
: Указывает количество строк, которые следует учитывать при анализе типов столбцов.

См. страницу справки R `help(read_csv)` для получения более подробной информации обо всех доступных опциях.

Если у вас есть файл данных, который использует точку с запятой (;) в качестве разделителей и запятые в качестве десятичного разделителя, как это обычно бывает за пределами Северной Америки, вам следует использовать функцию `read_csv2`, которая была создана именно для такой ситуации.

См. также

См. рецепт 4.9. Также *см.* сопроводительную документацию `vignette(readr)`.

4.9. Запись в файлы CSV

Задача

Вы хотите сохранить матрицу или таблицу данных в файле, используя формат значений, разделенных запятой.

Решение

Функция `write_csv` из пакета `readr` коллекции tidyverse может выполнить запись в файл CSV:

```
library(tidyverse)

write_csv(df, path = "outfile.csv")
```

Обсуждение

Функция `write_csv` записывает табличные данные в файл ASCII в формате CSV. Каждая строка данных создает одну строку в файле с элементами данных, разделенными запятыми (,). Можно начать с таблицы данных `tab1`, которую мы создали ранее в рецепте 4.7:

```
library(tidyverse)

write_csv(tab1, "./data/tab1.csv")
```

В этом примере мы создаем файл с именем *tab1.csv* в каталоге данных, который является подкаталогом текущего рабочего каталога. Вот как он выглядит:

```
last,first,birth,death
Fisher,R.A.,1890,1962
Pearson,Karl,1857,1936
Cox,Gertrude,1900,1978
Yates,Frank,1902,1994
Smith,Kirstine,1878,1939
```

У функции `write_csv` есть ряд параметров, которые обычно имеют очень хорошие значения по умолчанию. Если вы хотите настроить вывод, вот несколько параметров, которые можно изменить наряду с их значениями по умолчанию:

`col_names = TRUE`

Указывает, содержит ли первая строка имена столбцов.

`col_types = NULL`

Функция `write_csv` будет просматривать первые 1000 строк (изменяемые с помощью `guess_max`) и строить обоснованное предположение относительно того, какие типы данных использовать для столбцов. Если вы предпочитаете явно указывать типы столбцов, можете сделать это, передав вектор типов столбцов в параметр `col_types`.

`na = c("", "NA")`

Указывает, какие значения представляют собой отсутствующие или недоступные значения.

`comment = ""`

Указывает, какие строки игнорировать как комментарии или метаданные.

`trim_ws = TRUE`

Указывает, следует ли удалять пробелы в начале и/или конце полей.

`skip = 0`

Указывает количество пропускаемых строк в начале файла.

```
guess_max = min(1000, n_max)
```

Указывает количество строк, которые следует учитывать при приписывании типов столбцов.

См. также

См. рецепт 3.1 для получения дополнительной информации о текущем рабочем каталоге и рецепт 4.18, чтобы узнать о других способах сохранения данных в файлах. Для получения дополнительной информации о чтении и записи текстовых файлов *см.* `vignette(readr)`.

4.10. Чтение табличных или CSV-данных из интернета

Задача

Вы хотите читать данные непосредственно из интернета в рабочее пространство R.

Решение

Воспользуйтесь функциями `read_csv` или `read_table2` из пакета `readr`, используя URL-адрес вместо имени файла. Эти функции будут выполнять чтение непосредственно с удаленного сервера:

```
library(tidyverse)

berkley <- read_csv('http://bit.ly/barkley18', comment = '#')
#> Parsed with column specification:
#> cols(
#>   Name = col_character(),
#>   Location = col_character(),
#>   Time = col_time(format = "")
#> )
```

Вы также можете открыть соединение, используя URL-адрес, а затем выполнить чтение из этого соединения, что может быть предпочтительнее для сложных файлов.

Обсуждение

Интернет – золотое дно данных. Вы можете скачивать данные в файл, а затем читать его в R, но удобнее делать это напрямую из интернета. Присвойте URL-адрес `read_csv`, `read_table2` или другой функции чтения в `readr` (в зависимости от формата данных), и данные будут скачаны и проанализированы. Ни забот, ни хлопот.

За исключением использования URL-адреса, этот рецепт аналогичен чтению из файла CSV (см. рецепт 4.8) или сложного файла (рецепт 4.15), поэтому все комментарии в этих рецептах также применимы и здесь.

Помните, что URL-адреса подходят не только для HTTP-, но и для FTP-серверов. Это означает, что R также может читать данные с FTP-сайтов, используя URL-адреса:

```
tbl <- read_table2("ftp://ftp.example.com/download/data.txt")
```

См. также

См. рецепты 4.8 и 4.15.

4.11. Чтение данных из Excel

Задача

Вы хотите прочитать данные из файла Excel.

Решение

Пакет `openxlsx` облегчает чтение файлов Excel:

```
library(openxlsx)
df1 <- read.xlsx(xlsxFile = "file.xlsx",
                 sheet = 'sheet_name')
```

Обсуждение

Пакет `openxlsx` – неплохой вариант, который подходит как для чтения, так и для записи файлов Excel. Если мы читаем весь лист, можно просто использовать функцию `read.xlsx`. Нам нужно только передать имя файла и, при желании, имя листа, который мы хотим импортировать:

```
library(openxlsx)

df1 <- read.xlsx(xlsxFile = "data/iris_excel.xlsx",
                 sheet = 'iris_data')
head(df1, 3)
#>   Sepal.Length  Sepal.Width   Petal.Length  Petal.Width  Species
#> 1          5.1           3.5           1.4           0.2   setosa
#> 2          4.9           3.0           1.4           0.2   setosa
#> 3          4.7           3.2           1.3           0.2   setosa
```

Но `openxlsx` поддерживает и более сложные рабочие процессы.

Распространенным шаблоном является чтение именованной таблицы из файла Excel в таблицу данных R. Это более хитрый вариант, потому что лист, из которого мы читаем, может содержать значения за пределами именованной таблицы, но мы хотим выполнять чтение только в диапазоне именованной таблицы. Можно использовать функции в `openxlsx`, чтобы получить местоположение таблицы, а затем прочитать этот диапазон ячеек в таблицу данных.

Сначала мы загружаем всю книгу в R:

```
library(openxlsx)
wb <- loadWorkbook("data/excel_table_data.xlsx")
```

Затем можем использовать функцию `getTables`, чтобы получить имена и диапазоны всех таблиц Excel в листе `input_data` и выбрать нужную таблицу. В этом примере таблица Excel, которую мы ищем, носит название `example_table`:

```
tables <- getTables(wb, 'input_data')
table_range_str <- names(tables[tables == 'example_table'])
table_range_refs <- strsplit(table_range_str, ':')[[1]]

# Используем регулярное выражение, чтобы извлечь номера строк.
table_range_row_num <- gsub("[^0-9.]", "", table_range_refs)

# Извлекаем номера столбцов.
table_range_col_num <- convertFromExcelRef(table_range_refs)
```

Теперь вектор `col_vec` содержит номера столбцов нашей именованной таблицы, а `table_range_row_num` – номера строк. Теперь мы можем использовать функцию `read.xlsx`, чтобы получить только те строки и столбцы, которые нам нужны:

```
df <- read.xlsx(
  xlsxFile = "data/excel_table_data.xlsx",
  sheet = 'input_data',
  cols = table_range_col_num[1]:table_range_col_num[2],
  rows = table_range_row_num[1]:table_range_row_num[2]
)
```

Хотя это может показаться сложным, данный шаблон проектирования может избавить вас от мороки при совместном использовании данных с аналитиками, которые применяют структурированные файлы Excel, содержащие именованные таблицы.

См. также

Вы можете посмотреть сопроводительную документацию из `openxlsx`, установив `openxlsx` и выполнив команду `vignette('Introduction', package='openxlsx')`.

Пакет `readxl` (https://readxl.tidyverse.org) является частью tidyverse и обеспечивает быстрое и простое чтение файлов Excel. Однако в настоящее время он не поддерживает именованные таблицы Excel.

Пакет `writexl` (https://cran.r-project.org/web/packages/writexl/index.html) – это быстрый и легкий (без зависимостей) пакет для записи файлов Excel (обсуждается в рецепте 4.12).

4.12. Запись таблицы данных в Excel

Задача

Вам нужно записать таблицу данных в файл Excel.

Решение

Пакет `openxlsx` относительно упрощает запись в файлы Excel. Хотя в нем есть много опций, типичным шаблоном является указание имени файла Excel и имени листа:

```
library(openxlsx)
write.xlsx(df,
           sheetName = "some_sheet",
           file = "out_file.xlsx")
```

Обсуждение

Пакет `openxlsx` содержит огромное количество опций для управления множеством аспектов объектной модели Excel. Его можно использовать, например, для установки цветов ячеек, определения именованных диапазонов и контуров ячеек. Он также имеет ряд вспомогательных функций, таких как `write.xlsx`, которые сильно упрощают выполнение простых задач.

Когда компании работают с Excel, рекомендуется хранить все входные данные в файле Excel в именованной таблице, что упрощает доступ к данным

и снижает вероятность ошибок. Однако если вы используете `openxlsx` для перезаписи таблицы Excel на одном из листов, вы подвергаете себя риску, потому что новые данные могут содержать меньше строк, чем таблица Excel, которую они заменяют. Это может привести к ошибкам, поскольку вы получите старые и новые данные в последовательных строках. Решение состоит в том, чтобы сначала удалить существующую таблицу Excel, затем добавить новые данные обратно в то же место и присвоить новые данные именованной таблице Excel. Для этого нужно использовать более продвинутые возможности манипулирования Excel в `openxlsx`.

Сначала мы используем функцию `loadWorkbook`, чтобы полностью прочитать книгу Excel в R:

```
library(openxlsx)

wb <- loadWorkbook("data/excel_table_data.xlsx")
```

Прежде чем удалить таблицу, мы хотим извлечь начальную строку и столбец таблицы:

```
tables <- getTables(wb, 'input_data')
table_range_str <- names(tables[tables == 'example_table'])
table_range_refs <- strsplit(table_range_str, ':')[[1]]

# Используем регулярное выражение, чтобы извлечь номер начальной строки.
table_row_num <- gsub("[^0-9.]", "", table_range_refs)[[1]]

# Извлекаем номер начального столбца.
table_col_num <- convertFromExcelRef(table_range_refs)[[1]]
```

Затем можно использовать функцию `removeTable`, чтобы удалить существующую именованную таблицу Excel:

```
removeTable(wb = wb,
            sheet = 'input_data',
            table = 'example_table')
```

Теперь мы можем использовать функцию `writeDataTable` для записи таблицы данных `iris` (которая поставляется с R) обратно в наш объект workbook в R:

```
writeDataTable(
    wb = wb,
    sheet = 'input_data',
    x = iris,
    startCol = table_col_num,
    startRow = table_row_num,
    tableStyle = "TableStyleLight9",
    tableName = 'example_table'
)
```

На этом этапе мы можем сохранить рабочую книгу, и наша таблица будет обновлена. Однако рекомендуется сохранить некоторые метаданные в рабочей книге, чтобы другие точно знали, когда данные были обновлены. Это можно сделать с помощью функции `writeData`, затем сохранить книгу в файл и перезаписать исходный файл. В этом примере мы поместим текст метаданных в ячейку `B:5`, а затем сохраним книгу обратно в файл, перезаписав исходник:

```
writeData(
  wb = wb,
  sheet = 'input_data',
  x = paste('example_table data refreshed on:', Sys.time()),
  startCol = 2,
  startRow = 5
)

# Затем сохраняем рабочую книгу.
saveWorkbook(wb = wb,
             file = "data/excel_table_data.xlsx",
             overwrite = TRUE)
```

Результат показан на рис. 4-2.

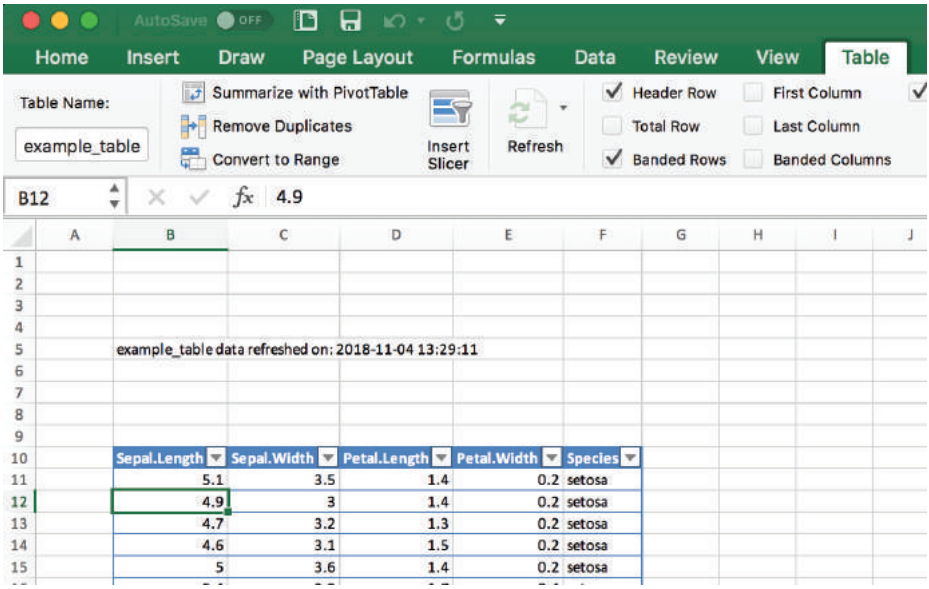


Рис. 4-2. Таблица Excel и текст метаданных

См. также

Вы можете посмотреть сопроводительную документацию из `openxlsx`, установив `openxlsx` и выполнив команду `vignette('Introduction', package='openxlsx')`.

Пакет `readxl` (https://readxl.tidyverse.org) является частью коллекции tidyverse и обеспечивает быстрое и простое чтение файлов Excel (обсуждается в рецепте 4.11).

Пакет `writexl` (https://cran.r-project.org/web/packages/writexl/index.html) – это быстрый и легкий (без зависимостей) пакет для записи файлов Excel.

4.13. Чтение данных из файла SAS

Задача

Вам нужно прочитать набор данных программного обеспечения для статистического анализа (SAS) в таблицу данных R.

Решение

Пакет `sas7bdat` поддерживает чтение файлов *.sas7bdat* в R:

```
library(haven)

sas_movie_data <- read_sas("data/movies.sas7bdat")
```

Обсуждение

SAS V7 и более поздние версии поддерживают формат файлов *.sas7bdat*. Функция `read_sas` в `haven` поддерживает чтение формата файла *.sas7bdat*, включая метки переменных. Если в вашем файле SAS есть переменные метки, то при импорте в R они будут сохранены в атрибутах `label` таблицы данных. По умолчанию эти метки выводиться не будут. Их можно увидеть, открыв таблицу данных в RStudio или вызвав функцию базовой версии R `attributes` для каждого столбца:

```
sapply(sas_movie_data, attributes)
#> $Movie
#> $Movie$label
#> [1] "Movie"
#>
#>
#> $Type
#> $Type$label
#> [1] "Type"
#>
#>
#> $Rating
#> $Rating$label
#> [1] "Rating"
#>
#>
#> $Year
#> $Year$label
#> [1] "Year"
#>
#>
#> $Domestic__
#> $Domestic__$label
#> [1] "Domestic $"
#>
#> $Domestic__$format.sas
#> [1] "F"
#>
#>
#> $Worldwide__
#> $Worldwide__$label
#> [1] "Worldwide $"
#>
#> $Worldwide__$format.sas
#> [1] "F"
#>
#>
#> $Director
#> $Director$label
#> [1] "Director"
```

См. также

Пакет `sas7bdat` намного медлительнее при работе с большими файлами по сравнению с `haven`, но имеет более сложную поддержку атрибутов файлов. Если вас интересуют метаданные SAS, вам следует изучить `sas7bdat::read.sas7bdat`.

4.14. Чтение данных из таблиц HTML

Задача

Вам нужно прочитать данные из таблицы HTML в интернете.

Решение

Используйте функции `read_html` и `html_table` в пакете `rvest`. Чтобы прочитать все таблицы на странице, сделайте следующее:

```
library(rvest)
library(tidyverse)

all_tables <-
  read_html("url") %>%
  html_table(fill = TRUE, header = TRUE)
```

Обратите внимание, что `rvest` устанавливается при запуске `install.packages('tidyverse')`, но это не основной пакет tidyverse. Поэтому вы должны явно загрузить его.

Обсуждение

Веб-страницы могут содержать несколько HTML-таблиц. Вызвав `read_html(url)`, а затем передав ее в `html_table`, вы прочитаете все таблицы на странице, и они вернутся в список. Это может быть полезно для изучения страницы, но раздражает, если вам нужна только одна конкретная таблица. В этом случае используйте `extract2(n)`, чтобы выбрать *n*-ю таблицу.
Например, здесь мы извлекаем все таблицы из статьи на сайте Википедии:

```
library(rvest)

all_tables <-
  read_html("https://en.wikipedia.org/wiki/Aviation_accidents_and_incidents") %>%
  html_table(fill = TRUE, header = TRUE)
```

Функция `read_html` помещает все таблицы из документа HTML в список вывода. Чтобы вытащить одну таблицу из этого списка, можно использовать функцию `extract2` из пакета `magrittr`:

```
out_table <-
  all_tables %>%
  magrittr::extract2(2)

head(out_table)
#>   Year  Deaths[53] # of incidents[54]
#> 1 2018  1,040               113[55]
#> 2 2017  399                 101
```

```
#> 3  2016   629                  102
#> 4  2015   898                  123
#> 5  2014   1,328                122
#> 6  2013   459                  138
```

Два общих параметра для функции `html_table` – это `fill=TRUE`, который заполняет пропущенные значения с помощью `NA`, и `header=TRUE`, который указывает, что первая строка содержит имена заголовков.

В следующем примере мы загружаем все таблицы со страницы Википедии под названием «Население Земли»:

```
url <- 'http://en.wikipedia.org/wiki/World_population'
tbls <- read_html(url) %>%
  html_table(fill = TRUE, header = TRUE)
```

Оказывается, эта страница содержит 23 таблицы (или то, что, по мнению `html_table`, может быть таблицами):

```
length(tbls)
#> [1] 23
```

В этом примере нас волнует только шестая таблица (в которой перечислены самые большие группы населения по странам), поэтому мы можем либо получить доступ к этому элементу с помощью квадратных скобок – `tbls[[6]]`, либо передать его в функцию `extract2` из пакета `magrittr`:

```
library(magrittr)
tbl <- tbls %>%
   extract2(6)

head(tbl, 2)
#>    Rank   Country / Territory  Population     Date % of world population
#> 1    1      China[note 4]    1,397,280,000  May 11, 2019       18.1%
#> 2    2      India            1,347,050,000  May 11, 2019       17.5%
#>    Source
#> 1  [84]
#> 2  [85]
```

Функция `extract2` является совместимой с командами организации конвейера версией синтаксиса R `[[i]]`: она извлекает из списка один элемент. Функция `extract` аналогична `[i]`, который возвращает элемент `i` из исходного списка в список длиной 1.

В этой таблице столбцы 2 и 3 содержат название страны и данные о численности населения соответственно:

```
tbl[, c(2, 3)]
#>      Country / Territory    Population
#> 1    China[note 4]        1,397,280,000
#> 2    India                1,347,050,000
#> 3    United States          329,181,000
#> 4    Indonesia              265,015,300
#> 5    Pakistan               212,742,631
#> 6    Brazil                 209,889,000
#> 7    Nigeria                188,500,000
#> 8    Bangladesh             166,532,000
#> 9    Russia[note 5]         146,877,088
#> 10   Japan                  126,440,000
```

Мы сразу же видим проблемы с данными: к названиям стран Китай и Россия добавлены `[note 4]` и `[note 5]`. На сайте Википедии это были ссылки на сноски, но теперь это всего лишь фрагменты нежелательного текста. Мало того, числа, обозначающие количество населения, содержат запятые, поэтому нельзя просто так преобразовать их в необработанные числа. Все эти проблемы можно решить обработкой строк, с каждой проблемой к процессу добавляется по крайней мере еще один шаг.

Это демонстрирует основное препятствие для чтения таблиц HTML. HTML был разработан для представления информации людям, а не компьютерам. Когда вы «извлекаете» информацию из HTML-страницы, то получаете вещи, которые полезны для людей, но раздражают компьютеры. Если у вас есть выбор, выберите вместо этого ориентированное на компьютер представление данных, такие форматы, как XML, JSON или CSV.

Функции `read_html(url)` и `html_table` являются частью большого и сложного (по необходимости) пакета `rvest`. Каждый раз, когда вы извлекаете данные с сайта, предназначенного для людей, а не для компьютеров, будьте готовы к тому, что вам придется выполнять постобработку, чтобы вычистить мусорные фрагменты.

См. также

См. рецепт 3.10 для скачивания и установки таких пакетов, как `rvest`.

4.15. Чтение файлов со сложной структурой

Задача

Вы читаете данные из файла, который имеет сложную или неправильную структуру.

Решение

Используйте функцию `readLines` для чтения отдельных строк; затем обработайте их как строки для извлечения элементов данных.

В качестве альтернативы используйте функцию `scan`, чтобы прочитать отдельные токены и использовать аргумент `what` для описания потока токенов в своем файле. Эта функция может преобразовывать токены в данные, а затем собирать данные в записи.

Обсуждение

Было бы легко и просто, если бы все наши файлы данных были организованы в аккуратные таблицы с четко разграниченными данными. Мы могли бы читать эти файлы, используя одну из функций из пакета `readr`, и продолжать радоваться жизни.

К сожалению, мы не живем в стране розовых единорогов.

В конечном итоге вы столкнетесь с необычным форматом файла, и ваша задача – прочитать содержимое этого файла в R.

Функции `read.table` и `read.csv` ориентированы на файлы и вряд ли помогут. Однако здесь полезны функции `readLines` и `scan`, поскольку они позволяют обрабатывать отдельные строки и даже токены файла.

Функция `readLines` довольно проста. Она читает строки из файла и возвращает их в виде списка символьных строк:

```
lines <- readLines("input.txt")
```

Можно ограничить количество строк, используя параметр `n`, который дает максимальное количество строк для чтения:

```
lines <- readLines("input.txt", n = 10) # Читаем 10 строк и останавливаемся.
```

Функция `scan` намного богаче. Она читает один токен за раз и обрабатывает его в соответствии с вашими инструкциями. Первый аргумент – это либо имя файла, либо соединение. Второй аргумент называется `what`. Он описывает токены, которые функция `scan` должна ожидать во входном файле. Описание запутанное, но довольно логичное:

`what=numeric(0)`

Интерпретирует следующий токен как число.

`what=integer(0)`

Интерпретирует следующий токен как целое число.

`what=complex(0)`

Интерпретирует следующий токен как комплексное число.

`what=character(0)`

Интерпретирует следующий токен как строку символов.

`what=logical(0)`

Интерпретирует следующий токен как логическое значение.

Функция `scan` будет применять данный шаблон несколько раз, пока не будут прочитаны все данные.

Предположим, ваш файл – это просто последовательность чисел, например:

```
2355.09 2246.73 1738.74 1841.01 2027.85
```

Используйте `what=numeric(0)`, чтобы сказать: «Мой файл – это последовательность токенов, каждый из которых является числом»:

```
singles <- scan("./data/singles.txt", what = numeric(0))
singles
#> [1] 2355.09 2246.73 1738.74 1841.01 2027.85
```

Ключевой особенностью функции `scan` является то, что `what` может быть списком, содержащим несколько типов токенов. Данная функция предполагает, что ваш файл – это повторяющаяся последовательность этих типов. Предположим, что ваш файл содержит триплеты данных, например:

```
15-Oct-87 2439.78 2345.63 16-Oct-87 2396.21 2207.73
19-Oct-87 2164.16 1677.55 20-Oct-87 2067.47 1616.21
21-Oct-87 2081.07 1951.76
```

Используйте список, чтобы сообщить функции `scan`, что она должна ожидать повторяющуюся последовательность из трех токенов:

```
triples <-
 scan("./data/triples.txt",
   what = list(character(0), numeric(0), numeric(0)))
triples
#> [[1]]
#> [1] "15-Oct-87" "16-Oct-87" "19-Oct-87" "20-Oct-87" "21-Oct-87"
#>
#> [[2]]
#> [1] 2439.78 2396.21 2164.16 2067.47 2081.07
#>
#> [[3]]
#> [1] 2345.63 2207.73 1677.55 1616.21 1951.76
```

Дайте имена элементам списка, а `scan` назначит эти имена данным:

```
triples <-
   scan("./data/triples.txt",
        what = list(
          date = character(0),
          high = numeric(0),
          low = numeric(0)
        ))
triples
#> $date
#> [1] "15-Oct-87" "16-Oct-87" "19-Oct-87" "20-Oct-87" "21-Oct-87"
#>
#> $high
#> [1] 2439.78 2396.21 2164.16 2067.47 2081.07
#>
#> $low
#> [1] 2345.63 2207.73 1677.55 1616.21 1951.76
```

Это легко можно превратить в таблицу данных с помощью команды `data.frame`:

```
df_triples <- data.frame(triples)
df_triples
#>      date    high    low
#> 1 15-Oct-87 2439.78 2345.63
#> 2 16-Oct-87 2396.21 2207.73
#> 3 19-Oct-87 2164.16 1677.55
#> 4 20-Oct-87 2067.47 1616.21
#> 5 21-Oct-87 2081.07 1951.76
```

У функции `scan` есть много «наворотов», но особенно полезны следующие:

`n=number`

Остановка после прочтения такого количества токенов (по умолчанию: остановка в конце файла).

`nlines=number`

Остановка после прочтения такого количества строк ввода (по умолчанию: остановка в конце файла).

`skip=number`

Количество строк ввода, пропущенных перед чтением данных.

`na.strings=list`

Список строк, которые следует интерпретировать как `NA`.

Пример

Давайте воспользуемся этим рецептом для чтения набора данных из StatLib, хранилища статистических данных и программного обеспечения, поддерживаемого Университетом Карнеги-Меллона. Джефф Уитмер предоставил набор данных под названием `wseries`, который показывает шаблон побед и поражений во всех Мировых сериях с 1903 года. Набор данных хранится в файле ASCII с 35 строками комментариев, за которыми следуют 23 строки данных. Сами данные выглядят так:

```
1903  LWLlwwwW   1927  wwWW    1950  wwWW      1973  WLwllWW
1905  wLwWW      1928  WWww    1951  LWlwwW    1974  wlWWW
1906  wLwLwW     1929  wwLWW   1952  lwLWLww   1975  lwWLWlw
1907  WWww       1930  WWllwW  1953  WWllwW    1976  WWww
1908  wWLww      1931  LWwlwLW 1954  WWww      1977  WLwwlW
.
. (etc.)
.
```

Данные кодируются следующим образом: `L` = поражение дома, `l` = поражение в гостях, `W` = победа дома, `w` = победа в гостях. Данные отображаются в порядке столбцов, а не строк, что немного усложняет нам жизнь.

Вот код для чтения необработанных данных:

```
# Читаем набор данных wseries:
# - Пропускаем первые 35 строк.
# - Затем читаем 23 строки данных.
# - Данные идут парами: год и шаблон (символьная строка).
#
world.series <- scan(
"http://lib.stat.cmu.edu/datasets/wseries",
skip = 35,
nlines = 23,
what = list(year = integer(0),
 pattern = character(0)),
)
```

Функция `scan` возвращает список, поэтому мы получаем список с двумя элементами: `year` и `pattern`. Она читает слева направо, но набор данных организован по столбцам, поэтому годы отображаются в странном порядке:

```
world.series$year
#> [1]  1903 1927 1950 1973 1905 1928 1951 1974 1906 1929 1952 1975 1907 1930
#> [15] 1953 1976 1908 1931 1954 1977 1909 1932 1955 1978 1910 1933 1956 1979
#> [29] 1911 1934 1957 1980 1912 1935 1958 1981 1913 1936 1959 1982 1914 1937
#> [43] 1960 1983 1915 1938 1961 1984 1916 1939 1962 1985 1917 1940 1963 1986
#> [57] 1918 1941 1964 1987 1919 1942 1965 1988 1920 1943 1966 1989 1921 1944
#> [71] 1967 1990 1922 1945 1968 1991 1923 1946 1969 1992 1924 1947 1970 1993
#> [85] 1925 1948 1971 1926 1949 1972
```

Это можно исправить, отсортировав элементы списка по годам:

```
perm <- order(world.series$year)
world.series <- list(year = world.series$year[perm],
                     pattern = world.series$pattern[perm])
```

Теперь данные отображаются в хронологическом порядке:

```
world.series$year
#>  [1] 1903 1905 1906 1907 1908 1909 1910 1911 1912 1913 1914 1915 1916 1917
#> [15] 1918 1919 1920 1921 1922 1923 1924 1925 1926 1927 1928 1929 1930 1931
#> [29] 1932 1933 1934 1935 1936 1937 1938 1939 1940 1941 1942 1943 1944 1945
#> [43] 1946 1947 1948 1949 1950 1951 1952 1953 1954 1955 1956 1957 1958 1959
#> [57] 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973
#> [71] 1974 1975 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987
#> [85] 1988 1989 1990 1991 1992 1993

world.series$pattern
#>  [1] "LWLlwwwW"  "wLwWW"    "wLwLwW"   "WWww"      "wWLww"    "WLwlWlw"
#>  [7] "WWwlw"     "lWwWlW"   "wLwWlLW"  "wLwWw"     "wwWW"     "lwWWw"
#> [13] "WWlwW"     WWllWw"    "wlwWLW"   "WWlwwLLw"  "wllWWWW"  "LlWwLwWw"
#> [19] "WWWW"      "LwLwWw"   "LWlwlWW"  "LWllwWW"   "lwWLLww"  "wwWW"
#> [25] "WWww"      "wwLWW"    "WWllwW"   "LWwlwLW"   "WWww"     "WWlww"
#> [31] "wlWLLww"   "LWwwlW"   "lwWWLw"   "WWwlw"     "wwWW"     "WWww"
#> [37] "LWlwlWW"   "WLwww"    "LWwww"    "WLWww"     "LWlwwW"   "LWLwwlw"
#> [43] "LWlwlww"   "WWllwLW"  "lwWWLw"   "WLwww"     "wwWW"     "LWlwwW"
#> [49] "lwLWLww"   "WWllwW"   "WWww"     "llWWWlw"   "llWWWlw"  "lwLWWlw"
#> [55] "llWLWww"   "lwWWLw"   "WLlwwLW"  "WLwww"     "wlWLWlw"  "wwWW"
#> [61] "WLlwwLW"   "llWWWlw"  "wwWW"     "wlWWLlw"   "lwLLWww"  "lwWWW"
#> [67] "wwWLW"     "llWWWlw"  "wwLWLlw"  "WLwllWW"   "wlWWW"    "lwWLWlw"
#> [73] "WWww"      "WLwwlW"   "llWWWw"   "lwLLWww"   "WWllwW"   "llWWWw"
#> [79] "LWwllWW"   "LWwww"    "wlWWW"    "LLwlwWW"   "LLwwlWW"  "WWlllWW"
#> [85] "WWlww"     "WWww"     "WWww"     "WWlllWW"   "lwWWLw"   "WLwwlW"
```

4.16. Чтение из баз данных MySQL

Задача

Вы хотите получить доступ к данным, хранящимся в базе данных MySQL.

Решение

Следуйте приведенным ниже шагам.

1. Установите пакет `RMySQL` на свой компьютер и добавьте пользователя и пароль.
2. Откройте соединение с базой данных, используя функцию `DBI::dbConnect`.
3. Применяйте функцию `dbGetQuery` для запуска команды `SELECT` и возврата наборов результатов.
4. Используйте функцию `dbDisconnect`, чтобы завершить соединение с базой данных, когда вы закончите.

Обсуждение

Данный рецепт требует, чтобы на вашем компьютере был установлен пакет `RMySQL`, который, в свою очередь, требует установки программы клиента MySQL. Если она еще не установлена и не настроена в вашей системе, обратитесь к документации MySQL или своему системному администратору.

Используйте функцию `dbConnect`, чтобы установить соединение с базой данных MySQL. Она возвращает объект подключения, который используется в последующих вызовах функций `RMySQL`:

```
library(RMySQL)

con <- dbConnect(
    drv = RMySQL::MySQL(),
    dbname = "your_db_name",
    host = "your.host.com",
    username = "userid",
    password = "pwd"
 )
```

Параметры `username`, `password` и `host` – это те же параметры, что используются для доступа к MySQL через клиентскую программу `mysql`. Приведенный здесь пример показывает, что они явно прописаны в вызове `dbConnect`, но на самом деле такая практика не рекомендуется. Ваш пароль помещается в простой текстовый документ, создавая проблему для безопасности. Это также причиняет серьезную головную боль всякий раз, когда ваш пароль или хост меняется, требуя, чтобы вы выискивали жестко закодированные значения. Вместо этого мы настоятельно рекомендуем использовать механизм безопасности MySQL. Восьмая версия MySQL представляет еще более продвинутые параметры безопасности, но в настоящее время они не встроены в клиента `RMySQL`. Итак, мы рекомендуем вам использовать собственные пароли MySQL, установив для `default-authentication-plugin` значение `mysql_native_password` в своем файле конфигурации MySQL. В Unix это *$HOME/.my.cnf*, а в Windows – *C:\my.cnf*. Мы используем `loose-local-infile = 1`, чтобы гарантировать, что у нас есть права на запись в базу данных. Убедитесь, что файл не будет прочитан никем, кроме вас. Файл разделен на разделы с маркерами `[mysqld]` и `[client]`. Поместите параметры подключения в раздел `[client]`, чтобы в вашем конфигурационном файле содержалось нечто вроде этого:

```
[mysqld]
default-authentication-plugin=mysql_native_password
loose-local-infile=1

[client]
loose-local-infile=1
user="jdl"
password="password"
host=127.0.0.1
port=3306
```

Как только параметры в файле конфигурации будут определены, вам больше не нужно указывать их в вызове `dbConnect`, который тогда становится намного проще:

```
con <- dbConnect(
drv = RMySQL::MySQL(),
dbname = "your_db_name")
```

Используйте функцию `dbGetQuery` для отправки вашего SQL в базу данных и чтения наборов результатов. Для этого требуется открытое соединение с базой данных:

```
sql <- "SELECT * from SurveyResults WHERE City = 'Chicago'"
rows <- dbGetQuery(con, sql)
```

Вы не ограничены операторами SELECT. Подойдет любой SQL-запрос, который генерирует набор результатов. Можно использовать вызов CALL, например если ваш SQL инкапсулирован в хранимых процедурах и эти процедуры содержат встроенные операторы SELECT.

Использовать функцию dbGetQuery удобно, потому что она упаковывает набор результатов в таблицу данных и возвращает эту таблицу. Это идеальное представление набора результатов SQL. Набор результатов представляет собой табличную структуру данных строк и столбцов, а также таблицу данных. Столбцы набора имеют имена, заданные SQL-оператором SELECT, и R использует их для именования столбцов таблицы данных.

Вызовите функцию dbGetQuery несколько раз, чтобы выполнить несколько запросов. Когда вы закончите, закройте соединение с базой данных, используя функцию dbDisconnect:

```
dbDisconnect(con)
```

Вот полный сеанс, который читает и выводит три строки из базы данных курсов акций. Запрос выбирает цену акций IBM за последние три дня 2008 года. Предполагается, что параметры username, password, dbname и host определены в файле *my.cnf*:

```
con <- dbConnect(RMySQL::MySQL())
sql <- paste(
  "select * from DailyBar where Symbol = 'IBM'",
  "and Day between '2008-12-29' and '2008-12-31'"
)
rows <- dbGetQuery(con, sql)

dbDisconnect(con)
print(rows)

##    Symbol   Day        Next         OpenPx   HighPx   LowPx    ClosePx    AdjClosePx
## 1    IBM 2008-12-29   2008-12-30   81.72    81.72    79.68    81.25      81.25
## 2    IBM 2008-12-30   2008-12-31   81.83    83.64    81.52    83.55      83.55
## 3    IBM 2008-12-31   2009-01-02   83.50    85.00    83.50    84.16      84.16
##    HistClosePx  Volume   OpenInt
## 1      81.25 6062600    NA
## 2      83.55 5774400    NA
## 3      84.16 6667700    NA
```

См. также

См. рецепт 3.10 и документацию по RMySQL, которая содержит более подробную информацию о настройке и использовании этого пакета.

См. рецепт 4.17, чтобы узнать, как получить данные из базы данных SQL без написания SQL-запросов.

R может выполнять чтение из других реляционных СУБД, включая Oracle, Sybase, PostgreSQL и SQLite. Для получения дополнительной информации *см.* руководство по импорту/экспорту данных R, которое поставляется с базовым дистрибутивом (рецепт 1.7) и также доступно по адресу https://cran.r-project.org/doc/manuals/R-data.pdf.

4.17. Доступ к базе данных с помощью dbplyr

Задача

Вам нужно получить доступ к базе данных, но вы не хотите писать код SQL, для манипулирования данными и возвращения результата в R.

Решение

В дополнение к набору команд манипулирования данными пакет dplyr из коллекции tidyverse может, в связке с пакетом `dbplyr`, превратить команды `dplyr` в команды SQL.

Давайте настроим в качестве примера базу данных с использованием `RSQLite`. Затем мы подключимся к ней и используем `dplyr` и серверную часть `dbplyr` для извлечения данных.

Сначала мы настроим таблицу, загрузив данные примера `msleep` в базу данных SQLite в памяти:

```
con <- DBI::dbConnect(RSQLite::SQLite(), ":memory:")
sleep_db <- copy_to(con, msleep, "sleep")
```

Теперь, когда у нас есть таблица в нашей базе данных, мы можем создать ссылку на нее из R:

```
sleep_table <- tbl(con, "sleep")
```

Объект `sleep_table` представляет собой тип указателя или псевдонима таблицы в базе данных. Тем не менее `dplyr` будет обрабатывать его как обычную таблицу данных, поэтому вы можете работать с ним, используя `dplyr` и другие команды R. Давайте выберем всех животных из данных, которые «спят» менее трех часов:

```
little_sleep <- sleep_table %>%
 select(name, genus, order, sleep_total) %>%
 filter(sleep_total < 3)
```

Серверная часть `dbplyr` не получает данные, когда мы выполняем предыдущие команды. Но она создает запрос и готовится. Чтобы увидеть запрос, созданный `dplyr`, можно использовать функцию `show_query`:

```
show_query(little_sleep)
#> <SQL>
#> SELECT *
#> FROM (SELECT 'name', 'genus', 'order', 'sleep_total'
#> FROM 'sleep')
#> WHERE ('sleep_total' < 3.0)
```

Чтобы вернуть данные обратно в локальный компьютер, используйте функцию `collect`:

```
local_little_sleep <- collect(little_sleep)
local_little_sleep
#> # Tibble: 3 x 4
#>    name    genus              order           sleep_total
#>    <chr>   <chr>              <chr>               <dbl>
#> 1 Horse   Equus              Perissodactyla        2.9
#> 2 Giraffe Giraffa            Artiodactyla          1.9
#> 3 Pilot whale Globicephalus  Cetacea              2.7
```

Обсуждение

Когда вы используете `dplyr` для доступа к базам данных SQL путем написания только команд `dplyr`, вы можете работать более продуктивно, не переключаясь с одного языка на другой и обратно. Альтернатива состоит в том, чтобы большие куски SQL-кода сохранялись в виде текстовых строк посреди R-сценария или чтобы SQL-код находился в отдельных файлах, которые читает R.

Позволяя `dplyr` явно создавать SQL-команды в фоновом режиме, вы освобождаетесь от необходимости поддерживать отдельный SQL-код для извлечения данных.

Пакет `dbplyr` использует `DBI` для подключения к вашей базе данных, поэтому вам понадобится бэкенд-пакет `DBI` для любой базы данных, к которой вы хотите получить доступ.

Вот некоторые часто используемые пакеты:

`odbc`

: Использует интерфейс Open Database Connectivity (ODBC) для подключения ко множеству различных баз данных. Обычно это наиболее подходящий вариант при подключении к Microsoft SQL Server. ODBC прост на компьютерах с Windows, но может потребовать значительных усилий для работы в Linux или macOS.

`RPostgreSQL`

: Для подключения к Postgres и Redshift.

`RMySQL`

: Для MySQL и MariaDB.

`RSQLite`

: Для подключения к базам данных SQLite на диске или в памяти.

`bigrquery`

: Для подключения к BigQuery от компании Google.

Каждый бэкенд-пакет `DBI`, о котором здесь идет речь, доступен на CRAN. Его можно установить с помощью стандартной команды `install.packages('имяпакета')`.

См. также

Для получения дополнительной информации о соединении баз данных с R и RStudio *см.* https://db.rstudio.com/.

Для получения более подробной информации о трансляции SQL-запросов в `dbplyr` *см.* сопроводительную документацию `sql-translation` в `vignette("sql-translation")` или на странице http://bit.ly/2wVCOKe.

4.18. Сохранение и транспортировка объектов

Задача

Вы хотите сохранить один или несколько объектов R в файле для последующего использования или скопировать объект R с одного компьютера на другой.

Решение

Запишите объекты в файл, используя функцию save:

```
save(tbl, t, file = "myData.RData")
```

Прочтите их обратно, используя функцию load на вашем компьютере либо на любой платформе, поддерживающей R:

```
load("myData.RData")
```

Функция save записывает двоичные данные. Чтобы выполнить сохранение в формате ASCII, используйте функции dput или dump:

```
dput(tbl, file = "myData.txt")
dump("tbl", file = "myData.txt")   # Обратите внимание на кавычки вокруг имени переменной.
```

Обсуждение

Предположим, что вы оказались один на один с большим сложным объектом данных, который хотите загрузить в другие рабочие пространства, или вы хотите перемещать объекты R между операционными средами Linux и Windows. Функции load и save позволяют вам делать все это: функция save сохранит объект в файле, который можно переносить с одного компьютера на другой, а функция load может читать эти файлы.

Когда вы запускаете функцию load, она не возвращает ваши данные как таковые; скорее, она создает переменные в вашем рабочем пространстве, загружает ваши данные в эти переменные, а затем возвращает имена переменных (в виде вектора). При первом запуске load у вас может возникнуть соблазн сделать это:

```
myData <- load("myData.RData") # Внимание! Возможно, получится не то, что вы думаете.
```

Давайте посмотрим, что такое myData:

```
myData
#> [1] "tbl" "t"
str(myData)
#> chr [1:2] "tbl" "t"
```

Это может вас озадачить, потому что у myData вообще не будет ваших данных, и может вызывать недоумение и разочарование, когда вы впервые столкнетесь с этим.

Есть еще несколько вещей, которые нужно иметь в виду. Во-первых, функция save выполняет запись в двоичном формате, чтобы сохранить размер файла небольшим. Иногда вам нужен формат ASCII. Когда вы отправляете вопрос в список рассылки или, например, на сайт Stack Overflow, включая дамп данных ASCII, то позволяете другим воссоздать свою проблему. В таких случаях используйте функции dput или dump, которые записывают представление ASCII.

Вы также должны быть осторожны при сохранении и загрузке объектов, созданных конкретным пакетом R. Когда вы загружаете объекты, R также автоматически не загружает требуемые пакеты, поэтому он не «узнает» объект, если вы предварительно не загрузили нужный пакет. Например, предположим, что у нас есть объект с именем z, созданный пакетом zoo, и мы сохраняем объект в файле с именем *z.RData*. Приведенная ниже последовательность функций вызовет путаницу:

```
load("./data/z.RData") # Создаем и заполняем переменную z.
plot(z) # Не строит график, как ожидалось: пакет zoo не загружен.
```

На рис. 4-3 показан получившийся в результате график. Это просто точки.

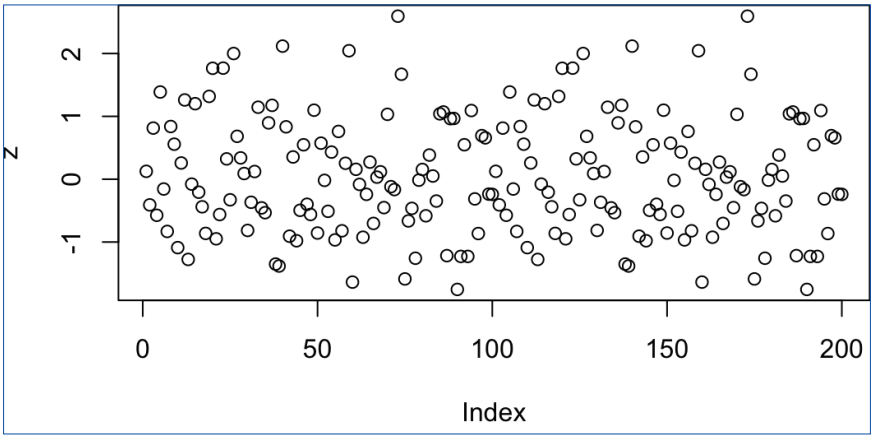


Рис. 4-3. Так выглядит график, когда zoo не загружен

Мы должны были загрузить пакет zoo, прежде чем выводить или наносить на график какие-либо объекты zoo, например так:

```
library(zoo)              # Загружаем пакет zoo в память.
load("./data/z.RData")    # Создаем и заполняем переменную z.
plot(z)                   # Ааа. Теперь все работает правильно.
```

Вы видите получившийся график на рис. 4-4.

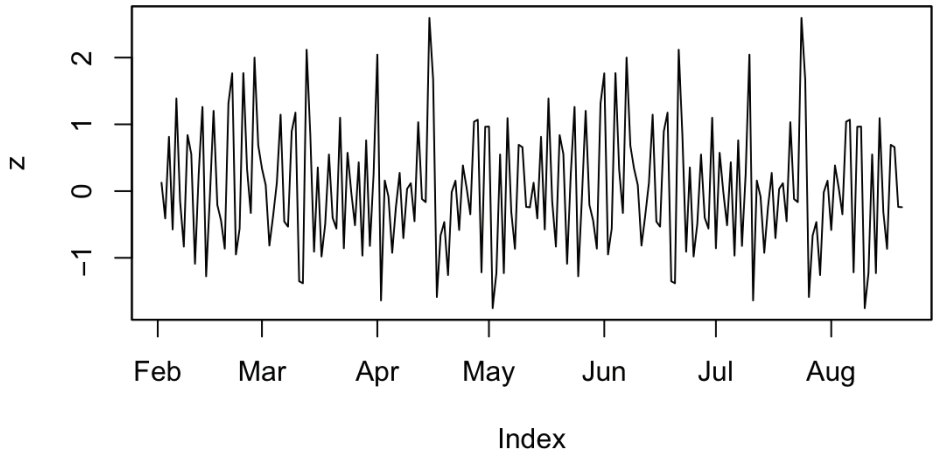


Рис. 4-4. Построение графика с помощью zoo

См. также

Если вы просто сохраняете и загружаете одну таблицу данных или другой объект R, вам следует обратить внимание на функции `write_rds` и `read_rds`. Они не имеют «побочных эффектов», в отличие от функции `load`.

Глава 5

Структуры данных

Вы можете продвинуться довольно далеко в R, используя исключительно векторы. Вот о чем глава 2. Эта же глава выходит за пределы векторов, и в ней приводятся рецепты матриц, списков, факторов, таблиц данных и tibble (которые представляют собой специальный тип таблицы данных). Если у вас есть предвзятые мнения относительно структур данных, то рекомендуем отложить их в сторону. R работает со структурами данных иначе, по сравнению со множеством других языков. Прежде чем мы перейдем к рецептам, приведенным в этой главе, мы кратко рассмотрим различные структуры данных в R.

Если вы хотите изучить технические аспекты структур данных R, мы предлагаем прочитать книгу *R in a Nutshell* (http://shop.oreilly.com/product/9780596801717.do) и посетить страницу https://cran.r-project.org/doc/manuals/R-lang.pdf. Примечания, данные здесь, более неформальные. Это то, что мы хотели бы знать, когда приступили к использованию R.

Векторы

Вот некоторые ключевые свойства векторов:

Векторы однородны
: Все элементы вектора должны иметь одинаковый тип или, используя терминологию R, один и тот же *режим* (mode).

Векторы можно индексировать по позициям.
: Таким образом, `v[2]` относится ко второму элементу `v`.

Векторы можно индексировать по нескольким позициям, возвращая подвектор.
: Таким образом, `v[c(2,3)]` является подвектором `v`, который состоит из второго и третьего элементов.

Векторные элементы могут иметь имена.
: У векторов есть свойство `names` той же длины, что и сам вектор, которое присваивает имена элементам:

```
v <- c(10, 20, 30)
names(v) <- c("Moe", "Larry", "Curly")
print(v)
#> Moe Larry Curly
#> 10  20    30
```

Если у векторных элементов есть имена, тогда можно выбирать их по имени.

Продолжая предыдущий пример:

```
v[["Larry"]]
#> [1] 20
```

Списки

Вот некоторые ключевые свойства списков:

Списки неоднородны.

Списки могут содержать элементы различных типов – используя терминологию R, можно сказать, что элементы списка могут иметь разные режимы. Списки могут даже содержать другие структурированные объекты, такие как списки и таблицы данных; это позволяет создавать рекурсивные структуры данных.

Списки можно индексировать по позициям.

Таким образом, `lst[[2]]` относится ко второму элементу `lst`. Обратите внимание на двойные квадратные скобки. Двойные скобки означают, что R вернет сам элемент без преобразования типа.

Списки позволяют извлекать подсписки.

Итак, `lst[c(2,3)]` – это подсписок `lst`, состоящий из второго и третьего элементов. Обратите внимание на одинарные квадратные скобки. Одинарные скобки означают, что R вернет элементы в списке. Если вы извлечете один элемент с одинарными скобками, например `lst[2]`, R вернет список длины 1 с первым элементом, который является желаемым элементом.

Элементы списка могут иметь имена.

И `lst[["Moe"]]`, и `lst$Moe` ссылаются на элемент с именем «Moe».

Поскольку списки неоднородны и их элементы могут быть получены по имени, список похож на словарь, хеш или таблицу поиска в других языках программирования (обсуждается в рецепте 5.9).

Удивительно (и это круто) то, что в R, в отличие от большинства других языков программирования, списки также можно индексировать по их месту.

Режим: физический тип

В R у каждого объекта есть режим (mode), который указывает, как он хранится в памяти: как число, как строка символов, как список указателей на другие объекты, функция и т. д. (*см.* табл. 5-1).

Таблица 5-1. R-объект: режим отображения

Объект	Пример	Режим
Число	3.1415	Числовой
Вектор чисел	c(2.7.182, 3.1415)	Числовой
Строка символов	«Moe»	Символьный
Вектор символьных строк	c(«Moe», «Larry», «Curly»)	Символьный
Фактор	factor(c(«NY», «CA», «IL»))	Числовой

Окончание табл. 5.1

Объект	Пример	Режим
Список	list(«Moe», «Larry», «Curly»)	Список
Таблица данных	data.frame(x=1:3, y=c(«NY», «CA», «IL»))	Список
Функция	Print	Функция

Функция `mode` дает нам эту информацию:

```
mode(3.1415)                      # Режим числа.
#> [1] "numeric"
mode(c(2.7182, 3.1415))           # Режим вектора чисел.
#> [1] "numeric"
mode("Moe")                       # Режим символьной строки.
#> [1] "character"
mode(list("Moe", "Larry", "Curly")) # Режим списка.
#> [1] "list
```

Решающее различие между векторами и списками можно подытожить следующим образом:

- в векторе все элементы должны иметь одинаковый режим;
- в списке элементы могут иметь разные режимы.

Класс: абстрактный тип

В R у каждого объекта также есть класс, который определяет его абстрактный тип. Эта терминология заимствована из объектно-ориентированного программирования. Одно число может обозначать множество разных вещей: например, расстояние, момент времени или вес. Все эти объекты имеют режим `"numeric"`, потому что они хранятся в виде числа, но у них могут быть разные классы для указания интерпретации.

Например, объект `Date` состоит из одного числа:

```
d <- as.Date("2010-03-15")
mode(d)
#> [1] "numeric"
length(d)
#> [1] 1
```

Но у него есть класс `Date`, говорящий нам, как интерпретировать это число, а именно как число дней, начиная с 1 января 1970 года:

```
class(d)
#> [1] "Date"
```

R использует класс объекта, чтобы решить, как обрабатывать этот объект. Например, у универсальной функции `print` есть специальные версии (*методы*) для вывода объектов в соответствии с их классом: `data.frame`, `Date`, `lm` и т. д. Когда вы выводите объект, R вызывает соответствующую функцию `print` в соответствии с классом объекта.

Скаляры

Что касается скаляров, то тут есть один причудливый момент – это их связь с векторами. В некоторых программах скаляры и векторы – это две разные вещи. В R это одно и то же: скаляр – это просто вектор, который содержит ровно один

элемент. В этой книге мы часто используем термин «скаляр», но это всего лишь сокращение словосочетания «вектор с одним элементом».

Давайте рассмотрим встроенную константу `pi`. Это скаляр:

```
pi
#> [1] 3.14
```

Поскольку скаляр является одноэлементным вектором, вы можете использовать векторные функции:

```
length(pi)
#> [1] 1
```

Можно проиндексировать ее. Конечно же, первый (и единственный) элемент – это `п`:

```
pi[1]
#> [1] 3.14
```

Если вы запросите второй элемент, то его нет:

```
pi[2]
#> [1] NA
```

Матрицы

В R матрица – это просто вектор, у которого есть размеры. Поначалу это может показаться странным, но вы можете преобразовать вектор в матрицу, просто задав ему размеры.

У вектора есть атрибут с именем `dim`, который изначально равен `NULL`, как показано здесь:

```
A <- 1:6
dim(A)
#> NULL
print(A)
#> [1] 1 2 3 4 5 6
```

Мы даем вектору размеры, когда устанавливаем его атрибут `dim`. Посмотрите, что произойдет, когда мы установим наши векторные размеры на 2×3 и выведем это:

```
dim(A) <- c(2, 3)
print(A)
#>      [,1] [,2] [,3]
#> [1,]    1    3    5
#> [2,]    2    4    6
```

Вуаля! Вектор был преобразован в матрицу 2×3.

Матрицу также можно создать из списка. Как и у вектора, у списка есть атрибут `dim`, который изначально равен `NULL`:

```
B <- list(1, 2, 3, 4, 5, 6)
dim(B)
#> NULL
```

Если мы установим атрибут `dim`, он придаст списку форму:

```
dim(B) <- c(2, 3)
print(B)
```

```
#>      [,1]  [,2]  [,3]
#> [1,] 1     3     5
#> [2,] 2     4     6
```

Вуаля! Мы превратили этот список в матрицу 2×3.

Массивы

Обсуждение матриц можно распространить на трехмерные или даже *n*-мерные структуры: просто присвойте базовому вектору (или списку) дополнительные размеры. В следующем примере мы создаем трехмерный массив с размерами 2×3×2:

```
D <- 1:12
dim(D) <- c(2, 3, 2)
print(D)
#> , , 1
#>
#>      [,1]  [,2]  [,3]
#> [1,]  1     3     5
#> [2,]  2     4     6
#>
#> , , 2
#>
#>      [,1]  [,2]  [,3]
#> [1,]  7     9    11
#> [2,]  8    10    12
```

Обратите внимание, что R выводит один «фрагмент» структуры за раз, поскольку невозможно вывести трехмерную структуру в двухмерной среде.

Мы можем превратить список в матрицу, просто присвоив списку атрибут `dim`, и это кажется очень странным. Но подождите: это еще не все.

Напомним, что список может быть неоднородным (смешанные режимы). Мы можем начать с неоднородного списка, дать ему размеры и, таким образом, создать неоднородную матрицу. В этом фрагменте кода мы создаем матрицу, которая представляет собой смесь числовых и символьных данных:

```
C <- list(1, 2, 3, "X", "Y", "Z")
dim(C) <- c(2, 3)
print(C)
#>      [,1]  [,2]  [,3]
#> [1,]  1     3     "Y"
#> [2,]  2     "X"   "Z"
```

Для нас это странно, потому что обычно мы предполагаем, что матрица является чисто числовой, а не смешанной. В R нет таких сильных ограничений.

Возможность наличия неоднородной матрицы может показаться мощной и удивительно захватывающей. Тем не менее это создает проблемы, когда вы делаете обычные, повседневные вещи с матрицами. Например, что происходит, когда матрица C (из предыдущего примера) используется для умножения матриц? Что произойдет, если она преобразуется в таблицу данных? Ответ: *может быть по-разному*.

В этой книге мы обычно игнорируем патологический случай неоднородной матрицы. Мы предполагаем, что у вас простые однородные по типу матрицы. Некоторые рецепты с участием матриц могут работать странно (или не работать вообще), если

ваша матрица содержит смешанные данные. Преобразование такой матрицы в вектор или таблицу данных, например, может быть проблематичным (*см*. рецепт 5.29).

Факторы

Фактор похож на символьный вектор, но у него есть особые свойства. R отслеживает уникальные значения в векторе, и каждое уникальное значение представляется градацией, или *уровнем* (level), соответствующего фактора. R использует компактное представление для факторов, что делает их эффективными для хранения в таблицах данных. В других языках программирования фактор обычно представлен вектором значений перечислимого типа.

Есть два ключевых варианта использования факторов:

Категориальные переменные

Фактор может представлять категориальную переменную. Категориальные переменные используются в таблицах сопряженности, линейной регрессии, дисперсионном анализе (ANOVA), логистической регрессии и многих других областях.

Группировка

Это метод маркировки ваших элементов данных в соответствии с их группой. *См*. главу 6.

Таблицы данных

Таблица данных – это мощная и гибкая структура. Наиболее серьезные R-приложения включают в себя таблицы данных. Таблица данных предназначена для имитации набора данных, такого как тот, который вы можете встретить в SAS или SPSS, или таблица в базе данных SQL.

Таблица данных представляет собой табличную (прямоугольную) структуру данных. Это означает, что у него есть строки и столбцы. Однако он не реализуется матрицей. Скорее, таблица данных представляет собой список со следующими характеристиками:

- элементы списка – это векторы и/или факторы[1];
- эти векторы и факторы являются столбцами таблицы данных;
- у всех векторов и факторов должна быть одинаковая длина; другими словами, все столбцы должны иметь одинаковую высоту;
- столбцы одинаковой высоты придают прямоугольную форму таблице данных;
- у столбцов должны быть имена.

Поскольку таблица данных – это и список, и прямоугольная структура, R предоставляет две разные парадигмы для доступа к его содержимому:

- вы можете использовать операторы списка для извлечения столбцов из таблицы данных, например `df[i]`, `df[[i]]` или `df$name`;
- вы можете использовать матричные обозначения, например `df[i,j]`, `df[i,]` или `df[,j]`.

---

[1] Таблицу данных можно создать из смеси векторов, факторов и матриц. Столбцы матриц становятся столбцами в таблице данных. Количество строк в каждой матрице должно соответствовать длине векторов и факторов. Другими словами, все элементы таблицы данных должны иметь одинаковую высоту.

Ваше восприятие таблицы данных, вероятно, зависит от окружающих условий:

Для специалиста по статистике

Таблица данных – это таблица наблюдений. Каждый ряд содержит одно наблюдение. Каждое наблюдение должно содержать одинаковые переменные. Эти переменные называются столбцами, и вы можете обращаться к ним по имени. Вы также можете ссылаться на содержимое по номеру строки и номеру столбца, как и в случае с матрицей.

Для программиста SQL

Таблица данных – это таблица. Таблица полностью находится в памяти, но вы можете сохранить ее в плоский файл и восстановить позже. Вам не нужно объявлять типы столбцов, потому что R определяет это за вас.

Для пользователя Excel

Таблица данных подобна рабочему листу или, возможно, диапазону в рабочем листе. Однако он более ограничен в том отношении, что у каждого столбца есть тип.

Для пользователя SAS

Таблица данных подобна набору данных SAS, для которого все данные находятся в памяти. R может читать и записывать таблицу данных на диск, но она должна находиться в памяти, пока R обрабатывает ее.

Для программиста R

Таблица данных – это гибридная структура данных, частично матрица и частично список. Столбец может содержать числа, символьные строки или факторы, но не их сочетание. Вы можете индексировать таблицу данных так же, как индексируете матрицу. Таблица данных также является списком, где элементы – это столбцы, поэтому вы можете получить доступ к столбцам с помощью операторов списка.

Для специалиста по информатике

Таблица данных представляет собой прямоугольную структуру данных. Столбцы набираются, и каждый столбец должен содержать числовые значения, символьные строки или факторы. Столбцы должны иметь метки; у строк могут быть метки. Таблицу можно индексировать по позициям, имени столбца и/или имени строки. К ней также могут обращаться операторы списков, и в этом случае R рассматривает таблицу данных как список, элементами которого являются столбцы таблицы данных.

Для члена правления

Вы можете поместить имена и номера в таблицу данных. Таблица данных похожа на маленькую базу данных. Вашим сотрудникам понравится использовать таблицы данных.

Tibble

Tibble – это современное переосмысление таблицы данных, введенное Хэдли Уикхемом в пакете `tibble`, который является базовым пакетом в `tidyverse`. Боль-

шинство распространенных функций, которые вы бы использовали с таблицами данных, также подходят и для tibble. Тем не менее tibble обычно делают меньше по сравнению с таблицами данных и жалуются больше. Такое поведение, возможно, напомнит вам о вашем наименее любимом сотруднике; однако мы думаем, что tibble станут одной из ваших любимых структур данных. Делать меньше и жаловаться больше – это может быть свойством, а не ошибкой.

В отличие от таблиц данных, tibble:

- не дают вам номера строк по умолчанию;
- не дают вам странных, неожиданных имен столбцов;
- не приводят ваши данные в факторы (если вы явно не просите об этом);
- переиспользуют векторы длины 1, но не других длин.

В дополнение к базовой функциональности таблиц данных tibble:

- по умолчанию выводят только четыре верхние строки и небольшое количество метаданных;
- всегда возвращают tibble при операции извлечения подмножества;
- никогда не выполняют частичное сопоставление: если вам нужен столбец из tibble, вы должны запросить его, используя его полное имя;
- больше жалуются, выдавая больше предупреждений и излишне избыточных сообщений, чтобы убедиться, что вы понимаете, что делает программа.

Все эти дополнения разработаны для того, чтобы вас ожидало меньше сюрпризов и вы делали меньше ошибок.

5.1. Добавление данных в вектор

Задача

Вы хотите добавить дополнительные элементы данных в вектор.

Решение

Используйте конструктор вектора (c), чтобы создать вектор с дополнительными элементами данных:

```
v <- c(1, 2, 3)
newItems <- c(6, 7, 8)
c(v, newItems)
#> [1] 1 2 3 6 7 8
```

В случае с отдельным элементом вы также можете присвоить новый элемент следующему элементу вектора. R автоматически расширит этот вектор:

```
v <- c(1, 2, 3)
v[length(v) + 1] <- 42
v
#> [1] 1 2 3 42
```

Обсуждение

Если вы спросите нас о добавлении элемента данных в вектор, мы, вероятно, предложим вам не делать этого.

R работает лучше, когда вы думаете о целых векторах, а не об отдельных элементах данных. Вы неоднократно добавляете элементы в вектор? Если так, то вы, вероятно, работаете внутри цикла. Это нормально для маленьких векторов, но в случае с большими векторами ваша программа будет работать медленно. Управление памятью в R работает плохо, когда вы многократно расширяете вектор на один элемент. Попробуйте заменить этот цикл операциями на векторном уровне. Вы будете писать меньше кода, а R будет работать намного быстрее.

Тем не менее иногда нужно добавлять данные в векторы. Наши эксперименты показывают, что наиболее эффективный способ сделать это – создать новый вектор, используя векторный конструктор (c) для объединения старых и новых данных. Это подходит для добавления отдельных или нескольких элементов:

```
v <- c(1, 2, 3)
v <- c(v, 4) # Добавляем отдельное значение к v.
v
#> [1] 1 2 3 4

w <- c(5, 6, 7, 8)
v <- c(v, w) # Добавляем целый вектор к v.
v
#> [1] 1 2 3 4 5 6 7 8
```

Вы также можете добавить элемент, присвоив его позиции после конца вектора, как показано в решении. На самом деле R очень либерален в отношении расширения векторов. Вы можете назначить любой элемент, и R расширит вектор, чтобы удовлетворить ваш запрос:

```
v <- c(1, 2, 3)    # Создаем вектор с тремя элементами.
v[10] <- 10        # Присваивание 10-му элементу.
v                  # R автоматически расширяет вектор.
#> [1] 1 2 3 NA NA NA NA NA NA 10
```

Обратите внимание на то, что R не жаловался на индекс, находящийся за пределами. Он просто расширил вектор до необходимой длины, заполнив его значениями NA.

В R есть функция append, которая создает новый вектор путем добавления элементов к существующему вектору. Однако наши эксперименты показывают, что эта функция работает медленнее, чем конструктор вектора и присваивание элементов.

5.2. Вставка данных в вектор

Задача

Вы хотите вставить один или несколько элементов данных в вектор.

Решение

Несмотря на свое имя, функция append вставляет данные в вектор с помощью параметра after, который задает точку вставки для нового элемента или элементов:

```
append(vec, newvalues, after = n)
```

Обсуждение

Новые элементы будут вставлены в позиции, указанной параметром `after`. В этом примере мы вставляем 99 посередине последовательности:

```
append(1:10, 99, after = 5)
#> [1] 1 2 3 4 5 99 6 7 8 9 10
```

Специальное значение `after=0` означает вставку новых элементов в *начало* вектора:

```
append(1:10, 99, after = 0)
#> [1] 99 1 2 3 4 5 6 7 8 9 10
```

Комментарии для рецепта 5.1 применимы и здесь. Если вы вставляете отдельные элементы в вектор, возможно, вы работаете на уровне элементов, тогда как на векторном уровне будет проще кодировать и быстрее запускать.

5.3. Правило повторного использования

Задача

Вы хотите понять таинственное правило повторного использования, которое определяет, как R обрабатывает векторы неравной длины.

Обсуждение

Когда вы занимаетесь векторной арифметикой, R выполняет поэлементные операции. Это хорошо работает, когда оба вектора имеют одинаковую длину: R соединяет элементы векторов и применяет эту операцию к этим парам.

Но что происходит, когда векторы имеют разную длину?

В этом случае R вызывает правило повторного использования. Оно обрабатывает векторные элементы попарно, начиная с первых элементов обоих векторов. В определенный момент более короткий вектор заканчивается, в то время как более длинный вектор все еще содержит необработанные элементы. Затем R возвращается к началу более короткого вектора, «переиспользуя» его элементы, продолжая принимать элементы из более длинного вектора, пока не завершит операцию. Он будет переиспользовать элементы более короткого вектора так часто, как это необходимо, до тех пор, пока операция не будет завершена.

Полезно увидеть правило повторного использования на наглядном примере. Вот диаграмма двух векторов `1:6` и `1:3`:

```
1:6  1:3
---- ----
 1   1
 2   2
 3   3
 4
 5
 6
```

Очевидно, что вектор `1:6` длиннее, чем вектор `1:3`. Если мы попытаемся добавить векторы, используя `(1:6) + (1:3)`, получается, что у `1:3` слишком мало эле-

ментов. Тем не менее R переиспользует элементы `1:3`, соединяя два вектора таким образом и получая вектор из шести элементов:

```
 1:6    1:3   (1:6) + (1:3)
----- ----- --------------
 1      1           2
 2      2           4
 3      3           6
 4                  5
 5                  7
 6                  9
```

Вот что вы видите в консоли R:

```
(1:6) + (1:3)
#> [1] 2 4 6 5 7 9
```

Не только векторные операции вызывают правило повторного использования; функции тоже могут это делать. Функция `cbind` может создавать векторы столбцов, такие как приведенные ниже векторы столбцов `1:6` и `1:3`. Конечно, у двух колонок разная высота:

```
cbind(1:6)

cbind(1:3)
```

Если мы попытаемся связать эти векторы столбцов в матрицу из двух столбцов, длины не будут совпадать. Вектор `1:3` слишком короткий, поэтому функция `cbind` вызывает правило повторного использования и переиспользует элементы `1:3`:

```
cbind(1:6, 1:3)
#>      [,1] [,2]
#> [1,]    1    1
#> [2,]    2    2
#> [3,]    3    3
#> [4,]    4    1
#> [5,]    5    2
#> [6,]    6    3
```

Если длина более длинного вектора не кратна длине более короткого вектора, R выдает предупреждение. Это хорошо, поскольку такая операция весьма подозрительна, и в вашей логике, скорее всего, есть ошибка:

```
(1:6) + (1:5) # Oops! 1:5 is one element too short
#> Warning in (1:6) + (1:5): longer object length is not a multiple of shorter
#> object length
#> [1] 2 4 6 8 10 7
```

Как только вы усвоите правило повторного использования, вы поймете, что операции между вектором и скаляром – всего лишь приложения к этому правилу. В этом примере `10` «переиспользуется» повторно до завершения добавления вектора:

```
(1:6) + 10
#> [1] 11 12 13 14 15 16
```

5.4. Создание фактора (категориальной переменной)

Задача

У вас есть вектор символьных строк или целых чисел. Вы хотите, чтобы R рассматривал их как фактор, который представляет собой термин для обозначения категориальной переменной.

Решение

Функция `factor` кодирует ваш вектор дискретных значений в фактор:

```
f <- factor(v) # v может быть вектором строк или целых чисел.
```

Если ваш вектор содержит только подмножество возможных значений, а не всю вселенную, включите сюда второй аргумент, который определяет возможные уровни фактора:

```
f <- factor(v, levels)
```

Обсуждение

В языке R каждое возможное значение категориальной переменной называется *уровнем*. Вектор уровней называется *фактором*. Факторы очень четко вписываются в векторную ориентацию R и используются мощными способами для обработки данных и создания статистических моделей.

В большинстве случаев преобразование ваших категориальных данных в фактор – это простой вызов функции `factor`, которая идентифицирует отдельные уровни категориальных данных и упаковывает их в фактор:

```
f <- factor(c("Win", "Win", "Lose", "Tie", "Win", "Lose"))
f
#> [1] Win Win Lose Tie Win Lose
#> Levels: Lose Tie Win
```

Обратите внимание, что когда мы вывели фактор, `f`, R не стал заключать значения в кавычки. Это уровни, а не строки. Также обратите внимание на то, что когда мы вывели фактор, R отобразил различные уровни ниже фактора.

Если ваш вектор содержит только подмножество всех возможных уровней, у R будет неполная картина возможных уровней. Предположим, у вас есть строковая переменная `wday`, которая дает день недели, в который наблюдались ваши данные:

```
wday <- c("Wed", "Thu", "Mon", "Wed", "Thu",
          "Thu", "Thu", "Tue", "Thu", "Tue")
f <- factor(wday)
f
#> [1] Wed Thu Mon Wed Thu Thu Thu Tue Thu Tue
#> Levels: Mon Thu Tue Wed
```

R считает, что понедельник, четверг, вторник и среда – единственно возможные уровни. Пятницы нет в списке. По-видимому, сотрудники лаборатории никогда не проводили наблюдения в пятницу, поэтому R не знает, что пятница является возможным значением. Следовательно, вам нужно явно перечислить возможные уровни `wday`:

```
f <- factor(wday, levels=c("Mon", "Tue", "Wed", "Thu", "Fri"))
f
#> [1] Wed Thu Mon Wed Thu Thu Thu Tue Thu Tue
#> Levels: Mon Tue Wed Thu Fri
```

Теперь R понимает, что `f` – это фактор с пятью возможными уровнями. Он также знает их правильный порядок. Первоначально он поставил четверг перед вторником, поскольку по умолчанию принимает алфавитный порядок. Явный аргумент `levels` определяет правильный порядок.

Во многих случаях нет необходимости вызывать функцию `factor` явно. Когда какая-то функция R требует фактора, она обычно автоматически преобразовывает ваши данные в фактор. Например, функция `table` работает только с факторами, поэтому она обычно преобразует свои входные данные в факторы, не спрашивая. Вы должны явно создать переменную фактора, когда хотите указать полный набор уровней или контролировать порядок уровней.

См. также

См. рецепт 12.5, чтобы создать фактор из текущих данных.

5.5. Объединение нескольких векторов в один вектор и фактор

Задача

У вас есть несколько групп данных с одним вектором для каждой группы. Вы хотите объединить векторы в один большой вектор и одновременно создать параллельный фактор, который идентифицирует исходную группу каждого значения.

Решение

Создайте список, содержащий векторы. Используйте функцию `stack`, чтобы объединить список в таблицу данных из двух столбцов:

```
comb <- stack(list(v1 = v1, v2 = v2, v3 = v3)) # Объединяем 3 вектора.
```

Столбцы таблицы данных носят названия `values` и значения `ind`. Первый столбец содержит данные, а второй – параллельный фактор.

Обсуждение

Для чего, скажите на милость, вам нужно объединять все свои данные в один большой вектор и параллельный фактор? Причина в том, что многие важные статистические функции требуют данных в этом формате.

Предположим, что вы опрашиваете первокурсников, второкурсников и третьекурсников на предмет их уровня уверенности («Какой процент времени вы чувствуете себя уверенно на занятиях?»). Теперь у вас есть три вектора: `freshmen`, `sophomores` и `juniors`. Вы хотите выполнить дисперсионный анализ (ANOVA) различий между группами. Функции ANOVA, `aov`, требуется один вектор с результатами опроса, а также параллельный фактор, который идентифицирует группу. Можно объединить группы, используя функцию `stack`:

```
freshmen <- c(1, 2, 1, 1, 5)
sophomores <- c(3, 2, 3, 3, 5)
juniors <- c(5, 3, 4, 3, 3)

comb <- stack(list(fresh = freshmen, soph = sophomores, jrs = juniors))
print(comb)
#>    values  ind
#> 1       1   fresh
#> 2       2   fresh
#> 3       1   fresh
#> 4       1   fresh
#> 5       5   fresh
#> 6       3   soph
#> 7       2   soph
#> 8       3   soph
#> 9       3   soph
#> 10      5   soph
#> 11      5   jrs
#> 12      3   jrs
#> 13      4   jrs
#> 14      3   jrs
#> 15      3   jrs
```

Теперь вы можете выполнить дисперсионный анализ на двух столбцах:

```
aov(values ~ ind, data = comb)
```

При построении списка мы должны предоставить теги для элементов списка. (В этом примере это теги `fresh`, `soph` и `jrs`.) Эти теги необходимы, поскольку функция `stack` использует их в качестве уровней параллельного фактора.

5.6. Создание списка

Задача

Вы хотите создать и заполнить список.

Решение

Чтобы создать список из отдельных элементов данных, используйте функцию `list`:

```
lst <- list(x, y, z)
```

Обсуждение

Списки могут быть довольно простыми, как, например, этот список из трех чисел:

```
lst <- list(0.5, 0.841, 0.977)
lst
#> [[1]]
#> [1] 0.5
#>
#> [[2]]
#> [1] 0.841
#>
#> [[3]]
#> [1] 0.977
```

Когда R выводит список, он идентифицирует каждый элемент списка по его позиции ([[1]], [[2]], [[3]]) и выводит значение элемента (т. е. [1]0,5) под его позицией.

Более целесообразно, когда списки могут, в отличие от векторов, содержать элементы разных режимов (типов). Ниже приводится экстремальный пример кода, созданного из скаляра, строки символов, вектора и функции:

```
lst <- list(3.14, "Moe", c(1, 1, 2, 3), mean)
lst
#> [[1]]
#> [1] 3.14
#>
#> [[2]]
#> [1] "Moe"
#>
#> [[3]]
#> [1] 1 1 2 3
#>
#> [[4]]
#> function (x, ...)
#> UseMethod("mean")
#> <bytecode: 0x7ff04b0bc900>
#> <environment: namespace:base>
```

Вы также можете создать список, создав пустой список и заполнив его. Ниже приводится наш пример кода, построенный таким образом:

```
lst <- list()
lst[[1]] <- 3.14
lst[[2]] <- "Moe"
lst[[3]] <- c(1, 1, 2, 3)
lst[[4]] <- mean
lst
#> [[1]]
#> [1] 3.14
#>
#> [[2]]
#> [1] "Moe"
#>
#> [[3]]
#> [1] 1 1 2 3
#>
#> [[4]]
#> function (x, ...)
#> UseMethod("mean")
#> <bytecode: 0x7ff04b0bc900>
#> <environment: namespace:base>
```

Элементы списка могут быть проименованы. Функция `list` позволяет указать имя для каждого элемента:

```
lst <- list(mid = 0.5, right = 0.841, far.right = 0.977)
lst
#> $mid
#> [1] 0.5
#>
#> $right
#> [1] 0.841
```

```
#>
#> $far.right
#> [1] 0.977
```

См. также

См. введение к этой главе для получения дополнительной информации о списках; *см.* рецепт 5.9 для получения дополнительной информации о создании и использовании списков с именованными элементами.

5.7. Выбор элементов по месту в списке

Задача

Вы хотите получить доступ к элементам по месту в списке.

Решение

Используйте один из этих способов. Здесь `lst` – это переменная списка:

```
lst[[n]]
```

Выбирает *n*-й элемент из списка.

```
lst [c(n1, n2, ..., nk)]
```

Возвращает список элементов, выбранных по их месту в списке.

Обратите внимание, что первая форма возвращает один элемент, а вторая возвращает список.

Обсуждение

Предположим, у нас есть список из четырех целых чисел с именем `years`:

```
years <- list(1960, 1964, 1976, 1994)
years
#> [[1]]
#> [1] 1960
#>
#> [[2]]
#> [1] 1964
#>
#> [[3]]
#> [1] 1976
#>
#> [[4]]
#> [1] 1994
```

Мы можем получить доступ к отдельным элементам, используя синтаксис двойных квадратных скобок:

```
years[[1]]
#> [1] 1960
```

Мы можем извлечь подсписки, используя синтаксис с одной квадратной скобкой:

```
years[c(1, 2)]
#> [[1]]
#> [1] 1960
```

```
#>
#> [[2]]
#> [1] 1964
```

Этот синтаксис может сбивать с толку из-за одного нюанса: между `lst[[n]]` и `lst[n]` есть важное различие. Это не одно и то же.

`lst[[n]]`

Это элемент, а не список. Это *n*-й элемент `lst`.

`lst[n]`

Это список, а не элемент. Этот список содержит один элемент, взятый из *n*-го элемента `lst`.

Вторая форма – это частный случай `lst [c(`n_1`, `n_2`, ..., `n_k`)]`, в котором мы исключили конструкцию `c(...)`, поскольку есть только один *n*. Разница становится очевидной, когда мы проверяем структуру результата: первое – это число, а второе – список:

```
class(years[[1]])
#> [1] "numeric"
class(years[1])
#> [1] "list"
```

Разница становится раздражающе очевидной, когда мы выводим значение с помощью функции `cat`. Напомним, что эта функция может выводить атомарные значения или векторы, но жалуется при выводе структурированных объектов:

```
cat(years[[1]], "\n")
#> 1960
cat(years[1], "\n")
#> Error in cat(years[1], "\n"): argument 1 (type 'list')
#> cannot be handled by 'cat'
```

Нам повезло, потому что R предупредил нас о проблеме. В других контекстах вы могли бы долго и усердно работать, чтобы выяснить, что вы получили доступ к подсписку, в то время когда вам нужен элемент, или наоборот.

5.8. Выбор элементов списка по имени

Задача

Вы хотите получить доступ к элементам списка по их именам.

Решение

Используйте одну из этих форм. Здесь `lst` – это переменная списка:

`lst[["name"]]`

Выбирает элемент с именем `name`. Возвращает `NULL`, если ни у одного элемента нет этого имени.

`lst$name`

То же, что и предыдущая форма, только другой синтаксис.

`lst[c(name1, name2, ..., namek)]`

Возвращает список, созданный из указанных элементов `lst`.

Обратите внимание, что первые две формы возвращают элемент, а третья возвращает список.

Обсуждение

У каждого элемента списка может быть имя. Если имя указано, элемент можно выбирать по имени. В этом присваивании мы создаем список из четырех именованных целых чисел:

```
years <- list(Kennedy = 1960, Johnson = 1964,
Carter = 1976, Clinton = 1994)
```

Эти два выражения возвращают одно и то же значение, а именно элемент с именем «Kennedy»:

```
years[["Kennedy"]]
#> [1] 1960
years$Kennedy
#> [1] 1960
```

Приведенные ниже выражения возвращают подсписки, извлеченные из списка `years`:

```
years[c("Kennedy", "Johnson")]
#> $Kennedy
#> [1] 1960
#>
#> $Johnson
#> [1] 1964

years["Carter"]
#> $Carter
#> [1] 1976
```

Как и при выборе элементов списка по позиции (*см.* рецепт 5.7), между `lst[["name"]]` и `lst["name"]` есть важное различие. Это не одно и то же:

`lst[["name"]]`

Это элемент, а не список.

`lst["name"]`

Это список, а не элемент.

Вторая форма – частный случай `lst[c(name`$_1$`, name`$_2$`, ..., name`$_k$`)]`, в котором нам не нужна конструкция `c(...)`, потому что здесь только одно имя.

См. также

См. рецепт 5.7, чтобы узнать, как получить доступ к элементам по позиции, а не по имени.

5.9. Создание ассоциативного списка «имя/значение»

Задание

Вы хотите создать список, который связывает имена и значения, например словарь, хеш или справочную таблицу, как на другом языке программирования.

Решение

Функция `list` позволяет присваивать имена элементам, создавая связь между каждым именем и его значением:

```
lst <- list(mid = 0.5, right = 0.841, far.right = 0.977)
```

Если у вас есть параллельные векторы имен и значений, вы можете создать пустой список, а затем заполнить его, используя векторизованный `lst[names] <- values` оператор присваивания:

```
values <- c(1, 2, 3)
names <- c("a", "b", "c")
lst <- list()
lst[names] <- values
```

Обсуждение

Каждый элемент списка может быть проименован, и вы можете получить элементы списка по имени. Это дает вам базовый инструмент программирования: возможность связывать имена и значения.

Вы можете присваивать имена элементов при создании списка. Функция `list` допускает аргументы вида `name=value`:

```
lst <- list(
far.left = 0.023,
left = 0.159,
mid = 0.500,
right = 0.841,
far.right = 0.977
)
lst
#> $far.left
#> [1] 0.023
#>
#> $left
#> [1] 0.159
#>
#> $mid
#> [1] 0.5
#>
#> $right
#> [1] 0.841
#>
#> $far.right
#> [1] 0.977
```

Один из способов именования элементов – создать пустой список, а затем заполнить его с помощью операторов присваивания:

```
lst <- list()
lst$far.left <- 0.023
lst$left <- 0.159
lst$mid <- 0.500
lst$right <- 0.841
lst$far.right <- 0.977
```

Иногда у вас есть вектор имен и вектор соответствующих значений:

```
values <- -2:2
names <- c("far.left", "left", "mid", "right", "far.right")
```

Вы можете связать имена и значения, создав пустой список, а затем заполнив его, используя векторизованный оператор присваивания:

```
lst <- list()
lst[names] <- values
lst
#> $far.left
#> [1] -2
#>
#> $left
#> [1] -1
#>
#> $mid
#> [1] 0
#>
#> $right
#> [1] 1
#>
#> $far.right
#> [1] 2
```

Как только связь будет установлена, список может «транслировать» имена в значения с помощью простого поиска в списке:

```
cat("The left limit is", lst[["left"]], "\n")
#> The left limit is -1
cat("The right limit is", lst[["right"]], "\n")
#> The right limit is 1

for (nm in names(lst)) cat("The", nm, "limit is", lst[[nm]], "\n")
#> The far.left limit is -2
#> The left limit is -1
#> The mid limit is 0
#> The right limit is 1
#> The far.right limit is 2
```

5.10. Удаление элемента из списка

Задача

Вы хотите удалить элемент из списка.

Решение

Присвойте элементу значение `NULL`. R удалит его из списка.

Обсуждение

Чтобы удалить элемент списка, выберите его по позиции или по имени, а затем присвойте выбранному элементу значение NULL:

```
years <- list(Kennedy = 1960, Johnson = 1964,
Carter = 1976, Clinton = 1994)
years
#> $Kennedy
#> [1] 1960
#>
#> $Johnson
#> [1] 1964
#>
#> $Carter
#> [1] 1976
#>
#> $Clinton
#> [1] 1994
years[["Johnson"]] <- NULL # Удаляем элемент с пометкой "Johnson".
years
#> $Kennedy
#> [1] 1960
#>
#> $Carter
#> [1] 1976
#>
#> $Clinton
#> [1] 1994
```

Вы также можете удалить несколько элементов:

```
years[c("Carter", "Clinton")] <- NULL # Remove two elements
years
#> $Kennedy
#> [1] 1960
```

5.11. Преобразование списка в вектор

Задача

Вы хотите преобразовать все элементы списка в вектор.

Решение

Используйте функцию unlist.

Обсуждение

Существует множество контекстов, где требуются векторы. Например, базовые статистические функции работают с векторами, но не со списками. Если iq.scores – это список чисел, мы не можем непосредственно рассчитать их среднее значение:

```
iq.scores <- list(100, 120, 103, 80, 99)
mean(iq.scores)
#> Warning in mean.default(iq.scores): argument is not numeric or logical:
```

```
#> returning NA
#> [1] NA
```

Вместо этого мы должны преобразовать список в вектор, используя функцию `unlist`, а затем вычислить среднее значение результата:

```
mean(unlist(iq.scores))
#> [1] 100
```

Вот еще один пример. Мы можем использовать функцию `cat` для скаляров и векторов, но не для списка:

```
cat(iq.scores, "\n")
#> Error in cat(iq.scores, "\n"): argument 1 (type 'list') cannot be
#> handled by 'cat'
```

Одним из решений является преобразование списка в вектор перед выводом на экран:

```
cat("IQ Scores:", unlist(iq.scores), "\n")
#> IQ Scores: 100 120 103 80 99
```

См. также

Подобные преобразования более подробно обсуждаются в рецепте 5.29.

5.12. Удаление элементов NULL из списка

Задача

Ваш список содержит значения `NULL`. Вы хотите удалить их.

Решение

Функция `compact` из пакета `purrr` удалит элементы `NULL`.

Обсуждение

Любопытному читателю может быть интересно, как список может содержать элементы `NULL`, учитывая, что мы удаляем элементы, присваивая им значение `NULL` (*см.* рецепт 5.10). Дело в том, что мы можем *создать* список, содержащий элементы `NULL`:

```
library(purrr) # Или library(tidyverse)

lst <- list("Moe", NULL, "Curly")
lst
#> [[1]]
#> [1] "Moe"
#>
#> [[2]]
#> NULL
#>
#> [[3]]
#> [1] "Curly"

compact(lst) # Удаляем элемент NULL
#> [[1]]
```

```
#> [1] "Moe"
#>
#> [[2]]
#> [1] "Curly"
```

На практике мы также могли бы получить элементы NULL в списке, применив преобразование.

Обратите внимание, что в R NA и NULL – не одно и то же. Функция compact удалит из списка значения NULL, но не NA. Чтобы узнать, как удалить значения NA, *см.* рецепт 5.13.

См. также

См. рецепт 5.10, чтобы узнать, как удалить элементы списка, и рецепт 5.13, чтобы узнать, как удалить элементы списка, используя условие.

5.13. Удаление элементов списка с использованием условия

Задача

Вы хотите удалить элементы из списка в соответствии с условным тестом, например элементы, которые не определены, являются отрицательными или меньше некоего порога.

Решение

Начните с функции, которая возвращает TRUE, когда ваши критерии выполнены, или FALSE в противном случае. Затем используйте функцию discard из purrr, чтобы удалить значения, соответствующие вашим критериям. В этом фрагменте кода, например, мы используем функцию is.na для удаления значений NA из lst:

```
lst <- list(NA, 0, NA, 1, 2)

lst %>%
discard(is.na)
#> [[1]]
#> [1] 0
#>
#> [[2]]
#> [1] 1
#>
#> [[3]]
#> [1] 2
```

Обсуждение

Функция discard удаляет элементы из списка с помощью *предиката*, функции, которая возвращает значение TRUE либо FALSE. Предикат применяется к каждому элементу списка. Если предикат возвращает значение TRUE, элемент отбрасывается; в противном случае он сохраняется.

Предположим, мы хотим удалить строки символов из `lst`. Функция `is.character` – это предикат, который возвращает значение `TRUE`, если его аргумент является символьной строкой, поэтому мы можем использовать его с функцией `discard`:

```
lst <- list(3, "dog", 2, "cat", 1)

lst %>%
discard(is.character)
#> [[1]]
#> [1] 3
#>
#> [[2]]
#> [1] 2
#>
#> [[3]]
#> [1] 1
```

Можно определить собственный предикат и использовать его с функцией `discard`. В этом примере мы удаляем значения `NA` и `NULL` из списка путем определения предиката `is_na_or_null`:

```
is_na_or_null <- function(x) {
is.na(x) || is.null(x)
}

lst <- list(1, NA, 2, NULL, 3)

lst %>%
discard(is_na_or_null)
#> [[1]]
#> [1] 1
#>
#> [[2]]
#> [1] 2
#>
#> [[3]]
#> [1] 3
```

Списки могут содержать сложные объекты, а не только атомарные значения. Предположим, что `mods` – это список линейных моделей, созданных функцией `lm`:

```
mods <- list(lm(x ~ y1),
lm(x ~ y2),
lm(x ~ y3))
```

Мы можем определить предикат `filter_r2`, чтобы идентифицировать модели, чьи значения R^2 меньше 0,70, а затем использовать предикат для удаления этих моделей из `mods`:

```
filter_r2 <- function(model) {
summary(model)$r.squared < 0.7
}

mods %>%
discard(filter_r2)
```

Существует функция `keep`, которая является инверсией функции `discard`. Она использует предикат для сохранения элементов списка, вместо того чтобы отбрасывать их.

См. также

См. рецепты 5.7, 5.10 и 15.3.

5.14. Инициализация матрицы

Задача

Вы хотите создать матрицу и инициализировать ее из заданных значений.

Решение

Зафиксируйте данные в векторе или списке, а затем используйте функцию `matrix`, чтобы сформировать данные в матрицу. В этом примере мы формируем вектор в матрицу 2×3 (т. е. две строки и три столбца):

```
vec <- 1:6
matrix(vec, 2, 3)
#> [,1] [,2] [,3]
#> [1,] 1 3 5
#> [2,] 2 4 6
```

Обсуждение

Первый аргумент функции `matrix` – это данные, второй аргумент – количество строк, а третий – количество столбцов. Обратите внимание, что матрица, используемая в решении, заполнялась столбец за столбцом, а не строка за строкой.

Обычно для инициализации всей матрицы используется одно значение, например `0` или `NA`. Если первый аргумент функции `matrix` – это единственное значение, R применит правило повторного использования и автоматически скопирует значение, чтобы заполнить всю матрицу:

```
matrix(0, 2, 3) # Создаем матрицу, содержащую все нули.
#> [,1] [,2] [,3]
#> [1,] 0 0 0
#> [2,] 0 0 0

matrix(NA, 2, 3) # Создаем матрицу, заполненную значениями NA.
#> [,1] [,2] [,3]
#> [1,] NA NA NA
#> [2,] NA NA NA
```

Конечно, можно создать матрицу с помощью одной строки кода, но ее станет трудно читать:

```
mat <- matrix(c(1.1, 1.2, 1.3, 2.1, 2.2, 2.3), 2, 3)
mat
#> [,1] [,2] [,3]
#> [1,] 1.1 1.3 2.2
#> [2,] 1.2 2.1 2.3
```

В R распространена идиома, при которой данные вводятся в виде прямоугольника, который раскрывает структуру матрицы:

```
theData <- c(
1.1, 1.2, 1.3,
2.1, 2.2, 2.3
```

```
)
mat <- matrix(theData, 2, 3, byrow = TRUE)
mat
#> [,1] [,2] [,3]
#> [1,] 1.1 1.2 1.3
#> [2,] 2.1 2.2 2.3
```

Установив для аргумента `byrow` значение `TRUE`, мы сообщаем функции `matrix`, что данные являются построчными, а не идут столбец за столбцом (как по умолчанию). В сжатом виде это будет выглядеть так:

```
mat <- matrix(c(1.1, 1.2, 1.3,
2.1, 2.2, 2.3),
2, 3,
byrow = TRUE)
```

Таким образом, легко увидеть две строки и три столбца данных.

Существует быстрый обходной способ превратить вектор в матрицу: просто присвойте вектору размеры. Это обсуждалось во введении к данной главе. В приведенном ниже примере мы создаем стандартный вектор, а затем формируем его в матрицу 2×3:

```
v <- c(1.1, 1.2, 1.3, 2.1, 2.2, 2.3)
dim(v) <- c(2, 3)
v
#> [,1] [,2] [,3]
#> [1,] 1.1 1.3 2.2
#> [2,] 1.2 2.1 2.3
```

Мы находим такой способ менее прозрачным, нежели использование функции `matrix`, тем более что здесь нет опции `byrow`.

См. также

См. рецепт 5.3.

5.15. Операции с матрицами

Задача

Вы хотите выполнить такие операции, как транспонирование, инверсия, умножение или построение единичной матрицы.

Решение

Выполните эти операции, используя следующие функции:

`t(A)`

Матричное транспонирование `A`.

`solve(A)`

Матрица, обратная к `A`.

`A %*% B`

Матричное умножение `A` и `B`.

`diag(n)`

Создает $n \times n$ диагональную (единичную) матрицу.

Обсуждение

Напомним, что `A*B` – это поэлементное умножение, тогда как `A% *% B` – матричное умножение (*см.* рецепт 2.11).

Все эти функции возвращают матрицу. Их аргументы могут быть матрицами или таблицами данных. Если это таблицы данных, R сначала преобразует их в матрицы (хотя это имеет смысл, только если таблица данных содержит исключительно числовые значения).

5.16. Задание описательных имен для строк и столбцов матрицы

Задача

Вы хотите присвоить описательные имена для строк или столбцов матрицы.

Решение

У каждой матрицы есть атрибуты `rownames` и `colnames`. Присвойте вектор символьных строк соответствующему атрибуту:

```
rownames(mat) <- c("rowname1", "rowname2", ..., "rownameN")
colnames(mat) <- c("colname1", "colname2", ..., "colnameN")
```

Обсуждение

R позволяет присваивать имена строкам и столбцам матрицы, что полезно для вывода матрицы. R будет отображать имена, если они определены, улучшая читабельность вашего вывода. Рассмотрим приведенную ниже матрицу корреляций между ценами на акции IBM, Microsoft и Google:

```
print(corr_mat)
#> [,1] [,2] [,3]
#> [1,] 1.000 0.556 0.390
#> [2,] 0.556 1.000 0.444
#> [3,] 0.390 0.444 1.000
```

В этой форме интерпретация матрицы не является самоочевидной. Мы можем дать имена строкам и столбцам, уточнив их значение:

```
colnames(corr_mat) <- c("AAPL", "MSFT", "GOOG")
rownames(corr_mat) <- c("AAPL", "MSFT", "GOOG")
corr_mat
#> AAPL MSFT GOOG
#> AAPL 1.000 0.556 0.390
#> MSFT 0.556 1.000 0.444
#> GOOG 0.390 0.444 1.000
```

Теперь сразу видно, какие строки и столбцы к каким акциям относятся.

Еще одно преимущество именования строк и столбцов заключается в том, что вы можете обращаться к элементам матрицы по этим именам:

```
# Какова корреляция между MSFT и GOOG?
corr_mat["MSFT", "GOOG"]
#> [1] 0.444
```

5.17. Выбор одной строки или столбца из матрицы

Задача

Вы хотите выбрать одну строку или один столбец из матрицы.

Решение

Решение зависит от того, что вы хотите. Если вы хотите, чтобы результат был простым вектором, просто используйте обычное индексирование:

```
mat[1, ] # Первая строка.
mat[, 3] # Третий столбец.
```

Если вы хотите, чтобы результат представлял собой однострочную матрицу или матрицу из одного столбца, включите в код аргумент `drop=FALSE`:

```
mat[1, , drop=FALSE] # Первая строка, однострочная матрица.
mat[, 3, drop=FALSE] # Третий столбец, матрица из одного столбца.
```

Обсуждение

Обычно, когда вы выбираете одну строку или столбец из матрицы, R удаляет размеры.

В результате получается безразмерный вектор:

```
mat[1, ]
#> [1] 1.1 1.2 1.3
mat[, 3]
#> [1] 1.3 2.3
```

Однако при включении аргумента `drop=FALSE` R сохраняет размеры. В этом случае при выборе строки возвращается вектор строки (матрица $1 \times n$):

```
mat[1, , drop=FALSE]
#>      [,1] [,2] [,3]
#> [1,]  1.1  1.2  1.3
```

Аналогично, при выборе столбца с помощью аргумента `drop=FALSE` возвращается вектор столбца (матрица $n \times 1$):

```
mat[, 3, drop=FALSE]
#>      [,1]
#> [1,]  1.3
#> [2,]  2.3
```

5.18. Инициализация таблицы данных из данных столбца

Задача

Ваши данные организованы по столбцам, и вы хотите собрать их в таблицу данных.

Решение

Если ваши данные зафиксированы в нескольких векторах и/или факторах, используйте функцию `data.frame`, чтобы собрать их в таблицу данных:

```
df <- data.frame(v1, v2, v3, f1)
```

Если ваши данные находятся в списке, содержащем векторы и/или факторы, используйте функцию `as.data.frame`:

```
df <- as.data.frame(list.of.vectors)
```

Обсуждение

Таблица данных – это набор столбцов, каждый из которых соответствует наблюдаемой переменной (в статистическом смысле, а не в смысле программирования). Если ваши данные уже организованы в столбцы, тогда создать таблицу данных легко.

Функция `data.frame` может построить таблицу данных из векторов, где каждый вектор – это наблюдаемая переменная. Предположим, у вас есть две числовые переменные, одна символьная переменная и одна переменная ответа. Функция `data.frame` может создать таблицу данных из ваших векторов:

```
data.frame(pred1, pred2, pred3, resp)
#> pred1 pred2 pred3 resp
#> 1 1.75 11.8 AM 13.2
#> 2 4.01 10.7 PM 12.9
#> 3 2.64 12.2 AM 13.9
#> 4 6.03 12.2 PM 14.9
#> 5 2.78 15.0 PM 16.4
```

Обратите внимание, что `data.frame` берет имена столбцов из ваших программных переменных. Можно переопределить это значение по умолчанию, указав явные имена столбцов:

```
data.frame(p1 = pred1, p2 = pred2, p3 = pred3, r = resp)
#> p1 p2 p3 r
#> 1 1.75 11.8 AM 13.2
#> 2 4.01 10.7 PM 12.9
#> 3 2.64 12.2 AM 13.9
#> 4 6.03 12.2 PM 14.9
#> 5 2.78 15.0 PM 16.4
```

Если вам больше нравится tibble, а не стандартная таблица данных, используйте функцию `tibble` из коллекции tidyverse:

```
tibble(p1 = pred1, p2 = pred2, p3 = pred3, r = resp)
#> # Tibble: 5 x 4
#> p1 p2 p3 r
#> <dbl> <dbl> <fct> <dbl>
#> 1 1.75 11.8 AM 13.2
#> 2 4.01 10.7 PM 12.9
```

```
#> 3 2.64 12.2 AM 13.9
#> 4 6.03 12.2 PM 14.9
#> 5 2.78 15.0 PM 16.4
```

Иногда ваши данные могут быть организованы в векторы, но эти векторы содержатся в списке, а не в отдельных программных переменных:

```
list.of.vectors <- list(p1=pred1, p2=pred2, p3=pred3, r=resp)
```

В этом случае используйте функцию `as.data.frame` для создания таблицы данных из списка:

```
as.data.frame(list.of.vectors)
#> p1 p2 p3 r
#> 1 1.75 11.8 AM 13.2
#> 2 4.01 10.7 PM 12.9
#> 3 2.64 12.2 AM 13.9
#> 4 6.03 12.2 PM 14.9
#> 5 2.78 15.0 PM 16.4
```

либо функцию `as_tibble` для создания tibble:

```
as_tibble(list.of.vectors)
#> # Tibble: 5 x 4
#> p1 p2 p3 r
#> <dbl> <dbl> <fct> <dbl>
#> 1 1.75 11.8 AM 13.2
#> 2 4.01 10.7 PM 12.9
#> 3 2.64 12.2 AM 13.9
#> 4 6.03 12.2 PM 14.9
#> 5 2.78 15.0 PM 16.4
```

Факторы в таблицах данных

Существует важное различие между созданием таблицы данных и созданием tibble.

Когда вы используете функцию `data.frame` для создания таблицы данных, R по умолчанию преобразует значения символов в факторы. Значение `pred3` в предыдущем примере было преобразовано в фактор, но в выводе этого не видно.

Однако функции `tibble` и `as_tibble` не изменяют символьные данные. Если вы посмотрите на пример с tibble, то увидите, что столбец `p3` имеет тип `chr`, что значит символ.

Об этом различии следует знать. Бывает невыносимо сложно отладить проблему, вызванную таким нюансом.

5.19. Инициализация таблицы данных из данных строки

Задача

Ваши данные организованы по строкам, и вы хотите собрать их в таблицу данных.

Решение

Сохраните каждую строку в однострочной таблице данных. Используйте функцию `rbind`, чтобы связать строки в одну большую таблицу данных:

```
rbind(row1, row2, ... , rowN)
```

Обсуждение

Данные часто поступают в виде набора наблюдений. Каждое наблюдение представляет собой запись или кортеж, который содержит несколько значений, по одному на каждую наблюдаемую переменную. Строки плоского файла обычно выглядят так: каждая строка представляет собой одну запись, каждая запись содержит несколько столбцов, а каждый столбец представляет собой отдельную переменную (*см.* рецепт 4.15). Такие данные организованы по *наблюдениям*, а не по *переменным*. Другими словами, вы получаете строки по одной за раз, а не столбцы по одному за раз.

Каждая такая строка может храниться несколькими способами. Одним из очевидных способов является вектор. Если у вас есть чисто числовые данные, используйте вектор.

Однако многие наборы данных представляют собой смесь числовых, символьных и категориальных данных, и в этом случае вектор не подойдет. Мы рекомендуем хранить каждую такую неоднородную строку в однострочной таблице данных. (Вы можете сохранить каждую строку в списке, но такой способ становится несколько сложнее.)

Нам нужно связать эти строки в таблицу данных. Вот что делает функция `rbind`. Она связывает свои аргументы таким образом, что каждый аргумент в результате становится одной строкой. Например, если возьмем три этих наблюдения и применим функцию `rbind`, то получим таблицу данных из трех строк:

```
r1 <- data.frame(a = 1, b = 2, c = "X")
r2 <- data.frame(a = 3, b = 4, c = "Y")
r3 <- data.frame(a = 5, b = 6, c = "Z")
rbind(r1, r2, r3)
#> a b c
#> 1 1 2 X
#> 2 3 4 Y
#> 3 5 6 Z
```

Когда вы работаете с большим количеством строк, они, вероятно, будут сохранены в списке; то есть у вас будет список строк. Функция `bind_rows` из пакета `dplyr` коллекции tidyverse обрабатывает этот случай, как показано в приведенном ниже маленьком примере:

```
list.of.rows <- list(r1, r2, r3)
bind_rows(list.of.rows)
#> Warning in bind_rows_(x, .id): Unequal factor levels: coercing to character
#> Warning in bind_rows_(x, .id): binding character and factor vector,
#> coercing into character vector

#> Warning in bind_rows_(x, .id): binding character and factor vector,
#> coercing into character vector

#> Warning in bind_rows_(x, .id): binding character and factor vector,
#> coercing into character vector
#> a b c
#> 1 1 2 X
#> 2 3 4 Y
#> 3 5 6 Z
```

Иногда по не зависящим от вас причинам каждая строка данных хранится в списке, а не в однострочных таблицах данных. Например, вы можете иметь дело со строками, которые возвращает функция или пакет базы данных. `bind_rows` может справиться и с этой ситуацией:

```
# Те же данные, но строки хранятся в списках.
l1 <- list(a = 1, b = 2, c = "X")
l2 <- list(a = 3, b = 4, c = "Y")
l3 <- list(a = 5, b = 6, c = "Z")
list.of.lists <- list(l1, l2, l3)

bind_rows(list.of.lists)
#> # Tibble: 3x3
#> a b c
#> <dbl> <dbl> <chr>
#> 1 1 2 X
#> 2 3 4 Y
#> 3 5 6 Z
```

Факторы в таблицах данных

Если вы предпочитаете получать строки вместо факторов, у вас есть несколько вариантов.

Один из них – установить для параметра `stringsAsFactors` значение `FALSE` при вызове функции `data.frame`:

```
data.frame(a = 1, b = 2, c = "a", stringsAsFactors = FALSE)
#> a b c
#> 1 1 2 a
```

Конечно, если вы унаследовали свои данные и они уже находятся в таблице данных с факторами, вы можете преобразовать все факторы в символы, используя этот рецепт в качестве бонуса:

```
# Те же настройки, что и в предыдущих примерах.
l1 <- list( a=1, b=2, c='X' )
l2 <- list( a=3, b=4, c='Y' )
l3 <- list( a=5, b=6, c='Z' )
obs <- list(l1, l2, l3)
df <- do.call(rbind, Map(as.data.frame, obs))

# Да, вы могли бы использовать stringsAsFactors = FALSE,
# но мы предполагаем, что data.frame
# пришел к вам уже с факторами.

i <- sapply(df, is.factor) # Определяем, какие столбцы являются факторами.
df[i] <- lapply(df[i], as.character) # Превращаем только факторы в символы.
```

Имейте в виду, что если вы используете tibble вместо таблицы данных, символы по умолчанию не будут переводиться в факторы.

См. также

См. рецепт 5.18, если ваши данные организованы по столбцам, а не по строкам.

5.20. Добавление строк в таблицу данных

Задача

Вы хотите добавить одну или несколько новых строк в таблицу данных.

Решение

Создайте вторую, временную таблицу данных, содержащую новые строки. Затем используйте функцию `rbind`, чтобы добавить временную таблицу в исходную.

Обсуждение

Предположим, у нас есть таблица данных пригородов Чикаго:

```
suburbs <- read_csv("./data/suburbs.txt")
#> Parsed with column specification:
#> cols(
#> city = col_character(),
#> county = col_character(),
#> state = col_character(),
#> pop = col_double()
#> )
```

Далее предположим, что нам нужно добавить новую строку. Сначала мы создаем однострочную таблицу данных с новыми данными:

```
newRow <- data.frame(city = "West Dundee", county = "Kane",
state = "IL", pop = 7352)
```

Далее используем функцию `rbind` для добавления этой таблицы данных к существующей:

```
rbind(suburbs, newRow)
#> # Tibble: 18x4
#> city county state pop
#> <chr> <chr> <chr> <dbl>
#> 1 Chicago Cook IL 2853114
#> 2 Kenosha Kenosha WI 90352
#> 3 Aurora Kane IL 171782
#> 4 Elgin Kane IL 94487
#> 5 Gary Lake(IN) IN 102746
#> 6 Joliet Kendall IL 106221
#> # ... with 12 more rows
```

Функция `rbind` сообщает R, что мы добавляем в `suburbs` новую строку, а не новый столбец. Для вас может быть очевидным, что `newRow` – это строка, а не столбец, но для R это не очевидно (используйте функцию `cbind` для добавления столбца).

В новой строке должны использоваться те же имена столбцов, что и в таблице данных.
В противном случае `rbind` потерпит неудачу.

Конечно, можно объединить эти два шага в один:

```
rbind(suburbs,
data.frame(city = "West Dundee", county = "Kane",
state = "IL", pop = 7352))
#> # Tibble: 18x4
#> city county state pop
#> <chr> <chr> <chr> <dbl>
#> 1 Chicago Cook IL 2853114
#> 2 Kenosha Kenosha WI 90352
#> 3 Aurora Kane IL 171782
#> 4 Elgin Kane IL 94487
#> 5 Gary Lake(IN) IN 102746
#> 6 Joliet Kendall IL 106221
#> # ... with 12 more rows
```

Мы даже можем расширить данный метод на несколько новых строк, потому что функция `rbind` допускает использование нескольких аргументов:

```
rbind(suburbs,
data.frame(city = "West Dundee", county = "Kane",
state = "IL", pop = 7352),
data.frame(city = "East Dundee", county = "Kane",
state = "IL", pop = 3192)
)
#> # Tibble: 19×4
#> city county state pop
#> <chr> <chr> <chr> <dbl>
#> 1 Chicago Cook IL 2853114
#> 2 Kenosha Kenosha WI 90352
#> 3 Aurora Kane IL 171782
#> 4 Elgin Kane IL 94487
#> 5 Gary Lake(IN) IN 102746
#> 6 Joliet Kendall IL 106221
#> # ... with 13 more rows
```

Стоит отметить, что в предыдущих примерах мы незаметно смешивали tibble и таблицы данных. `suburbs` – это tibble, потому что мы использовали функцию `read_csv`, которая создает tibble, а `newRow` была создана с использованием функции `data.frame`, которая возвращает традиционную для R таблицу данных. И обратите внимание, что таблицы данных содержат факторы, у tibble их нет:

```
str(suburbs) # Tibble
#> Classes 'spec_tbl_df', 'tbl_df', 'tbl' and 'data.frame': 17 obs. of
#> 4 variables:
#> $ city : chr "Chicago" "Kenosha" "Aurora" "Elgin" ...
#> $ county: chr "Cook" "Kenosha" "Kane" "Kane" ...
#> $ state : chr "IL" "WI" "IL" "IL" ...
#> $ pop : num 2853114 90352 171782 94487 102746 ...
#> - attr(*, "spec")=
#> .. cols(
#> .. city = col_character(),
#> .. county = col_character(),
#> .. state = col_character(),
#> .. pop = col_double()
#> .. )
str(newRow) # a data.frame
#> 'data.frame': 1 obs. of 4 variables:
#> $ city : Factor w/ 1 level "West Dundee": 1
```

```
#> $ county: Factor w/ 1 level "Kane": 1
#> $ state : Factor w/ 1 level "IL": 1
#> $ pop : num 7352
```

Когда входные данные для функции `rbind` представляют собой смесь объектов `data.frame` и объектов `tibble`, результат будет иметь тот же тип, что и *первый* аргумент `rbind`. Таким образом, это приведет к появлению tibble:

```
rbind(some_tibble, some_data.frame)
```

а это даст таблицу данных:

```
rbind(some_data.frame, some_tibble)
```

5.21. Выбор столбцов по их месту в таблице данных

Задача

Вы хотите выбрать столбцы из таблицы данных в соответствии с их местом.

Решение

Используйте функцию `select`:

```
df %>% select(n1, n2, ..., nk)
```

где `df` – таблица данных, а n_1, n_2, …, n_k – целые числа со значениями от 1 до количества столбцов.

Обсуждение

Давайте используем первые три строки из набора данных о населении 16 крупнейших пригородов Большого Чикаго:

```
suburbs <- read_csv("data/suburbs.txt") %>% head(3)
#> Parsed with column specification:
#> cols(
#> city = col_character(),
#> county = col_character(),
#> state = col_character(),
#> pop = col_double()
#> )
suburbs
#> # Tibble: 3×4
#> city county state pop
#> <chr> <chr> <chr> <dbl>
#> 1 Chicago Cook IL 2853114
#> 2 Kenosha Kenosha WI 90352
#> 3 Aurora Kane IL 171782
```

Сразу же видно, что это tibble. Мы извлечем первый столбец (и только первый столбец):

```
suburbs %>%
dplyr::select(1)
#> # Tibble: 3×1
#> city
#> <chr>
#> 1 Chicago
```

```
#> 2 Kenosha
#> 3 Aurora
```

А здесь мы извлечем несколько столбцов:

```
suburbs %>%
dplyr::select(1, 3, 4)
#> # Tibble: 3x3
#> city state pop
#> <chr> <chr> <dbl>
#> 1 Chicago IL 2853114
#> 2 Kenosha WI 90352
#> 3 Aurora IL 171782
suburbs %>%
dplyr::select(2:4)
#> # Tibble: 3×3
#> county state pop
#> <chr> <chr> <dbl>
#> 1 Cook IL 2853114
#> 2 Kenosha WI 90352
#> 3 Kane IL 171782
```

Операторы списка

Глагол `select` является частью пакета `dplyr` коллекции tidyverse. Базовая версия R также обладает собственной богатой функциональностью для выбора столбцов за счет дополнительного синтаксиса. Эти варианты выбора могут сбивать с толку, пока вы не поймете логику, стоящую за этими альтернативами.

В одной из альтернатив используется синтаксис, как в случае списков. Это может показаться странным, пока вы не вспомните, что таблица данных представляет собой список столбцов. Выражение списка выбирает столбцы из этого списка. Читая это объяснение, обратите внимание на то, что изменение синтаксиса – двойные скобки по сравнению с одинарными – меняет значение оператора.

Мы можем выбрать ровно один столбец, используя двойные скобки ([[и]]):

`df[[n]]`

Возвращает *вектор*, а именно вектор в *n*-м столбце `df`.

Мы можем выбрать один или несколько столбцов, используя одинарные скобки ([и]).

`df[n]`

Возвращает *таблицу данных,* состоящую исключительно из *n*-го столбца `df`.

`df[c(n1, n2, ..., nk)]`

Возвращает *таблицу данных*, созданную из столбцов в позициях n_1, n_2, ..., n_k.

Например, мы можем использовать нотацию списка для выбора первого столбца из `suburbs`, столбца `city`:

```
suburbs[[1]]
#> [1] "Chicago" "Kenosha" "Aurora"
```

Этот столбец является символьным вектором, поэтому вот что возвращает `suburbs[[1]]`: вектор.

Результат изменяется, когда мы используем нотацию с одинарными скобками, как в `suburbs[1]` или `suburbs[c(1,3)]`. Мы по-прежнему получаем запрошенные столбцы, но R оставляет их в таблице данных. В этом примере мы возвращаем первый столбец в качестве таблицы данных с одним столбцом:

```
suburbs[1]
#> # Tibble: 3×1
#> city
#> <chr>
#> 1 Chicago
#> 2 Kenosha
#> 3 Aurora
```

А в этом примере мы возвращаем первый и третий столбцы в виде таблицы данных:

```
suburbs[c(1, 3)]
#> # Tibble: 3×2
#> city state
#> <chr> <chr>
#> 1 Chicago IL
#> 2 Kenosha WI
#> 3 Aurora IL
```

На самом деле выражение `suburbs[1]` – это сокращенная форма `suburbs[c(1)]`. Нам не нужна оболочка `c(...)`, потому что здесь только один *n*.

Основным источником путаницы является то, что `suburbs[[1]]` и `suburbs[1]` выглядят похоже, но дают совсем разные результаты:

```
suburbs[[1]]
```

Возвращает один столбец.

```
suburbs[1]
```

Возвращает таблицу данных, которая содержит ровно один столбец.

Дело в том, что фраза «один столбец» отличается от фразы «таблица данных, который содержит один столбец». Первое выражение возвращает вектор. Второе выражение возвращает таблицу данных, которая представляет собой другую структуру данных.

Матричный стиль индексации

Вы можете использовать матричную индексацию для выбора столбцов из таблицы данных:

```
df[, n]
```

Возвращает вектор, взятый из *n*-го столбца (при условии что `n` содержит ровно одно значение).

```
df[, c(n1, n2, ..., nk)]
```

Возвращает *таблицу данных*, созданную из столбцов в позициях n_1, n_2,…, n_k.

Здесь вы можете столкнуться со странной особенностью: вы можете получить вектор столбца или таблицу данных, в зависимости от того, сколько индексов используете и работаете ли вы с tibble или таблицей данных. Tibble всегда будут возвращать tibble во время индексации.

Однако функция `data.frame` может вернуть вектор, если вы используете индекс. В простом случае, когда используется один индекс, вы получите вектор:

```
# suburbs - это tibble, поэтому мы выполняем конвертацию для данного примера.
suburbs_df &lt;- as.data.frame(suburbs)
suburbs_df[, 1]
#> [1] "Chicago" "Kenosha" "Aurora"
```

Но при использовании того же синтаксиса с несколькими индексами мы получаем таблицу данных:

```
suburbs_df[, c(1, 4)]
#> city pop
#> 1 Chicago 2853114
#> 2 Kenosha 90352
#> 3 Aurora 171782
```

Это создает проблему. Предположим, в каком-то старом коде R вы видите это выражение:

```
df[, vec]
```

Скажите быстро, оно возвращает столбец или таблицу данных? Как посмотреть. Если `vec` содержит одно значение, вы получите столбец; в противном случае вы получите таблицу данных. Это нельзя определить по одному только синтаксису.

Чтобы избежать подобной проблемы, вы можете включить в индексы `drop=FALSE`, заставляя R возвращать таблицу данных:

```
df[, vec, drop = FALSE]
```

Теперь нет никакой неоднозначности в отношении возвращаемой структуры данных. Это таблица данных.

В конечном итоге использование матричной нотации для выбора столбцов из таблиц данных может оказаться непростым делом. Используйте функцию `select`, когда у вас есть такая возможность.

См. также

См. рецепт 5.17 для получения более подробной информации об использовании `drop=FALSE`.

5.22. Выбор столбцов таблицы данных по имени

Задача

Вы хотите выбрать столбцы из таблицы данных в соответствии с их именем.

Решение

Используйте функцию `select` и присвойте ей имена столбцов.

```
df %>% select(name1, name2, ..., namek)
```

Обсуждение

У всех столбцов в таблице данных должны быть имена. Если вы знаете имя, обычно удобнее и читабельнее осуществлять выбор по имени, а не по позиции. Обратите внимание, что при использовании функции `select` имена столбцов не берутся в кавычки.

Решения, описанные здесь, аналогичны решениям, приведенным в рецепте 5.21, где мы выбирали столбцы по позиции. Разница состоит лишь в том, что здесь мы используем имена столбцов вместо их номеров. Все замечания, сделанные в том рецепте, применимы и здесь.

Выражения списка

Функция `select` входит в состав пакета dplyr. В базовой версии R также имеется ряд богатых методов для выбора столбцов за счет дополнительного синтаксиса.

Чтобы выбрать один столбец, используйте один из этих операторов списка. Обратите внимание, что они используют *двойные* скобки ([[и]]):

```
df[["name"]]
```

Возвращает один столбец, столбец с именем `name`.

```
df$name
```

То же, что и предыдущее, только другой синтаксис.

Чтобы выбрать один или несколько столбцов, используйте эти операторы списка. Обратите внимание, что они используют одинарные скобки ([и]):

```
df["name"]
```

Выбирает один столбец из таблицы данных.

```
df[c("name1", "name2", ..., "namek")]
```

Выбирает несколько столбцов.

Матричный стиль индексации

Базовая версия R также позволяет использовать матричную индексацию для выбора одного или нескольких столбцов из таблицы данных по имени:

```
df[, "name"]
```

Возвращает именованный столбец.

```
df[, c("name1", "name2", ..., "namek")]
```

Выбирает несколько столбцов в таблице данных.

При таком типе индексации может быть возвращен либо столбец, либо таблица данных, поэтому будьте осторожны с тем, сколько имен вы указали. *См.* комментарии в рецепте 5.21, где обсуждается эта «ловушка» и использование `drop=FALSE`.

См. также

См. рецепт 5.21, чтобы узнать, как осуществлять выбор по позиции, а не по имени.

5.23. Изменение имен столбцов таблицы данных

Задача

Вы хотите изменить имена столбцов таблицы данных.

Решение

Функция `rename` из пакета `dplyr` делает процесс переименования довольно простым:

```
df %>% rename(newname1 = oldname1, ... , newnamen = oldnamen
```

где `df` – это таблица данных, $\texttt{oldname}_i$ – это имена столбцов в `df`, а $\texttt{newname}_i$ – необходимые новые имена.

Обратите внимание на порядок аргументов: `newname = oldname`.

Обсуждение

У столбцов таблиц данных должны быть имена. Их можно изменить с помощью функции `rename`:

```
df <- data.frame(V1 = 1:3, V2 = 4:6, V3 = 7:9)
df %>% rename(tom = V1, dick = V2)
#> tom dick V3
#> 1 1 4 7
#> 2 2 5 8
#> 3 3 6 9
```

Имена столбцов хранятся в атрибуте с именем `colnames`, поэтому еще один способ переименовать столбцы – изменить этот атрибут:

```
colnames(df) <- c("tom", "dick", "V2")
df
#> tom dick V2
#> 1 1 4 7
#> 2 2 5 8
#> 3 3 6 9
```

Если вы используете функцию `select` для выбора отдельных столбцов, можете переименовать эти столбцы одновременно:

```
df <- data.frame(V1 = 1:3, V2 = 4:6, V3 = 7:9)
df %>% select(tom = V1, V2)
#> tom V2
#> 1 1 4
#> 2 2 5
#> 3 3 6
```

Разница между переименованием с использованием функции `select` и переименованием с использованием функции `rename` состоит в том, что функция `rename` переименует то, что вы укажете, оставив все остальные столбцы без изменений, в то время как `select` оставляет только выбранные вами столбцы. В предыдущем примере `V3` отбрасывается, потому что его нет в операторе `select`. И та, и другая функции используют один и тот же порядок аргументов: `новоеимя = староеимя`.

См. также

См. рецепт 5.29.

5.24. Удаление значений NA из таблицы данных

Задача

Ваша таблица данных содержит значения NA, что создает для вас проблемы.

Решение

Используйте функцию na.omit для удаления строк, содержащих любые значения NA:

```
clean_dfrm <- na.omit(dfrm)
```

Обсуждение

Мы часто сталкиваемся с ситуациями, когда всего несколько значений NA в таблице данных приводят к тому, что все выходит из строя. Одно из решений – просто удалить все строки, которые содержат какие-либо значения NA. Именно это и делает функция na.omit.

Рассмотрим таблицу данных со встроенными значениями NA:

```
df <- data.frame(
x = c(1, NA, 3, 4, 5),
y = c(1, 2, NA, 4, 5)
)
df
#> x y
#> 1 1 1
#> 2 NA 2
#> 3 3 NA
#> 4 4 4
#> 5 5 5
```

Функция cumsum должна вычислять кумулятивные суммы, но наталкивается на значения NA:

```
colSums(df)
#> x y
#> NA NA
```

Если мы удалим строки со значениями NA, cumsum может завершить суммирование:

```
cumsum(na.omit(df))
#> x y
#> 1 1 1
#> 4 5 5
#> 5 10 10
```

Но будьте осторожны! Функция na.omit удаляет строки *целиком*. Значения, не являющиеся NA, в этих строках также исчезают, изменяя значение «кумулятивной суммы».

Этот рецепт подходит и для удаления значений NA из векторов и матриц, но не из списков.

Очевидная опасность здесь заключается в том, что простое удаление наблюдений из ваших данных может сделать результаты численно или статистически бессмысленными. Убедитесь, что пропущенные данные имеют смысл в вашем контексте. Помните, что функция `na.omit` удалит строки целиком, а не только значения `NA`, в результате чего будет удалена полезная информация.

5.25. Исключение столбцов по имени

Задача

Вы хотите исключить столбец из таблицы данных, используя его имя.

Решение

Используйте функцию `select` из пакета `dplyr` с дефисом (знаком минус) перед именем столбца, чтобы исключить его:

```
select(df, -bad) # Выбираем все столбцы из df, кроме плохих.
```

Обсуждение

Поместив знак минус перед именем переменной, мы даем функции `select` указание удалить эту переменную.

Это может пригодиться, когда мы вычисляем матрицу корреляции из таблицы данных и хотим исключить столбцы вне данных, например метки:

```
cor(patient_data)
#> patient_id pre dosage post
#> patient_id 1.0000 0.159 -0.0486 0.391
#> pre 0.1590 1.000 0.8104 -0.289
#> dosage -0.0486 0.810 1.0000 -0.526
#> post 0.3912 -0.289 -0.5262 1.000
```

Эта матрица корреляции включает в себя бессмысленную «корреляцию» между `patient_id` и другими переменными, что раздражает. Можно исключить столбец `patient_id`, чтобы очистить вывод:

```
patient_data %>%
select(-patient_id) %>%
cor
#> pre dosage post
#> pre 1.000 0.810 -0.289
#> dosage 0.810 1.000 -0.526
#> post -0.289 -0.526 1.000
```

Также можно исключить несколько столбцов:

```
patient_data %>%
select(-patient_id, -dosage) %>%
cor()
#> pre post
#> pre 1.000 -0.289
#> post -0.289 1.000
```

5.26. Объединение двух таблиц данных

Задача

Вы хотите объединить содержимое двух таблиц данных в одну.

Решение

Чтобы объединить столбцы двух таблиц данных бок о бок, используйте функцию `cbind` (связывание столбцов):

```
all.cols <- cbind(df1, df2)
```

Чтобы «сложить» строки двух таблиц данных, используйте функцию `rbind` (связывание строк):

```
all.rows <- rbind(df1, df2)
```

Обсуждение

Таблицы данных можно объединить одним из двух способов: либо расположив столбцы рядом, чтобы создать более широкую таблицу данных, либо «сложив» строки, чтобы создать более высокую таблицу.

Функция `cbind` будет объединять таблицы данных бок о бок:

```
df1 <- data.frame(a = c(1,2))
df2 <- data.frame(b = c(7,8))

cbind(df1, df2)
#> a b
#> 1 1 7
#> 2 2 8
```

Обычно объединяются столбцы с одинаковой высотой (числом строк). Однако с технической точки зрения функция `cbind` не требует соответствия высот. Если одна из таблиц данных короткая, R вызовет правило повторного использования, чтобы при необходимости расширить короткие столбцы (*см.* рецепт 5.3). Это, возможно, то, что вам нужно, а может быть, и нет.

Функция `rbind` «сложит» строки двух таблиц данных:

```
df1 <- data.frame(x = c("a", "a"), y = c(5, 6))
df2 <- data.frame(x = c("b", "b"), y = c(9, 10))
rbind(df1, df2)
#> x y
#> 1 a 5
#> 2 a 6
#> 3 b 9
#> 4 b 10
```

Функция `rbind` требует, чтобы таблицы данных имели одинаковую ширину – одинаковое количество столбцов и одинаковые имена столбцов. Однако столбцы не обязательно должны идти в том же *порядке*; `rbind` уладит это.

Наконец, этот рецепт немного более общий, чем предполагает название. Во-первых, вы можете объединить более двух таблиц данных, потому что функции `rbind` и `cbind` принимают несколько аргументов. Во-вторых, этот рецепт можно

применять и к другим типам данных, поскольку эти функции работают также с векторами, списками и матрицами.

5.27. Объединение таблиц данных по общему столбцу

Задача

У вас есть две таблицы данных, у которых есть общий столбец. Вы хотите объединить или соединить их строки в одну таблицу данных путем сопоставления в общем столбце.

Решение

Мы можем использовать функции `join` из пакета `dplyr`, чтобы объединить наши таблицы данных в общем столбце. Если вам нужны только строки, которые присутствуют в *обеих* таблицах, используйте функцию `inner_join`:

```
inner_join(df1, df2, by = "col")
```

где `"col"` – столбец, присутствующий в обеих таблицах данных.

Если вам нужны все строки, присутствующие в *любой* из таблиц, используйте функцию `full_join`:

```
full_join(df1, df2, by = "col")
```

Если вам нужны все строки из `df1` и только те строки из `df2`, что совпадают, используйте функцию `left_join`:

```
left_join(df1, df2, by = "col")
```

Или, чтобы получить все записи из `df2` и только совпадающие записи из `df1`, используйте функцию `right_join`:

```
right_join(df1, df2, by = "col")
```

Обсуждение

Предположим, у нас есть две таблицы данных, `born` и `died`, каждая из которых содержит столбец `name`:

```
born <- tibble(
name = c("Moe", "Larry", "Curly", "Harry"),
year.born = c(1887, 1902, 1903, 1964),
place.born = c("Bensonhurst", "Philadelphia", "Brooklyn", "Moscow")
)

died <- tibble(
name = c("Curly", "Moe", "Larry"),
year.died = c(1952, 1975, 1975)
)
```

Мы можем объединить их в одну таблицу данных, используя столбец `name` для объединения совпадающих строк:

```
inner_join(born, died, by="name")
#> # Tibble: 3x4
```

```
#> name year.born place.born year.died
#> <chr> <dbl> <chr> <dbl>
#> 1 Moe 1887 Bensonhurst 1975
#> 2 Larry 1902 Philadelphia 1975
#> 3 Curly 1903 Brooklyn 1952
```

Обратите внимание, что функция `inner_join` не требует сортировки строк или их размещения в одном и том же порядке. Она нашла совпадающие строки для `Curly`, хотя они и находятся в разных местах, а также отбросила строку для `Harry`, который присутствует только в таблице `born`.

Приведенный ниже код с использованием функции `full_join` включает в себя все строки, даже строки без совпадающих значений:

```
full_join(born, died, by="name")
#> # Tibble: 4×4
#> name year.born place.born year.died
#> <chr> <dbl><chr> <dbl>
#> 1 Moe 1887 Bensonhurst 1975
#> 2 Larry 1902 Philadelphia 1975
#> 3 Curly 1903 Brooklyn 1952
#> 4 Harry 1964 Moscow NA
```

Если у таблицы данных нет совпадающего значения, его столбцы заполняются значением `NA`: в случае с `Harry` `year.died` – `NA`.

Если мы не предоставим функции `join` поле для присоединения, она попытается соединиться любым полем с совпадающими именами в обеих таблицах данных и вернет информационный ответ, в котором указано, к какому полю она присоединяется:

```
full_join(born, died)
#> Joining, by = "name"
#> # Tibble: 4×4
#> name year.born place.born year.died
#> <chr> <dbl><chr> <dbl>
#> 1 Moe 1887 Bensonhurst 1975
#> 2 Larry 1902 Philadelphia 1975
#> 3 Curly 1903 Brooklyn 1952
#> 4 Harry 1964 Moscow NA
```

Если мы хотим объединить две таблицы данных в поле, у которого нет одинакового имени в обеих таблицах данных, нам нужно, чтобы наш параметр `by` был вектором равенства:

```
df1 <- data.frame(key1 = 1:3, value=2)
df2 <- data.frame(key2 = 1:3, value=3)
inner_join(df1, df2, by = c("key1" = "key2"))
#> key1 value.x value.y
#> 1 1 2 3
#> 2 2 2 3
#> 3 3 2 3
```

Обратите внимание, что в предыдущем примере в обеих таблицах есть поле с именем `value`, которое переименовывается в выводе. Поле из первой таблицы становится `value.x`, а поле из второй таблицы становится `value.y`. Семейство функций `join` из `dplyr` всегда будет переименовывать результат таким образом при наличии конфликта имен, когда столбцы не объединяются.

См. также

См. рецепт 5.26, чтобы узнать о других способах соединения таблиц данных.

В этом примере использовался один столбец `name`, но данные функции также могут работать и с несколькими столбцами. Для получения дополнительной информации *см.* документацию по функциям, набрав `? dplyr::join`.

Эти операции объединения были основаны на SQL. Как и в SQL, в `dplyr` существует несколько типов соединений, в том числе внутреннее объединение, левое, правое, полное, полуобъединение и антиобъединение. Для получения исчерпывающей информации обратитесь к документации по этим функциям.

5.28. Приведение атомарных типов данных

Задача

У вас есть значение данных, которое имеет атомарный тип данных: символьный, комплексный, двойной, целочисленный или логический. Вы хотите преобразовать это значение в один из других атомарных типов данных.

Решение

Для каждого атомарного типа данных есть функция для преобразования значений в этот тип. Функции приведения атомарных типов данных включают в себя:

- `as.character(x)`
- `as.complex(x)`
- `as.numeric(x)` или `as.double(x)`
- `as.integer(x)`
- `as.logical(x)`

Обсуждение

Процесс приведения атомарных типов данных обычно довольно прост. Если это сработает, вы получите то, что ожидаете. Если нет, вы получите `NA`:

```
as.numeric(" 3.14 ")
#> [1] 3.14
as.integer(3.14)
#> [1] 3
as.numeric("foo")
#> Warning: NAs introduced by coercion
#> [1] NA
as.character(101)
#> [1] "101"
```

Если у вас есть вектор атомарных типов, эти функции применяются к каждому значению.

Таким образом, предыдущие примеры преобразования скаляров легко распространяются и на преобразование целых векторов:

```
as.numeric(c("1", "2.718", "7.389", "20.086"))
#> [1] 1.00 2.72 7.39 20.09
as.numeric(c("1", "2.718", "7.389", "20.086", "etc."))
```

```
#> Warning: NAs introduced by coercion
#> [1] 1.00 2.72 7.39 20.09 NA
as.character(101:105)
#> [1] "101" "102" "103" "104" "105"
```

При преобразовании логических значений в числовые R преобразует FALSE в 0, а TRUE в 1:

```
as.numeric(FALSE)
#> [1] 0
as.numeric(TRUE)
#> [1] 1
```

Такое поведение полезно, когда вы подсчитываете вхождения TRUE в векторах логических значений. Если logvec – вектор логических значений, то sum(logvec) выполняет неявное преобразование из логических в целочисленные значения и возвращает число значений TRUE:

```
logvec <- c(TRUE, FALSE, TRUE, TRUE, TRUE, FALSE)
sum(logvec) ## Число значений TRUE.
#> [1] 4
length(logvec) - sum(logvec) ## Число значений, не являющихся TRUE.
#> [1] 2
```

5.29. Приведение структурированных типов данных

Задача

Вы хотите преобразовать переменную из одного структурированного типа данных в другой, например преобразовать вектор в список или матрицу в таблицу данных.

Решение

Эти функции преобразуют свои аргументы в соответствующий структурированный тип данных:

- `as.data.frame(x)`
- `as.list(x)`
- `as.matrix(x)`
- `as.vector(x)`

Однако некоторые из этих типов преобразований могут вас удивить. Мы предлагаем вам просмотреть табл. 5-2 для получения более подробной информации.

Обсуждение

Преобразование из одного структурированного типа данных в другой может быть сложным. Некоторые преобразования ведут себя так, как вы и ожидаете. Например, если вы преобразуете матрицу в таблицу данных, строки и столбцы матрицы становятся строками и столбцами таблицы данных. Нет проблем.

В других случаях результаты могут вас удивить. В табл. 5-2 приведено несколько заслуживающих внимания примеров.

Таблица 5-2. Преобразования данных

Тип преобразования	Способ выполнения	Примечания
Вектор → список	`as.list(vec)`	Не используйте `list(vec)` – у вас получится список, единственным элементом которого является копия `vec`
Вектор → матрица	Чтобы создать матрицу из одного столбца: `cbind(vec)` или `as.matrix(vec)` Чтобы создать однострочную матрицу: `rbind(vec)` Чтобы создать матрицу n×m: `matrix(vec,n,m)`	См. рецепт 5.14
Вектор → таблица данных	Чтобы создать матрицу из одного столбца: `as.data.frame(vec)` Чтобы создать однострочную матрицу: `as.data.frame(rbind(vec))`	
Список → вектор	`unlist(lst)`	Используйте функцию `unlist` вместо `as.vector`; см. примечание 1 и рецепт 5.11
Список → матрица	Чтобы создать матрицу из одного столбца: `as.matrix(lst)` Чтобы создать однострочную матрицу: `as.matrix(rbind(lst))` Чтобы создать матрицу n×m: `matrix(lst,n,m)`	
Список → таблица данных	Если элементы списка – столбцы данных: `as.data.frame(lts)` Если элементы списка – строки данных, см. рецепт 5.19	
Матрица → вектор	`as.vector(mat)`	Возвращает все элементы матрицы в векторе
Матрица → список	`as.list(mat)`	Возвращает все элементы матрицы в списке
Матрица → таблица данных	`as.data.frame(mat)`	
Таблица данных → вектор	Чтобы преобразовать однострочную таблицу данных: `df[1,]` Чтобы преобразовать таблицу данных, состоящую из одного столбца: `df[, 1]` или `df[[1]]`	См. примечание 2
Таблица данных → список	`as.list(df)`	См. примечание 3
Таблица данных → матрица	`as.matrix(df)`	См. примечание 4

В таблице приводятся следующие примечания.

1. Когда вы преобразуете список в вектор, преобразование работает корректно, если ваш список содержит атомарные значения, относящиеся к одному и тому же режиму. Ситуация усложняется, если ваш список содержит смешанные режимы (например, числовой и символьный). В этом случае все преобразуется в символы, или ваш список содержит другие структурированные типы данных (например, подсписки или таблицы данных). Тогда происходят очень странные вещи, поэтому не делайте этого.
2. Преобразование таблицы данных в вектор имеет смысл, только если таблица данных содержит одну строку или один столбец. Чтобы извлечь все его элементы в один длинный вектор, используйте `as.vector(as.matrix(df))`. Но даже это имеет смысл, только если таблица данных полностью числовая или полностью символьная; в противном случае сначала все преобразуется в символьные строки.
3. Преобразование таблицы данных в список может показаться странным, поскольку таблица данных уже является списком (т. е. списком столбцов). Использование `as.list` по существу удаляет класс (`data.frame`) и, таким образом, предоставляет базовый список. Это полезно, когда вы хотите, чтобы R рассматривал вашу структуру данных как список, скажем, для вывода на экран.
4. Будьте осторожны при преобразовании таблицы данных в матрицу. Если таблица данных содержит только числовые значения, вы получите числовую матрицу. Если она содержит только символьные значения, вы получите символьную матрицу. Но если таблица данных представляет собой сочетание чисел, символов и/или факторов, все значения сначала будут преобразованы в символы. В результате получится матрица символьных строк.

Особые соображения касательно матриц

Описанные здесь преобразования матриц предполагают, что ваша матрица однородна, то есть все элементы имеют одинаковый режим (например, все они числовые или все символы). Матрица также может быть неоднородной, если создается из списка. В этом случае результат преобразования может быть непредсказуемым. Например, при преобразовании матрицы смешанного режима в таблицу данных столбцы таблицы данных фактически являются списками (для размещения смешанных данных).

См. также

См. рецепт 5.28 для получения информации о преобразовании атомарных типов данных; *см.* введение к этой главе, чтобы ознакомиться с замечаниями касательно проблемных преобразований.

Глава 6

Преобразование данных

В то время как традиционные языки программирования используют циклы, R традиционно рекомендует применять векторизованные операции и семейство функций `apply` для обработки данных в пакетном режиме, что значительно упрощает вычисления. Ничто не мешает вам писать циклы в R, которые разбивают ваши данные на нужные вам фрагменты, а затем выполнять операции с каждым фрагментом. Однако использование векторизованных функций во многих случаях может повысить скорость, читабельность и удобство сопровождения вашего кода.

Тем не менее недавно в рамках коллекции tidyverse, в частности в пакетах `purrr` и `dplyr`, добавили в R новые идиомы, которые облегчают изучение этих концепций и делают их несколько более последовательными. Название `purrr` происходит от фразы «чистое P» (Pure R).

Это функция, результат которой определяется только ее входными данными и которая не вызывает побочных эффектов. Это не концепция функционального программирования, которую по-хорошему вам следует понимать, чтобы получить большую выгоду от использования `purrr`. Все, что нужно знать большинству пользователей, – это то, что `purrr` содержит функции, которые помогают обрабатывать наши данные «фрагмент за фрагментом» таким образом, чтобы это хорошо сочеталось с другими пакетами tidyverse, такими как `dplyr`.

В базовой версии R имеется множество функций `apply` – `apply`, `lapply`, `sapply`, `tapply` и `mapply`, а также их двоюродные сестры, `by` и `split`. Это надежные функции, которые в течение многих лет были рабочими лошадками в базовой версии R. Мы сомневались относительно того, сколько внимания уделить функциям `apply` в базовой версии R и в какой степени фокусироваться на более новом «аккуратном» подходе. После множества обсуждений мы решили попытаться проиллюстрировать подход с использованием `purrr` и продемонстрировать подходы, применяемые в базовой версии R и, где-то, показать и то, и то другое. Интерфейс `purrr` и `dplyr` очень понятный и, как мы полагаем, в большинстве случаев более интуитивный.

6.1. Применение функции ко всем элементам списка

Задача

У вас есть список, и вы хотите применить функцию к каждому элементу списка.

Решение

Используйте функцию `map`, чтобы применить функцию к каждому элементу списка:

```
library(tidyverse)

lst %>%
map(fun)
```

Обсуждение

Давайте рассмотрим конкретный пример, где берется среднее значение всех чисел в каждом элементе списка:

```
library(tidyverse)

lst <- list(
a = c(1,2,3),
b = c(4,5,6)
)
lst %>%
map(mean)
#> $a
#> [1] 2
#>
#> $b
#> [1] 5
```

Функция `map` будет вызывать вашу функцию один раз для каждого элемента в вашем списке. Ваша функция должна ожидать один аргумент, элемент из списка. Функции `map` будут собирать возвращаемые значения и возвращать в списке.

Пакет `purrr` содержит целое семейство функций `map`, которые принимают список или вектор, а затем возвращают объект с тем же числом элементов, что и ввод. Тип возвращаемого ими объекта зависит от того, какая функция `map` используется. *См.* файл справки по функциям `map`, чтобы увидеть полный список, а ниже приводится несколько наиболее распространенных из них:

`map`
: Всегда возвращает список, и элементы списка могут быть разных типов. Очень напоминает функцию `lapply` из базовой версии R.

`map_chr`
: Возвращает символьный вектор.

`map_int`
: Возвращает целочисленный вектор.

`map_dbl`
: Возвращает числовой вектор с плавающей точкой.

Давайте кратко рассмотрим вымышленную ситуацию, когда у нас есть функция, которая может дать результат в виде символа или целого числа:

```
fun <- function(x) {
if (x > 1) {
1
} else {
"Less Than 1"
}
}
```

```
fun(5)
#> [1] 1
fun(0.5)
#> [1] "Less Than 1"
```

Давайте создадим список элементов, к которым можно применить `map(lst, fun)`, и посмотрим, как ведут себя некоторые варианты функции `map`:

```
lst <- list(.5, 1.5, .9, 2)

map(lst, fun)
#> [[1]]
#> [1] "Less Than 1"
#>
#> [[2]]
#> [1] 1
#>
#> [[3]]
#> [1] "Less Than 1"
#>
#> [[4]]
#> [1] 1
```

Видно, что функция `map` создала список, и это смешанные типы данных.
Функция `map_chr` создаст вектор символов и преобразует числа в символы:

```
map_chr(lst, fun)
#> [1] "Less Than 1" "1.000000" "Less Than 1" "1.000000"

## или используя конвейеры
lst %>%
map_chr(fun)
#> [1] "Less Than 1" "1.000000" "Less Than 1" "1.000000"
```

в то время как `map_dbl` будет пытаться преобразовать строку символов в числовой вектор с плавающей точкой и у нее ничего не выйдет:

```
map_dbl(lst, fun)
#> Error: Can't coerce element 1 from a character to a double
```

Как упоминалось ранее, функция `lapply` действует очень похоже на `map`. Функция `sapply` больше похожа на другие функции `map`, которые мы обсуждали ранее, в том смысле, что она пытается упростить результаты до вектора или матрицы.

См. также

См. рецепт 15.3.

6.2. Применение функции к каждой строке таблицы данных

Задача

У вас есть функция, и вы хотите применить ее к каждой строке в таблице данных.

Решение

Функция `mutate` создаст новую переменную на базе вектора значений. Но если мы используем функцию, которая не может принять вектор и вывести вектор, мы должны выполнить построчную операцию, применяя функцию `rowwise`.

Мы можем использовать эту функцию в конвейере, чтобы дать `dplyr` указание выполнять все последующие команды построчно:

```
df %>%
rowwise() %>%
row_by_row_function()
```

Обсуждение

Давайте создадим функцию и применим ее построчно к таблице данных. Наша функция просто вычислит сумму последовательности от `a` к `b` по `c`:

```
fun <- function(a, b, c) {
sum(seq(a, b, c))
}
```

Давайте создадим данные, чтобы применить эту функцию, а затем используем функцию `rowwise`, чтобы применить к ней нашу функцию `fun`:

```
df <- data.frame(mn = c(1, 2, 3),
mx = c(8, 13, 18),
rng = c(1, 2, 3))

df %>%
rowwise %>%
 mutate(output = fun(a = mn, b = mx, c = rng))
#> Source: local data frame [3 x 4]
#> Groups: <by row>
#>
#> # Tibble: 3×4
#> mn mx rng output
#> <dbl> <dbl> <dbl> <dbl>
#> 1 1 8 1 36
#> 2 2 13 2 42
#> 3 3 18 3 63
```

Если бы мы попытались запустить эту функцию без `rowwise`, она бы выбросила ошибку, потому что функция `seq` не может обработать весь вектор:

```
df %>%
 mutate(output = fun(a = mn, b = mx, c = rng))
#> Error in seq.default(a, b, c): 'from' must be of length 1
```

6.3. Применение функции к каждой строке матрицы

Задача

У вас есть матрица. Вы хотите применить функцию к каждой строке, вычисляя результат функции для каждой строки.

Решение

Используйте функцию apply. Установите для второго аргумента значение 1, чтобы указать на построчное применение функции:

```
results <- apply(mat, 1, fun) # mat - это матрица, fun - это функция.
```

Функция apply будет вызывать fun один раз для каждой строки матрицы, собирать возвращаемые значения в вектор и затем возвращать этот вектор.

Обсуждение

Вы, возможно, заметили, что здесь мы показываем только использование функции apply из базовой версии R, в то время как другие рецепты иллюстрируют альтернативы purrr. На момент написания этих строк операции с матрицами выходили за рамки purrr, поэтому мы используем очень надежную функцию apply. Если вам действительно нравится синтаксис apply, вы можете использовать эти функции, если сначала преобразуете свою матрицу в таблицу данных или tibble. Но если у вас большая матрица, вы заметите значительное замедление во время выполнения, используя purrr.

Предположим, у нас фрагмент кода long <- matrix, содержащий продольные данные, поэтому в каждой строке есть данные для одного субъекта, а столбцы содержат повторяющиеся наблюдения с течением времени:

```
long <- matrix(1:15, 3, 5)
long
#> [,1] [,2] [,3] [,4] [,5]
#> [1,] 1 4 7 10 13
#> [2,] 2 5 8 11 14
#> [3,] 3 6 9 12 15
```

Мы могли бы рассчитать среднее наблюдение для каждого субъекта, применив функцию mean к каждой строке. В результате получится вектор:

```
apply(long, 1, mean)
#> [1] 7 8 9
```

Если в нашей матрице есть имена строк, функция apply использует их для идентификации элементов результирующего вектора, что удобно:

```
rownames(long) <- c("Moe", "Larry", "Curly")
apply(long, 1, mean)
#> Moe Larry Curly
#> 7 8 9
```

Вызываемая функция должна ожидать один аргумент, вектор, который будет одной строкой из матрицы. Функция может возвращать скаляр или вектор. В случае с вектором apply собирает результаты в матрицу. Функция range возвращает вектор из двух элементов, минимума и максимума, поэтому если применить ее к long, получится матрица:

```
apply(long, 1, range)
#> Moe Larry Curly
#> [1,] 1 2 3
#> [2,] 13 14 15
```

Этот рецепт также можно использовать и с таблицами данных. Это сработает, если таблица данных однородна, т. е. все данные либо численные, либо символьные. Когда в таблице данных есть столбцы разных типов, извлечение векторов из строк не имеет смысла, поскольку векторы должны быть однородными.

6.4. Применение функции к каждому столбцу

Задача

У вас есть матрица или таблица данных, и вы хотите применить функцию к каждому столбцу.

Решение

В случае с матрицей используйте функцию `apply`. Установите для второго аргумента значение 2, которое указывает на применение функции «столбец за столбцом». Итак, если бы наша матрица или таблица данных называлась `mat` и мы бы хотели применить функцию с именем `fun` к каждому столбцу, это выглядело бы так:

```
apply(mat, 2, fun)
```

В случае с таблицей данных используйте функцию `map_df` из `purrr`:

```
df2 <- map_df(df, fun)
```

Обсуждение

Давайте рассмотрим пример с действительными числами и применим функцию `mean` к каждому столбцу матрицы:

```
mat <- matrix(c(1, 3, 2, 5, 4, 6), 2, 3)
colnames(mat) <- c("t1", "t2", "t3")
mat
#> t1 t2 t3
#> [1,] 1 2 4
#> [2,] 3 5 6

apply(mat, 2, mean) # Вычисляем среднее значение каждого столбца.
#> t1 t2 t3
#> 2.0 3.5 5.0
```

В базовой версии R функция `apply` предназначена для обработки матрицы или таблицы данных. Второй аргумент этой функции определяет направление:

- 1 – обработка строка за строкой;
- 2 – обработка столбец за столбцом.

Это выглядит более мнемонически, чем кажется. Мы говорим о матрицах, используя термины «строки и столбцы», поэтому строки идут первыми, а столбцы – вторыми: 1 и 2 соответственно.

Таблица данных – это более сложная структура данных, чем матрица, поэтому здесь вариантов больше. Можно просто использовать функцию `apply`, и в этом случае R преобразует вашу таблицу данных в матрицу и затем применит вашу функцию. Это сработает, если ваша таблица данных содержит только один тип данных,

но, вероятно, не будет делать то, что вы хотите, если какие-то столбцы являются числами, а какие-то из них символами. В этом случае R заставит все столбцы иметь одинаковые типы, что может привести к нежелательному преобразованию.

К счастью, есть несколько альтернатив. Напомним, что таблица данных – это своего рода список: это список столбцов таблицы данных. В `purrr` существует целое семейство функций `map`, которые возвращают различные типы объектов. Особый интерес здесь представляет функция `map_df`, которая возвращает `data.frame` (отсюда и `df` в имени):

```
df2 <- map_df(df, fun) # Возвращаем data.frame.
```

Функция `fun` должна ожидать один аргумент: столбец из таблицы данных.

Ниже приводится распространенный рецепт для проверки типов столбцов в таблицах данных. В этом примере, на первый взгляд, столбец `batch` этой таблицы данных содержит числа:

```
load("./data/batches.rdata")
head(batches)
#> batch clinic dosage shrinkage
#> 1 3 KY IL -0.307
#> 2 3 IL IL -1.781
#> 3 1 KY IL -0.172
#> 4 3 KY IL 1.215
#> 5 2 IL IL 1.895
#> 6 2 NJ IL -0.430
```

Но использование функции `map_df` для вывода класса каждого столбца показывает, что столбец `batch` – это фактор:

```
map_df(batches, class)
#> # Tibble: 1×4
#> batch clinic dosage shrinkage
#> <chr> <chr> <chr> <chr>
#> 1 factor factor factor numeric
```

Обратите внимание, что в третьей строке вывода все время повторяется <chr>. Это связано с тем, что вывод `class` помещается в таблицу данных и затем выводится. Промежуточная таблица данных – это все символьные поля. Именно последняя строка сообщает нам, что у нашей исходной таблицы данных есть три столбца *factor* и одно поле *numeric*.

См. также

См. рецепты 5.21, 6.1 и 6.3.

6.5. Применение скалярной функции к векторам или спискам

Задача

У вас есть функция, которая принимает несколько аргументов. Вы хотите применить эту функцию поэлементно к векторам и получить векторный результат.

К сожалению, функция не векторизована; то есть она работает со скалярами, но не с векторами.

Решение

Используйте одну из функций – `map` или `pmap` – из пакета `purrr`. Самое обычное решение – поместить ваши векторы в список, а затем использовать функцию `pmap`:

```
lst <- list(v1, v2, v3)
pmap(lst, fun)
```

`pmap` возьмет элементы `lst` и передаст их в качестве входных данных в `fun`.

Если у вас есть только два вектора, которые вы передаете в качестве входных данных в вашу функцию, удобно будет использовать семейство функций `map2`. Это избавляет вас от необходимости сначала помещать свои векторы в список. `map2` вернет список:

```
map2(v1, v2, fun)
```

в то время как типизированные варианты (`map2_chr`, `map2_dbl` и т. д.) возвращают векторы того типа, который подразумевает их имя. Итак, если `fun` возвращает только числовой вектор с плавающей точкой, используйте вместо этого типизированный вариант `map2`:

```
map2_dbl(v1, v2, fun)
```

Типизированные варианты в функциях `purrr` обращаются к типу *вывода*, ожидаемого от функции. Все типизированные варианты возвращают векторы соответствующего типа, в то время как нетипизированные варианты возвращают списки, которые позволяют смешивать типы.

Обсуждение

Основные операторы R, такие как `x + y`, векторизованы; это означает, что они вычисляют свой элемент результата по элементу и возвращают вектор результатов. Кроме того, многие функции R векторизованы.

Однако не все функции являются векторизованными, а те, что не типизированы, работают только со скалярами. Использование векторных аргументов в лучшем случае приводит к ошибкам, а в худшем – к бессмысленным результатам. В таких случаях функции `map` от `purrr` могут эффективно векторизовать функцию за вас.

Рассмотрим функцию `gcd` из рецепта 15.3, которая принимает два аргумента:

```
gcd <- function(a, b) {
if (b == 0) {
return(a)
} else {
return(gcd(b, a %% b))
}
}
```

Если мы применим ее к двум векторам, результатом будут неправильные ответы и куча сообщений об ошибках:

```
gcd(c(1, 2, 3), c(9, 6, 3))
#> Warning in if (b == 0) {: the condition has length > 1 and only the first
#> element will be used
#> Warning in if (b == 0) {: the condition has length > 1 and only the first
#> element will be used
#> Warning in if (b == 0) {: the condition has length > 1 and only the first
#> element will be used
#> [1] 1 2 0
```

Функция не векторизована, но мы можем использовать функцию `map`, чтобы «векторизовать» ее. В этом случае, поскольку у нас есть два входа, мы должны использовать функцию `map2`. Это дает поэлементные наибольшие общие делители (GCD) между двумя векторами:

```
a <- c(1, 2, 3)
b <- c(9, 6, 3)
my_gcds <- map2(a, b, gcd)
my_gcds
#> [[1]]
#> [1] 1
#>
#> [[2]]
#> [1] 2
#>
#> [[3]]
#> [1] 3
```

Обратите внимание, что `map2` возвращает список списков. Если бы мы хотели получить вывод в векторе, то могли бы использовать функцию `unlist`:

```
unlist(my_gcds)
#> [1] 1 2 3
```

или один из типизированных вариантов, например `map2_dbl`.

Семейство функций `map` предоставляет вам серию вариантов, которые возвращают определенные типы вывода. Суффиксы в именах функций сообщают тип вектора, который они будут возвращать. В то время как `map` и `map2` возвращают списки, поскольку специфичные для типа варианты возвращают объекты, которые гарантированно имеют одинаковый тип, их можно поместить в атомарные векторы. Например, мы могли бы использовать функцию `map_chr`, чтобы попросить R преобразовать результаты в вывод символа, или `map2_dbl`, дабы убедиться, что результаты – это числовые векторы с плавающей точкой:

```
map2_chr(a, b, gcd)
#> [1] "1.000000" "2.000000" "3.000000"
map2_dbl(a, b, gcd)
#> [1] 1 2 3
```

Если в наших данных более двух векторов или эти данные уже находятся в списке, мы можем использовать семейство функций `pmap`, которые принимают список в качестве ввода:

```
lst <- list(a,b)
pmap(lst, gcd)
```

```
#> [[1]]
#> [1] 1
#>
#> [[2]]
#> [1] 2
#>
#> [[3]]
#> [1] 3
```

Или если нам нужен типизированный вектор в качестве вывода:

```
lst <- list(a,b)
pmap_dbl(lst, gcd)
#> [1] 1 2 3
```

Работая с функциями purrr, помните, что семейство pmap – это параллельные преобразователи, которые принимают в качестве входных данных *список*, а функции map2 принимают два и только два *вектора* в качестве входных данных.

См. также

Это действительно особый случай нашего самого первого рецепта в данной главе, рецепта 6.1. *См.* этот рецепт, где варианты функций map обсуждаются более подробно. Кроме того, у Дженни Брайан есть потрясающая коллекция уроков по purrr на ее сайте в GitHub.

6.6. Применение функции к группам данных

Задача

Ваши элементы данных могут группироваться по какому-то признаку. Вы хотите обрабатывать данные по группам – например, выполняя суммирование или усреднение по группам.

Решение

Самый простой способ сделать это – использовать функцию group_by из пакета dplyr в сочетании с функцией summarize. Если наша таблица данных – это df, и у нее есть переменная, которую мы хотим сгруппировать по grouping_var, и мы хотим применить функцию fun ко всем комбинациям v1 и v2, это можно сделать с помощью функции group_by:

```
df %>%
group_by(v1, v2) %>%
summarize(
result_var = fun(value_var)
)
```

Обсуждение

Давайте рассмотрим конкретный пример, где наша таблица входных данных, df, содержит переменную my_group, по которой мы хотим сгруппировать данные, и поле с именем values, для которого мы хотели бы рассчитать статистику:

```
df <- tibble(
my_group = c("A", "B","A", "B","A", "B"),
```

```
values = 1:6
)

df %>%
group_by(my_group) %>%
summarize(
avg_values = mean(values),
tot_values = sum(values),
count_values = n()
)
#> # Tibble: 2×4
#>   my_group avg_values tot_values count_values
#>   <chr>       <dbl>      <int>        <int>
#> 1 A               3          9            3
#> 2 B               4         12            3
```

В выводе содержится по одной записи на группу наряду с вычисленными значениями для трех итоговых полей, которые мы определили.

Стоит отметить, что функция `summarize` бесшумно удалит последнюю из переменных группировки `group_by`. Это делается автоматически, потому что если оставить группировку на месте по определению, это будет означать, что ни в одной группе нет более чем одной строки. Последняя переменная группировки удаляется только из вектора группировки, а не из таблицы данных. Это поле все еще там, но оно не используется для группировки. Это может вас удивить, когда вы впервые видите это в действии.

6.7. Создание нового столбца по условию

Задача

Вы хотите создать новый столбец в таблице данных на базе некоего условия.

Решение

Используя пакет `dplyr`, можно создавать новые столбцы таблицы данных с помощью функции `mutate`, а затем использовать функцию `case_when` для реализации условной логики:

```
df %>%
mutate(
case_when(my_field == "something" ~ "result",
my_field != "something else" ~ "other result",
TRUE ~ "all other results")
)
```

Обсуждение

Функция `case_when` из пакета `dplyr` аналогична операторам `CASE WHEN` в SQL или вложенным операторам `IF` в Excel. Она проверяет каждый элемент и, когда находит истинное условие, возвращает значение по правую сторону от ~ (тильды).

Давайте рассмотрим пример, где нам нужно добавить текстовое поле, которое описывает значение. Сначала настроим несколько простых примеров данных в таблице данных с одним столбцом с именем `vals`:

```
df <- data.frame(vals = 1:5)
```

Теперь реализуем логику, которая создает поле с именем `new_vals`. Если `vals` меньше или равен 2, мы вернем `2 or less`; если значение больше 2 и меньше или равно 4, мы вернем `2 to 4`, а в противном случае вернем `over 4`:

```
df %>%
mutate(new_vals = case_when(vals <= 2 ~ "2 or less",
 vals > 2 & vals <= 4 ~ "2 to 4",
 TRUE ~ "over 4"))
#>   vals new_vals
#> 1    1 2 or less
#> 2    2 2 or less
#> 3    3 2 to 4
#> 4    4 2 to 4
#> 5    5 over 4
```

В этом примере видно, что условие идет слева от ~, а полученное возвращаемое значение – справа. Каждое условие разделяется запятыми. Функция `case_when` выполнит последовательную оценку каждого условия и прекратит оценку, как только один из критериев вернет значение `TRUE`. Наша последняя строка – это оператор `or else`. Установив для условия значение `TRUE`, мы гарантируем, что, несмотря ни на что, это условие будет выполнено, если ни одно из условий выше не вернуло это значение.

См. также

См. рецепт 6.2, где приводятся дополнительные примеры использования функции `mutate`.

Глава 7

Строки и даты

Строки? Даты? В пакете статистического программирования?

Когда вы читаете файлы или распечатываете отчеты, вам нужны строки. Когда вы работаете с проблемами реального мира, вам нужны даты.

В R есть возможности и для того, и для другого. Они неуклюжи по сравнению со строчно-ориентированными языками, такими как Perl, но это вопрос выбора правильного инструмента для работы. Нам бы не хотелось выполнять логистическую регрессию в Perl.

Некоторые из этих неудобств, связанные со строками и датами, были улучшены благодаря пакетам `stringr` и `lubridate` из коллекции tidyverse. Как и в других главах этой книги, приведенные здесь примеры будут взяты из базовой версии R, а также из дополнительных пакетов, которые делают жизнь проще, быстрее и удобнее.

Классы дат и времени

В R есть множество классов для работы с датами и временем, что хорошо, если вы предпочитаете иметь выбор, но раздражает, если вы предпочитаете просто жить. Существует важное различие между классами: некоторые из них предназначены только для дат, а другие – только для времени. Все классы могут обрабатывать календарные даты (например, 15 марта 2019 г.), но не все могут обозначать время (11:45 1 марта 2019 г.).

Приведенные ниже классы включены в базовый дистрибутив R:

`Date`

Класс `Date` может обозначать календарную дату, но не время. Это надежный универсальный класс для работы с датами, включая преобразования, форматирование, базовую арифметику дат и обработку часовых поясов. Большинство рецептов, связанных с датами в этой книге, основаны на классе `Date`.

`POSIXct`

Это временной класс. Он может обозначать момент времени с точностью до одной секунды. Внутри дата и время хранятся как количество секунд с 1 января 1970 года, поэтому это очень компактное представление. Этот класс рекомендуется для хранения информации о времени и дате (например, в таблицах данных).

`POSIXlt`

Это тоже временной класс, но представление хранится в списке из девяти элементов, который включает в себя год, месяц, день, час, минуту и секунду. Это

представление облегчает извлечение частей даты, таких как месяц или час. Очевидно, что он гораздо менее компактен, по сравнению с классом `POSIXct`; следовательно, он обычно используется для промежуточной обработки, а не для хранения данных.

Базовый дистрибутив также предоставляет функции для простого преобразования представлений: `as.Date`, `as.POSIXct` и `as.POSIXlt`.

Приведенные ниже полезные пакеты доступны для скачивания на сайте CRAN:

`chron`

Пакет `chron` может обозначать и дату, и время, но без дополнительных сложностей обработки часовых поясов и перехода на летнее время. Поэтому его проще использовать, чем `Date`, но он не такой мощный, как `POSIXct` и `POSIXlt`. Он мог бы быть полезен для работы в эконометрике или при анализе временных рядов.

`lubridate`

Это пакет tidyverse, разработанный для облегчения работы с датами и временем, сохраняя при этом важные «навороты», такие как часовые пояса. Он особенно подходит для арифметических действий с датами и временем. Этот пакет представляет некоторые полезные конструкции, такие как длительности, периоды и интервалы. `lubridate` является частью tidyverse, поэтому он устанавливается при выполнении `install.packages ('tidyverse')`, но не является частью `core tidyverse`, поэтому не загружается при запуске `library(tidyverse)`. Это означает, что вы должны явно загрузить его, выполнив `library(lubridate)`.

`mondate`

Это специализированный пакет для обработки дат в единицах месяцев в дополнение к дням и годам. Он может быть полезен в бухгалтерском учете и страховом деле, например когда требуются помесячные расчеты.

`timeDate`

Это мощный пакет с хорошо продуманными средствами обработки дат и времени, включая арифметические операции с датами, рабочие дни, праздники, преобразования и обобщенную обработку часовых поясов. Первоначально он был частью программного обеспечения Rmetrics для финансового моделирования, где точность дат и времени имеет решающее значение. Если вам необходимы средства для работы с данными, рассмотрите этот пакет.

Какой класс выбрать? В статье Габора Гротендика и Томаса Петцольдта «Классы дат и времени в R» (https://cran.r-project.org/doc/Rnews/Rnews_2004-1.pdf) предлагается вот такой общий совет:

> Когда вы рассматриваете, какой класс использовать, всегда выбирайте наименее сложный класс, который будет поддерживать приложение. То есть если это возможно, используйте класс `Date`, в противном случае применяйте `chron` либо классы `POSIX`. Такая стратегия значительно снизит вероятность ошибок и повысит надежность вашего приложения.

См. также

См. `help(DateTimeClasses)` для получения более подробной информации о встроенных средствах. *См.* статью Габора Гротендика и Томаса Петцольдта «Классы

дат и времени в R» (https://cran.r-project.org/doc/Rnews/Rnews_2004-1.pdf) за июнь 2004 года, где можно прекрасно познакомиться с возможностями для работы с датами и временем. В статье Брайана Рипли и Курта Хорника «Классы дат и времени» (https://cran.r-project.org/doc/Rnews/Rnews_2001-2.pdf), опубликованной в июне 2001 года, в частности, рассматриваются два класса POSIX. Глава 16 «Даты и время» из книги Гаррета Гроулмунда и Хэдли Уикхэма (издательство *O'Reilly*) *R for Data Science* (http://shop.oreilly.com/product/0636920034407.do) предлагает отличное введение в `lubridate`.

7.1. Получение длины строки

Задача

Вы хотите знать длину строки.

Решение

Используйте функцию `nchar` вместо функции `length`.

Обсуждение

Функция `nchar` берет строку и возвращает количество символов в строке:

```
nchar("Moe")
#> [1] 3
nchar("Curly")
#> [1] 5
```

Если применить эту функцию к вектору строк, он вернет длину каждой строки:

```
s <- c("Moe", "Larry", "Curly")
nchar(s)
#> [1] 3 5 5
```

Вы, наверное, думаете, что функция `length` возвращает длину строки. Нет. Она возвращает длину вектора. Когда вы применяете функцию `length` к одной строке, R возвращает значение 1, потому что она рассматривает эту строку как синглтонный вектор – вектор с одним элементом:

```
length("Moe")
#> [1] 1
length(c("Moe", "Larry", "Curly"))
#> [1] 3
```

7.2. Конкатенация строк

Задача

Вы хотите объединить две или более строк в одну.

Решение

Используйте функцию `paste`.

Обсуждение

Функция `paste` объединяет несколько строк воедино. Другими словами, она создает новую строку, соединяя указанные строки непрерывной цепью:

```
paste("Everybody", "loves", "stats.")
#> [1] "Everybody loves stats."
```

По умолчанию функция `paste` вставляет один пробел между парами строк, что удобно, если вы этого хотите. В противном случае это раздражает. Аргумент `sep` позволяет указать другой разделитель. Используйте пустую строку («»), чтобы запустить строки вместе без разделения:

```
paste("Everybody", "loves", "stats.", sep = "-")
#> [1] "Everybody-loves-stats."
paste("Everybody", "loves", "stats.", sep = "")
#> [1] "Everybodylovesstats."
```

Общепринято желание объединять строки вообще без разделителя. Функция `paste0` делает этот процесс очень удобным:

```
paste0("Everybody", "loves", "stats.")
#> [1] "Everybodylovesstats."
```

Эта функция довольно снисходительна в отношении нестроковых аргументов. Она пытается преобразовать их в строки, используя за кулисами функцию `as.character`:

```
paste("The square root of twice pi is approximately", sqrt(2 * pi))
#> [1] "The square root of twice pi is approximately 2.506628274631"
```

Если один или несколько аргументов являются векторами строк, функция `paste` сгенерирует все комбинации аргументов (из-за повторного использования):

```
stooges <- c("Moe", "Larry", "Curly")
paste(stooges, "loves", "stats.")
#> [1] "Moe loves stats." "Larry loves stats." "Curly loves stats."
```

Иногда вам нужно объединить даже эти комбинации в одну большую строку. Параметр `collapse` позволяет определить разделитель верхнего уровня и дает функции `paste` команду объединить созданные строки с использованием этого разделителя:

```
paste(stooges, "loves", "stats", collapse = ", and ")
#> [1] "Moe loves stats, and Larry loves stats, and Curly loves stats"
```

7.3. Извлечение подстрок

Задача

Вам нужно извлечь часть строки в соответствии с расположением.

Решение

Используйте `substr(string,start,end)` для извлечения подстроки, которая начинается в `start` и заканчивается в `end`.

Обсуждение

Функция `substr` принимает строку, отправную точку и конечную. Она возвращает подстроку между отправной и конечной точками:

```
substr("Statistics", 1, 4) # Extract first 4 characters
#> [1] "Stat"
substr("Statistics", 7, 10) # Extract last 4 characters
#> [1] "tics"
```

Как и многие функции в R, `substr` позволяет первому аргументу быть вектором строк. В этом случае она применяется к каждой строке и возвращает вектор подстрок:

```
ss <- c("Moe", "Larry", "Curly")
substr(ss, 1, 3) # Extract first 3 characters of each string
#> [1] "Moe" "Lar" "Cur"
```

Фактически все аргументы могут быть векторами. В этом случае `substr` будет рассматривать их как параллельные векторы. Из каждой строки она извлекает подстроку, разделенную соответствующими записями в отправной и конечной точках, что дает возможность для некоторых полезных уловок. Например, в приведенном ниже фрагменте кода мы извлекаем последние два символа из каждой строки; каждая подстрока начинается на предпоследнем символе исходной строки и заканчивается на последнем символе:

```
cities <- c("New York, NY", "Los Angeles, CA", "Peoria, IL")
substr(cities, nchar(cities) - 1, nchar(cities))
#> [1] "NY" "CA" "IL"
```

Можно распространить этот трюк на умопомрачительную территорию, воспользовавшись правилом повторного использования, но мы предлагаем вам избежать соблазна.

7.4. Разбиение строки по разделителю

Задача

Вы хотите разбить строку на подстроки. Подстроки разделены разделителем.

Решение

Используйте функцию `strsplit`, которая принимает два аргумента, строку и разделитель подстрок:

```
strsplit(string, delimiter)
```

Разделитель может быть простой строкой или регулярным выражением.

Обсуждение

Обычно строка содержит несколько подстрок, разделенных одним и тем же разделителем.

Один из примеров – путь к файлу, компоненты которого разделены косой чертой (/):

```
path <- "/home/mike/data/trials.csv"
```

Можно разделить этот путь на компоненты, используя функцию `strsplit` с разделителем /:

```
strsplit(path, "/")
#> [[1]]
#> [1] "" "home" "mike" "data" "trials.csv"
```

Обратите внимание, что первый «компонент» на самом деле является пустой строкой, потому что перед первой косой чертой ничего нет.

Также обратите внимание на то, что функция `strsplit` возвращает список и что каждый элемент этого списка – это вектор подстрок. Такая двухуровневая структура необходима, потому что первый аргумент может быть вектором строк. Каждая строка разбивается на свои подстроки (вектор), а затем эти векторы возвращаются в списке.

Если вы работаете только с одной строкой, можно открыть первый элемент следующим образом:

```
strsplit(path, "/")[[1]]
#> [1] "" "home" "mike" "data" "trials.csv"
```

В этом примере мы разделяем три пути к файлам и возвращаем список из трех элементов:

```
paths <- c(
"/home/mike/data/trials.csv",
"/home/mike/data/errors.csv",
"/home/mike/corr/reject.doc"
)
strsplit(paths, "/")
#> [[1]]
#> [1] "" "home" "mike" "data" "trials.csv"
#> [[2]]
#> [1] "" "home" "mike" "data" "errors.csv"
#>
#> [[3]]
#> [1] "" "home" "mike" "corr" "reject.doc"
#>
```

Второй аргумент `strsplit` (аргумент `delimeter`) на самом деле намного мощнее, чем показывают эти примеры. Он может быть регулярным выражением, позволяющим выполнять сопоставление с образцами гораздо более сложным образом, чем простая строка. Фактически, чтобы отключить функцию регулярных выражений (и ее интерпретацию специальных символов), нужно включить аргумент `fixed=TRUE`.

См. также

Чтобы узнать больше о регулярных выражениях в R, *см.* страницу справки по регулярным выражениям. *См.* книгу Джеффри Э. Ф. Фридла «Регулярные выражения» (http://shop.oreilly.com/product/9780596528126.do) издательства *O'Reilly*, чтобы узнать больше о регулярных выражениях в целом.

7.5. Замена подстрок

Задача

Внутри строки вы хотите заменить одну подстроку другой.

Решение

Используйте функцию sub для замены первого экземпляра подстроки:

```
sub(old, new, string)
```

Используйте функцию gsub для замены всех экземпляров подстроки:

```
gsub(old, new, string)
```

Обсуждение

Функция sub находит первый экземпляр подстроки old в string и заменяет ее подстрокой new:

```
str <- "Curly is the smart one. Curly is funny, too."
sub("Curly", "Moe", str)
#> [1] "Moe is the smart one. Curly is funny, too."
```

Функция gsub делает то же самое, но заменяет *все* экземпляры подстроки (глобальная замена), а не только первый экземпляр:

```
gsub("Curly", "Moe", str)
#> [1] "Moe is the smart one. Moe is funny, too."
```

Чтобы полностью удалить подстроку, просто сделайте новую подстроку пустой:

```
sub(" and SAS", "", "For really tough problems, you need R and SAS.")
#> [1] "For really tough problems, you need R."
```

Аргумент old может быть регулярным выражением, которое позволяет выполнять сопоставление с образцом гораздо более сложным образом, чем простая строка. Это фактически предполагается по умолчанию, поэтому вы должны установить аргумент fixed=TRUE, если не хотите, чтобы функции sub и gsub интерпретировали old как регулярное выражение.

См. также

Чтобы узнать больше о регулярных выражениях в R, *см.* страницу справки по регулярным выражениям. *См.* книгу «*Регулярные выражения*», чтобы узнать больше о регулярных выражениях в целом.

7.6. Генерация всех попарных комбинаций строк

Задача

У вас есть два набора строк, и вы хотите сгенерировать все комбинации из этих двух наборов (их декартово произведение).

Решение

Используйте функции `outer` и `paste` вместе, чтобы сгенерировать матрицу всех возможных комбинаций:

```
m <- outer(strings1, strings2, paste, sep = "")
```

Обсуждение

Функция `outer` предназначена для формирования внешнего произведения. Однако она позволяет третьему аргументу заменить простое умножение на любую функцию. В этом рецепте мы заменяем умножение на конкатенацию строк (`paste`), и в результате получаются все комбинации строк.

Предположим, у нас есть четыре тестовые площадки и три процедуры (treatments):

```
locations <- c("NY", "LA", "CHI", "HOU")
treatments <- c("T1", "T2", "T3")
```

Мы можем применить функции `outer` и `paste`, чтобы сгенерировать все комбинации тестовых площадок и процедур, например:

```
outer(locations, treatments, paste, sep = "-")
#> [,1] [,2] [,3]
#> [1,] "NY-T1" "NY-T2" "NY-T3"
#> [2,] "LA-T1" "LA-T2" "LA-T3"
#> [3,] "CHI-T1" "CHI-T2" "CHI-T3"
#> [4,] "HOU-T1" "HOU-T2" "HOU-T3"
```

Четвертый аргумент функции `outer` передается в `paste`. В этом случае мы передали `sep="-"`, чтобы определить дефис в качестве разделителя между строками.

Результатом функции `outer` является матрица. Если вместо этого вы хотите использовать комбинации в векторе, преобразуйте матрицу с помощью функции `as.vector`.

В особом случае, когда вы комбинируете набор с самим собой и порядок не имеет значения, результатом будут дублированные комбинации:

```
outer(treatments, treatments, paste, sep = "-")
#> [,1] [,2] [,3]
#> [1,] "T1-T1" "T1-T2" "T1-T3"
#> [2,] "T2-T1" "T2-T2" "T2-T3"
#> [3,] "T3-T1" "T3-T2" "T3-T3"
```

Или можно использовать функцию `expand.grid`, чтобы получить пару векторов, представляющих все комбинации:

```
expand.grid(treatments, treatments)
#> Var1 Var2
#> 1 T1 T1
#> 2 T2 T1
#> 3 T3 T1
#> 4 T1 T2
#> 5 T2 T2
#> 6 T3 T2
#> 7 T1 T3
```

```
#> 8 T2 T3
#> 9 T3 T3
```

Но предположим, что нам нужны *все* уникальные попарные комбинации процедур. Можно устранить дубликаты, удалив нижний треугольник (или верхний). Функция `lower.tri` идентифицирует этот треугольник, поэтому при его инвертировании все элементы идентифицируются *за пределами* нижнего треугольника:

```
m <- outer(treatments, treatments, paste, sep = "-")
m[!lower.tri(m)]
#> [1] "T1-T1" "T1-T2" "T2-T2" "T1-T3" "T2-T3" "T3-T3"
```

См. также

См. рецепт 13.3, чтобы узнать, как использовать функцию `paste` для генерации комбинаций строк. В пакете `gtools` в CRAN (https://cran.r-project.org/web/packages/gtools/index.html) есть функции `combinations` и `permutation`, которые могут помочь в связанных задачах.

7.7. Получение текущей даты

Задача

Вам нужно знать сегодняшнюю дату.

Решение

Функция `Sys.Date` возвращает текущую дату:

```
Sys.Date()
#> [1] "2019-05-13"
```

Обсуждение

Функция `Sys.Date` возвращает объект `Date`. В предыдущем примере кажется, что она возвращает строку, потому что результат выводится в двойных кавычках. Однако в действительности функция возвращает объект `Date`, а затем R преобразует этот объект в строку для вывода. Можете убедиться в этом, проверив класс результата из `Sys.Date`:

```
class(Sys.Date())
#> [1] "Date"
```

См. также

См. рецепт 7.9.

7.8. Преобразование строки в дату

Задача

У вас есть строковое представление даты, например `"2018-12-31"`, и вы хотите преобразовать ее в объект `Date`.

Решение

Можно использовать функцию `as.Date`, но вы должны знать формат строки. По умолчанию эта функция предполагает, что строка имеет формат *гггг-мм-дд*. Для обработки других форматов необходимо указать параметр `format` функции `as.Date`. Используйте `format="%m/%d/%Y"`, если дата идет, например, в американском стиле.

Обсуждение

В этом примере показан формат по умолчанию, принятый `as.Date`, который является стандартным форматом ISO 8601 *гггг-мм-дд*:

```
as.Date("2018-12-31")
#> [1] "2018-12-31"
```

Функция `as.Date` возвращает объект `Date`, который (как и в предыдущем рецепте) преобразуется здесь обратно в строку для вывода; это объясняет наличие двойных кавычек.

Строка может быть в других форматах, но вы должны предоставить аргумент `format`, чтобы `as.Date` могла интерпретировать вашу строку. *См.* страницу справки по функции `strftime` для получения подробной информации о разрешенных форматах.

Будучи простыми американцами, мы часто по ошибке пытаемся преобразовать обычный американский формат даты (*мм/дд/гггг*) в объект `Date`, что приводит к таким печальным результатам:

```
as.Date("12/31/2018")
#> Error in charToDate(x): character string is not in a standard
#> unambiguous format
```

Вот правильный способ конвертировать дату в американском стиле:

```
as.Date("12/31/2018", format = "%m/%d/%Y")
#> [1] "2018-12-31"
```

Обратите внимание, что символ `Y` в строке формата указан в прописном варианте для обозначения четырехзначного года. Если вы используете двузначные годы, укажите строчный вариант буквы `y`.

7.9. Преобразование даты в строку

Задача

Вы хотите преобразовать объект `Date` в символьную строку обычно потому, что желаете вывести дату.

Решение

Используйте функцию `format` либо функцию `as.character`:

```
format(Sys.Date())
#> [1] "2019-05-13"
as.character(Sys.Date())
#> [1] "2019-05-13"
```

Обе функции допускают наличие аргумента `format`, который управляет форматированием. Используйте `format="%m/%d/%Y"`, чтобы получить даты в американском стиле, например:

```
format(Sys.Date(), format = "%m/%d/%Y")
#> [1] "05/13/2019"
```

Обсуждение

Аргумент `format` определяет внешний вид результирующей строки. Обычные символы, такие как косая черта (/) или дефис (-), просто копируются в строку вывода. Каждая двухбуквенная комбинация, состоящая из знака процента (%), за которым следует еще один символ, имеет особое значение. Вот некоторые из них:

`%b`
: Сокращенное название месяца («Jan»).

`%B`
: Полное название месяца («January»).

`%d`
: День в виде двузначного числа.

`%m`
: Месяц в виде двузначного числа.

`%y`
: Год без века (00–99).

`%Y`
: Год с веком.

См. страницу справки по функции `strftime`, чтобы увидеть полный список кодов форматирования.

7.10. Преобразование года, месяца и дня в объект Date

Задача

У вас есть дата в виде года, месяца и дня в разных переменных. Вы хотите объединить эти элементы в единое представление объекта `Date`.

Решение

Используйте функцию `ISOdate`:

```
ISOdate(year, month, day)
```

В результате получается объект `POSIXct`, который можно преобразовать в объект `Date`:

```
year <- 2018
month <- 12
day <- 31
```

```
as.Date(ISOdate(year, month, day))
#> [1] "2018-12-31"
```

Обсуждение

Входные данные обычно содержат даты, закодированные в виде трех чисел: год, месяц и день. Функция `ISOdate` может объединить их в объект `POSIXct`:

```
ISOdate(2020, 2, 29)
#> [1] "2020-02-29 12:00:00 GMT"
```

Вы можете сохранить свою дату в формате `POSIXct`. Однако при работе с чистыми датами (не датами и временем) мы часто выполняем преобразование в объект `Date` и усекаем неиспользуемую информацию о времени:

```
as.Date(ISOdate(2020, 2, 29))
#> [1] "2020-02-29"
```

Попытка преобразовать недопустимую дату приводит к появлению значения `NA`:

```
ISOdate(2013, 2, 29) # Ой! 2013 год не является високосным
#> [1] NA
```

Функция `ISOdate` может обрабатывать целые векторы лет, месяцев и дней, что очень удобно для массового преобразования входных данных. Приведенный ниже пример начинается с цифр, обозначающих год/месяц/день для третьей среды января на протяжении нескольких лет, а затем все они объединяются в объекты `Date`:

```
years <- 2010:2014
months <- rep(1, 5)
days <- 5:9
ISOdate(years, months, days)
#> [1] "2010-01-05 12:00:00 GMT" "2011-01-06 12:00:00 GMT"
#> [3] "2012-01-07 12:00:00 GMT" "2013-01-08 12:00:00 GMT"
#> [5] "2014-01-09 12:00:00 GMT"
as.Date(ISOdate(years, months, days))
#> [1] "2010-01-05" "2011-01-06" "2012-01-07" "2013-01-08" "2014-01-09"
```

Борцы за чистоту заметят, что вектор месяцев избыточен и что последнее выражение можно еще более упростить путем вызова правила повторного использования:

```
as.Date(ISOdate(years, 1, days))
#> [1] "2010-01-05" "2011-01-06" "2012-01-07" "2013-01-08" "2014-01-09"
```

Вы также можете расширить этот рецепт для обработки годов, месяцев, дней, часов, минут и секунд, используя функцию `ISOdatetime` (см. подробности на странице справки):

```
ISOdatetime(year, month, day, hour, minute, second)
```

7.11. Получение даты по юлианскому календарю

Задача

Имеется объект `Date`. Вы хотите извлечь юлианскую дату – в R это число дней с 1 января 1970 года.

Решение

Преобразуйте объект `Date` в целое число либо используйте функцию `julian`:

```
d <- as.Date("2019-03-15")
as.integer(d)
#> [1] 17970
jd <- julian(d)
jd
#> [1] 17970
#> attr(,"origin")
#> [1] "1970-01-01"
attr(jd, "origin")
#> [1] "1970-01-01"
```

Обсуждение

Юлианская «дата» – это просто число дней с момента произвольной отправной точки. В случае с R эта отправная точка – 1 января 1970 года, та же отправная точка, что и в системах Unix. Таким образом, юлианская дата на 1 января 1970 года равна нулю, как показано здесь:

```
as.integer(as.Date("1970-01-01"))
#> [1] 0
as.integer(as.Date("1970-01-02"))
#> [1] 1
as.integer(as.Date("1970-01-03"))
#> [1] 2
```

7.12. Извлечение частей даты

Задача

Для данного объекта `Date` вы хотите извлечь часть даты, например день недели, день года, календарный день, календарный месяц или календарный год.

Решение

Преобразуйте объект `Date` в объект `POSIXlt`, который представляет собой список частей даты. Затем извлеките нужную часть из этого списка:

```
d <- as.Date("2019-03-15")
p <- as.POSIXlt(d)
p$mday # Day of the month
#> [1] 15
p$mon # Month (0 = January)
#> [1] 2
p$year + 1900 # Year
#> [1] 2019
```

Обсуждение

Объект `POSIXlt` представляет дату в виде списка ее частей. Преобразуйте свой объект `Date` в `POSIXlt` с помощью функции `as.POSIXlt`, которая выдаст вам список этих членов:

`sec`
: Секунды (0–61).

`min`
: Минуты (0–59).

`hour`
: Часы (0–23).

`mday`
: День месяца (1–31).

`mon`
: Месяц (0–11).

`year`
: Лет с 1900.

`wday`
: День недели (0–6, 0 = воскресенье).

`yday`
: День года (0–365).

`isdst`
: Флаг перехода на летнее время.

Используя эти части даты, мы можем узнать, что 2 апреля 2020 года – четверг (`wday = 4`) и 93-й день года (потому что `yday = 0` 1 января):

```
d <- as.Date("2020-04-02")
as.POSIXlt(d)$wday
#> [1] 4
as.POSIXlt(d)$yday
#> [1] 92
```

Распространенной ошибкой является упущение добавления 1900 к году, в результате чего создается впечатление, что вы живете очень-очень давно:

```
as.POSIXlt(d)$year # Ой!
#> [1] 120
as.POSIXlt(d)$year + 1900
#> [1] 2020
```

7.13. Создание последовательности дат

Задача

Вы хотите создать последовательность дат, например последовательность ежедневных, ежемесячных или годовых дат.

Решение

Функция `seq` – это обобщенная функция, у которой есть версия для объектов `Date`. Она может создать последовательность `Date` аналогично тому, как создает последовательность чисел.

Обсуждение

При типичном использовании функции `seq` указывается дата начала (`from`), дата окончания (`to`) и приращение (`by`). Приращение 1 указывает на ежедневные даты:

```
s <- as.Date("2019-01-01")
e <- as.Date("2019-02-01")
seq(from = s, to = e, by = 1) # Один месяц с датами.
#> [1]    "2019-01-01" "2019-01-02" "2019-01-03" "2019-01-04" "2019-01-05"
#> [6]    "2019-01-06" "2019-01-07" "2019-01-08" "2019-01-09" "2019-01-10"
#> [11]   "2019-01-11" "2019-01-12" "2019-01-13" "2019-01-14" "2019-01-15"
#> [16]   "2019-01-16" "2019-01-17" "2019-01-18" "2019-01-19" "2019-01-20"
#> [21]   "2019-01-21" "2019-01-22" "2019-01-23" "2019-01-24" "2019-01-25"
#> [26]   "2019-01-26" "2019-01-27" "2019-01-28" "2019-01-29" "2019-01-30"
#> [31]   "2019-01-31" "2019-02-01"
```

При другом варианте типичного использования мы указываем начальную дату (`from`), приращение (`by`) и количество дат (`length.out`):

```
seq(from = s, by = 1, length.out = 7) # Даты, одна неделя.
#> [1] "2019-01-01" "2019-01-02" "2019-01-03" "2019-01-04" "2019-01-05"
#> [6] "2019-01-06" "2019-01-07"
```

Приращение (`by`) можно указать в днях, неделях, месяцах или годах:

```
seq(from = s, by = "month", length.out = 12) # Первый месяц для одного года.
#> [1] "2019-01-01" "2019-02-01" "2019-03-01" "2019-04-01" "2019-05-01"
#> [6] "2019-06-01" "2019-07-01" "2019-08-01" "2019-09-01" "2019-10-01"
#> [11] "2019-11-01" "2019-12-01"
seq(from = s, by = "3 months", length.out = 4) # Квартальные даты для одного года.
#> [1] "2019-01-01" "2019-04-01" "2019-07-01" "2019-10-01"
seq(from = s, by = "year", length.out = 10) # Даты начала года для одного десятилетия.
#> [1] "2019-01-01" "2020-01-01" "2021-01-01" "2022-01-01" "2023-01-01"
#> [6] "2024-01-01" "2025-01-01" "2026-01-01" "2027-01-01" "2028-01-01"
```

Будьте осторожны с `by="month"` рядом с концом месяца. В этом примере конец февраля переходит в март, и, вероятно, это не то, что вам нужно:

```
seq(as.Date("2019-01-29"), by = "month", len = 3)
#> [1] "2019-01-29" "2019-03-01" "2019-03-29"
```

Глава 8

Вероятность

Теория вероятностей является основой статистики, и в R существует множество механизмов для работы с вероятностями, распределениями вероятностей и случайными переменными. Рецепты, приведенные в этой главе, показывают, как рассчитать вероятности по квантилям, рассчитать квантили по вероятностям, сгенерировать случайные величины, взятые из распределений, построить графики распределений и т. д.

Названия распределений

В R существует сокращенное обозначение для каждого распределения вероятностей. Это название используется для идентификации функций, связанных с распределением. Например, имя нормального распределения – «norm». Это корень имен функций, перечисленных в табл. 8-1.

Таблица 8-1. Функции нормального распределения

Функция	Назначение
`dnorm`	Плотность нормального распределения
`pnorm`	Функция плотности нормального распределения
`qnorm`	Функция для вычисления квантилей нормального распределения
`rnorm`	Нормальные случайные величины

В табл. 8-2 описано несколько распространенных дискретных распределений, а в табл. 8-3 приводится ряд распространенных непрерывных распределений.

Таблица 8-2. Распространенные дискретные распределения

Дискретное распределение	Как оно называется в R	Параметры
Биномиальное	`binom`	`n` = количество испытаний; `p` = вероятность успеха для одного испытания
Геометрическое	`geom`	`p` = вероятность успеха для одного испытания
Гипергеометрическое	`hyper`	`m` = количество белых шаров в урне; `n` = количество черных шаров в урне; `k` = количество шаров, извлеченных из урны
Отрицательное биноминальное	`nbinom`	`размер` = количество успешных испытаний, либо `prob` = вероятность успешного испытания, либо `mu` = среднее
Пуассона	`pois`	`лямбда` = среднее

Таблица 8-3. Распространенные непрерывные распределения

Непрерывное распределение	Как оно называется в R	Параметры
Бета	`beta`	`shape1; shape2`
Коши	`cauchy`	`location; scale`
Хи-квадрат	`chisq`	`df` = степени свободы
Экспоненциальное	`exp`	`rate`
Фишера	`f`	`df1` и `df2` = степени свободы
Гамма	`gamma`	`rate` или `scale`
Логнормальное	`lnorm`	`meanlog` = среднее значение на логарифмической шкале; `sdlog` = стандартное отклонение на логарифмической шкале
Логистическое	`logis`	`location; scale`
Нормальное	`norm`	`mean; sd` = стандартное отклонение
Стьюдента	`t`	`df` = степени свободы
Равномерное	`unif`	`min` = нижний предел; `max` = верхний предел
Вейбулла	`weibull`	`shape; scale`
Уилкоксона	`wilcox`	`m` = количество наблюдений в первой выборке; `n` = количество наблюдений во второй выборке

Все функции, связанные с распределением, требуют параметров распределения, таких как `size` и `prob` для биномиального распределения или `prob` для геометрического. Возможная проблема заключается в том, что параметры распределения могут не соответствовать вашим ожиданиям. Например, мы ожидаем, что параметр экспоненциального распределения – это β, среднее значение. Однако в соглашении R экспоненциальное распределение определяется скоростью = 1/β, поэтому мы часто задаем неправильное значение. Мораль такова: изучите страницу справки, прежде чем использовать функцию, связанную с распространением. Убедитесь, что вы правильно поняли параметры.

Получение справки о распределениях вероятностей

Чтобы увидеть функции R, связанные с конкретным распределением вероятностей, используйте команду `help` и полное имя распределения. Например, с помощью этого вы увидите функции, связанные с нормальным распределением:

```
?Normal
```

В некоторых дистрибутивах есть имена, которые плохо работают с командой `help`, такие как «Student's t». У них есть специальные справочные имена, как отмечено в табл. 8-2 и 8-3: Neg-Binomial, Chisquare, Lognormal и TDist. Таким образом, чтобы получить справку по распределению Стьюдента, используйте это:

```
?TDist
```

См. также

Есть много других распределений, которые реализованы в пакетах, доступных для скачивания; *см.* представление задач CRAN, посвященное распределению вероятностей (https://cran.r-project.org/web/views/Distributions.html). Пакет `SuppDists` входит в базовую часть R и включает в себя 10 дополнительных распределений. Пакет `MASS`, который также входит в базовую часть, обеспечивает дополнительную поддержку распределений, например построение методом максимального правдоподобия для некоторых распространенных распределений, а также выборку из многомерного нормального распределения.

8.1. Подсчет количества комбинаций

Задача

Вы хотите рассчитать количество комбинаций *n* элементов, взятых по *k* за раз.

Решение

Используйте функцию `choose`.

Обсуждение

Распространенной проблемой при вычислении вероятностей дискретных переменных является подсчет комбинаций: количество различных подмножеств *k* размера, которые можно создать из *n* элементов. Число задается как `n!/r!(n - r)!`, но гораздо удобнее использовать функцию `choose`, особенно по мере увеличения *n* и *k*:

```
choose(5, 3)    # Сколькими способами мы можем выбрать 3 элемента из 5?
#> [1] 10
choose(50, 3)   # Сколькими способами мы можем выбрать 3 элемента из 50?
#> [1] 19600
choose(50, 30)  # Сколькими способами мы можем выбрать 30 элементов из 50?
#> [1] 4.71e+13
```

Эти числа также известны как *биномиальные коэффициенты*.

См. также

В этом рецепте мы просто считаем комбинации; *см.* рецепт 8.2, чтобы узнать, как сгенерировать их.

8.2. Генерация комбинаций

Задача

Вы хотите сгенерировать все комбинации из *n* элементов, взятых по *k* за раз.

Решение

Используйте функцию `combn`:

```
items <- 2:5
k <- 2
```

```
combn(items, k)
#>      [,1][,2][,3][,4][,5][,6]
#> [1,]  2   2   2   3   3   4
#> [2,]  3   4   5   4   5   5
```

Обсуждение

Мы можем использовать `combn(1:5,3)` для генерации всех комбинаций чисел от 1 до 5, взятых по три за раз:

```
combn(1:5, 3)
#>      [,1][,2][,3][,4][,5][,6][,7][,8][,9][,10]
#> [1,]  1   1   1   1   1   1   2   2   2   3
#> [2,]  2   2   2   3   3   4   3   3   4   4
#> [3,]  3   4   5   4   5   5   4   5   5   5
```

Эта функция не ограничивается числами. Мы также можем генерировать комбинации строк. Вот все комбинации из пяти процедур, взятых по три одновременно:

```
combn(c("T1", "T2", "T3", "T4", "T5"), 3)
#>      [,1]  [,2]  [,3]  [,4]  [,5]  [,6]  [,7]  [,8]  [,9]  [,10]
#> [1,] "T1"  "T1"  "T1"  "T1"  "T1"  "T1"  "T2"  "T2"  "T2"  "T3"
#> [2,] "T2"  "T2"  "T2"  "T3"  "T3"  "T4"  "T3"  "T3"  "T4"  "T4"
#> [3,] "T3"  "T4"  "T5"  "T4"  "T5"  "T5"  "T4"  "T5"  "T5"  "T5"
```

По мере того как количество элементов, *n*, увеличивается, количество комбинаций может увеличиться взрывным образом – особенно если *k* далеко от 1 или *n*.

См. также

См. рецепт 8.1, чтобы подсчитать количество возможных комбинаций, *прежде чем* приступить к генерации большого числа комбинаций.

8.3. Генерация случайных чисел

Задача

Вы хотите сгенерировать случайные числа.

Решение

Простой случай генерации равномерного случайного числа в промежутке от 0 до 1 обрабатывается функцией `runif`. В этом примере мы генерируем одно равномерное случайное число:

```
runif(1)
#> [1] 0.915
```

Если вы говорите `runif` вслух (или даже в своей голове), следует произносить его как «are unif» вместо «run if». Термин `runif` – это слияние слов «random uniform», поэтому оно не должно звучать так, словно это функция управления потоком.

R может генерировать случайные переменные и из других распределений. Для данного распределения имя генератора случайных чисел имеет префикс «r» перед сокращенным именем распределения (например, `rnorm` для генератора случайных чисел нормального распределения).

В этом примере мы генерируем одно случайное значение из стандартного нормального распределения:

```
rnorm(1)
#> [1] 1.53
```

Обсуждение

У большинства языков программирования имеется только простейший генератор случайных чисел, который генерирует одно случайное число, равномерно распределенное между 0,0 и 1,0, и это все. Но не R.

R может генерировать случайные числа из множества распределений вероятностей, отличных от равномерного распределения. Простой случай генерации равномерных случайных чисел от 0 до 1 обрабатывается функцией `runif`:

```
runif(1)
#> [1] 0.83
```

Аргумент этой функции – число случайных значений, которые будут сгенерированы. Сгенерировать вектор из 10 таких значений так же просто, как и из одного:

```
runif(10)
#> [1] 0.642 0.519 0.737 0.135 0.657 0.705 0.458 0.719 0.935 0.255
```

Существуют генераторы случайных чисел для всех встроенных распределений. Просто добавьте к имени дистрибутива букву «r», и вы получите имя соответствующего генератора случайных чисел. Вот некоторые из них:

```
runif(1, min = -3, max = 3)      # Равномерная случайная величина в промежутке от -3 до +3.
#> [1] 2.49
rnorm(1)                         # Одна стандартная нормальная случайная величина.
#> [1] 1.53
rnorm(1, mean = 100, sd = 15)    # Одна нормальная случайная величина, среднее значение 100
и SD 15.
#> [1] 114
rbinom(1, size = 10, prob = 0.5) # Одна биномиальная случайная величина.
#> [1] 5
rpois(1, lambda = 10)            # Одна случайная величина Пуассона.
#> [1] 12
rexp(1, rate = 0.1)              # Одна экспоненциальная случайная величина.
#> [1] 3.14
rgamma(1, shape = 2, rate = 0.1) # Одна гамма-случайная величина.
#> [1] 22.3
```

Как и в случае с `runif`, первый аргумент – это количество генерируемых случайных значений. Последующие аргументы – это параметры распределения, такие как `mean` и `sd` для нормального распределения или `size` и `prob` для биномиального. Подробнее *см.* страницу справки по функции.

В приведенных выше примерах используются простые скаляры для параметров распределения. Тем не менее параметры также могут быть векторами, и в этом случае R будет циклически проходить через вектор, генерируя случайные значения. В приведенном ниже примере мы генерируем три нормальных случайных значения, взятых из распределений со средними значениями –10, 0 и +10 соответственно (все распределения имеют стандартное отклонение 1,0):

```
rnorm(3, mean = c(-10, 0, +10), sd = 1)
#> [1] -9.420 -0.658 11.555
```

Это мощная возможность для таких случаев, как иерархические модели, где параметры сами являются случайными. В следующем примере мы вычисляем 30 значений нормальной случайной величины, среднее значение которой само распределено случайным образом с гиперпараметрами $\mu = 0$ и $\sigma = 0{,}2$:

```
means <- rnorm(30, mean = 0, sd = 0.2)
rnorm(30, mean = means, sd = 1)
#> [1]   -0.5549 -2.9232 -1.2203 0.6962  0.1673  -1.0779 -0.3138 -3.3165
#> [9]   1.5952  0.8184  -0.1251 0.3601  -0.8142 0.1050  2.1264  0.6943
#> [17]  -2.7771 0.9026  0.0389  0.2280  -0.5599 0.9572  0.1972  0.2602
#> [25]  -0.4423 1.9707  0.4553  0.0467  1.5229  0.3176
```

Если вы генерируете множество случайных значений и вектор параметров слишком короткий, R применит правило повторного использования к вектору параметров.

См. также

См. введение к этой главе.

8.4. Генерация воспроизводимых случайных чисел

Задача

Вам нужно сгенерировать последовательность случайных чисел, но вы хотите воспроизводить одну и ту же последовательность при каждом запуске вашей программы.

Решение

Перед запуском кода вызовите функцию `set.seed`, чтобы инициализировать генератор случайных чисел в известное состояние:

```
set.seed(42) # Или используйте любое другое положительное целое число...
```

Обсуждение

После генерации случайных чисел у вас может появиться желание воспроизводить одну и ту же последовательность «случайных» чисел при каждом запуске вашей программы. Таким образом, вы получаете те же результаты от запуска

к запуску. Один из авторов как-то сопровождал сложный анализ методом Монте-Карло огромного портфеля ценных бумаг. Пользователи жаловались на то, что при каждом запуске программы получались немного разные результаты. Без шуток! Анализ проводился исключительно по случайным числам, поэтому, конечно, в выводе была случайность. Было решено установить генератор случайных чисел в известное состояние в начале программы. Таким образом, он будет генерировать одни и те же (квази)случайные числа каждый раз и, следовательно, давать согласованные, воспроизводимые результаты.

В R функция `set.seed` устанавливает генератор случайных чисел в известное состояние. Эта функция принимает один аргумент, целое число. Подойдет любое положительное целое число, но вы должны использовать одно и то же число, чтобы получить то же начальное состояние.

Эта функция ничего не возвращает. Она работает за кулисами, инициализируя (или повторно инициализируя) генератор случайных чисел. Ключевым моментом здесь является то, что при использовании одного и того же начального числа генератор случайных чисел перезапускается обратно в том же месте:

```
set.seed(165) # Инициализируем генератор в известное состояние.
runif(10) # Генерируем десять случайных чисел.
#> [1] 0.116 0.450 0.996 0.611 0.616 0.426 0.666 0.168 0.788 0.442

set.seed(165) # Выполняем повторную инициализацию в то же известное состояние.
runif(10) # Генерируем все те же десять "случайных" чисел.
#> [1] 0.116 0.450 0.996 0.611 0.616 0.426 0.666 0.168 0.788 0.442
```

Когда вы устанавливаете начальное значение (seed value) и замораживаете свою последовательность случайных чисел, вы устраняете источник случайности, который может иметь решающее значение для таких алгоритмов, как методы Монте-Карло. Прежде чем вызывать функцию `set.seed` в своем приложении, спросите себя: не подрываю ли я ценность своей программы или, возможно, даже нарушаю ее логику?

См. также

См. рецепт 8.3 для получения дополнительной информации касательно генерации случайных чисел.

8.5. Генерация случайной выборки

Задача

Вы хотите выбрать набор данных случайным образом.

Решение

Функция `sample` случайным образом выберет *n* элементов из набора:

```
sample(set, n)
```

Обсуждение

Предположим, что ваши данные о Мировой серии содержат вектор лет, когда проводились игры. Вы можете выбрать 10 лет наугад, используя функцию `sample`:

```
world_series <- read_csv("./data/world_series.csv")
sample(world_series$year, 10)
#> [1] 2010 1961 1906 1992 1982 1948 1910 1973 1967 1931
```

Элементы выбираются случайным образом, поэтому повторное использование этой функции (обычно) дает другой результат:

```
sample(world_series$year, 10)
#> [1] 1941 1973 1921 1958 1979 1946 1932 1919 1971 1974
```

Функция `sample` обычно выполняет выборку без замены, то есть она не будет выбирать один и тот же элемент дважды. Некоторые статистические процедуры (в особенности бутстрэп) требуют выборку *с* заменой, а это означает, что один элемент может появляться в выборке несколько раз. Используйте `replace=TRUE` для выборки с заменой.

Простой бутстрэп-метод легко реализовать, используя выборку с заменой. Предположим, у нас есть вектор `x` из 1000 случайных чисел, построенный из нормального распределения со средним значением 4 и стандартным отклонением 10:

```
set.seed(42)
x <- rnorm(1000, 4, 10)
```

В этом фрагменте кода мы делаем выборку 1000 раз из вектора `x` и вычисляем медиану каждой выборки:

```
medians <- numeric(1000) # empty vector of 1000 numbers
for (i in 1:1000) {
medians[i] <- median(sample(x, replace = TRUE))
}
```

Исходя из оценок бутстрэпа, можно рассчитать доверительный интервал для медианы:

```
ci <- quantile(medians, c(0.025, 0.975))
cat("95% confidence interval is (", ci, ")\n")
#> 95% confidence interval is ( 3.16 4.49 )
```

Мы знаем, что вектор `x` был создан из нормального распределения со средним значением 4, и, следовательно, медиана выборки также должна составлять 4. (В симметричном распределении, подобном этому, среднее значение и медиана одинаковы.) Наш доверительный интервал легко содержит это значение.

См. также

См. рецепт 8.7, чтобы узнать о случайной перестановке вектора, и рецепт 13.8 для получения дополнительной информации о бутстрэппинге. В рецепте 8.4 обсуждается назначение начальных чисел (seeds) при генерации квазислучайных чисел.

8.6. Генерация случайных последовательностей

Задача

Вы хотите сгенерировать случайную последовательность, например серию смоделированных бросков монет или смоделированную последовательность испытаний Бернулли.

Решение

Используйте функцию `sample`. Используйте *n* из набора возможных значений и установите для `replace` значение `TRUE`:

```
sample(set, n, replace = TRUE)
```

Обсуждение

Функция `sample` случайным образом выбирает элементы из набора. Обычно она выполняет выборку *без* замены. Это означает, что она не будет выбирать один и тот же элемент дважды и вернет ошибку, если вы попытаетесь выбрать больше элементов, чем имеется в наборе. Однако, когда для `replace` установлено значение `TRUE`, эта функция может выбирать элементы снова и снова, что позволяет генерировать длинные случайные последовательности элементов.

В следующем примере мы генерируем случайную последовательность из 10 симулированных бросков монеты:

```
sample(c("H", "T"), 10, replace = TRUE)
#> [1] "H" "T" "H" "T" "T" "T" "H" "T" "T" "H"
```

В этом примере мы генерируем последовательность из 20 испытаний Бернулли – случайные успехи или неудачи. Мы используем значение `TRUE`, чтобы обозначить успех:

```
sample(c(FALSE, TRUE), 20, replace = TRUE)
#> [1] TRUE FALSE TRUE TRUE FALSE TRUE FALSE FALSE TRUE TRUE FALSE
#> [12] TRUE TRUE FALSE TRUE TRUE FALSE FALSE FALSE FALSE
```

По умолчанию функция `sample` будет одинаково выбирать среди заданных элементов, поэтому вероятность выбора `TRUE` или `FALSE` равна 0,5. При испытании Бернулли вероятность успеха p не обязательно равна 0,5. Вы можете изменить выборку с помощью аргумента `prob` для функции `sample`; этот аргумент представляет собой вектор вероятностей, по одной на каждый элемент набора. Предположим, мы хотим сгенерировать 20 испытаний Бернулли с вероятностью успеха $p = 0{,}8$. Мы устанавливаем вероятность `FALSE` равной 0,2, а вероятность `TRUE` – 0,8:

```
sample(c(FALSE, TRUE), 20, replace = TRUE, prob = c(0.2, 0.8))
#> [1] TRUE TRUE FALSE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
#> [12] TRUE TRUE TRUE TRUE TRUE FALSE FALSE TRUE TRUE
```

Получившаяся в результате последовательность явно смещена в сторону `TRUE`. Мы выбрали этот пример, потому что это простая демонстрация общей техники. Для частного случая двоичной последовательности с двумя исходами можно использовать функцию `rbinom`, генератор случайных чисел для биномиальных величин:

```
rbinom(10, 1, 0.8)
#> [1] 1 0 1 1 1 1 1 0 1 1
```

8.7. Случайная перестановка вектора

Задача

Вы хотите сгенерировать случайную перестановку вектора.

Решение

Если `v` – это ваш вектор, то `sample(v)` возвращает случайную перестановку.

Обсуждение

Обычно, когда речь идет о выборке из больших наборов данных, на ум нам приходит функция `sample`. Однако параметры по умолчанию позволяют создать случайную перестановку набора данных. Пример вызова `sample(v)` эквивалентен:

```
sample(v, size = length(v), replace = FALSE)
```

что означает «выбрать все элементы `v` в случайном порядке, используя каждый элемент ровно один раз». Это случайная перестановка. Вот случайная перестановка 1, ..., 10:

```
sample(1:10)
#> [1] 7 3 6 1 5 2 4 8 10 9
```

См. также

См. рецепт 8.5 для получения дополнительной информации о функции `sample`.

8.8. Расчет вероятностей для дискретных распределений

Задача

Вы хотите вычислить простую либо интегральную вероятность, связанную с дискретной случайной величиной.

Решение

В случае с простой вероятностью $P(X = x)$ используйте функцию плотности. У всех встроенных распределений вероятностей есть функция плотности, имя которой представляет собой префикс «d», стоящий перед именем распределения; например, `dbinom` для биномиального распределения.

В случае с интегральной вероятностью $P(X \leqslant x)$ используйте функцию распределения. У всех встроенных распределений вероятностей есть функция распределения, имя которой представляет собой префикс «p», стоящий перед именем распределения; таким образом, `pbinom` – это функция распределения биномиального распределения.

Обсуждение

Предположим, у нас есть биномиальная случайная величина X на 10 испытаний, где вероятность успеха каждого испытания равна 1/2. Тогда мы можем вычислить вероятность наблюдения $x = 7$, вызвав функцию `dbinom`:

```
dbinom(7, size = 10, prob = 0.5)
#> [1] 0.117
```

Мы вычислили вероятность около 0,117. В R функция `dbinom` носит название *функции плотности*. В некоторых учебниках она называется *функцией распределения масс*, или *функцией вероятности*. Называя ее функцией плотности, мы сохраняем терминологию, согласованную между дискретным и непрерывным распределениями (*см.* рецепт 8.9).

Интегральная вероятность, $P(X \leqslant x)$, задается *функцией распределения*, которую иногда называют *интегральной функцией распределения вероятностей*. Функция распределения для биномиального распределения – это `pbinom`. Ниже приводится интегральная вероятность для $x = 7$ (т. е. $P(X \leqslant 7)$):

```
pbinom(7, size = 10, prob = 0.5)
#> [1] 0.945
```

Похоже, вероятность наблюдения $X \leqslant 7$ составляет около 0,945.

Функции плотности и функции распределения для некоторых распространенных дискретных распределений показаны в табл. 8-4.

Таблица 8-4. Дискретные распределения

Распределение	Функция плотности: $P(X = x)$	Функция распределения: $P(X \leqslant x)$
Биномиальное	`dbinom(x, size, prob)`	`pbinom(x, size, prob)`
Геометрическое	`dgeom(x, prob)`	`pgeom(x, prob)`
Пуассона	`dpois(x, lambda)`	`ppois(x, lambda)`

Дополнением интегральной вероятности является *функция выживания*, $P(X > x)$. Все функции распределения позволяют найти эту правостороннюю вероятность, просто указав для `lower.tail` значение `FALSE`:

```
pbinom(7, size = 10, prob = 0.5, lower.tail = FALSE)
#> [1] 0.0547
```

Таким образом, мы видим, что вероятность наблюдения $X > 7$ составляет около 0,055.

Интервальная вероятность, $P(x_1 < X \leqslant x_2)$, – это вероятность наблюдения X между пределами x_1 и x_2. Она рассчитывается как разница между двумя интегральными вероятностями: $P(X \leqslant x_2) - P(X \leqslant x_1)$. Ниже приводится $P(3 < X \leqslant 7)$ для нашей биномиальной величины:

```
pbinom(7, size = 10, prob = 0.5) - pbinom(3, size = 10, prob = 0.5)
#> [1] 0.773
```

R позволяет указать несколько значений x для этих функций и возвращает вектор соответствующих вероятностей. Здесь мы вычисляем две интегральные вероятности, $P(X \leqslant 3)$ и $P(X \leqslant 7)$, за один вызов функции `pbinom`:

```
pbinom(c(3, 7), size = 10, prob = 0.5)
#> [1] 0.172 0.945
```

Это приводит к появлению однострочного сценария для вычисления интервальных вероятностей. Функция `diff` вычисляет разницу между последователь-

ными элементами вектора. Мы применяем ее к выводу функции pbinom, чтобы получить разницу в интегральных вероятностях – другими словами, интервальную вероятность:

```
diff(pbinom(c(3, 7), size = 10, prob = 0.5))
#> [1] 0.773
```

См. также

См. введение к этой главе для получения дополнительной информации о встроенных распределениях вероятностей.

8.9. Расчет вероятностей для непрерывных распределений

Задача

Вы хотите вычислить функцию распределения (DF) или интегральную функцию распределения вероятностей (CDF) для непрерывной случайной величины.

Решение

Используйте функцию распределения, которая вычисляет $P(X \leqslant x)$. У всех встроенных распределений вероятностей есть функция распределения, имя которой представляет собой префикс «p», стоящий перед аббревиатурой имени распределения; например, pbinom для биномиального распределения.

Допустим, можно рассчитать вероятность отрисовки из случайного стандартного нормального распределения ниже 0,8 следующим образом:

```
pnorm(q = .8, mean = 0, sd = 1)
#> [1] 0.788
```

Обсуждение

Функции распределений вероятностей в R следуют последовательной схеме, поэтому решение этого рецепта, по существу, идентично решению, используемому для дискретных случайных величин (*см.* рецепт 8.8). Существенным отличием является то, что непрерывные величины не имеют «вероятности» в одной точке, $P(X = x)$. Вместо этого у них есть «плотность» в точке.

Учитывая такую согласованность, обсуждение функций распределения в рецепте 8.8 применимо и здесь. В табл. 8-5 приведены функции распределения для ряда непрерывных распределений.

Таблица 8-5. Непрерывные распределения

Распределение	Функция распределения: P(X ⩽ x)
Нормальное	`pnorm(x, mean, sd)`
Стьюдента	`pt(x, df)`
Экспоненциальное	`pexp(x, rate)`
Гамма	`pgamma(x, shape, rate)`
Хи-квадрат ($\chi 2$)	`pchisq(x, df)`

Мы можем использовать функцию `pnorm` для расчета вероятности того, что некий человек ниже 66 дюймов, предполагая, что его рост обычно распределяется со средним значением в 70 дюймов и стандартным отклонением в 3 дюйма. Говоря математически, мы хотим $P(X \leqslant 66)$, учитывая, что $X \sim N(70, 3)$:

```
pnorm(66, mean = 70, sd = 3)
#> [1] 0.0912
```

Аналогично, можно использовать функцию `pexp` для вычисления вероятности того, что экспоненциальная величина со средним значением 40 может быть меньше 20:

```
pexp(20, rate = 1 / 40)
#> [1] 0.393
```

Как и в случае с дискретными вероятностями, функции для непрерывных вероятностей используют `lower.tail` со значением `FALSE` для определения функции выживания, $P(X > x)$. Этот вызов функции `pexp` дает вероятность того, что одна и та же экспоненциальная величина может быть больше 50:

```
pexp(50, rate = 1 / 40, lower.tail = FALSE)
#> [1] 0.287
```

Так же, как и дискретные вероятности, интервальная вероятность для непрерывной величины, $P(x_1 < X < x_2)$, вычисляется как разница между двумя интегральными вероятностями, $P(X < x_2) - P(X < x_1)$. В случае с той же экспоненциальной величиной в приведенном ниже примере мы видим $P(20 < X < 50)$, вероятность того, что она может упасть между 20 и 50:

```
pexp(50, rate = 1 / 40) - pexp(20, rate = 1 / 40)
#> [1] 0.32
```

См. также

См. введение к этой главе для получения дополнительной информации о встроенных распределениях вероятностей.

8.10. Преобразование вероятностей в квантили

Задача

Имеется вероятность p и распределение. Вы хотите определить соответствующий квантиль для p: значение x как в $P(X \leqslant x) = p$.

Решение

Каждое встроенное распределение включает в себя квантильную функцию, которая преобразует вероятности в квантили. Имя функции представляет собой префикс «q», добавленный к имени распределения; таким образом, например, `qnorm` – это квантильная функция нормального распределения.

Первый аргумент квантильной функции – вероятность. Остальные аргументы – это параметры распределения, такие как `mean`, `shape` или `rate`:

```
qnorm(0.05, mean = 100, sd = 15)
#> [1] 75.3
```

Обсуждение

Типичный пример вычисления квантилей – вычисление границ доверительного интервала. Если мы хотим знать 95%-ный доверительный интервал ($\alpha = 0{,}05$) стандартной нормальной величины, нам нужны квантили с вероятностями $\alpha/2 = 0{,}025$ и $(1 - \alpha) / 2 = 0{,}975$:

```
qnorm(0.025)
#> [1] -1.96
qnorm(0.975)
#> [1] 1.96
```

В истинном духе R первый аргумент квантильных функций может быть вектором вероятностей, и в этом случае мы получаем вектор квантилей. Можно упростить этот пример и написать его одной строкой:

```
qnorm(c(0.025, 0.975))
#> [1] -1.96 1.96
```

Все встроенные распределения вероятностей предоставляют квантильную функцию. В табл. 8-6 показаны квантильные функции ряда распространенных дискретных распределений.

Таблица 8-6. Дискретные квантильные распределения

Распределение	Квантильная функция
Биномиальное	`qbinom(p, size, prob)`
Геометрическое	`qgeom(p, prob)`
Пуассона	`qpois(p, lambda)`

В табл. 8-7 приводятся квантильные функции распространенных непрерывных распределений.

Таблица 8-7. Непрерывные квантильные распределения

Распределение	Квантильная функция
Нормальное	`qnorm(p, mean, sd)`
Стьюдента	`qt(p, df)`
Экспоненциальное	`qexp(p, rate)`
Гамма	`qgamma(p, shape, rate)` или `qgamma(p, shape, scale)`
Хи-квадрат ($\chi 2$)	`qchisq(p, df)`

См. также

Определение квантилей набора данных отличается от определения квантилей распределения – *см.* рецепт 9.5.

8.11. Построение графика функции плотности

Задача

Вы хотите построить график функции плотности распределения вероятностей.

Решение

Определите вектор x на области определения. Примените функцию плотности распределения к этому вектору, а затем нанесите результат на график. Если x – это вектор точек на области определения, которую вы рассматриваете для построения графика, вы затем рассчитываете плотность, используя одну из функций плотности d____, например dlnorm для логнормального распределения или dnorm для нормального:

```
dens <- data.frame(x = x,
                   y = d_____(x))
ggplot(dens, aes(x, y)) + geom_line()
```

Вот конкретный пример, гле показано стандартное нормальное распределение для интервала от –3 до +3:

```
library(ggplot2)
x <- seq(-3, +3, 0.1)
dens <- data.frame(x = x, y = dnorm(x))
ggplot(dens, aes(x, y)) + geom_line()
```

На рис. 8-1 показана плавная функция плотности.

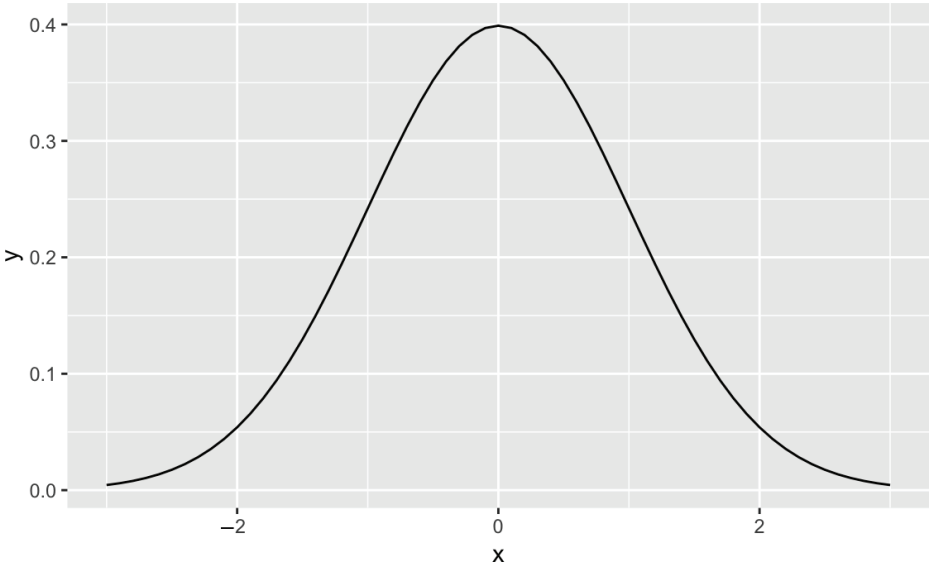


Рис. 8-1. Плавная функция плотности

Обсуждение

Все встроенные распределения вероятностей включают в себя функцию плотности. Для определенной плотности имя функции «d» ставится перед именем распределения. Функция плотности нормального распределения – это dnorm, для гамма-распределения – dgamma и т. д.

Если первый аргумент функции плотности – это вектор, функция вычисляет плотность в каждой точке и возвращает вектор плотностей.

В приведенном ниже коде мы создаем график 2×2 четырех плотностей (рис. 8-2):

```
x <- seq(from = 0, to = 6, length.out = 100) # Определяем области плотности.
ylim <- c(0, 0.6)

# Создаем data.frame с плотностями нескольких распределений.
df <- rbind(
  data.frame(x = x, dist_name = "Uniform"=, y = dunif(x, min = 2, max = 4)),
  data.frame(x = x, dist_name = "Normal"=, y = dnorm(x, mean = 3, sd = 1)),
  data.frame(x = x, dist_name = "Exponential", y = dexp(x, rate = 1 / 2)),
  data.frame(x = x, dist_name = "Gamma"=, y = dgamma(x, shape = 2, rate = 1)) )

# Строим линейный график, как и прежде, но используем функцию facet_wrap для создания
сетки.
ggplot(data = df, aes(x = x, y = y)) +
  geom_line() +
  facet_wrap(~dist_name) # Используем переменную dist_name.
```

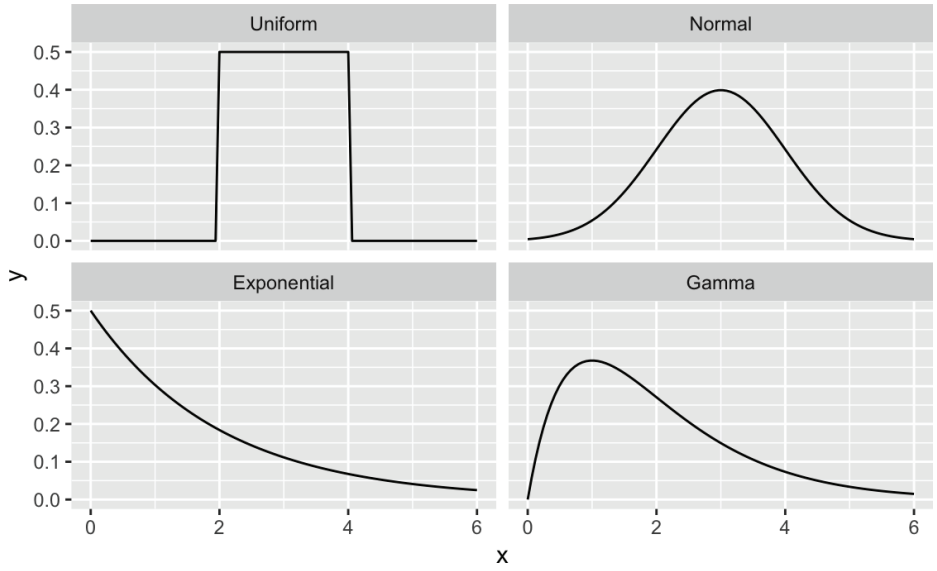


Рис. 8-2. Графики нескольких плотностей распределений

На рис. 8-2 показано четыре графика плотности. Тем не менее сырой график плотности распределения редко бывает полезным или интересным сам по себе, и мы часто заштриховываем интересующую область.

На рис. 8-3 показано нормальное распределение с затенением от 75-го до 95-го процентиля.

Мы создаем график, нанося на него плотность, а затем создаем заштрихованную область с помощью функции `geom_ribbon` из пакета `ggplot2`.

Сначала мы создаем некие данные и рисуем кривую плотности, подобную той, что показана на рис. 8-4:

```
x <- seq(from = -3, to = 3, length.out = 100)
df <- data.frame(x = x, y = dnorm(x, mean = 0, sd = 1))
```

```
p <- ggplot(df, aes(x, y)) +
  geom_line() +
  labs(
    title = "Standard Normal Distribution",
    y = "Density",
    x = "Quantile"
  )
p
```

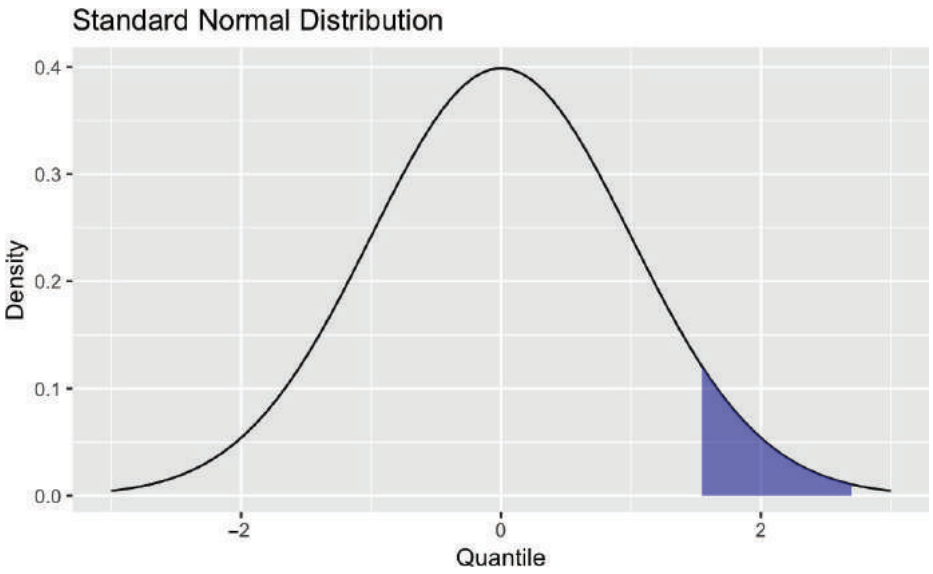


Рис. 8-3. Стандартное нормальное распределение с затенением

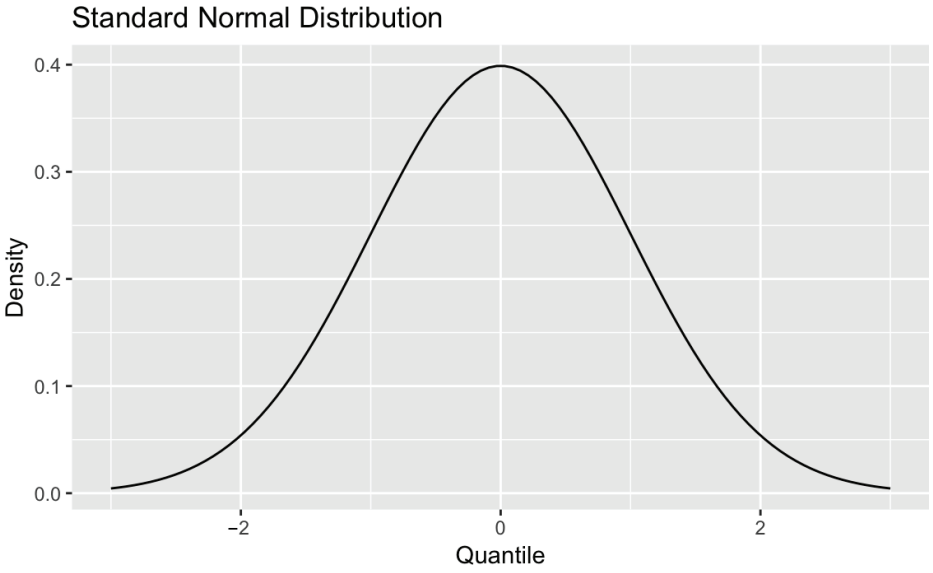


Рис. 8-4. График распределения

Затем мы определяем интересующую область, вычисляя значения x для интересующих нас квантилей. Наконец, мы используем функцию geom_ribbon, чтобы добавить подмножество наших исходных данных в качестве окрашенной области:

```
q75 <- quantile(df$x, .75)
q95 <- quantile(df$x, .95)
p +
geom_ribbon(
data = subset(df, x > q75 & x < q95),
aes(ymax = y),
ymin = 0,
fill = "blue",
color = NA,
alpha = 0.5
)
```

Получившийся в результате график показан на рис. 8-5.

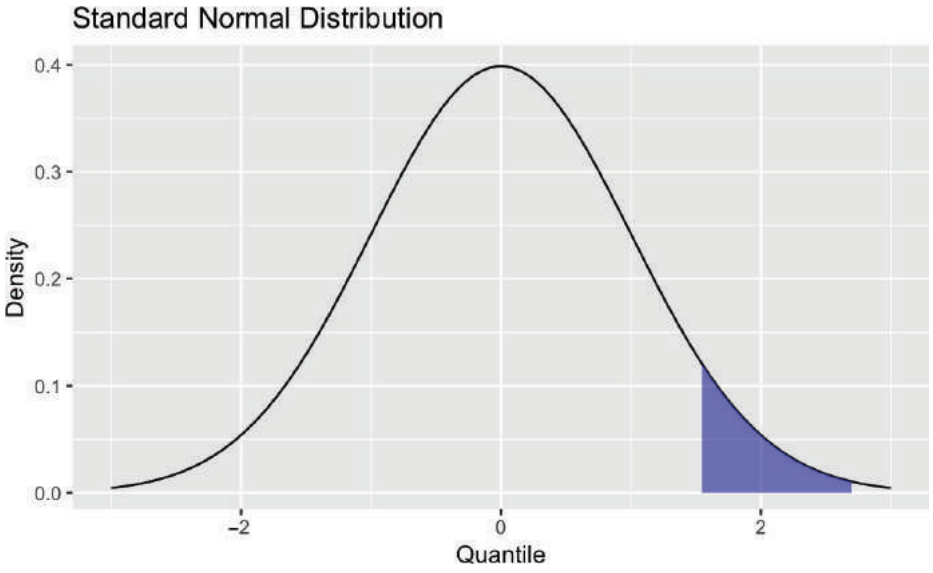


Рис. 8-5. Плотность нормального распределения с затенением

Глава 9

Общая статистика

Любое значительное применение R включает в себя статистику, модели или графику. Эта глава посвящена статистике. Некоторые рецепты просто описывают, как рассчитать статистику, например относительную частоту. В большинстве рецептов используются статистические тесты или доверительные интервалы. Статистические тесты позволяют сделать выбор между двумя конкурирующими гипотезами; эта парадигма описывается далее. Доверительные интервалы отражают вероятный диапазон параметра генеральной совокупности и рассчитываются на основе вашей выборки данных.

Нулевые гипотезы, альтернативные гипотезы и p-значения

Многие статистические тесты, приведенные в этой главе, используют проверенную временем парадигму статистического вывода. В этой парадигме у нас есть один или два образца данных. У нас также есть две конкурирующие гипотезы, каждая из которых может быть обоснованно верна.

Одна из гипотез, которая носит название *нулевая гипотеза*, состоит в том, что *ничего не произошло*: среднее значение не изменилось; лечение не имело эффекта; вы получили ожидаемый ответ; модель не улучшилась и т. д.

Другая гипотеза, которая носит название *альтернативная гипотеза*, состоит в том, что что-то произошло: среднее значение выросло; в ходе лечения здоровье пациентов улучшилось; вы получили неожиданный ответ; модель подходит лучше и т. д.

Мы хотим определить, какая гипотеза более вероятна в свете данных. Вот как мы это делаем.

1. Для начала предположим, что нулевая гипотеза верна.
2. Подсчитаем тестовую статистику. Это может быть что-то простое, например среднее значение выборки, или это может быть что-то довольно сложное. Крайне важно знать распределение статистики. Мы можем узнать распределение среднего значения выборки, например вызвав центральную предельную теорему.
3. Исходя из статистики и ее распределения, мы можем вычислить *p*-значение, вероятность значения тестовой статистики как экстремального или более экстремального, чем наблюдаемое нами, при условии что нулевая гипотеза верна.
4. Если *p*-значение слишком мало, у нас есть веские доказательства против нулевой гипотезы.

Это называется *опровержением* нулевой гипотезы.

5. Если *p*-значение велико, у нас нет таких доказательств. Это называется *неспособностью опровергнуть* нулевую гипотезу.

Здесь есть один вопрос: когда *p*-значение «слишком мало»?

В этой книге мы следуем распространенному соглашению, согласно которому отвергаем нулевую гипотезу, когда $p < 0{,}05$, и не способны опровергнуть ее, когда $p > 0{,}05$. В статистической терминологии мы выбираем уровень значимости $\alpha = 0{,}05$, чтобы определить границу между убедительными и недостаточными доказательствами нулевой гипотезы.

Но в действительности ответ будет таким: «Как посмотреть». Выбранный вами уровень значимости зависит от вашей проблемной области. Обычный предел $p < 0{,}05$ подходит для многих проблем. В нашей работе данные особенно зашумлены, поэтому мы часто удовлетворяемся $p < 0{,}10$. Для тех, кто работает в зонах повышенного риска, может потребоваться $p < 0{,}01$ или $p < 0{,}001$.

В рецептах мы упоминаем, какие тесты содержат *p*-значение, чтобы вы могли сравнить его с выбранным вами уровнем значимости α. Мы сформулировали рецепты, чтобы помочь вам интерпретировать сравнение. Вот формулировка из рецепта 9.4, теста на независимость двух факторов.

Обычно *p*-значение меньше 0,05 указывает на то, что переменные, вероятно, не являются независимыми, в то время как *p*-значение, превышающее 0,05, не может предоставить такие доказательства.

Это лаконичный способ сказать:

- нулевая гипотеза состоит в том, что переменные являются независимыми;
- альтернативная гипотеза состоит в том, что переменные не являются независимыми;
- в случае $\alpha = 0{,}05$, если $p < 0{,}05$, мы отвергаем нулевую гипотезу, давая убедительные доказательства того, что переменные не являются независимыми; если $p > 0{,}05$, мы не можем ее опровергнуть;
- конечно, вы можете выбрать собственное значение α, в этом случае ваше решение об отклонении или неспособности сделать это может отличаться.

Помните, рецепт утверждает *неформальную интерпретацию* результатов теста, а не строгую математическую интерпретацию. Мы используем разговорный язык в надежде, что он приведет вас к практическому пониманию и применению теста. Если точная семантика проверки гипотез имеет решающее значение для вашей работы, мы настоятельно рекомендуем вам обратиться к ссылке, приведенной в разделе «См. также», или к одному из других отличных учебников по математической статистике.

Доверительные интервалы

Проверка гипотез – это хорошо понятная математическая процедура, но она может разочаровать. Во-первых, семантика – вещь хитрая. Тест не дает однозначного, полезного заключения. Вы можете получить веские доказательства против нулевой гипотезы, но и только. Во-вторых, он не дает вам число, только доказательства.

Если вам нужны цифры, используйте доверительные интервалы, которые ограничивают оценку параметра совокупности при данном уровне достоверности. Рецепты, приведенные в этой главе, могут рассчитать доверительные интервалы для средних значений, медиан и пропорций совокупности.

Например, в рецепте 9.9 мы рассчитываем 95%-ный доверительный интервал для среднего значения совокупности на основе выборочных данных. Интервал составляет $97{,}16 < \mu < 103{,}98$, а это означает, что существует 95%-ная вероятность того, что среднее значение совокупности, μ, находится между 97,16 и 103,98.

См. также

Статистическая терминология и условные обозначения могут различаться. Эта книга в целом следует соглашениям *Mathematical Statistics with Applications*, 6-е изд., Денниса Вакерли и соавт. (Duxbury Press). Мы рекомендуем эту книгу также для того, чтобы узнать больше о статистических тестах, описанных в данной главе.

9.1. Получение сводки данных

Задача

Вам нужна базовая статистическая сводка ваших данных.

Решение

Функция `summary` дает некоторую полезную статистику по векторам, матрицам, факторам и таблицам данных:

```
summary(vec)
#> Min. 1st Qu. Median Mean 3rd Qu. Max.
#> 0.0 0.5 1.0 1.6 1.9 33.0
```

Обсуждение

В решении показана сводка вектора. `1st Qu.` и `3rd Qu.` – это первый и третий квартили соответственно. Когда у вас и среднее значение, и медиана, это полезно, потому что вы можете быстро обнаружить асимметрию. Например, результат в решении показывает среднее значение, которое больше медианы, что указывает на возможный перекос вправо, как и следовало ожидать от логнормального распределения.

Сводка матрицы работает столбец за столбцом. Здесь мы видим сводку матрицы `mat` с тремя столбцами `Samp1`, `Samp2` и `Samp3`:

```
summary(mat)
#>       Samp1        Samp2            Samp3
#>  Min.    : 1.0   Min.    :-2.943  Min. : 0.04
#>  1st Qu. : 25.8  1st Qu. :-0.774  1st Qu.: 0.39
#>  Median  : 50.5  Median  :-0.052  Median : 0.85
#>  Mean    : 50.5  Mean    :-0.067  Mean : 1.60
#>  3rd Qu. : 75.2  3rd Qu. : 0.684  3rd Qu.: 2.12
#>  Max.    :100.0  Max.    : 2.150  Max. :13.18
```

Сводка фактора дает следующую информацию:

```
summary(fac)
#> Maybe No Yes
#> 38 32 30
```

Сводка символьного вектора довольно бесполезна, она дает только длину вектора:

```
summary(char)
#> Length Class Mode
#> 100 character character
```

Сводка таблицы данных включает в себя все эти функции. Она работает столбец за столбцом, давая соответствующую сводку в соответствии с типом столбца. Числовые значения получают статистическую сводку, а факторы подсчитываются (строки символов не участвуют в этом):

```
suburbs <- read_csv("./data/suburbs.txt")
summary(suburbs)
#>    city              county             state
#> Length:17          Length:17          Length:17
#> Class :character   Class :character   Class :character
#> Mode :character    Mode :character    Mode :character
#>
#>
#>
#>        pop
#> Min.     : 5428
#> 1st Qu.  : 72616
#> Median   : 83048
#> Mean     : 249770
#> 3rd Qu.  : 102746
#> Max.     : 2853114
```

«Сводка» списка довольно забавная: вы получаете тип данных каждого члена списка. Вот краткая сводка списка векторов:

```
summary(vec_list)
#>    Length  Class   Mode
#> x 100    -none-  numeric
#> y 100    -none-  numeric
#> z 100    -none-  character
```

Чтобы получить сводку данных в списке векторов, отобразите функцию `summary` в каждый элемент списка:

```
library(purrr)
map(vec_list, summary)
#> $x
#>    Min.  1st Qu.  Median  Mean 3rd Qu. Max.
#> -2.572  -0.686  -0.084  -0.043 0.660 2.413
#>
#> $y
#>    Min.  1st Qu.  Median  Mean 3rd Qu.  Max.
#> -1.752  -0.589  0.045   0.079 0.769   2.293
#>
#> $z
#> Length  Class   Mode
#>    100 character character
```

К сожалению, функция summary не вычисляет какую-либо меру изменчивости, например стандартное отклонение или медианное абсолютное отклонение. Это серьезный недостаток, поэтому сразу после вызова summary мы обычно вызываем функции sd или mad (среднее абсолютное отклонение).

См. также

См. рецепты 2.6 и 6.1.

9.2. Расчет относительных частот

Задача

Вы хотите посчитать относительную частоту определенных наблюдений в своей выборке.

Решение

Определите интересные наблюдения, используя логическое выражение; затем используйте функцию mean для вычисления доли наблюдений, которые она идентифицирует. Например, для вектора *x* можно найти относительную частоту положительных значений следующим образом:

```
mean(x > 3)
#> [1] 0.12
```

Логическое выражение, такое как *x* > 3, создает вектор логических значений (TRUE и FALSE), по одному на каждый элемент *x*. Функция mean преобразует эти значения в единицы и нули соответственно и вычисляет среднее. В результате получается доля значений, которые являются истинными, другими словами, относительная частота интересных значений. Например, в решении это относительная частота значений больше 3.

Концепция здесь довольно проста. Сложная часть – придумать подходящее логическое выражение. Вот некоторые примеры:

```
mean(lab == "NJ")
```

Доля значений lab, которые находятся в Нью-Джерси.

```
mean(after > before)
```

Доля наблюдений, для которых эффект увеличивается.

```
mean(abs(x-mean(x)) > 2*sd(x))
```

Доля наблюдений, которые превышают два стандартных отклонения от среднего значения.

```
mean(diff(ts) > 0)
```

Доля наблюдений во временном ряде, которые больше, чем в предыдущем наблюдении.

9.3. Представление факторов в виде таблицы и создание таблиц сопряженности

Задача

Вы хотите свести один из факторов в таблицу или построить таблицу сопряженности из нескольких факторов.

Решение

Функция `table` дает следующие цифры:

```
table(f1)
#> f1
#> a   b   c   d   e
#> 14  23  24  21  18
```

Она также может создавать таблицы сопряженности (таблицы кросс-табуляции) из двух или более факторов:

```
table(f1, f2)
#>    f2
#> f1  f   g   h
#> a   6   4   4
#> b   7   9   7
#> c   4   11  9
#> d   7   8   6
#> e   5   10  3
```

Эта функция также работает и с символами, а не только с факторами:

```
t1 <- sample(letters[9:11], 100, replace = TRUE)
table(t1)
#> t1
#> i   j   k
#> 20  40  40
```

Обсуждение

Функция `table` подсчитывает уровни одного фактора или символов, как, например, здесь (`initial` и `outcome` – это факторы):

```
set.seed(42)
initial <- factor(sample(c("Yes", "No", "Maybe"), 100, replace = TRUE))
outcome <- factor(sample(c("Pass", "Fail"), 100, replace = TRUE))

table(initial)
#> initial
#> Maybe No Yes
#> 39 31 30

table(outcome)
#> outcome
#> Fail Pass
#> 56 44
```

Большая сила этой функции заключается в создании таблиц сопряженности, также известных как таблицы кросс-табуляции. Каждая ячейка в таблице сопряженности подсчитывает, сколько раз встречалась эта строка/столбец:

```
table(initial, outcome)
#> outcome
#> initial Fail Pass
#> Maybe 23 16
#> No 20 11
#> Yes 13 17
```

Эта таблица показывает, что комбинация `initial = Yes` и `outcome = Fail` встречается 13 раз, комбинация `initial = Yes` и `outcome = Pass` встречается 17 раз и т. д.

См. также

Функция `xtabs` также может создавать таблицу сопряженности. У нее есть интерфейс формулы, который предпочитают некоторые.

9.4. Проверка категориальных переменных на независимость

Задача

У вас есть две категориальные переменные, представленные факторами. Вы хотите проверить их на независимость, используя тест хи-квадрат.

Решение

Используйте функцию `table` для создания таблицы сопряженности из двух факторов. Затем применяйте функцию `summary`, чтобы выполнить тест хи-квадрат таблицы сопряженности. В этом примере у нас есть два вектора значений факторов, которые мы создали в предыдущем рецепте:

```
summary(table(initial, outcome))
#> Number of cases in table: 100
#> Number of factors: 2
#> Test for independence of all factors:
#> Chisq = 3, df = 2, p-value = 0.2
```

Вывод включает в себя значение *p*. Обычно *p*-значение меньше 0,05 указывает на то, что переменные, вероятно, не являются независимыми, в то время как *p*-значение, превышающее 0,05, не может предоставить такие доказательства.

Обсуждение

В этом примере мы выполняем тест хи-квадрат для таблицы сопряженности из рецепта 9.3 и получаем *p*-значение, равное 0,2:

```
summary(table(initial, outcome))
#> Number of cases in table: 100
#> Number of factors: 2
#> Test for independence of all factors:
#> Chisq = 3, df = 2, p-value = 0.2
```

Большое *p*-значение указывает на то, что два фактора, `initial` и `outcome`, вероятно, независимы. В сущности, мы заключаем, что между переменными нет никакой связи. Это имеет смысл, так как данные этого примера были созданы простым построением случайных данных с использованием функции `sample` из предыдущего рецепта.

См. также

Функция `chisq.test` также может выполнить этот тест.

9.5. Расчет квантилей (и квартилей) набора данных

Задача

Имеется доля *f*. Вы хотите узнать соответствующий квантиль своих данных. То есть вы ищете наблюдение *x*, чтобы доля наблюдений ниже *x* была *f*.

Решение

Используйте функцию `quantile`. Второй аргумент – это доля *f*:

```
quantile(vec, 0.95)
#> 95%
#> 1.43
```

В случае с квартилями просто опустите второй аргумент:

```
quantile(vec)
#>      0%     25%     50%     75%    100%
#> -2.0247 -0.5915 -0.0693  0.4618 2.7019
```

Обсуждение

Предположим, что `vec` содержит 1000 наблюдений между 0 и 1. Функция `quantile` может сказать вам, какое наблюдение ограничивает нижние 5 % данных:

```
vec <- runif(1000)
quantile(vec, .05)
#> 5%
#> 0.0451
```

В документации по функции `quantile` второй аргумент называется «вероятностью», что естественно, когда мы думаем о вероятности как об относительной частоте.

В характерном для R стиле вторым аргументом может быть вектор вероятностей; в этом случае функция возвращает вектор соответствующих квантилей, по одному для каждой вероятности:

```
quantile(vec, c(.05, .95))
#>      5%     95%
#> 0.0451 0.9363
```

Это удобный способ определить средние 90 % (в данном случае) наблюдений.

Если вы опускаете вероятности, R предполагает, что вам нужны вероятности 0, 0,25, 0,50, 0,75 и 1,0 – другими словами, квартили:

```
quantile(vec)
#> 0% 25% 50% 75% 100%
#> 0.000405 0.235529 0.479543 0.737619 0.999379
```

Удивительно, но функция `quantile` реализует девять (да, девять) различных алгоритмов для вычисления квантилей. Изучите страницу справки, прежде чем предположить, что алгоритм по умолчанию – наиболее подходящий для вас вариант.

9.6. Инвертирование квантиля

Задача

У вас есть наблюдение *x* из ваших данных. Вы хотите узнать соответствующий квантиль.

То есть вы хотите знать, какая доля данных меньше *x*.

Решение

Предполагая, что ваши данные находятся в векторе `vec`, сравните данные с наблюдением, а затем используйте функцию `mean` для вычисления относительной частоты значений, меньших *x*, скажем 1.6, как в этом примере:

```
mean(vec < 1.6)
#> [1] 0.948
```

Обсуждение

Выражение `vec <x` сравнивает каждый элемент `vec` с `x` и возвращает вектор логических значений, где *n*-е логическое значение равно `TRUE`, если `vec [n] <x`. Функция mean преобразует эти логические значения в нули и единицы: `0` для `FALSE` и `1` для `TRUE`. Среднее значение всех этих единиц и нулей – это доля `vec`, которая меньше `x`, или обратный квантиль `x`.

См. также

Это применение общего подхода, описанного в рецепте 9.2.

9.7. Преобразование данных в *z*-оценки

Задача

У вас есть набор данных, и вы хотите рассчитать соответствующие *z*-оценки для всех элементов данных. (Иногда это называют *нормализацией* данных.)

Решение

Используйте функцию `scale`:

```
scale(x)
#> [,1]
#> [1,]  0.8701
#> [2,] -0.7133
#> [3,] -1.0503
```

```
#> [4,]  0.5790
#> [5,]  0.6324
#> [6,]  0.0991
#> [7,]  2.1495
#> [8,]  0.2481
#> [9,] -0.8155
#> [10,] -0.7341
#> attr(,"scaled:center")
#> [1] 2.42
#> attr(,"scaled:scale")
#> [1] 2.11
```

Она работает с векторами, матрицами и таблицами данных. В случае с вектором эта функция возвращает вектор нормализованных значений. В случае с матрицами и таблицами данных она нормализует каждый столбец независимо и возвращает столбцы нормализованных значений в матрице.

Обсуждение

Вы также можете нормализовать единственное значение *y* относительно набора данных *x*. Это можно сделать, используя векторизованные операции:

```
(y - mean(x)) / sd(x)
#> [1] -0.633
```

9.8. Проверка среднего значения выборки (*t*-критерий)

Задача

У вас есть выборка из генеральной совокупности. Учитывая эту выборку, вы хотите знать, может ли среднее значение совокупности разумно быть конкретным значением *m*.

Решение

Примените функцию `t.test` к выборке *x* с аргументом `mu = m`:

```
t.test(x, mu = m)
```

Вывод включает в себя значение *p*. Традиционно, если $p < 0{,}05$, то среднее значение совокупности вряд ли будет равно *m*, тогда как $p > 0{,}05$ не дает таких доказательств.

Если размер выборки *n* невелик, то основная совокупность должна быть распределена нормально, чтобы получить значимые результаты из *t*-критерия. Есть хорошее эмпирическое правило, согласно которому «невелик» означает $n < 30$.

Обсуждение

t-критерий является рабочей лошадкой статистики, и это одно из его основных применений: делать выводы о значении совокупности из выборки. В следующем примере моделируется выборка из нормальной совокупности со средним значением $\mu = 100$. В нем используется *t*-критерий, чтобы узнать, может ли среднее значение генеральной совокупности составлять 95, а функция `t.test` вычисляет *p*-значение, равное 0,005:

```
x <- rnorm(75, mean = 100, sd = 15)
t.test(x, mu = 95)
#>
#> One Sample t-test
#>
#> data: x
#> t = 3, df = 70, p-value = 0.005
#> alternative hypothesis: true mean is not equal to 95
#> 95 percent confidence interval:
#>    96.5  103.0
#> sample estimates:
#> mean of x
#>      99.7
```

p-значение невелико, поэтому маловероятно (на основе выборочных данных), что 95 – это среднее значение совокупности.

Неформально мы могли бы интерпретировать низкое *p*-значение следующим образом. Если бы среднее значение генеральной совокупности действительно составляло 95, тогда вероятность наблюдения нашей тестовой статистики ($t = 2{,}8898$ или что-то более экстремальное) была бы только 0,005. Это очень невероятно, но это то значение, которое мы наблюдали. Отсюда мы заключаем, что нулевая гипотеза неверна; следовательно, данные выборки не поддерживают утверждение, согласно которому среднее значение совокупности составляет 95.

В противоположность этому тестирование среднего значения 100 дает *p*-значение 0,9:

```
t.test(x, mu = 100)
#>
#> One Sample t-test
#>
#> data: x
#> t = -0.2, df = 70, p-value = 0.9
#> alternative hypothesis: true mean is not equal to 100
#> 95 percent confidence interval:
#>    96.5  103.0
#> sample estimates:
#> mean of x
#>      99.7
```

Большое *p*-значение указывает на то, что выборка согласуется с предположением, что среднее значение совокупности μ составляет 100. Говоря терминами статистики, данные не дают доказательств в отношении истинного среднего значения, равного 100.

Распространен случай проверки на среднее значение нуля. Если опустить аргумент `mu`, по умолчанию это будет 0.

См. также

Функция `t.test` – роскошная вещь. *См.* рецепты 9.9 и 9.15, где описываются другие варианты ее использования.

9.9. Формирование доверительного интервала для среднего значения

Задача

У вас есть выборка из генеральной совокупности. Учитывая эту выборку, вы хотите определить доверительный интервал для среднего значения совокупности.

Решение

Примените функцию `t.test` к своей выборке `x`:

```
t.test(x)
```

Выходные данные включают в себя доверительный интервал на уровне достоверности 95 %. Чтобы увидеть интервалы на других уровнях, используйте аргумент `conf.level`.

Как и в рецепте 9.8, если размер выборки n невелик, основная совокупность должна быть нормально распределена, чтобы существовал полноценный доверительный интервал. Опять же, есть хорошее эмпирическое правило, которое гласит, что «невелик» значит $n < 30$.

Обсуждение

Когда вы применяете функцию `t.test` к вектору, появляется много выходных данных, где находится доверительный интервал:

```
t.test(x)
#>
#> One Sample t-test
#>
#> data: x
#> t = 50, df = 50, p-value <2e-16
#> alternative hypothesis: true mean is not equal to 0
#> 95 percent confidence interval:
#> 94.2 101.5
#> sample estimates:
#> mean of x
#> 97.9
```

В этом примере доверительный интервал составляет приблизительно $94{,}2 < \mu < 101{,}5$, что иногда записывается просто как (94,2, 101,5).

Можно повысить уровень достоверности до 99 %, установив для `conf.level` значение `0.99`:

```
t.test(x, conf.level = 0.99)
#>
#> One Sample t-test
#>
#> data: x
#> t = 50, df = 50, p-value <2e-16
#> alternative hypothesis: true mean is not equal to 0
#> 99 percent confidence interval:
#> 92.9 102.8
```

```
#> sample estimates:
#> mean of x
#> 97.9
```

В результате этого изменения доверительный интервал расширяется до 92,9 < μ < 102,8.

9.10. Формирование доверительного интервала для медианы

Задача

У вас есть выборка данных, и вы хотите знать доверительный интервал для медианы.

Решение

Используйте функцию `wilcox.test`, установив для `conf.int` значение `TRUE`:

```
wilcox.test(x, conf.int = TRUE)
```

Вывод будет содержать доверительный интервал для медианы.

Обсуждение

Процедура вычисления доверительного интервала среднего значения хорошо определена и широко известна. К сожалению, это не относится к медиане. Существует несколько процедур для расчета доверительного интервала медианы. Ни одна из них не является «той самой процедурой», но критерий знаковых рангов Уилкоксона является довольно стандартным.

Функция `wilcox.test` реализует эту процедуру. В выводе заложен доверительный интервал 95 %, который в этом случае приблизительный (–0,102, 0,646):

```
wilcox.test(x, conf.int = TRUE)
#>
#> Wilcoxon signed rank test
#>
#> data: x
#> V = 200, p-value = 0.1
#> alternative hypothesis: true location is not equal to 0
#> 95 percent confidence interval:
#> -0.102 0.646
#> sample estimates:
#> (pseudo)median
#>          0.311
```

Можно изменить уровень достоверности, настроив `conf.level`, например `conf.level=0.99` или другие подобные значения.

Вывод также включает в себя нечто, именуемое *псевдомедианой*, определение которой дается на странице справки. Не думайте, что это то же самое, что и медиана; это разные вещи:

```
median(x)
#> [1] 0.314
```

См. также

Процедура бутстрэппинга также полезна для оценки доверительного интервала медианы.

См. рецепты 8.5 и 13.8.

9.11. Тестирование доли выборки

Проблема

У вас есть выборка значений из генеральной совокупности, состоящая из успехов и неудач.

Вы полагаете, что истинная совокупность успехов равна p, и хотите проверить эту гипотезу, используя данные выборки.

Решение

Используйте функцию `prop.test`. Предположим, что размер выборки равен n, и образец содержит x успехов:

```
prop.test(x, n, p)
```

Вывод включает в себя p-значение. Обычно p-значение менее 0,05 указывает на то, что истинная доля вряд ли будет равна p, тогда как p-значение, превышающее 0,05, не может предоставить такие доказательства.

Обсуждение

Предположим, в начале бейсбольного сезона вы столкнулись с каким-то крикливым фанатом клуба Chicago Cubs. Игроки клуба сыграли 20 игр и выиграли 11 из них, или 55 % своих игр. Основываясь на этих доказательствах, фанат «убежден», что они выиграют больше половины своих игр в этом году. Стоит ли быть таким уверенным?

Функция `prop.test` может оценить логику фаната. Здесь число наблюдений равно $n = 20$, количество успехов – $x = 11$, а p – истинная вероятность выигрыша. Мы хотим знать, разумно ли на основании этих данных делать вывод, что $p > 0{,}5$. Как правило, функция `prop.test` проверяет на $p \neq 0{,}05$, но мы можем проверить на $p > 0{,}5$, установив для `alternative` значение `"greater"`:

```
prop.test(11, 20, 0.5, alternative = "greater")
#>
#> 1-sample proportions test with continuity correction
#>
#> data: 11 out of 20, null probability 0.5
#> X-squared = 0.05, df = 1, p-value = 0.4
#> alternative hypothesis: true p is greater than 0.5
#> 95 percent confidence interval:
#> 0.35 1.00
#> sample estimates:
#> p
#> 0.55
```

Вывод `prop.test` показывает большое *p*-значение, равное 0,55, поэтому мы не можем отклонить нулевую гипотезу; то есть мы не можем разумно заключить, что *p* больше 1/2. Фанат Chicago Cubs чрезмерно уверен в себе, основываясь на слишком малом количестве данных. Нечему удивляться.

9.12. Формирование доверительного интервала для доли

Задача

У вас есть выборка значений из генеральной совокупности, состоящая из успехов и неудач.

Основываясь на данных выборки, вы хотите сформировать доверительный интервал для доли успехов.

Решение

Используйте функцию `prop.test`. Предположим, что размер выборки равен *n*, а выборка содержит *x* успехов:

```
prop.test(x, n)
```

Вывод функции включает в себя доверительный интервал для *p*.

Обсуждение

Мы подписываемся на хорошо подготовленную новостную рассылку, посвященную фондовому рынку, но она включает в себя раздел, предназначенный для определения акций, которые могут вырасти. Это делается путем поиска определенного шаблона в цене акций. Например, недавно сообщалось, что определенная акция следовала этому шаблону. Также сообщалось, что акции выросли в шесть раз после того, как последние девять раз имел место тот шаблон. Авторы пришли к выводу, что вероятность повторного роста акций составила 6/9, или 66,7 %.

Используя функцию `prop.test`, мы можем получить доверительный интервал для истинной доли раз, когда акции растут в соответствии с шаблоном. Здесь число наблюдений – $n = 9$, а количество успехов – $x = 6$. Выходные данные показывают доверительный интервал (0,309,0,910) при уровне достоверности 95 %:

```
prop.test(6, 9)
#> Warning in prop.test(6, 9): Chi-squared approximation may be incorrect
#>
#> 1-sample proportions test with continuity correction
#>
#> data: 6 out of 9, null probability 0.5
#> X-squared = 0.4, df = 1, p-value = 0.5
#> alternative hypothesis: true p is not equal to 0.5
#> 95 percent confidence interval:
#> 0.309 0.910
#> sample estimates:
#> p
#> 0.667
```

Авторы довольно глупы, утверждая, что вероятность роста акций составляет 66,7 %. Они могут вынудить своих читателей сделать очень плохую ставку.

По умолчанию функция `prop.test` вычисляет доверительный интервал на уровне достоверности 95 %.

Используйте аргумент `conf.level`, чтобы изменить уровень достоверности:

```
prop.test(x, n, p, conf.level = 0.99) # Уровень достоверности 99 %.
```

См. также

См. рецепт 9.11.

9.13. Проверка на нормальность

Задача

Вам нужен статистический тест, чтобы определить, получена ли ваша выборка данных из нормально распределенной совокупности.

Решение

Используйте функцию `shapiro.test`:

```
shapiro.test(x)
```

Вывод включает в себя p-значение. Традиционно $p < 0{,}05$ указывает на то, что совокупность, вероятно, не распределена нормально, в то время как $p > 0{,}05$ не предоставляет таких доказательств.

Обсуждение

В этом примере показано p-значение, равное 0,4 для `x`:

```
shapiro.test(x)
#>
#> Shapiro-Wilk normality test
#>
#> data: x
#> W = 1, p-value = 0.4
```

Большое p-значение указывает на то, что основная совокупность может быть нормально распределена.

В следующем примере показано очень маленькое p-значение для `y`, поэтому маловероятно, что эта выборка была получена из нормально распределенной совокупности:

```
shapiro.test(y)
#>
#> Shapiro-Wilk normality test
#>
#> data: y
#> W = 0.7, p-value = 7e-13
```

Мы выделили тест Шапиро–Уилка, потому что это стандартная функция R. Вы также можете установить пакет `nortest`, который целиком посвящен тестам на нормальность.

Этот пакет включает в себя следующие тесты:

- критерий Андерсона–Дарлинга (`ad.test`);
- критерий Крамера–фон Мизеса (`cvm.test`);
- критерий Лиллиефорса (`lillie.test`);
- критерий согласия Пирсона для проверки сложной гипотезы нормальности (`pearson.test`);
- критерий Шапиро–Франсиа (`sf.test`).

Проблема всех этих тестов – их нулевая гипотеза: все они предполагают, что совокупность нормально распределена, пока не доказано обратное. В результате совокупность должна быть явно ненормальной, прежде чем тест сообщит о небольшом *p*-значении, и вы можете отклонить эту нулевую гипотезу, что делает эти тесты достаточно консервативными, склонными к ошибкам на стороне нормальности.

Вместо того чтобы полагаться исключительно на статистический тест, мы предлагаем также использовать гистограммы (рецепт 10.19) и квантиль-квантиль графики (рецепт 10.21) для оценки нормальности любых данных. Хвосты слишком толстые? Вершина слишком острая? Ваше мнение, вероятно, будет лучше, чем отдельный статистический тест.

См. также

См. рецепт 3.10 для установки нормального пакета.

9.14. Тест последовательностей

Задача

Ваши данные представляют собой последовательность двоичных значений: да/нет, 0/1, истина/ложь или другие двузначные данные. Вы хотите знать: является ли последовательность случайной?

Решение

Пакет `tseries` содержит функцию `runs.test`, которая проверяет, является ли последовательность случайной. Последовательность должна быть фактором с двумя уровнями:

```
library(tseries)
runs.test(as.factor(s))
```

Функция `runs.test` вычисляет *p*-значение. Обычно *p*-значение менее 0,05 указывает на то, что последовательность, скорее всего, не является случайной, тогда как значение, превышающее 0,05, не дает таких доказательств.

Обсуждение

Серия (run) – это подпоследовательность, состоящая из одинаковых значений, таких как только единицы или только нули. Случайная последовательность

должна быть правильно перемешана, и в ней не должно быть слишком большого количества серий. Она также не должна содержать слишком мало серий – последовательность совершенно чередующихся значений (0, 1, 0, 1, 0, 1, …) не содержит серий, но вы бы назвали ее случайной?

Функция `runs.test` проверяет количество серий в вашей последовательности. Если их слишком много или слишком мало, она вычисляет маленькое *p*-значение.

В первом примере мы генерируем случайную последовательность нулей и единиц, а затем проверяем серии. Неудивительно, что функция `runs.test` вычисляет большое *p*-значение. Это указывает на то, что последовательность, скорее всего, является случайной:

```
s <- sample(c(0, 1), 100, replace = T)
runs.test(as.factor(s))
#>
#> Runs Test
#>
#> data: as.factor(s)
#> Standard Normal = 0.1, p-value = 0.9
#> alternative hypothesis: two.sided
```

Приведенная ниже последовательность, однако, состоит из трех серий, поэтому вычисляемое *p*-значение довольно низкое:

```
s <- c(0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0)
runs.test(as.factor(s))
#>
#> Runs Test
#>
#> data: as.factor(s)
#> Standard Normal = -2, p-value = 0.02
#> alternative hypothesis: two.sided
```

См. также

См. рецепты 5.4 и 8.6.

9.15. Сравнение средних значений двух выборок

Задача

У вас есть одна выборка из двух совокупностей. Вы хотите знать, могут ли две совокупности иметь одинаковое среднее значение.

Решение

Выполните *t*-тест, вызвав функцию `t.test`:

```
t.test(x, y)
```

По умолчанию эта функция предполагает, что ваши наблюдения не зависимы друг от друга. В противном случае (то есть если каждый x_i зависим от y_i) укажите для `paired` значение `TRUE`:

```
t.test(x, y, paired = TRUE)
```

В любом случае `t.test` вычислит *p*-значение. Традиционно, если $p < 0{,}05$, средние значения, вероятно, будут разные, тогда как $p > 0{,}05$ не дает таких доказательств:

- если размер выборки невелик, то совокупности должны быть нормально распределены. В данном случае «невелик» означает менее 20 точек данных;
- если у двух совокупностей одинаковая дисперсия, укажите для `var.equal` значение `TRUE`, чтобы получить менее консервативный тест.

Обсуждение

Мы часто используем *t*-критерий, чтобы быстро понять разницу между двумя средними значениями совокупностей. Для этого нужно, чтобы выборки были достаточно большими (то есть обе выборки имели 20 или более наблюдений) или чтобы основные совокупности были нормально распределены. Мы не воспринимаем фразу «нормально распределена» слишком буквально. Быть колоколообразным и разумно симметричным вполне достаточно.

Ключевым отличием здесь является то, содержат ваши данные зависимые наблюдения или нет, поскольку результаты в этих двух случаях могут быть разными. Предположим, мы хотим знать, улучшает ли употребление кофе по утрам результаты тестов SAT. Мы могли бы провести этот эксперимент двумя способами:

- случайным образом выбрать одну группу людей. Дайте им тест SAT дважды, один раз с утренним кофе и один раз без него. Для каждого участника у нас будет по два балла. Это зависимые наблюдения;
- случайно выберите две группы людей. Одна группа выпивает чашку утреннего кофе и выполняет SAT. Другая группа просто проходит тест. У нас есть балл для каждого участника, но эти баллы никак не связаны.

По статистике, эти эксперименты совершенно разные. В первом эксперименте есть два наблюдения для каждого человека (с кофеином и без), и они не являются статистически независимыми. Во втором эксперименте наблюдения являются независимыми.

Если у вас есть зависимые наблюдения (первый эксперимент) и вы по ошибке проанализировали их как наблюдения, которые не являются таковыми (второй эксперимент), то можете получить такой результат с *p*-значением 0,3:

```
load("./data/sat.rdata")
t.test(x, y)
#>
#> Welch Two Sample t-test
#>
#> data: x and y
#> t = -1, df = 200, p-value = 0.3
#> alternative hypothesis: true difference in means is not equal to 0
#> 95 percent confidence interval:
#> -46.4 16.2
#> sample estimates:
#> mean of x mean of y
#>      1054      1069
```

Большое *p*-значение заставляет вас сделать вывод, что между этими группами нет разницы. Сравните этот результат с тем, что следует из анализа тех же данных, но правильно идентифицируя их как зависимые:

```
t.test(x, y, paired = TRUE)
#>
#> Paired t-test
#>
#> data: x and y
#> t = -20, df = 100, p-value <2e-16
#> alternative hypothesis: true difference in means is not equal to 0
#> 95 percent confidence interval:
#> -16.8 -13.5
#> sample estimates:
#> mean of the differences
#>                   -15.1
```

p-значение резко падает до 2e – 16, и мы приходим к совершенно противоположному выводу.

См. также

Если совокупности не нормально распределены (в форме колокола) и все выборки маленькие, рассмотрите возможность использования критерия Уилкоксона–Манна–Уитни, описанного в рецепте 9.16.

9.16. Непараметрическое сравнение местоположений двух выборок

Задача

У вас есть выборки из двух совокупностей. Вам неизвестно распределение совокупностей, но вы знаете, что у них схожие формы. Вы хотите знать: смещена одна из совокупностей влево или вправо по сравнению с другой?

Решение

Можно использовать непараметрический критерий, критерий Уилкоксона–Манна–Уитни, который реализуется функцией `wilcox.test`. Для зависимых наблюдений (каждый x_i зависит от y_i) установите для `paired` значение `TRUE`:

```
wilcox.test(x, y, paired = TRUE)
```

Для независимых наблюдений оставьте для `paired` значение по умолчанию `FALSE`:

```
wilcox.test(x, y)
```

Результат теста включает в себя *p*-значение. Обычно *p*-значение менее 0,05 указывает на то, что вторая совокупность, вероятно, смещена влево или вправо относительно первой, тогда как *p*-значение, превышающее 0,05, не дает таких доказательств.

Обсуждение

Когда мы перестаем строить предположения относительно распределений совокупностей, мы попадаем в мир непараметрической статистики. Критерий Уил-

коксона–Манна–Уитни является непараметрическим, поэтому может применяться к бóльшему количеству наборов данных, чем *t*-критерий, который требует нормального распределения данных (для небольших выборок). Единственное предположение этого теста состоит в том, что у двух совокупностей схожая форма.

В этом рецепте мы задаемся вопросом: смещена вторая совокупность влево или вправо относительно первой? Это похоже на вопрос о том, является ли среднее второй совокупности меньше или больше, чем у первой. Однако критерий Уилкоксона–Манна–Уитни отвечает на другой вопрос: он говорит нам, значительно ли различаются центральные местоположения двух совокупностей или, что эквивалентно, различаются ли их относительные частоты.

Предположим, мы случайным образом выбираем группу сотрудников и просим каждого из них выполнить одну и ту же задачу при двух разных обстоятельствах: в благоприятных условиях и в неблагоприятных, например в шумной обстановке. Мы измеряем время завершения в обоих условиях, поэтому у нас есть два варианта измерения для каждого сотрудника. Мы хотим знать, значительно ли отличаются два этих периода времени, но мы не можем предположить, что они нормально распределены.

Наблюдения являются зависимыми, поэтому мы должны установить для `paired` значение `TRUE`:

```
load(file = "./data/workers.rdata")

wilcox.test(fav, unfav, paired = TRUE)
#>
#> Wilcoxon signed rank test
#>
#> data: fav and unfav
#> V = 10, p-value = 1e-04
#> alternative hypothesis: true location shift is not equal to 0
```

p-значение, по существу, равно нулю. Говоря статистически, мы отвергаем предположение, что время завершения в обоих случаях было одинаковым. На практике можно сделать вывод, что оно было разным.

В этом примере установка для `paired` значения `TRUE` является важной. Неверно относиться к этим данным как к независимым, поскольку данные наблюдения не являются таковыми, а это, в свою очередь, приведет к фиктивным результатам. Когда мы выполняем пример, где `paired=FALSE`, это дает нам *p*-значение, равное 0.1022, что приводит к неверному выводу.

См. также

См. рецепт 9.15, где приводится параметрический тест.

9.17. Проверка значимости корреляции

Задача

Вы рассчитали корреляцию между двумя величинами, но не знаете, является ли эта корреляция статистически значимой.

Решение

Функция `cor.test` может рассчитать и *p*-значение, и доверительный интервал корреляции. Если величины получены из нормально распределенных групп совокупностей, используйте показатель корреляции по умолчанию, а именно метод Пирсона:

```
cor.test(x, y)
```

В случае с ненормальными совокупностями используйте метод Спирмена:

```
cor.test(x, y, method = "spearman")
```

Функция `cor.test` возвращает несколько значений, включая *p*-значение из критерия значимости. Традиционно, $p < 0{,}05$ указывает на то, что корреляция, вероятно, является значимой, тогда как $p > 0{,}05$ указывает на то, что это не так.

Обсуждение

По нашему опыту люди часто не могут проверить значимость корреляции. На самом деле многие не знают, что корреляция может быть незначимой. Они запихивают свои данные в компьютер, вычисляют корреляцию и слепо верят результату. Однако они должны спросить себя: было ли достаточно данных? Достаточно ли велика корреляция? К счастью, функция `cor.test` дает ответы на эти вопросы.

Предположим, у нас есть два вектора, `x` и `y`, со значениями из нормальных совокупностей. Мы были бы очень рады, если бы их корреляция превышала 0,75:

```
cor(x, y)
#> [1] 0.751
```

Но это наивно. Если мы выполняем функцию `cor.test`, она сообщает об относительно большом *p*-значении, равном 0,09:

```
cor.test(x, y)
#>
#> Pearson's product-moment correlation
#>
#> data: x and y
#> t = 2, df = 4, p-value = 0.09
#> alternative hypothesis: true correlation is not equal to 0
#> 95 percent confidence interval:
#> -0.155 0.971
#> sample estimates:
#> cor
#> 0.751
```

p-значение выше обычного порога 0,05, поэтому мы заключаем, что корреляция вряд ли будет значимой.

Корреляцию также можно проверить, используя доверительный интервал. В этом примере доверительный интервал составляет (–0,155, 0,971). Интервал содержит ноль, поэтому возможно, что корреляция равна нулю, и в этом случае ее не будет. Опять же, вы не можете быть уверены, что сообщенная корреляция является значимой.

Вывод функции cor.test также включает в себя точечную оценку, которую сообщает функция cor (вывод после sample estimates), что избавляет вас от дополнительного шага, связанного с запуском cor.

По умолчанию функция cor.test рассчитывает корреляцию Пирсона, которая предполагает, что основные совокупности нормально распределены. Метод Спирмена не делает такого предположения, потому что он непараметрический. Установите для method значение "Spearman" при работе с ненормальными данными.

См. также

См. рецепт 2.6, чтобы узнать, как рассчитать простые корреляции.

9.18. Проверка групп на предмет наличия равных пропорций

Задача

У вас есть выборки из двух или более групп. Элементы групп имеют двоичные значения: либо успех, либо неудача. Вы хотите знать, имеют ли группы равные пропорции успехов.

Решение

Используйте функцию prop.test с двумя векторными аргументами:

```
ns <- c(48, 64)
nt <- c(100, 100)
prop.test(ns, nt)
#>
#> 2-sample test for equality of proportions with continuity
#> correction
#>
#> data: ns out of nt
#> X-squared = 5, df = 1, p-value = 0.03
#> alternative hypothesis: two.sided
#> 95 percent confidence interval:
#> -0.3058 -0.0142
#> sample estimates:
#> prop 1 prop 2
#>   0.48   0.64
```

Это параллельные векторы. Первый вектор, ns, дает количество успехов в каждой группе. Второй вектор, nt, дает размер соответствующей группы (это часто называют *количеством испытаний*).

Вывод включает в себя *p*-значение. Обычно *p*-значение менее 0,05 указывает на то, что, скорее всего, пропорции групп различны, тогда как *p*-значение, превышающее 0,05, не дает таких доказательств.

Обсуждение

В рецепте 9.11 мы проверили пропорцию на базе одной выборки. Здесь у нас выборки из нескольких групп, и мы хотим сравнить пропорции в основных группах.

Один из авторов недавно преподавал статистику 38 студентам и поставил отметку «очень хорошо» четырнадцати из них. Его коллега давал тот же урок

40 ученикам и поставил отметку «очень хорошо» только десяти. Мы хотели узнать: стимулировал ли автор обесценивание оценок, поставив значительно больше отметок «очень хорошо» по сравнению с другим преподавателем?

Мы использовали функцию `prop.test`. «Успех» (success) означает, что ученику поставили отметку «очень хорошо», поэтому вектор успехов содержит два элемента: число, которое поставил автор, и число, которое поставил его коллега:

```
successes <- c(14, 10)

trials - это количество учащихся в соответствующем классе:
trials <- c(38, 40)
```

Вывод функции `prop.test` дает *p*-значение 0,4:

```
prop.test(successes, trials)
#>
#> 2-sample test for equality of proportions with continuity
#> correction
#>
#> data: successes out of trials
#> X-squared = 0.8, df = 1, p-value = 0.4
#> alternative hypothesis: two.sided
#> 95 percent confidence interval:
#> -0.111 0.348
#> sample estimates:
#> prop 1 prop 2
#>  0.368   0.250
```

Относительно большое *p*-значение говорит, что мы не можем отвергнуть нулевую гипотезу: доказательства не предполагают каких-либо различий между оценками учителей.

См. также

См. рецепт 9.11.

9.19. Парные сравнения между средними значениями групп

Задача

У вас есть несколько выборок, и вы хотите выполнить парное сравнение их средних значений. То есть вы хотите сравнить среднее значение каждой выборки со средним значением любой другой выборки.

Решение

Поместите все данные в один вектор и создайте параллельный фактор для идентификации групп. Используйте функцию `pairwise.t.test` для выполнения парного сравнения средних значений:

```
pairwise.t.test(x, f) # x - это данные, f - фактор группировки.
```

Вывод содержит таблицу *p*-значений, по одному на каждую пару групп. Традиционно, если $p < 0{,}05$, то две группы, вероятно, имеют разные средние значения, тогда как $p > 0{,}05$ не дает таких доказательств.

Обсуждение

Этот пример сложнее, чем в рецепте 9.15, где мы сравнивали средние значения двух выборок. Здесь выборок несколько, и мы хотим сравнить среднее значение каждой выборки со средним значением всех других выборок.

Говоря статистически, парные сравнения – вещь непростая. Это не то же самое, что просто выполнить *t*-тест для каждой возможной пары. *p*-значения должны быть скорректированы, иначе вы получите слишком оптимистичный результат. На страницах, где содержится справочная информация по функциям `pairwise.t.test` и `p.adjust`, описаны алгоритмы настройки, доступные в R. Всем, кто выполняет серьезные парные сравнения, настоятельно рекомендуется просмотреть соответствующую справочную информацию и обратиться к хорошему учебнику.

Предположим, что мы используем большую выборку данных из рецепта 5.5, где мы объединили данные по первокурсникам, второкурсникам и третьекурсникам в таблицу данных с именем `comb`. У этой таблицы данных два столбца: данные в столбце `values` и фактор группировки в столбце `ind`. Мы можем использовать функцию `pairwise.t.test` для выполнения парных сравнений между группами:

```
pairwise.t.test(comb$values, comb$ind)
#>
#> Pairwise comparisons using t-tests with pooled SD
#>
#> data: comb$values and comb$ind
#>
#>      fresh soph
#> soph 0.001 -
#> jrs 3e-04 0.592
#>
#> P value adjustment method: holm
```

Обратите внимание на таблицу *p*-значений. Сравнение третьекурсников с первокурсниками и второкурсников с первокурсниками дает небольшие *p*-значения: 0,001 и 0,0003 соответственно. Можно заключить, что между этими группами есть существенные различия. Тем не менее сравнение второкурсников и третьекурсников дало (относительно) большое *p*-значение, 0,592, поэтому они отличаются незначительно.

См. также

См. рецепты 5.5 и 9.15.

9.20. Проверка двух выборок, чтобы определить, принадлежат ли они одному закону распределения

Задача

У вас есть две выборки, и вам интересно, принадлежат ли они одному закону распределения?

Решение

Критерий Колмогорова–Смирнова сравнивает две выборки и проверяет, принадлежат ли они одному и тому же закону распределения. Функция `ks.test` реализует этот критерий:

```
ks.test(x, y)
```

Вывод включает в себя *p*-значение. Обычно *p*-значение менее 0,05 указывает на то, что две выборки (*x* и *y*) принадлежат разным законам распределения, тогда как *p*-значение, превышающее 0,05, не дает таких доказательств.

Обсуждение

Критерий Колмогорова–Смирнова – замечательная вещь по двум причинам. Во-первых, это непараметрический критерий, поэтому вам не нужно делать никаких предположений относительно базовых распределений: он подходит для всех распределений. Во-вторых, он проверяет местоположение, дисперсию и форму совокупностей на основе выборок. Если эти характеристики не совпадают, критерий обнаружит это, что позволит вам сделать вывод, что базовые распределения отличаются.

Предположим, мы подозреваем, что векторы `x` и `y` принадлежат разным законам распределения. Здесь функция `ks.test` сообщает, что *p*-значение равно 0,04:

```
ks.test(x, y)
#>
#> Two-sample Kolmogorov-Smirnov test
#>
#> data: x and y
#> D = 0.2, p-value = 0.04
#> alternative hypothesis: two-sided
```

Исходя из небольшого *p*-значения, можно сделать вывод, что выборки принадлежат разным распределениям. Однако когда мы проверяем `x` на другой выборке, `z`, *p*-значение намного больше (0,6); это говорит о том, что `x` и `z` могут иметь одинаковое базовое распределение:

```
          z <- rnorm(100, mean = 4, sd = 6)
ks.test(x, z)
#>
#> Two-sample Kolmogorov-Smirnov test
#>
#> data: x and z
#> D = 0.1, p-value = 0.6
#> alternative hypothesis: two-sided
```

Глава 10

Графики

Графики – это мощная сила языка R. Пакет `graphics` является частью стандартного дистрибутива и содержит много полезных функций для создания разнообразных графических дисплеев. Базовая функциональность была расширена и улучшена с помощью `ggplot2`, являющегося частью пакетов коллекции tidyverse. В этой главе мы сосредоточимся на примерах использования `ggplot2`, а иногда будем предлагать и другие пакеты. В разделах «См. также» этой главы мы упоминаем функции из других пакетов, которые выполняют ту же работу иным способом. Мы предлагаем вам изучить эти альтернативы, если вы недовольны тем, что предлагает `ggplot2`, или базовые возможности.

Графики – обширная тема, и здесь мы можем лишь коснуться верхушки. Второе издание книги *R Graphics Cookbook* (http://shop.oreilly.com/product/0636920063704.do) Уинстона Чанга является частью серии книг рецептов от издательства O'Reilly. В ней содержится множество полезных рецептов с упором на `ggplot2`. Если вы хотите глубже вникнуть, мы рекомендуем книгу Пола Маррелла *R Graphics* (издательство *Chapman & Hall*); в ней обсуждаются парадигмы, лежащие в основе графиков R, и объясняется, как использовать функции для построения графиков. Книга содержит многочисленные примеры, включая код для их повторного создания. Некоторые примеры просто поразительны.

Иллюстрации

Графики, приведенные в этой главе, в большинстве своем простые и незамысловатые. Мы сделали их такими намеренно.

Когда вы вызываете функцию `ggplot`, как здесь:

```
library(tidyverse)

df <- data.frame(x = 1:5, y = 1:5)
ggplot(df, aes(x, y)) +
  geom_point()
```

вы получаете простое графическое представление `x` и `y`, как показано на рис. 10-1.

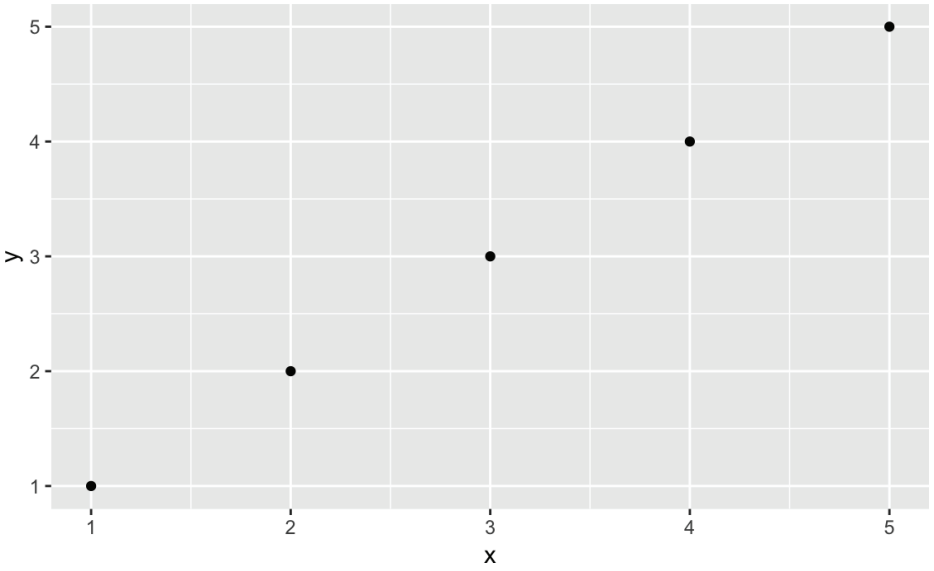


Рис. 10-1. Простой график

Можно сделать этот график цветным, украсить его заголовком, метками, условными обозначениями, текстом и т. д., но тогда вызов `ggplot` будет становиться все более и более перегруженным, скрывая основную цель:

```
ggplot(df, aes(x, y)) +
 geom_point() +
 labs(
   title = "Simple Plot Example",
   subtitle = "with a subtitle",
   x = "x-values",
   y = "y-values"
 ) +
 theme(panel.background = element_rect(fill = "white", color = "grey50"))
```

Полученный график показан на рис. 10-2. Мы хотим сохранить рецепты в чистоте, поэтому делаем акцент на базовом графике, а затем покажем (как в рецепте 10.2), как добавить украшения.

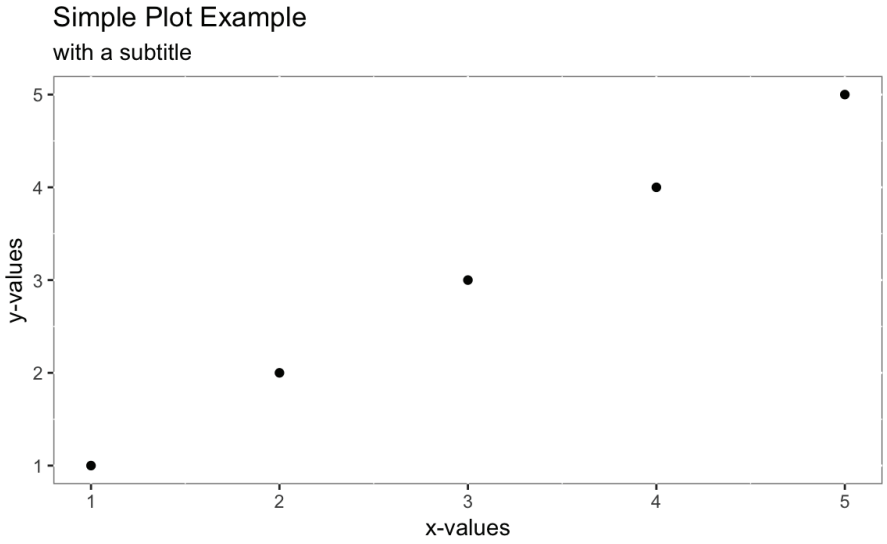


Рис. 10-2. Чуть более сложный график

Заметки по основам ggplot2

Хотя пакет носит название `ggplot2`, основная функция построения графиков из этого пакета называется `ggplot`. Важно понимать основные фрагменты графика, созданного с помощью этой функции. В предыдущих примерах видно, что мы передаем данные в `ggplot`, а затем определяем, как создается график, составляя вместе небольшие фразы, которые описывают некоторые аспекты графика. Это объединение фраз является частью «грамматики графиков» (вот откуда аббревиатура `gg`). Чтобы узнать больше, можете прочитать статью *A Layered Grammar of Graphics* (http://vita.had.co.nz/papers/layered-grammar.pdf), написанную автором `ggplot` Хэдли Уикхемом. Эта концепция возникла у Леланда Уилкинсона, который сформулировал идею построения графиков из набора примитивов (то есть глаголов и существительных). Используя `ggplot`, базовые данные не нужно существенно изменять для каждого типа графического представления. Как правило, данные остаются неизменными, а пользователь слегка меняет синтаксис, чтобы по-разному проиллюстрировать данные, что значительно более последовательно, по сравнению с базовыми графиками, которые часто требуют изменения данных для изменения способа их визуализации.

Поскольку мы говорим о графиках `ggplot`, стоит определить компоненты графика:

Функции геометрических объектов

Это геометрические объекты, которые описывают тип создаваемого графика. Их имена начинаются с `geom_`; в качестве примера можно упомянуть `geom_line`, `geom_boxplot` и `geom_point`, а также десятки других.

Эстетика

Эстетика, или эстетическое отображение, сообщает `ggplot`, какие поля в исходных данных в какие визуальные элементы на графике отображаются. Это строка `aes` в вызове `ggplot`.

Статистические преобразования

Это статистические преобразования, которые выполняются перед показом данных. Они будут не у всех графиков, но среди нескольких распространенных из них можно упомянуть: `stat_ecdf` (эмпирическая функция интегрального распределения) и `stat_identity`, которая говорит `ggplot` передать данные без какой-либо статистических преобразований вообще.

Панельные функции

Панели – это вспомогательные участки, где каждый небольшой график представляет подгруппу данных. Панельные функции включают в себя `facet_wrap` и `facet_grid`.

Темы

Темы – это визуальные элементы графика, которые не привязаны к данным. Они могут включать в себя заголовки, поля, таблицы расположений содержимого или варианты выбора шрифтов.

Слой

Слой – это сочетание данных, эстетики, геометрического объекта, статистических преобразований и других параметров для создания визуального слоя в графике `ggplot`.

«Длинные» и «широкие» данные с помощью ggplot

Одним из первых источников путаницы для новых пользователей `ggplot` является то, что они склонны изменять свои данные, чтобы они стали «широкими», прежде чем наносить их на график. Слово «широкий» здесь означает, что каждая переменная, которая наносится на график, является собственным столбцом в базовой таблице данных. Это подход, который многие пользователи вырабатывают при использовании Excel, а затем привносят его в R. `ggplot` проще всего работать с «длинными» данными, где дополнительные переменные добавляются в виде строк в таблице данных, а не в столбцах. Сильным побочным эффектом добавления большего количества измерений в виде строк является то, что любые правильно построенные графики `ggplot` будут автоматически обновляться для отражения новых данных без изменения кода `ggplot`. Если бы каждая дополнительная переменная была добавлена в виде столбца, пришлось бы изменить код построения графика, чтобы ввести дополнительные переменные. Эта идея «длинных» и «широких» данных станет более очевидной в примерах, которые приводятся в оставшейся части данной главы.

R – легко программируемый язык, и его механизм построения графиков был расширен дополнительными функциями. Довольно часто пакеты включают в себя специальные функции для построения своих результатов и объектов. Например, пакет `zoo` реализует объект временного ряда. Если вы создаете объект `z` и вызываете `plot(z)`, то пакет `zoo` выполняет построение графика; он создает график, настроенный для отображения временного ряда. `zoo` использует базовые графики, поэтому полученный график не будет графиком `ggplot`.

Есть даже целые пакеты, посвященные расширению R с помощью новых парадигм, относящихся к созданию графиков. Пакет `lattice` является альтернативой базовым графикам, предшествующей `ggplot2`. Он использует мощную парадигму,

которая позволяет легче создавать информативные графики. Он был реализован Дипаяном Саркаром, который также является автором книги *Lattice: Multivariate Data Visualization with R* (издательство Springer), где подробно рассказывается об этом пакете и о том, как его использовать. Пакет `lattice` также описан в книге *R in a Nutshell* (O'Reilly).

В превосходной книге Хэдли Уикхэма и Гаррета Гроулмунда *R for Data Science* есть две главы, посвященные графикам. В главе 7 «Разведочный анализ данных» основное внимание уделяется изучению данных с помощью `ggplot2`, а в главе 28 «Графики для общения» рассматривается общение с помощью графиков. Данная книга доступна в печатном виде или онлайн (https://r4ds.had.co.nz).

10.1. Создание точечной диаграммы

Задача

У вас есть зависимые наблюдения: (x_1, y_1), (x_2, y_2), …, (x_n, y_n). Вы хотите создать точечную диаграмму этих пар.

Решение

Мы можем отобразить данные, вызвав `ggplot`, передав таблицу данных и вызвав функцию геометрической точки:

```
ggplot(df, aes(x, y)) +
  geom_point()
```

В этом примере таблица данных называется `df`, а данные *x* и *y* находятся в полях с именами `x` и `y`, которые мы передаем в вызове `aes(x,y)`.

Обсуждение

Точечная диаграмма – это обычная первая атака на новый набор данных. Это быстрый способ увидеть связи, если таковые имеются, между *x* и *y*.

Для построения графиков с помощью `ggplot` необходимо указать ему, какую таблицу данных использовать, а затем какой тип графика создавать и какое эстетическое отображение (`aes`) использовать. `aes` в этом случае определяет, какое поле из `df` в какую ось ходит на графике. Затем команда `geom_point` сообщает, что вам нужен точечный график, а не линейный или другой тип.

Можно использовать встроенный набор данных `mtcars`, чтобы проиллюстрировать создание диаграммы лошадиных сил (`hp`) по оси *x* и экономии топлива (`mpg`) по оси *y*:

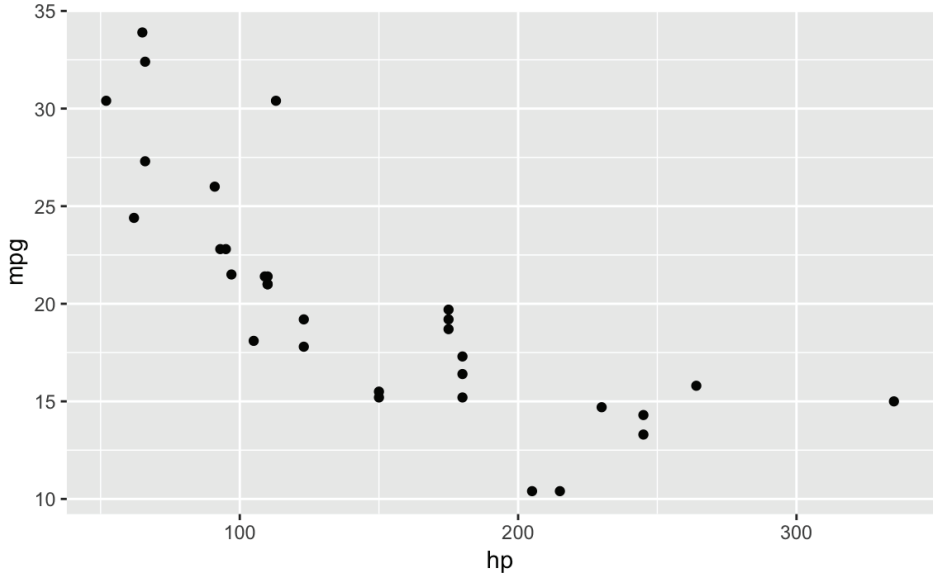


Рис. 10-3. Точечная диаграмма

См. также

См. рецепт 10.2, чтобы узнать, как добавить заголовок и метки, рецепт 10.3, чтобы узнать, как добавить сетку, и рецепт 10.6, чтобы узнать, как добавить условные обозначения. *См.* рецепт 10.8, чтобы узнать, как нанести на график несколько переменных.

10.2. Добавление заголовка и меток

Задача

Вам нужно добавить заголовок к своему графику или добавить метки для осей.

Решение

С помощью `ggplot` мы добавляем элемент `labs`, который контролирует метки для заголовка и осей.

При вызове `labs` в `ggplot` укажите:

`title`

Желаемый текст заголовка.

`x`

Метка оси *x*.

`y`

Метка оси *y*.

Например:

```
ggplot(df, aes(x, y)) +
  geom_point() +
  labs(title = "The Title",
       x = "X-axis Label",
       y = "Y-axis Label")
```

Обсуждение

График, созданный в рецепте 10.1, довольно простой. Заголовок и более подходящие метки сделают его интереснее и проще для интерпретации.

Обратите внимание, что в `ggplot` вы создаете элементы графика, соединяя части знаком плюс, +. Итак, мы добавляем дополнительные элементы, связывая фразы воедино. Это можно увидеть в приведенном ниже коде, где используется встроенный набор данных `mtcars` и отображается мощность в лошадиных силах и расход топлива на точечной диаграмме, показанной на рис. 10-4.

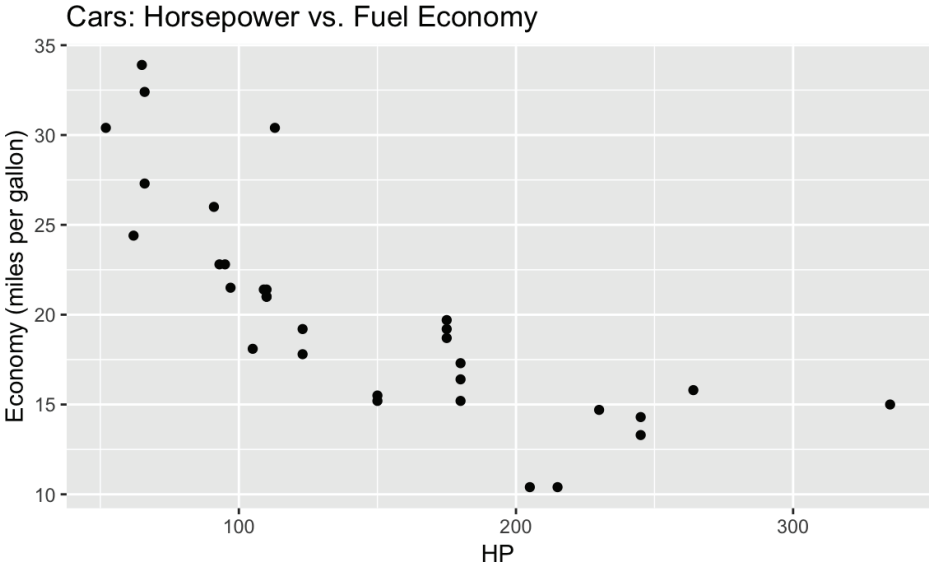


Рис. 10-4. Помеченные оси и заголовок

10.3. Добавление (или удаление) координатной сетки

Задача

Вы хотите изменить фоновую сетку своего графика.

Решение

В `ggplot` фоновые сетки идут по умолчанию, как вы видели в предыдущих рецептах. Однако можно изменить фоновую сетку, используя функцию `theme` или применив предварительно упакованную тему к нашему графику.

Мы можем использовать функцию `theme`, чтобы изменить фоновую панель нашего графика. В этом примере мы удаляем ее, как показано на рис. 10-5:

```
ggplot(df) +
  geom_point(aes(x, y)) +
  theme(panel.background = element_rect(fill = "white", color = "grey50"))
```

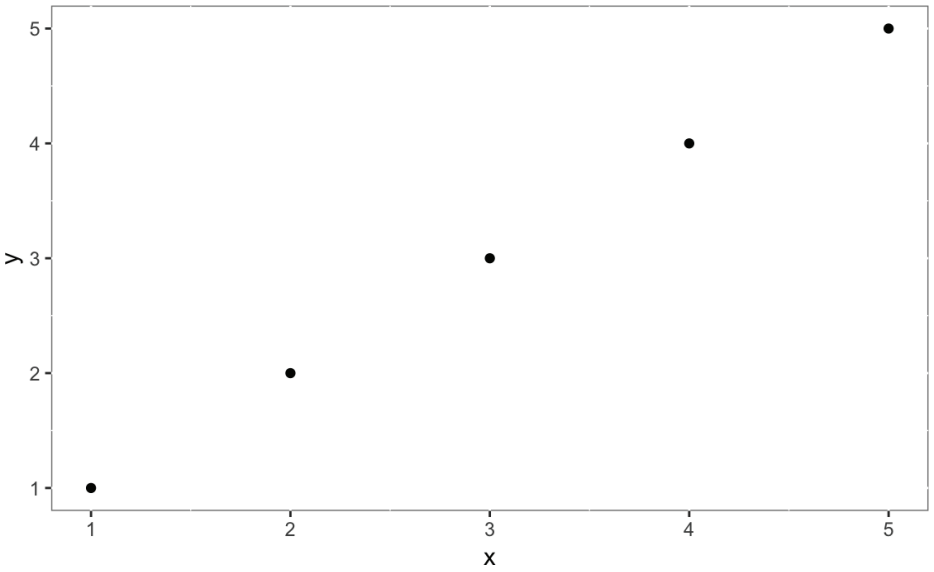


Рис. 10-5. Белый фон

Обсуждение

По умолчанию `ggplot` заполняет фон серой сеткой. У вас может возникнуть желание полностью удалить эту сетку или заменить ее на что-то другое. Давайте создадим график, а затем постепенно изменим стиль фона.

Мы можем добавлять или изменять внешний вид нашего графика, создавая объект `ggplot`, а затем вызывая этот объект и используя +, чтобы выполнить добавление. Затенение фона в графике на самом деле представляет собой три разных элемента графика:

`panel.grid.major`

Основная сетка по умолчанию белая и жирная.

`panel.grid.minor`

Вспомогательная сетка по умолчанию белая и светлая.

`panel.background`

Фон серый по умолчанию.

Можно увидеть эти элементы, если внимательно посмотреть на фон на рис. 10-4.

Если мы установим фон как `element_blank`, тогда основная и вспомогательная сетки по-прежнему будут там, но они будут белыми на белом фоне, поэтому на рис. 10-6 их не видно:

```
g1 <- ggplot(mtcars, aes(hp, mpg)) +
  geom_point() +
  labs(title = "Cars: Horsepower vs. Fuel Economy",
       x = "HP",
       y = "Economy (miles per gallon)") +
  theme(panel.background = element_blank())
g1
```

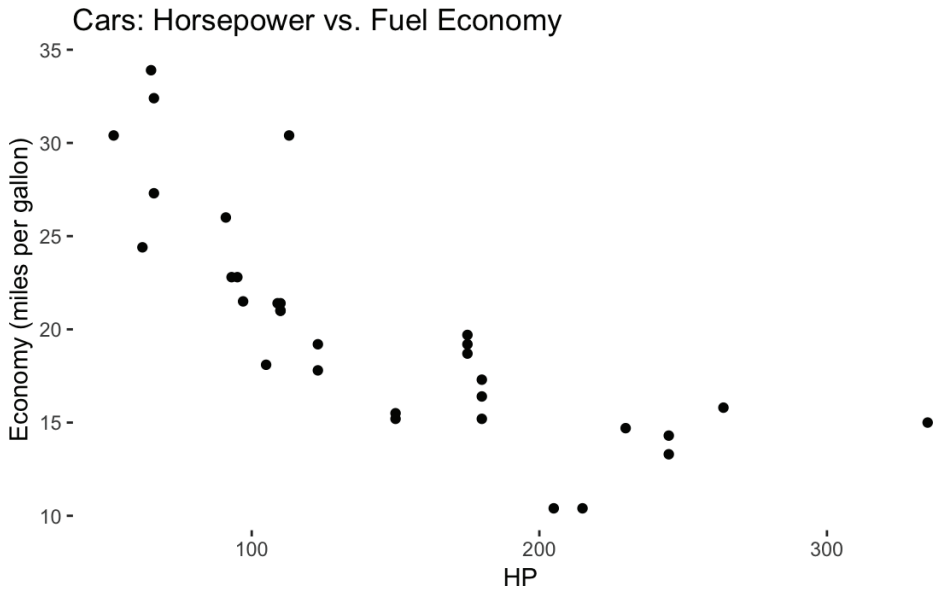


Рис. 10-6. Пустой фон

Обратите внимание, что в предыдущем коде мы помещаем график в переменную с именем `g1`. Затем мы вывели на экран график, просто вызвав эту переменную. Наличие графика внутри `g1` означает, что мы можем добавить дополнительные компоненты, не перестраивая график.

Если бы мы хотели показать фоновую сетку с необычными узорами для иллюстрации, это так же просто, как установить цвет ее компонентов и тип линии, как в этом примере (*см.* рис. 10-7):

```
g2 <- g1 + theme(panel.grid.major =
                   element_line(color = "black", linetype = 3)) +
  # linetype = 3 - это пунктир.
  theme(panel.grid.minor =
          element_line(color = "darkgrey", linetype = 4))
  # linetype = 4 - это штрихпунктир.
g2
```

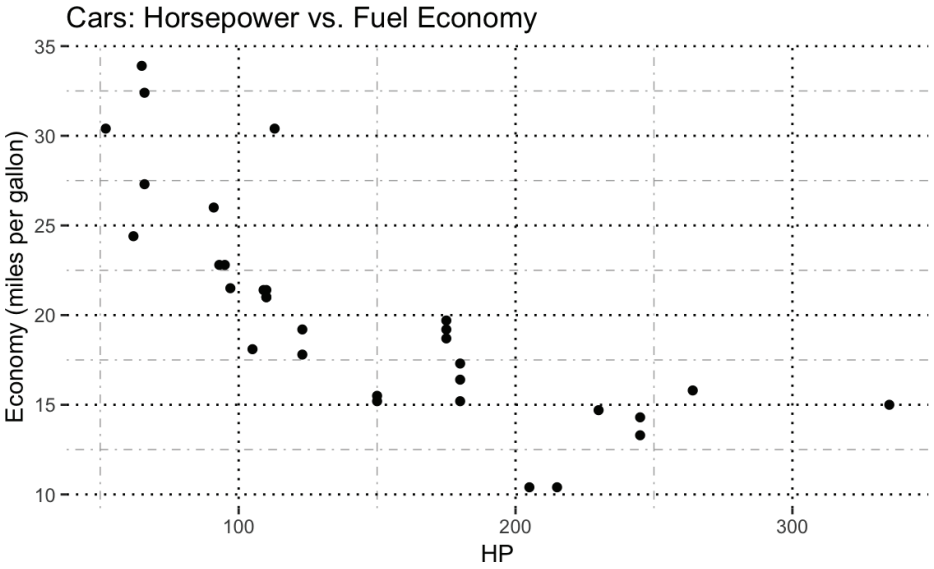


Рис. 10-7. Основные и вспомогательные линии координатной сетки

Рисунку 10-7 недостает визуальной привлекательности, но ясно видно, что точечные черные линии образуют основную сетку, а пунктирные серые линии – вспомогательную.

Или мы могли бы сделать что-то менее броское, взять ранее созданный объект `g1` и добавить серые линии сетки на белый фон, как показано на рис. 10-8:

```
g1 +
  theme(panel.grid.major = element_line(color = "grey"))
```

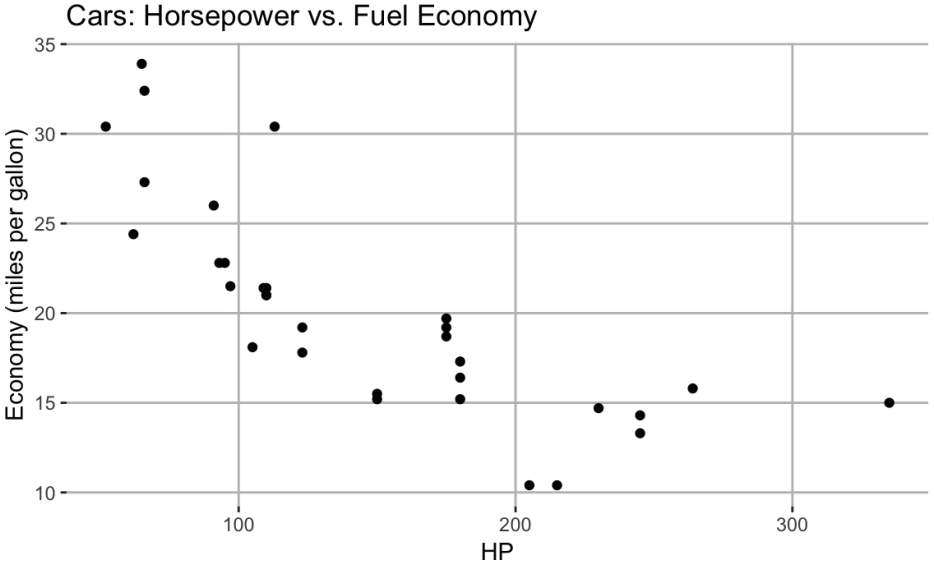


Рис. 10-8. Основные линии координатной сетки серого цвета

См. также

См. рецепт 10.4, чтобы увидеть, как применить полностью подготовленную тему к своему графику.

10.4. Применение темы к графику ggplot

Задача

Вы хотите, чтобы ваш график использовал предустановленную коллекцию цветов, стилей и форматирования.

Решение

`ggplot` поддерживает темы, которые представляют собой коллекции настроек для ваших диаграмм. Чтобы использовать одну из тем, просто добавьте нужную функцию `theme` с помощью знака +:

```
ggplot(df, aes(x, y)) +
  geom_point() +
  theme_bw()
```

Пакет `ggplot2` содержит следующие темы:

```
theme_bw()
theme_dark()
theme_classic()
theme_gray()
theme_linedraw()
theme_light()
theme_minimal()
theme_test()
theme_void()
```

Обсуждение

Давайте начнем с простой диаграммы, а затем покажем, как она выглядит, с помощью нескольких встроенных тем. На рис. 10-9 показан базовый вариант без темы:

```
p <- ggplot(mtcars, aes(x = disp, y = hp)) +
  geom_point() +
  labs(title = "mtcars: Displacement vs. Horsepower",
       x = "Displacement (cubic inches)",
       y = "Horsepower")
p
```

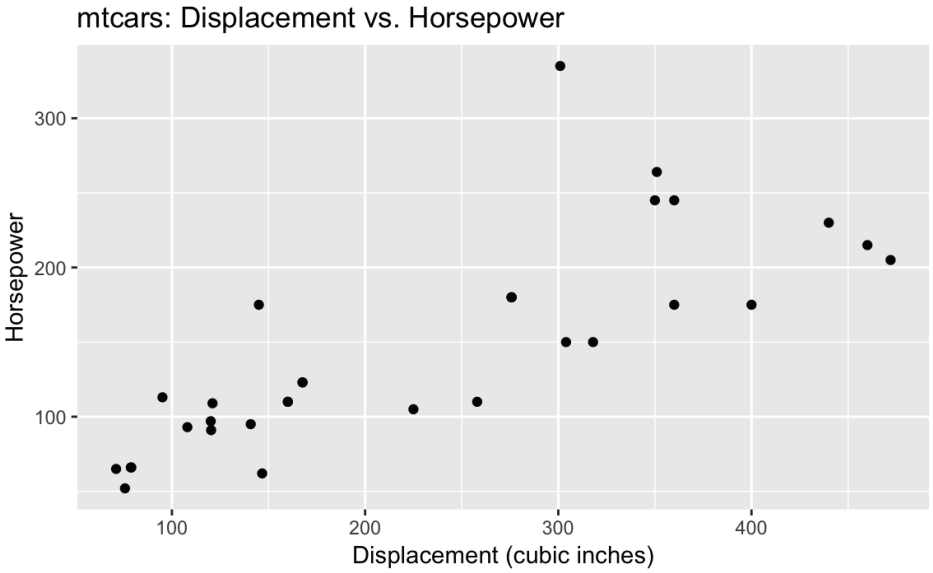


Рис. 10-9. Начальный график

Давайте создадим один и тот же график несколько раз, применяя каждый раз разные темы.

На рис. 10-10 показан внешний вид графика с применением черно-белой темы:

```
p + theme_bw()
```

А на рис. 10-11 показана классическая тема:

```
p + theme_classic()
```

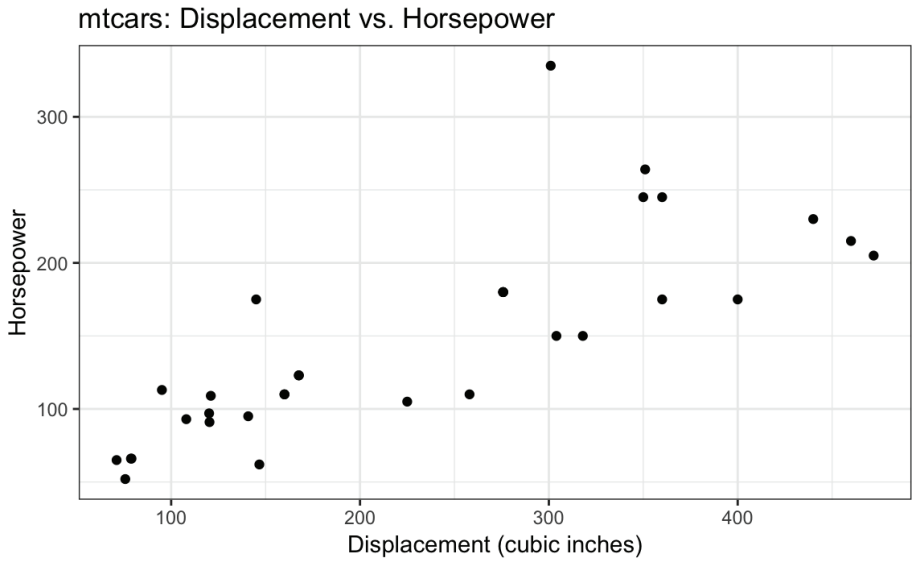


Рис. 10-10. theme_bw

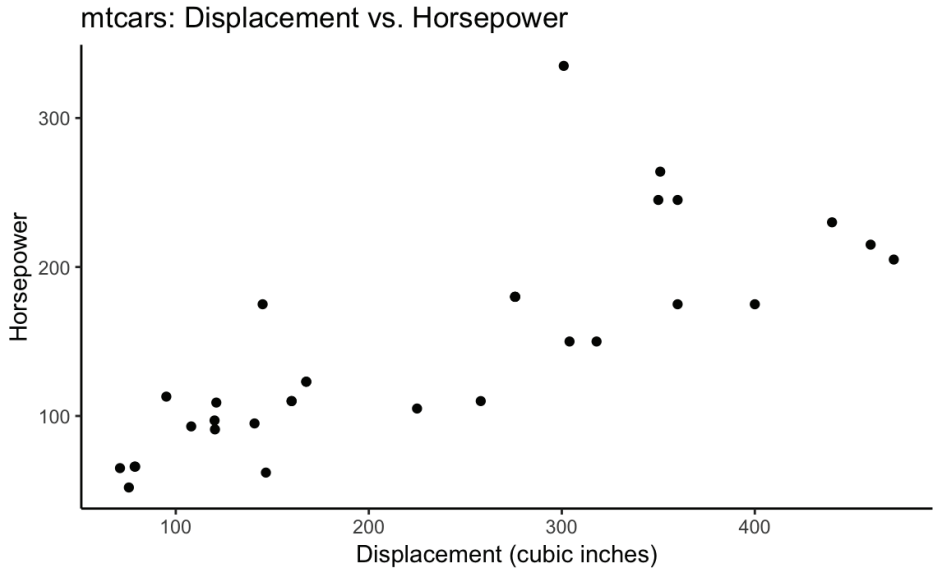


Рис. 10-11. theme_classic

На рис. 10-12 показана минимальная тема:

```
p + theme_minimal()
```

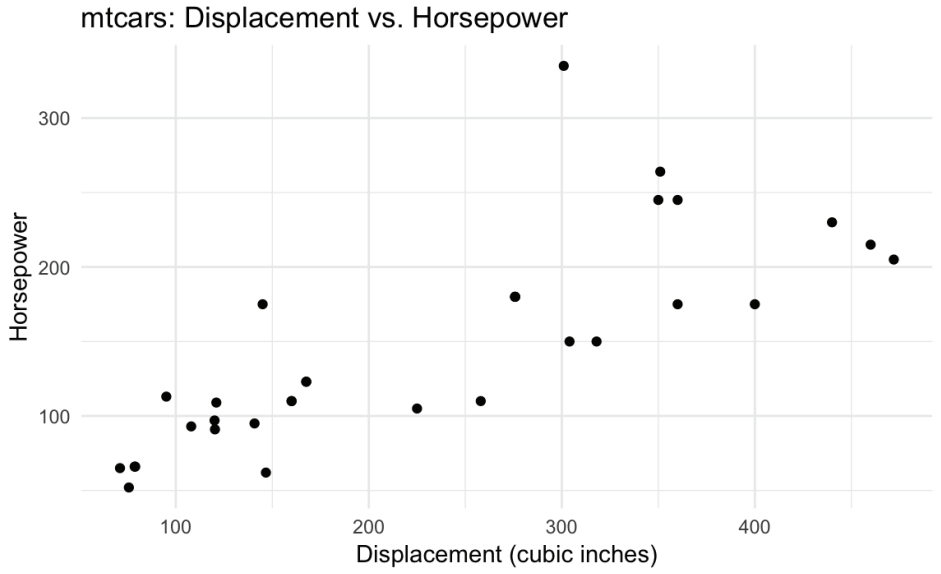


Рис. 10-12. theme_minimal

А на рис. 10-13 – пустая тема:

```
p + theme_void()
```

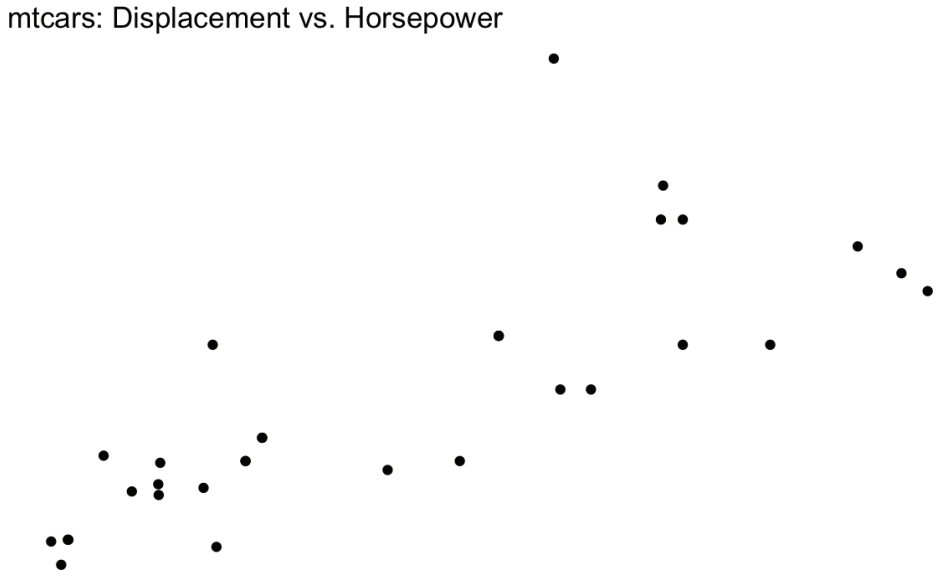


Рис. 10-13. theme_void

В дополнение к темам, которые включены в `ggplot2`, существуют такие пакеты, как `ggtheme`, содержащие темы, которые помогут вам сделать ваши диаграммы более похожими на те, что можно встретить в популярных инструментах и изданиях, таких как Stata или *The Economist*.

См. также

См. рецепт 10.3, чтобы узнать, как изменить один элемент темы.

10.5. Создание точечной диаграммы из нескольких групп

Задача

У вас есть данные в таблице данных с несколькими наблюдениями на запись: *x*, *y* и фактор *f*, который указывает группу. Вы хотите создать точечную диаграмму *x* и *y*, которая различает группы.

Решение

С помощью `ggplot` мы контролируем отображение фигур в фактор `f`, передавая `shape = f` функции `aes`:

```
ggplot(df, aes(x, y, shape = f)) +
  geom_point()
```

Обсуждение

Построение нескольких групп в одной точечной диаграмме приводит к бесполезному беспорядку, если мы не будем отличать одну группу от другой. Это различие осуществляется в `ggplot` путем настройки параметра `shape` функции `aes`.

Встроенный набор данных `iris` содержит зависимые меры `Petal.Length` и `Petal.Width`. Каждое измерение также имеет свойство `Species`, указывающее на вид цветка, который был измерен. Если мы нанесем все данные на график за один раз, то просто получим точечную диаграмму, изображенную на рис. 10-14:

```
ggplot(data = iris,
       aes(x = Petal.Length,
           y = Petal.Width)) +
 geom_point()
```

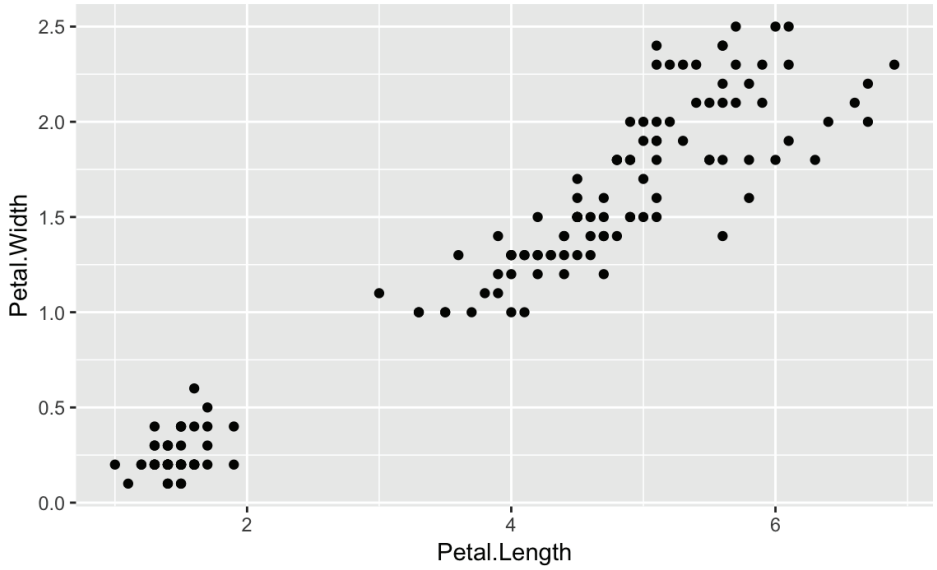


Рис. 10-14. Набор данных iris: длина и ширина

График был бы гораздо более информативным, если бы мы различали точки по видам. В дополнение к различению видов по форме мы также можем различать их по цвету. Можно добавить в вызов функции `aes shape = Species` и `color = Species`, чтобы получить каждый вид с разной формой и цветом, как показано на рис. 10-15:

```
ggplot(data = iris,
     aes(
        x = Petal.Length,
        y = Petal.Width,
      shape = Species,
       color = Species
      )) +
 geom_point()
```

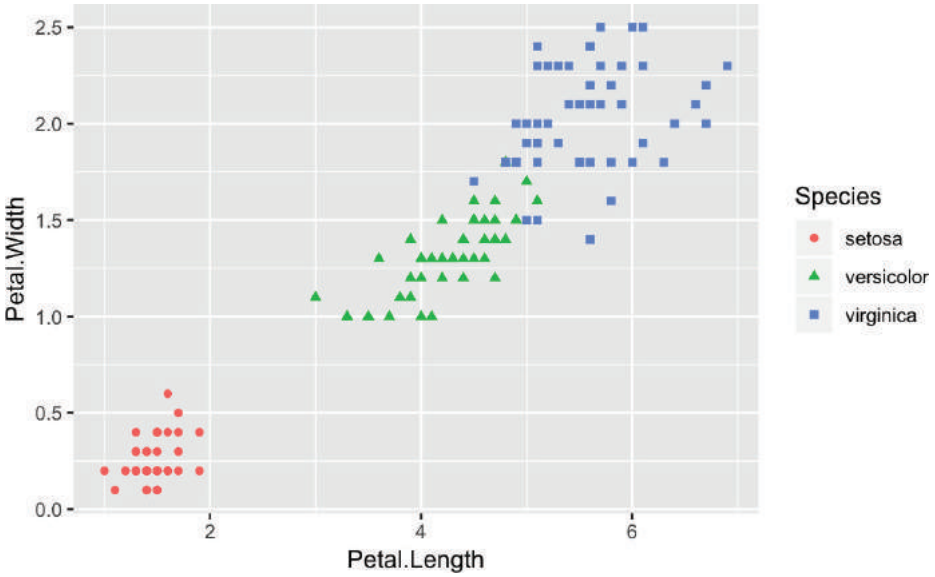


Рис. 10-15. Набор данных iris: форма и цвет

`ggplot` также с легкостью создает за вас условные обозначения, что очень удобно.

См. также

См. рецепт 10.6 для получения дополнительной информации о том, как добавить условные обозначения.

10.6. Добавление (или удаление) условных обозначений

Задача

Вы хотите, чтобы ваша диаграмма содержала *условные обозначения*, маленькие окошки, которые расшифровывают график для зрителя.

Решение

В большинстве случаев `ggplot` будет добавлять условные обозначения автоматически, как видно из предыдущего рецепта. Но если у нас нет явной группировки в функции `aes`, `ggplot` не будет показывать их по умолчанию. Если мы хотим заставить `ggplot` показывать условные обозначения, можно установить константу формы или типа линии нашей диаграммы. Затем `ggplot` покажет условные обозначения с одной группой. Мы используем функцию `guides`, чтобы направлять `ggplot` относительно того, как их подписывать.

Это можно проиллюстрировать на нашей точечной диаграмме `iris`:

```
g <- ggplot(data = iris,
      aes(x = Petal.Length,
          y = Petal.Width,
          shape="Observation")) +
  geom_point() +
  guides(shape=guide_legend(title="My Legend Title"))
g
```

На рис. 10-16 показан результат установки для формы строкового значения и последующего повторного подписывания условных обозначений с помощью функции `guides`:

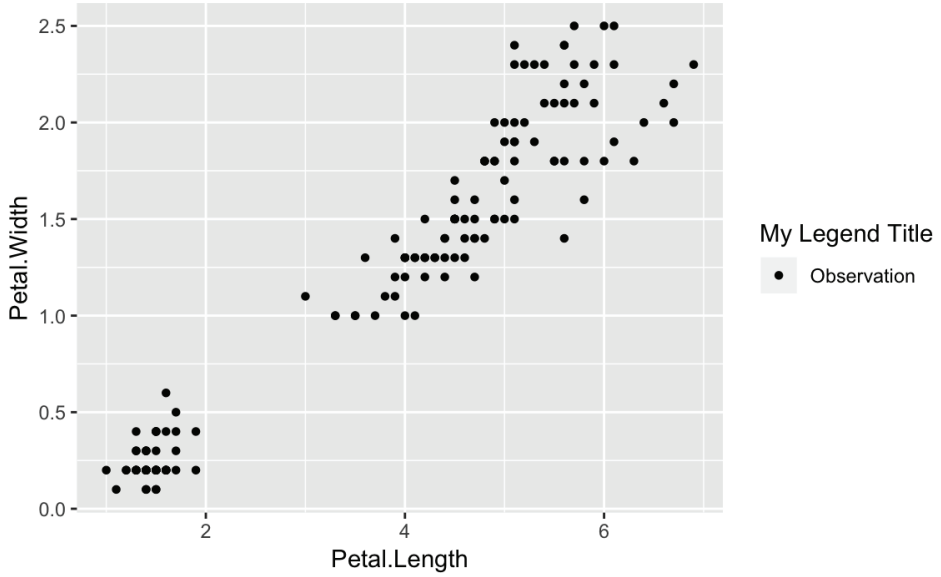


Рис. 10-16. Условные обозначения добавлены

Чаще всего у вас может возникнуть желание отключить условные обозначения, что можно сделать, вызвав функцию `theme` с `legend.position = "none"`. На рис. 10-17 показан результат, когда мы добавляем этот вызов к диаграмме `iris` из предыдущего рецепта:

```
g <- ggplot(data = iris,
           aes(
             x = Petal.Length,
            y = Petal.Width,
            shape = Species,
            color = Species
          )) +
  geom_point() +
  theme(legend.position = "none")
g
```

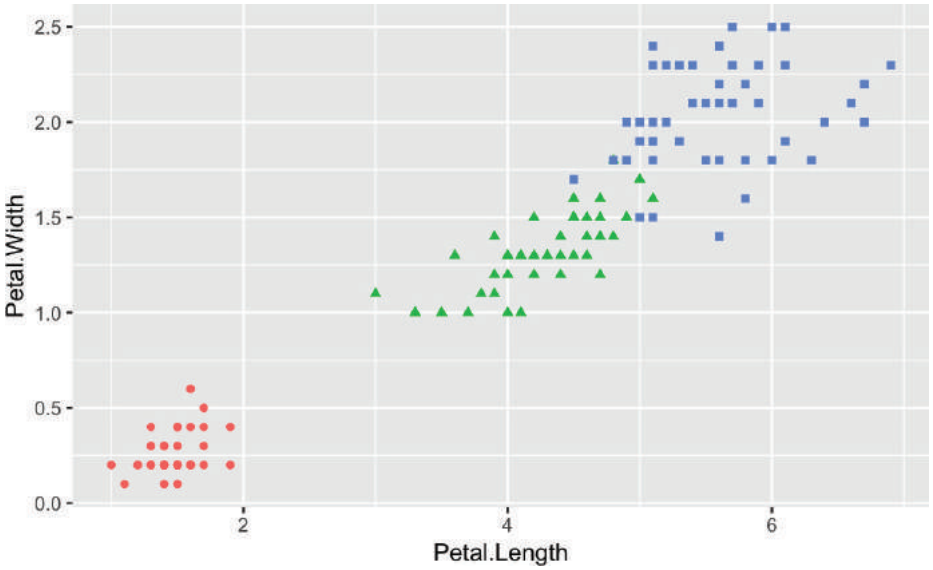


Рис. 10-17. Условные обозначения удалены

Обсуждение

Добавление условных обозначений в `ggplot`, когда нет группировки, – упражнение в «обмане» `ggplot`, чтобы показать эти обозначения, передав строку в параметр группировки в функции `aes`. Хотя это не изменит группировку (поскольку существует только одна группа), это приведет к отображению условных обозначений с именем.

Затем можно использовать функцию `guides`, чтобы изменить заголовок условных обозначений. Стоит отметить, что мы ничего не меняем в данных, просто используем настройки, чтобы заставить `ggplot` показывать условные обозначения, когда в обычной ситуации они бы не были показаны. Одно из огромных преимуществ `ggplot` – очень хорошие значения по умолчанию. Получение позиций и соответствия между метками и их типами точек выполняется автоматически, но при необходимости может быть отменено. Чтобы полностью удалить условные обозначения, мы устанавливаем параметры функции `theme` как `theme(legend.position = "none")`. Мы также можем установить для `legend.position` значения `"left"`, `"right"`, `"bottom"`, `"top"` или двухэлементный числовой вектор. Используйте двухэлементный числовой вектор, чтобы передать `ggplot` конкретные координаты, где вам нужны условные обозначения. Если вы используете координаты, переданные значения находятся в диапазоне от 0 до 1 для позиций *x* и *y* в указанном порядке.

На рис. 10-18 показан пример условных обозначений, расположенных внизу и созданных так:

```
g + theme(legend.position = "bottom")
```

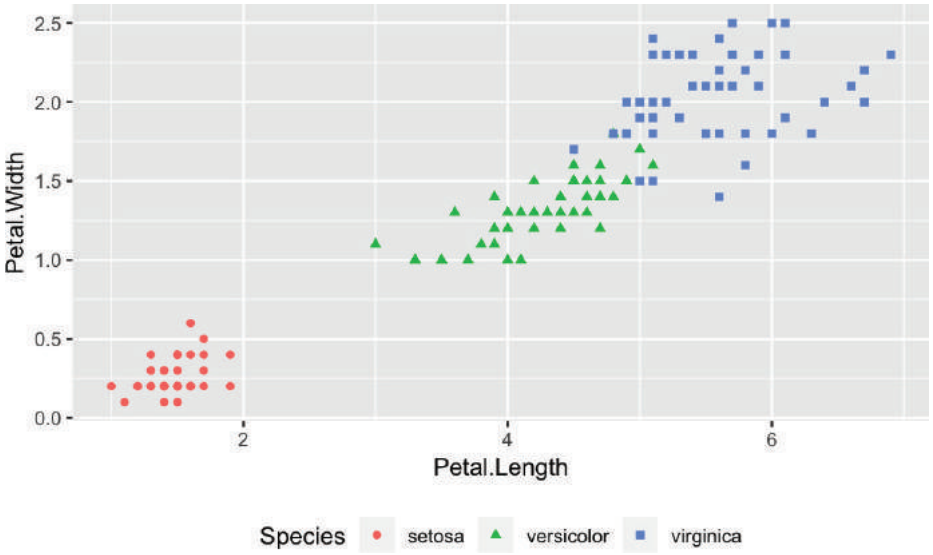


Рис. 10-18. Условные обозначения, расположенные внизу

Либо можно было бы использовать двухэлементный числовой вектор, чтобы поместить условные обозначения в определенном месте, как показано на рис. 10-19. В этом примере центр условных обозначений смещен на 80 % вправо и на 20 % вверх от нижней части:

```
g + theme(legend.position = c(.8, .2))
```

Во многих аспектах, помимо условных обозначений, `ggplot` использует разумные значения по умолчанию, но предлагает гибкость, чтобы переопределить их и настроить детали. Более подробную информацию об опциях `ggplot`, связанных с условными обозначениями, можно найти в справке по функциям `theme`, набрав `?theme` либо посетив страницу по адресу https://ggplot2.tidyverse.org/reference/theme.html.

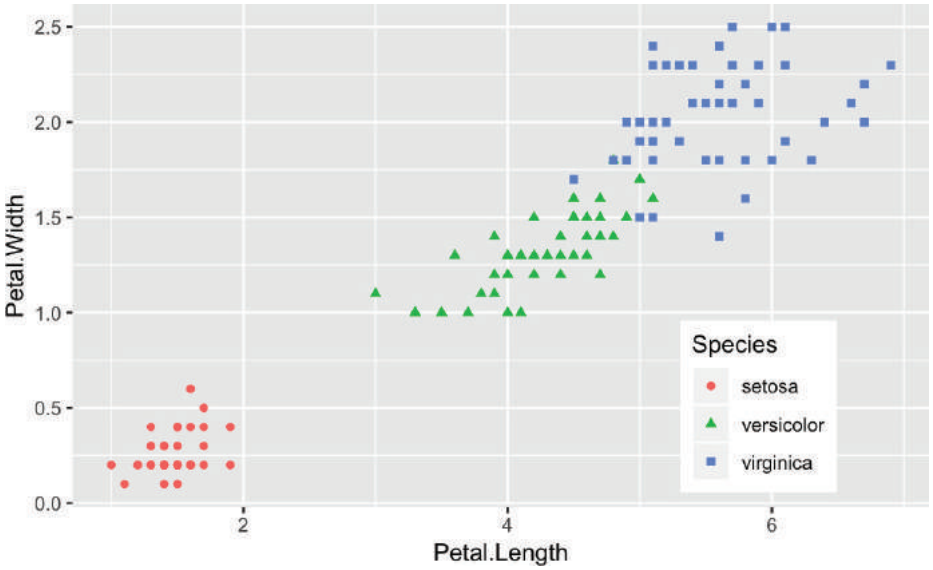


Рис. 10-19. Условные обозначения в определенной точке

10.7. Построение регрессионной линии точечной диаграммы

Задача

Вы наносите на график пары точек данных и хотите добавить линию, которая иллюстрирует их линейную регрессию.

Решение

При использовании `ggplot` нет необходимости сначала рассчитывать линейную модель, используя функцию R `lm`. Вместо этого можно использовать функцию `geom_smooth` для вычисления линейной регрессии внутри вызова `ggplot`.

Если наши данные находятся в таблице данных `df`, а данные *x* и *y* – в столбцах `x` и `y`, мы строим линию регрессии следующим образом:

```
ggplot(df, aes(x, y)) +
  geom_point() +
  geom_smooth(method = "lm",
              formula = y ~ x,
              se = FALSE)
```

Параметр `se = FALSE` дает `ggplot` указание не строить стандартные полосы ошибок вокруг нашей линии регрессии.

Обсуждение

Предположим, что мы моделируем набор данных `strongx` из пакета `faraway`. Мы можем создать линейную модель, используя встроенную функцию `lm`. Мы можем предсказать переменную `crossx` как линейную функцию `energy`. Во-первых, давайте посмотрим на простую точечную диаграмму наших данных (рис. 10-20):

```
library(faraway)
data(strongx)

ggplot(strongx, aes(energy, crossx)) +
 geom_point()
```

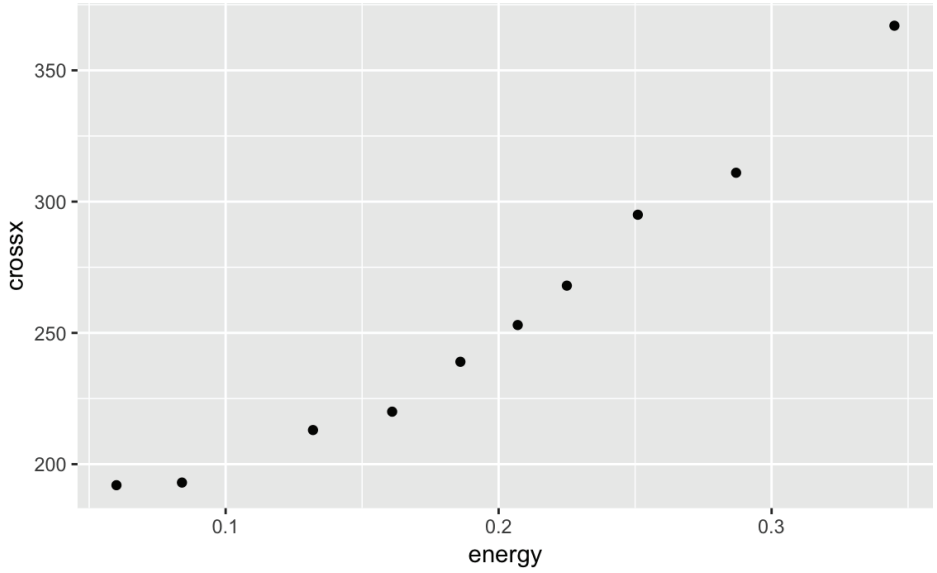


Рис. 10-20. Точечная диаграмма strong

`ggplot` может вычислить линейную модель на лету, а затем построить линию регрессии вместе с нашими данными (рис. 10-21):

```
g <- ggplot(strongx, aes(energy, crossx)) +
 geom_point()

g + geom_smooth(method = "lm",
                formula = y ~ x)
```

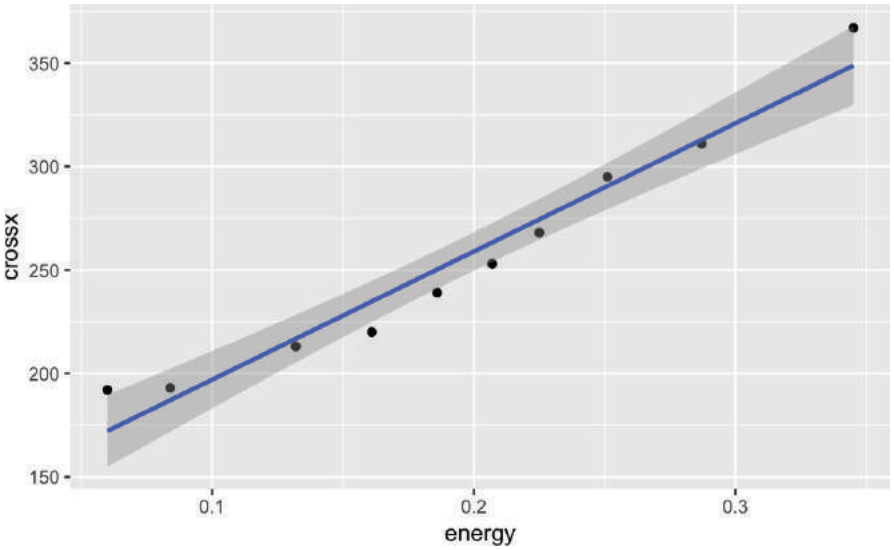


Рис. 10-21. Простая линейная модель ggplot

Мы можем отключить доверительные интервалы, добавив параметр `se = FALSE`, как показано на рис. 10-22:

```
g + geom_smooth(method = "lm",
 formula = y ~ x,
 se = FALSE)
```

Обратите внимание, что в функции `geom_smooth` мы используем `x` и `y`, а не имена переменных. `ggplot` установил `x` и `y` внутри графика на основе эстетики. `geom_smooth` поддерживает несколько методов сглаживания. Вы можете изучить эти и другие параметры в справке, набрав `?geom_smooth`.

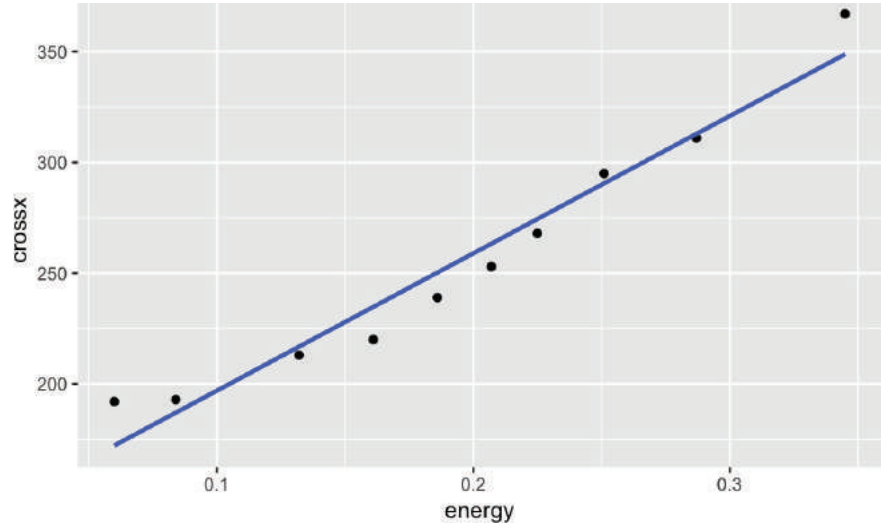


Рис. 10-22. Простая линейная модель ggplot без se

Если бы у нас была линия, которую мы хотели построить, сохраненная в другом объекте R, можно было бы использовать функцию `geom_abline`. В приведенном ниже примере мы извлекаем свободный член и угол наклона из регрессионной модели `m` и добавляем их в наш график (*см.* рис. 10-23):

```
m <- lm(crossx ~ energy, data = strongx)

ggplot(strongx, aes(energy, crossx)) +
 geom_point() +
 geom_abline(
   intercept = m$coefficients[1],
   slope = m$coefficients[2]
 )
```

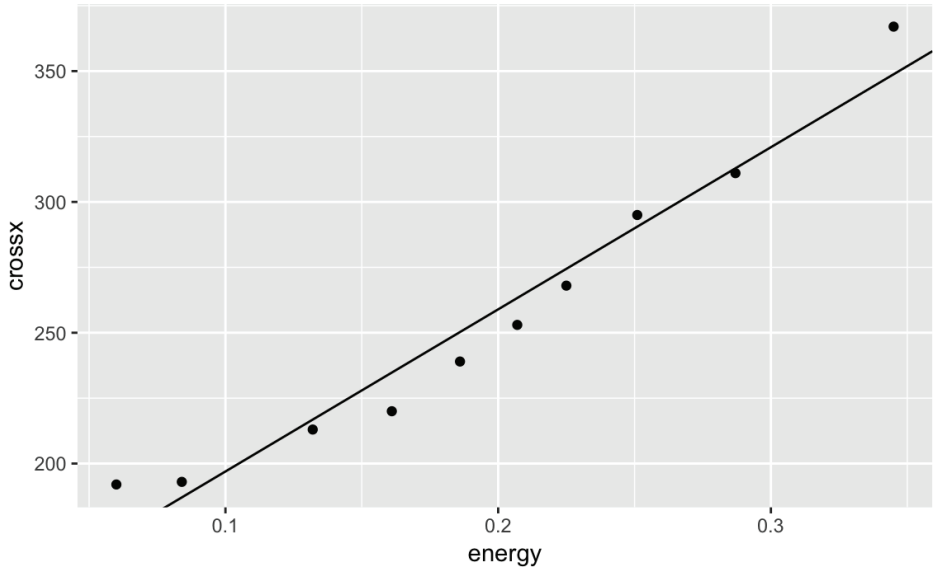


Рис. 10-23. Простая линия

Это дает график, очень похожий на тот, что изображен на рис. 10-22. Метод `geom_abline` может быть полезен, если вы строите линию из источника, отличного от простой линейной модели.

См. также

См. главу 11 для получения дополнительной информации о линейной регрессии и функции `lm`.

10.8. Построение точечных диаграмм всех пар переменных

Задача

Ваш набор данных содержит несколько числовых переменных. Вы хотите увидеть точечные диаграммы всех пар переменных.

Решение

У ggplot нет встроенного метода для создания парных графиков; однако пакет GGally предоставляет такую возможность с помощью функции ggpairs:

```
library(GGally)
ggpairs(df)
```

Обсуждение

Когда у вас есть большое количество переменных, найти взаимосвязь между ними сложно. Одним из полезных методов является просмотр точечных диаграмм всех пар переменных. Было бы довольно утомительно, если бы кодирование выполнялось попарно, но функция ggpairs из пакета GGally предоставляет простой способ создания всех этих точечных диаграмм за один раз.

Набор данных iris содержит четыре числовые переменные и одну категориальную:

```
head(iris)
#>   Sepal.Length  Sepal.Width  Petal.Length  Petal.Width  Species
#> 1          5.1          3.5           1.4          0.2   setosa
#> 2          4.9          3.0           1.4          0.2   setosa
#> 3          4.7          3.2           1.3          0.2   setosa
#> 4          4.6          3.1           1.5          0.2   setosa
#> 5          5.0          3.6           1.4          0.2   setosa
#> 6          5.4          3.9           1.7          0.4   setosa
```

Какая связь, если таковая имеется, между столбцами? Построение столбцов с помощью функции ggpairs дает несколько точечных диаграмм, как показано на рис. 10-24:

```
library(GGally)
ggpairs(iris)
```

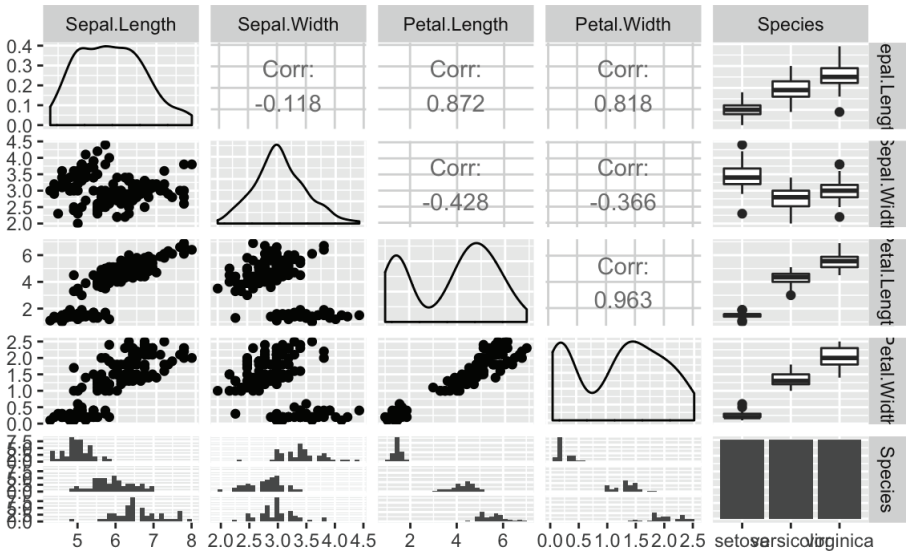


Рис. 10-24. Диаграмма набора данных iris

Функция `ggpairs` хороша, но она не особенно быстрая. Если вы просто выполняете интерактивную работу и хотите быстро взглянуть на данные, функция `plot` из базовой версии R обеспечивает более быстрый вывод (*см.* рис. 10-25):

```
plot(iris)
```

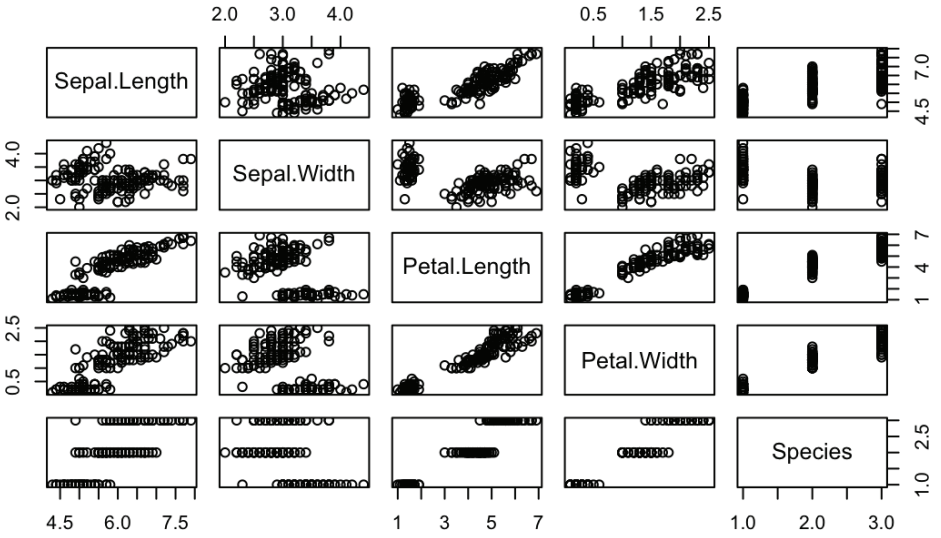


Рис. 10-25. Результат, полученный с помощью функции plot

Несмотря на то что функция `ggpairs` не такая быстрая, как функция `plot`, она создает графики плотности по диагонали и вычисляет корреляцию в верхнем треугольнике графика. Когда присутствуют факторы или символьные столбцы, `ggpairs` создает гистограммы в нижнем треугольнике графика и диаграммы размаха в верхнем. Это прекрасные дополнения, позволяющие понять связи в ваших данных.

10.9. Создание по одной точечной диаграмме для каждой группы

Задача

Ваш набор данных содержит (как минимум) две числовые переменные и поле фактора или символа, определяющее группу. Вы хотите создать несколько точечных диаграмм для числовых переменных, по одной на каждый уровень поля.

Решение

В `ggplot` данный тип диаграммы, которая носит название *условной*, создается путем добавления функции `facet_wrap`. В этом примере мы используем таблицу данных `df`, которая содержит три столбца, *x*, *y* и *f*, где *f* – это фактор (или символьная строка):

```
ggplot(df, aes(x, y)) +
 geom_point() +
 facet_wrap( ~ f)
```

Обсуждение

Условные диаграммы – это еще один способ изучить и проиллюстрировать влияние фактора или сравнить разные группы друг с другом.

Набор данных `Cars93` содержит 27 переменных, описывающих 93 модели автомобилей по состоянию на 1993 г. Две числовые переменные: `MPG.city`, количество милей на галлон в городе, и `Horsepower`, мощность двигателя. Одной из категориальных переменных является `Origin`, которая может быть США или страной за пределами США в зависимости от того, где была собрана модель.

Изучая взаимосвязь между MPG и лошадиными силами, можно было бы спросить: существует ли разница в связях между моделями из США и моделями из других стран?

Давайте посмотрим на это как на панельную диаграмму (рис. 10-26):

```
data(Cars93, package = "MASS")
ggplot(Cars93, aes(MPG.city, Horsepower)) +
 geom_point() +
 facet_wrap( ~ Origin)
```

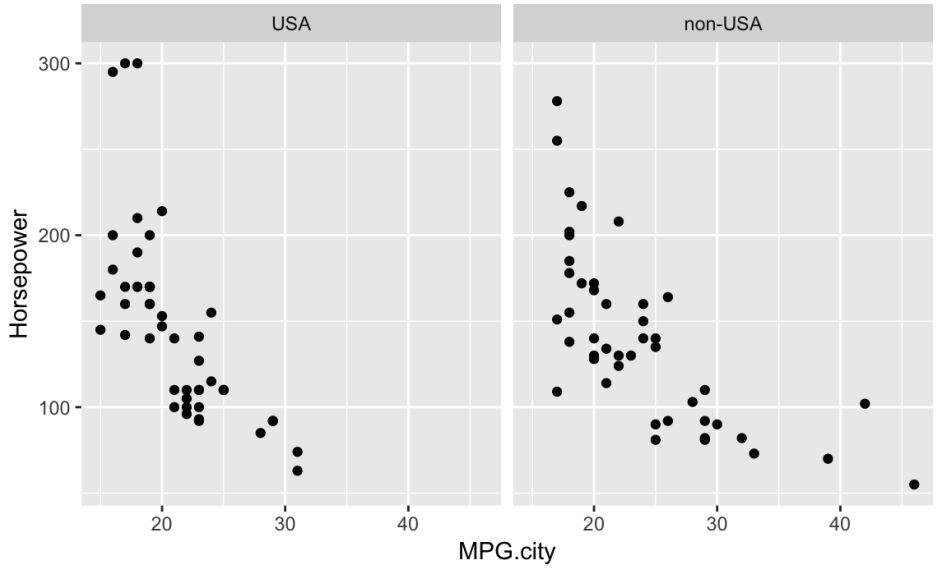


Рис. 10-26. Набор данных Cars93 с использованием функции facet_wrap

Получившаяся диаграмма приводит к ряду выводов. Если мы действительно жаждем этого монстра с двигателем мощностью 300 лошадиных сил, нам придется купить автомобиль, произведенный в США; но если нам нужно больше милей на галлон, у нас есть больше вариантов выбора среди неамериканских моделей. Такие выводы можно сделать на основе статистического анализа, но визуальное представление делает это быстрее.

Обратите внимание, что использование функции `facet_wrap` приводит к созданию вспомогательных участков с одинаковыми диапазонами осей *x* и *y*. Это помогает гарантировать, что визуальный осмотр данных не вводит в заблуждение из-за разных диапазонов.

См. также

Функция `coplot` из базовой версии R может создавать очень похожие диаграммы, используя только базовую графику.

10.10. Создание гистограммы

Задача

Вам нужно создать гистограмму.

Решение

Распространенной ситуацией является наличие столбца данных, обозначающего группу, а затем другого столбца, обозначающего показатель этой группы. Данный формат представляет собой «длинные» данные, потому что данные работают вертикально, вместо того чтобы иметь столбец для каждой группы.

Используя функцию `geom_bar` из `ggplot`, мы можем построить высоту в виде столбцов. Если данные уже агрегированы, мы добавляем `stat = "identity"`, чтобы `ggplot` знал, что он не должен агрегировать группы значений перед построением графика:

```
ggplot(data = df, aes(x, y)) +
 geom_bar(stat = "identity")
```

Обсуждение

В качестве примера используем автомобили марки Ford из набора данных `Cars93`:

```
ford_cars <- Cars93 %>%
 filter(Manufacturer == "Ford")

ggplot(ford_cars, aes(Model, Horsepower)) +
 geom_bar(stat = "identity")
```

На рис. 10-27 показана получившаяся гистограмма.

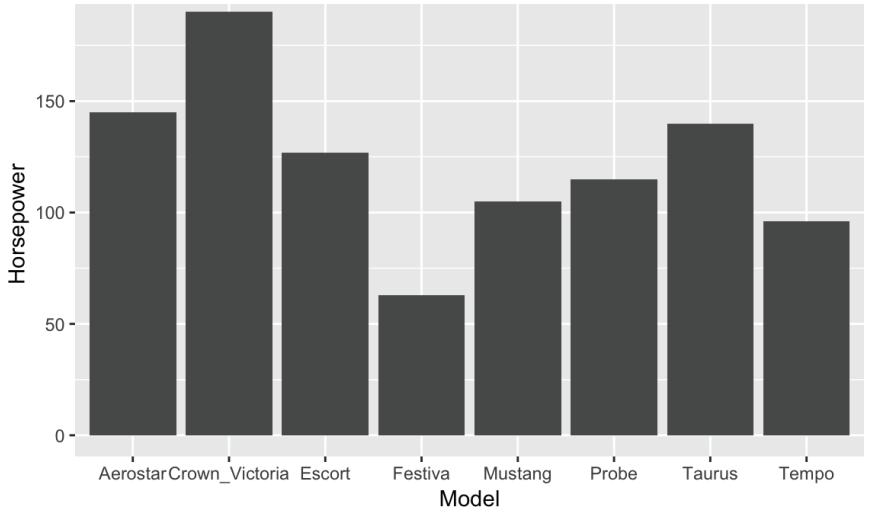


Рис. 10-27. Гистограмма автомобилей марки Ford

В этом примере используется `stat = "identity"`, а это предполагает, что высоту ваших столбцов удобно хранить в виде значения в одном поле, содержащем только одну запись на столбец. Однако это не всегда так. Часто у вас есть вектор числовых данных и параллельный коэффициент или символьное поле, которое группирует данные, и вы хотите создать гистограмму групповых средних значений или итоговых данных.

Давайте рассмотрим пример с использованием встроенного набора данных `airquality`, который содержит данные о суточной температуре для одного местоположения за пять месяцев. В таблице данных есть числовой столбец `Temp` и столбцы `Month` и `Day`. Если мы хотим построить среднюю температуру по месяцам с помощью `ggplot`, нам не нужно предварительно вычислять среднее значение; вместо этого мы можем заставить `ggplot` сделать это в логике команды `plot`. Чтобы дать `ggplot` указание вычислить среднее значение, мы передаем `stat = "summary"`, `fun.y = "mean"` команде `geom_bar`. Мы также можем превратить числа месяцев в даты, используя встроенную константу `month.abb`, которая содержит аббревиатуры месяцев:

```
ggplot(airquality, aes(month.abb[Month], Temp)) +
  geom_bar(stat = "summary", fun.y = "mean") +
  labs(title = "Mean Temp by Month",
       x = "",
       y = "Temp (deg. F)")
```

На рис. 10-28 показан получившийся график. Но вы, наверное, обратили внимание на то, что месяцы сортируются по алфавиту, а это не тот вид, в котором мы обычно привыкли их видеть.

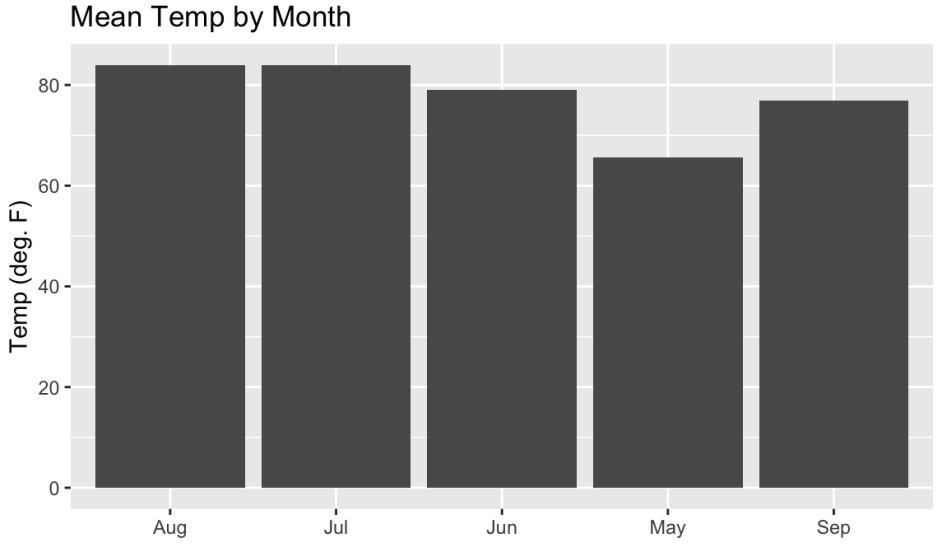


Рис. 10-28. Гистограмма: температура по месяцам

Можно решить проблему сортировки, используя несколько функций из `dplyr` в сочетании с функцией `fct_inorder` из пакета tidyverse `forcats`. Чтобы месяцы шли в правильном порядке, мы можем отсортировать таблицу данных по `Month`, номеру месяца. Затем можно применить функцию `fct_inorder`, которая упорядочит

наши факторы в порядке их появления в данных. На рис. 10-29 видно, что столбцы теперь отсортированы правильно:

```
library(forcats)
aq_data <- airquality %>%
 arrange(Month) %>%
 mutate(month_abb = fct_inorder(month.abb[Month]))

ggplot(aq_data, aes(month_abb, Temp)) +
 geom_bar(stat = "summary", fun.y = "mean") +
 labs(title = "Mean Temp by Month",
      x = "",
      y = "Temp (deg. F)")
```

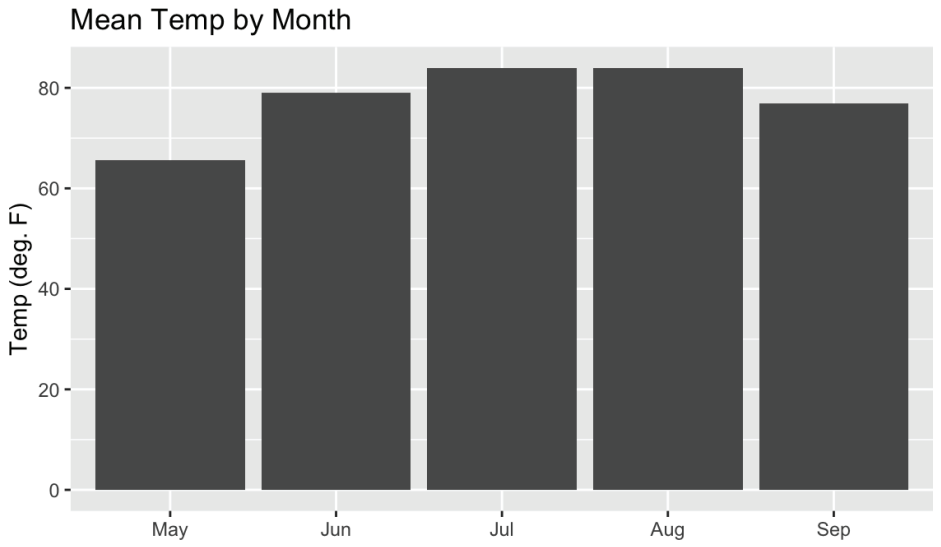


Рис. 10-29. Гистограмма отсортирована правильно

См. также

См. рецепт 10.11, чтобы узнать, как добавить доверительные интервалы, и рецепт 10.12, чтобы узнать, как добавить цвет.

Введите `?geom_bar` для получения помощи с гистограммами в `ggplot`.

Вы также можете использовать функцию `barplot` для создания гистограмм из базовой версии R или функцию `barchart` из пакета `lattice`.

10.11. Добавление доверительных интервалов в гистограмму

Задача

Вы хотите дополнить гистограмму доверительными интервалами.

Решение

Предположим, у нас есть таблица данных `df` со столбцами `group` (имена групп), `stat` (столбец статистики), а также `lower` и `upper` (которые обозначают соответствующие

пределы для доверительных интервалов). Мы можем отобразить гистограмму `stat` для каждого столбца `group` и ее доверительный интервал, используя функцию `geom_bar` в сочетании с функцией `geom_errorbar`:

```
ggplot(df, aes(group, stat)) +
 geom_bar(stat = "identity") +
 geom_errorbar(aes(ymin = lower, ymax = upper), width = .2)
```

На рис. 10-30 показана полученная гистограмма с доверительными интервалами.

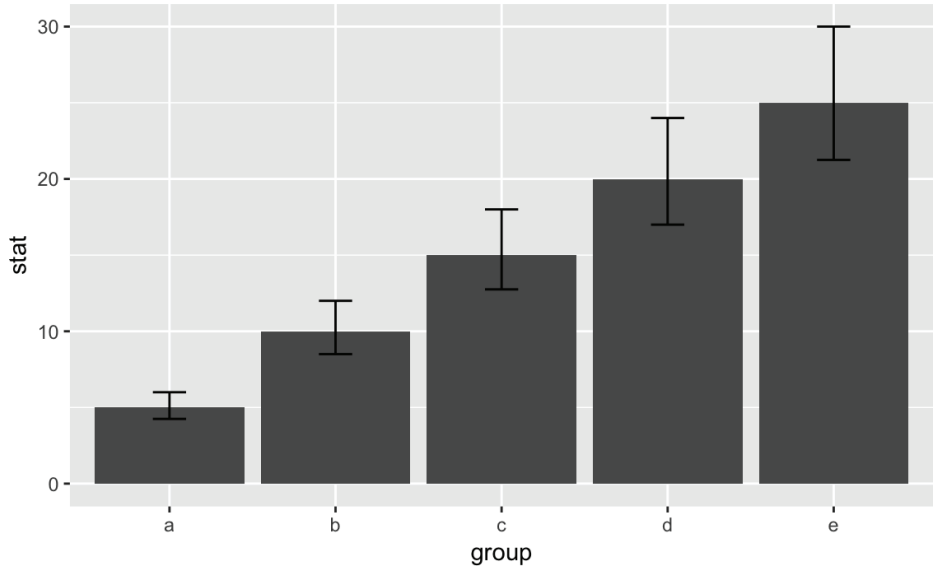


Рис. 10-30. Гистограмма с доверительными интервалами

Обсуждение

Большинство гистограмм показывают точечные оценки, которые отображаются по высоте столбцов, но они редко включают в себя доверительные интервалы. Наши специалисты по статистике очень не любят этого. Точная оценка – только половина картины; доверительный интервал дает ее полностью.

К счастью, можно построить планки погрешностей с помощью `ggplot`. Сложность состоит в расчете интервалов. В предыдущих примерах у наших данных был простой интервал –15 % и +20 %. Тем не менее в рецепте 10.10 мы рассчитали групповые средние значения перед их построением. Если мы позволим `ggplot` выполнять расчеты за нас, то можем использовать встроенную функцию `mean_se` наряду с функцией `stat_summary`, чтобы получить стандартные ошибки средних значений.

Воспользуемся данными `airquality`, которые мы использовали ранее. Сначала мы выполним процедуру отсортированного фактора (из предыдущего рецепта), чтобы получить названия месяцев в нужном порядке:

```
aq_data <- airquality %>%
 arrange(Month) %>%
 mutate(month_abb = fct_inorder(month.abb[Month]))
```

Теперь можно построить планки вместе со стандартными ошибками, как показано на рис. 10-31:

```
ggplot(aq_data, aes(month_abb, Temp)) +
 geom_bar(stat = "summary",
          fun.y = "mean",
          fill = "cornflowerblue") +
 stat_summary(fun.data = mean_se, geom = "errorbar") +
 labs(title = "Mean Temp by Month",
      x = "",
      y = "Temp (deg. F)")
```

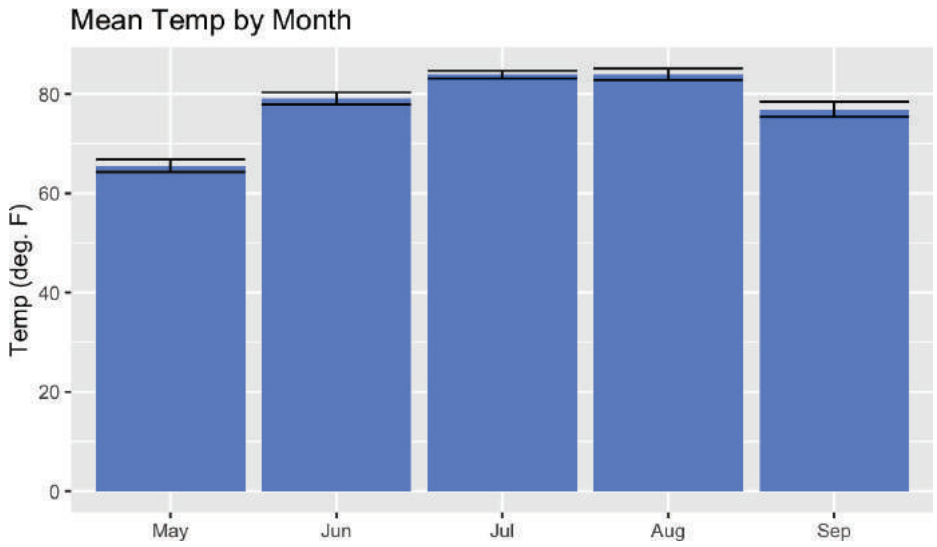


Рис. 10-31. Средняя температура по месяцам с планками погрешностей

Иногда вам может понадобиться отсортировать столбцы на гистограмме в порядке убывания, основываясь на их высоте, как показано на рис. 10-32. Это может немного сбивать с толку, когда вы используете сводную статистику в `ggplot`, но секрет состоит в том, чтобы использовать `mean` в операторе `reorder`, дабы отсортировать фактор по среднему значению температуры. Обратите внимание на то, что когда `mean` используется с `reorder`, он не закавычен, но когда он используется с функцией `geom_bar`, кавычки стоят:

```
ggplot(aq_data, aes(reorder(month_abb, -Temp, mean), Temp)) +
 geom_bar(stat = "summary",
          fun.y = "mean",
          fill = "tomato") +
 stat_summary(fun.data = mean_se, geom = "errorbar") +
 labs(title = "Mean Temp by Month",
      x = "",
      y = "Temp (deg. F)")
```

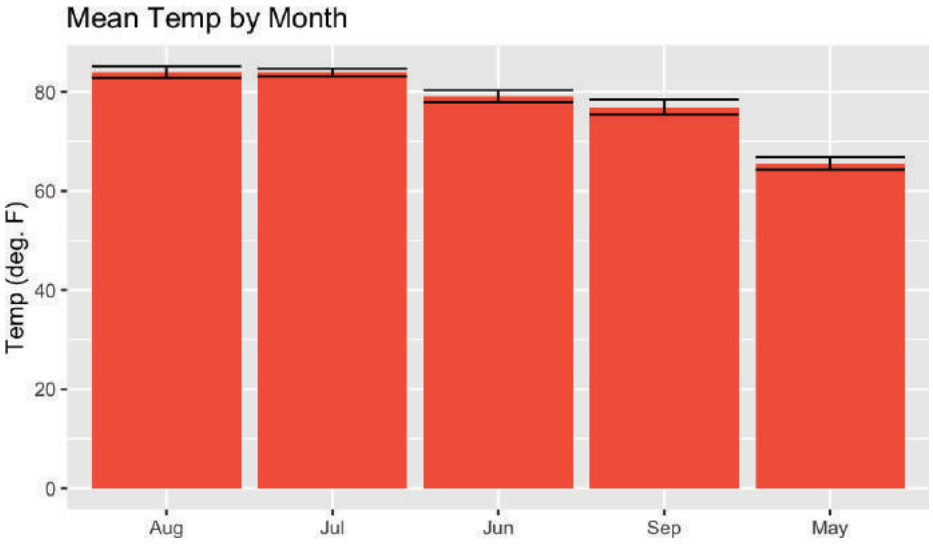


Рис. 10-32. Средняя температура по месяцам в порядке убывания

Вы можете посмотреть на этот пример и на результат на рис. 10-32 и спросить себя: «Почему они не использовали `reorder(month_abb, Month)` в первом примере вместо всей этой сортировки с помощью `forcats::fct_inorder`, чтобы месяцы шли в правильном порядке?» Ну, мы могли бы это сделать. Но сортировка с использованием функции `fct_inorder` – это шаблон проектирования, который обеспечивает гибкость для более сложных вещей. Кроме того, это довольно легко читать в сценарии. Использование `reorder` внутри `aes` немного плотнее, и в дальнейшем читать это будет сложнее, но любой из данных подходов является разумным.

См. также

См. рецепт 9.9 для получения дополнительной информации о функции `t.test`.

10.12. Раскраска гистограммы

Задача

Вы хотите раскрасить или затенить столбцы гистограммы.

Решение

С помощью `ggplot` мы добавляем параметр `fill` к нашему вызову `aes` и позволяем `ggplot` выбирать цвета для нас:

```
ggplot(df, aes(x, y, fill = group))
```

Обсуждение

Мы можем использовать параметр `fill` в `aes`, чтобы указать `ggplot`, на каком поле основывать цвета. Если мы передадим числовое поле в `ggplot`, то получим непрерывный градиент цветов, а если передадим поле фактора или символа в параметр

fill, то получим контрастные цвета для каждой группы. Здесь мы передаем имя символа каждого месяца параметру fill:

```
aq_data <- airquality %>%
 arrange(Month) %>%
 mutate(month_abb = fct_inorder(month.abb[Month]))

ggplot(data = aq_data, aes(month_abb, Temp, fill = month_abb)) +
 geom_bar(stat = "summary", fun.y = "mean") +
 labs(title = "Mean Temp by Month",
      x = "",
      y = "Temp (deg. F)") +
 scale_fill_brewer(palette = "Paired")
```

Мы определяем цвета в итоговой гистограмме (рис. 10-33), вызвав scale_fill_brewer (palette = "Paired"). Цветовая палитра "Paired" поставляется вместе со множеством других цветовых палитр в пакете RColorBrewer.

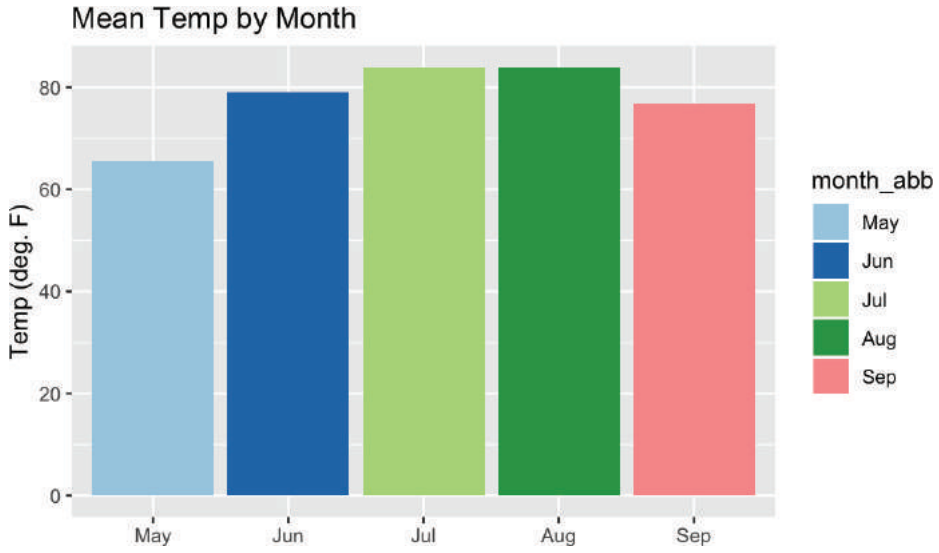


Рис. 10-33. Гистограмма месячных температур в цвете

Если нам нужно изменить цвет всех столбцов в зависимости от температуры, нельзя просто установить для параметра fill значение Temp – как может показаться интуитивно понятным, – потому что ggplot не поймет, что мы хотим получить среднюю температуру после группировки по месяцам. Это можно обойти, обратившись к специальному полю внутри нашего графика, с именем ..y.., которое представляет собой вычисляемое значение на оси y. Но нам не нужно условное обозначение с надписью ..y.., поэтому мы добавляем fill = "Temp" в вызов функции labs, чтобы изменить название условного обозначения. Результат показан на рис. 10-34:

```
ggplot(airquality, aes(month.abb[Month], Temp, fill = ..y..)) +
 geom_bar(stat = "summary", fun.y = "mean") +
 labs(title = "Mean Temp by Month",
```

```
x = "",
y = "Temp (deg. F)",
fill = "Temp")
```

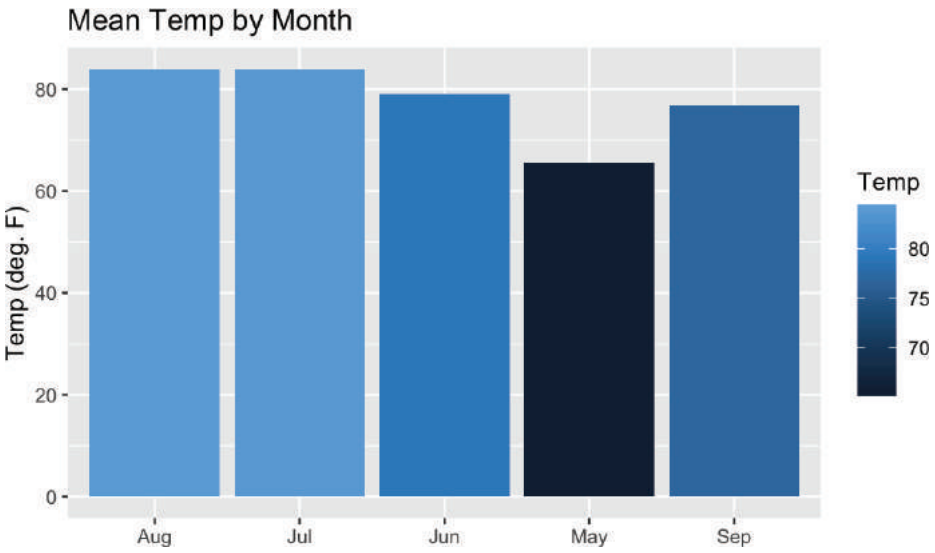


Рис. 10-34. Гистограмма, затененная по значению

Если мы хотим изменить цветовую шкалу, можно просто добавить знак «-» перед полем, которое мы заполняем, например `fill = - .. y ...`

См. также

См. рецепт 10.10, чтобы узнать, как создать гистограмму.

10.13. Построение линии из точек *x* и *y*

Задача

У вас есть зависимые наблюдения в таблице данных: $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$. Вы хотите построить серию отрезков, соединяющих точки данных.

Решение

Можно использовать функцию `geom_point` для построения точек:

```
ggplot(df, aes(x, y)) +
 geom_point()
```

Поскольку графики `ggplot` строятся элемент за элементом, у нас легко может быть и точка, и линия на одном графике, если мы будем использовать функции `geom_point()` и `geom_line()`.

```
ggplot(df, aes(x , y)) +
 geom_point() +
 geom_line()
```

Обсуждение

Чтобы проиллюстрировать это, рассмотрим ряд примеров экономических данных США, которые поставляются с `ggplot2`. В этом примере таблицы данных есть столбец с именем `date`, который мы будем строить на оси x, и поле с именем `unemploy`, которое представляет собой число безработных:

```
ggplot(economics, aes(date , unemploy)) +
 geom_point() +
 geom_line()
```

На рис. 10-35 показана получившаяся диаграмма, которая содержит и линии, и точки, потому что мы использовали обе функции.

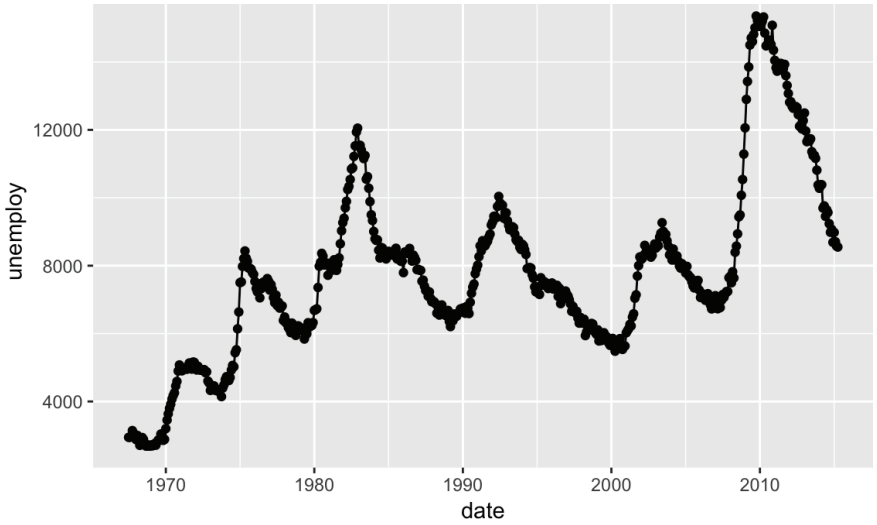


Рис. 10-35. Линейная диаграмма

См. также

См. рецепт 10.1.

10.14. Изменение типа, ширины или цвета линии

Задача

Вы строите линию и хотите изменить ее тип, ширину или цвет.

Решение

`ggplot` использует параметр `linetype` для управления внешним видом линий. Ниже приводятся его варианты:

- `linetype="solid"` или `linetype=1` (по умолчанию);
- `linetype="dashed"` или `linetype=2`;
- `linetype="dotted"` или `linetype=3`;
- `linetype="dotdash"` или `linetype=4`;

- `linetype="longdash"` или `linetype=5`;
- `linetype="twodash"` или `linetype=6`;
- `linetype="blank"` или `linetype=0` (запрещает рисование).

Мы можем изменить характеристики линии, передав `linetype`, `col` и/или `size` в качестве параметров функции `geom_line`. Например, если мы хотим изменить тип линии на пунктирную, красную и жирную, можно передать `geom_line` следующие параметры:

```
ggplot(df, aes(x, y)) +
 geom_line(linetype = 2,
           size = 2
           col = "red")
```

Обсуждение

Этот пример синтаксиса показывает, как нарисовать одну линию и указать ее стиль, ширину или цвет. Обычный сценарий включает в себя рисование нескольких линий, каждая из которых имеет свой стиль, ширину или цвет.

В `ggplot` для многих пользователей это может стать головоломкой. Проблема состоит в том, что `ggplot` лучше всего работает с «длинными» данными, а не с «широкими», как было упомянуто во введении к этой главе.

Возьмем какие-нибудь данные в качестве примера:

```
x <- 1:10
y1 <- x**1.5
y2 <- x**2
y3 <- x**2.5
df <- data.frame(x, y1, y2, y3)
```

У нашей таблицы данных есть четыре столбца широких данных:

```
head(df, 3)
#>    x   y1  y2  y3
#> 1 1 1.00  1   1.00
#> 2 2 2.83  4   5.66
#> 3 3 5.20  9   15.59
```

Мы можем сделать наши широкие данные длинными, используя функцию `gather` из базового пакета `tidyr` коллекции tidyverse. В этом примере мы используем функцию `gather` для создания нового столбца с именем `bucket` и помещаем туда имена наших столбцов, сохраняя при этом наши переменные `x` и `y`:

```
df_long <- gather(df, bucket, y, -x)
head(df_long, 3)
#>    x bucket y
#> 1 1  y1    1.00
#> 2 2  y1    2.83
#> 3 3  y1    5.20
tail(df_long, 3)
#>    x bucket y
#> 28 8   y3  181
#> 29 9   y3  243
#> 30 10 y3 316
```

Теперь мы можем передать `bucket` параметру `col` и получить несколько строк, у каждой из которых свой цвет:

```
ggplot(df_long, aes(x, y, col = bucket)) +
  geom_line()
```

На рис. 10-36 показан получившийся график, в котором каждая переменная обозначена разным цветом.

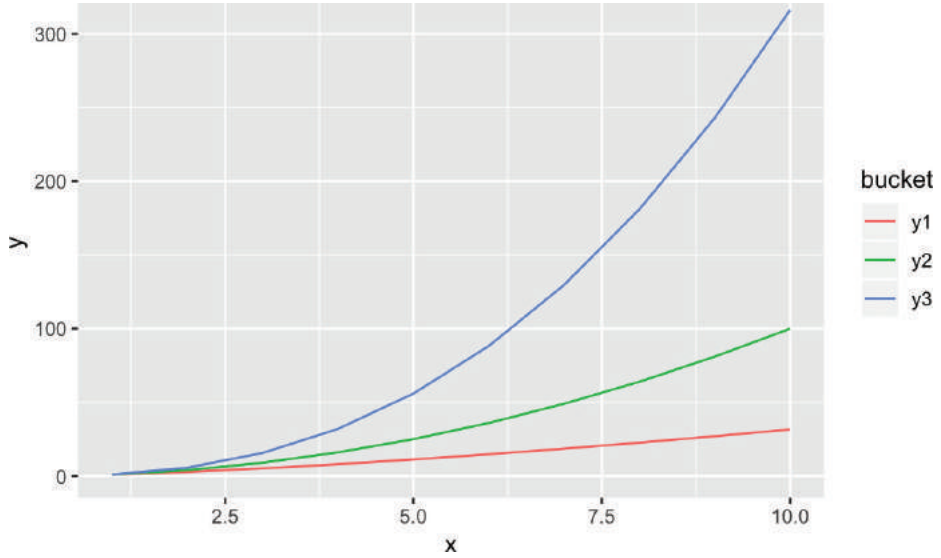


Рис. 10-36. Диаграмма из нескольких линий

Изменить толщину строки с помощью переменной очень просто – просто передайте числовую переменную в `size`.

Результат изменения толщины с помощью *x* показан на рис. 10-37.

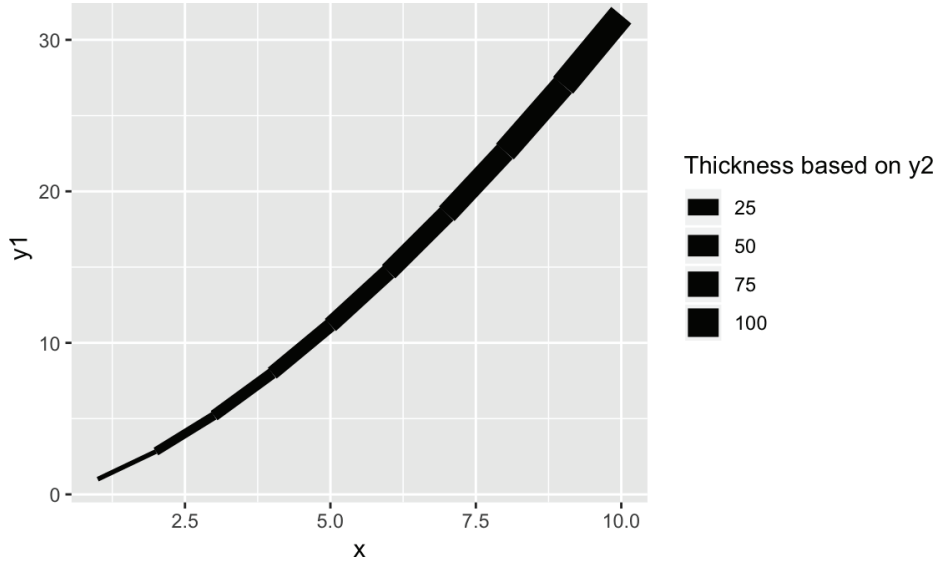


Рис. 10-37. Толщина как функция x

См. также

См. рецепт 10.13, чтобы узнать, как построить базовую линию.

10.15. Построение нескольких наборов данных

Задача

Вы хотите показать несколько наборов данных на одном графике.

Решение

Мы можем добавить несколько таблиц данных к графику `ggplot`, создав пустой график, а затем добавив к нему две разные функции семейства `geom`:

```
ggplot() +
  geom_line(data = df1, aes(x1, y1)) +
  geom_line(data = df2, aes(x2, y2))
```

Здесь мы использовали функцию `geom_line`, но вы можете применять любую функцию из этого семейства.

Обсуждение

Мы могли бы объединить данные в одну таблицу перед построением, используя одну из функций соединения из пакета `dplyr`. Однако мы создадим две отдельные таблицы данных, а затем добавим их в график `ggplot`.

Сначала давайте настроим наши таблицы данных, `df1` и `df2`:

```
# Пример данных.
n <- 20
x1 <- 1:n

y1 <- rnorm(n, 0, .5)
df1 <- data.frame(x1, y1)

x2 <- (.5 * n):((1.5 * n) - 1)
y2 <- rnorm(n, 1, .5)
df2 <- data.frame(x2, y2)
```

Обычно мы передаем таблицы данных непосредственно в вызов функции `ggplot`. Поскольку нам нужны две функции `geom_` с двумя разными источниками данных, мы запустим график с помощью `ggplot`, а затем добавим два вызова функции `geom_line`, и в обоих случаях у каждой функции будет свой источник данных:

```
ggplot() +
  geom_line(data = df1, aes(x1, y1), color = "darkblue") +
  geom_line(data = df2, aes(x2, y2), linetype = "dashed")
```

`ggplot` позволяет нам делать несколько вызовов различным функциям `geom_`, у каждой из которых есть собственный источник данных, если это необходимо. Затем `ggplot` рассмотрит все данные, которые мы наносим, и скорректирует диапазоны, чтобы вместить все данные.

График с расширенными пределами показан на рис. 10-38.

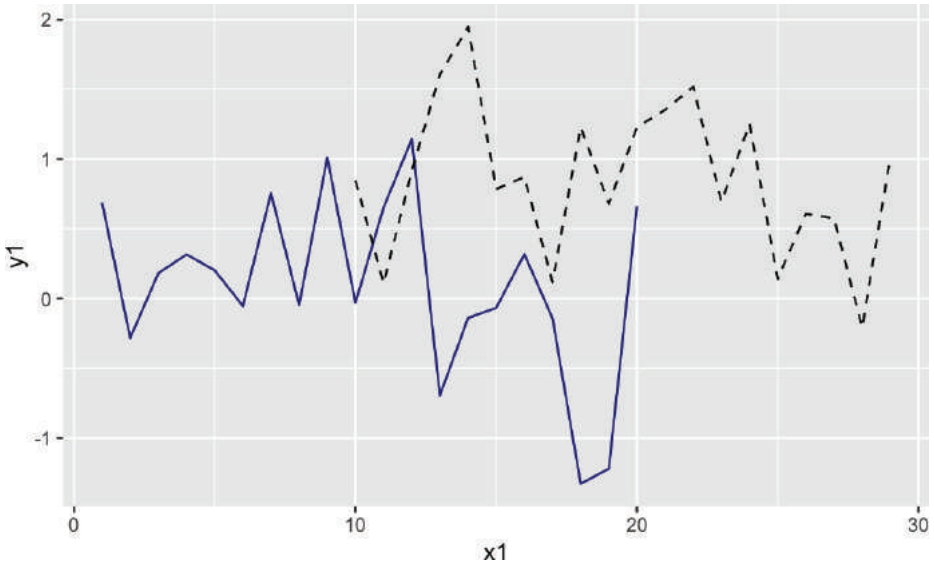


Рис. 10-38. Две линии, один график

10.16. Добавление вертикальных или горизонтальных линий

Задача

Вы хотите добавить вертикальную или горизонтальную линию к своему графику, например ось, проходящую через начало координат или указатель порога.

Решение

Функции `geom_vline` и `geom_hline` создают вертикальные и горизонтальные линии соответственно. Они также могут принимать параметры `color`, `line type` и `size` для установки стиля линии:

```
# Используем таблицу данных df1 из предыдущего рецепта.
ggplot(df1) +
 aes(x = x1, y = y1) +
 geom_point() +
 geom_vline(
   xintercept = 10,
   color = "red",
   linetype = "dashed",
   size = 1.5
 ) +
 geom_hline(yintercept = 0, color = "blue")
```

На рис. 10-39 показан получившийся график с добавленными горизонтальными и вертикальными линиями.

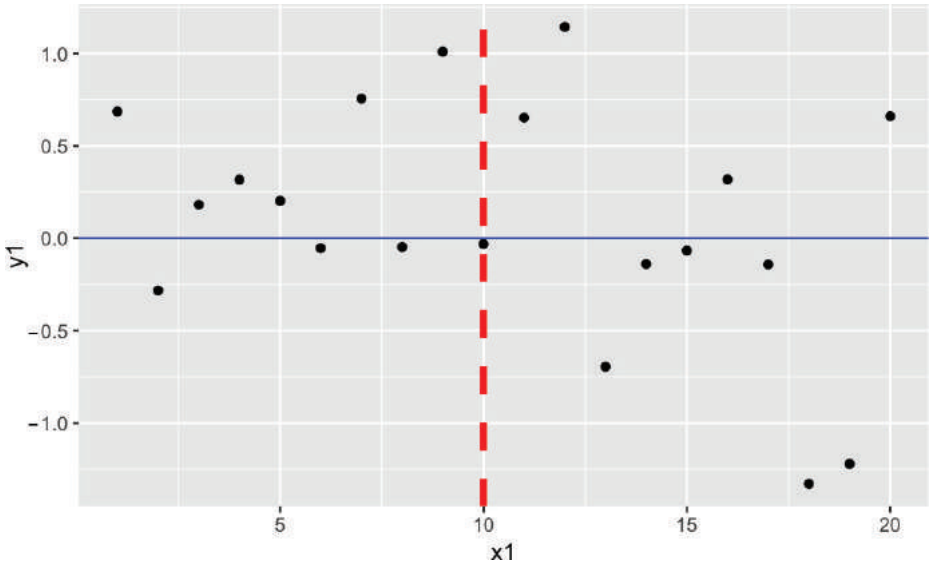


Рис. 10-39. Вертикальные и горизонтальные линии

Обсуждение

Типичное использование линий – рисование линий, расположенных через равные промежутки времени. Предположим, у нас есть выборка точек, `samp`. Сначала мы наносим их сплошной линией через среднее значение. Затем рассчитываем и рисуем пунктирные линии на ±1 и ±2 стандартных отклонения от среднего значения. Мы можем добавить линии в наш график с помощью функции `geom_hline`:

```
samp <- rnorm(1000)
samp_df <- data.frame(samp, x = 1:length(samp))

mean_line <- mean(samp_df$samp)
sd_lines <- mean_line + c(-2, -1, +1, +2) * sd(samp_df$samp)

ggplot(samp_df) +
  aes(x = x, y = samp) +
  geom_point() +
  geom_hline(yintercept = mean_line, color = "darkblue") +
  geom_hline(yintercept = sd_lines, linetype = "dotted")
```

На рис. 10-40 показаны данные выборки вместе с линиями среднего значения и стандартного отклонения.

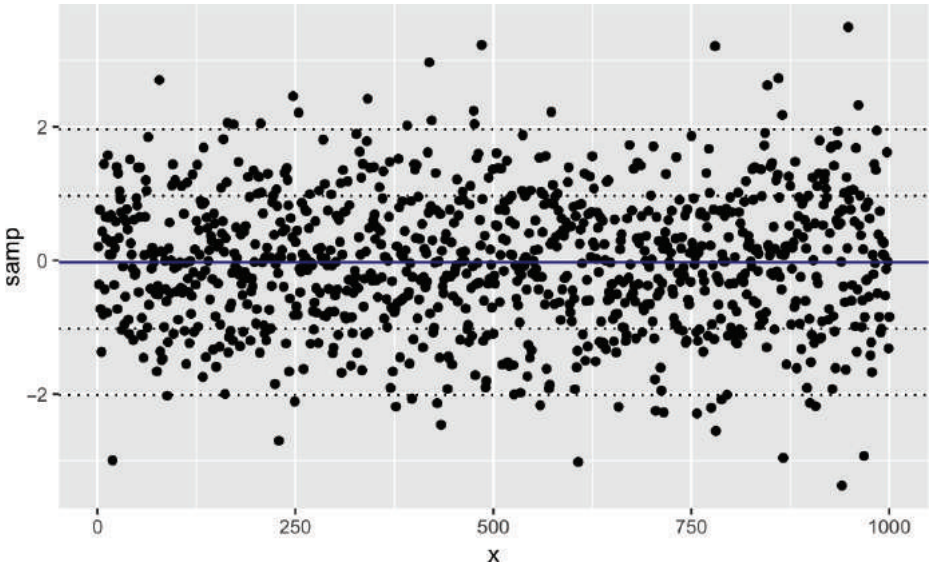


Рис. 10-40. Линии среднего значения и стандартного отклонения на графике

См. также

См. рецепт 10.14 для получения дополнительной информации об изменении типов линий.

10.17. Создание диаграммы размаха

Задача

Вы хотите создать диаграмму размаха из своих данных.

Решение

Используйте функцию `geom_boxplot` из `ggplot`, чтобы добавить диаграмму размаха на график. Используя таблицу данных `samp_df` из предыдущего рецепта, мы можем создать диаграмму значений в столбце `x`. Результат показан на рис. 10-41:

```
ggplot(samp_df) +
 aes(y = samp) +
 geom_boxplot()
```

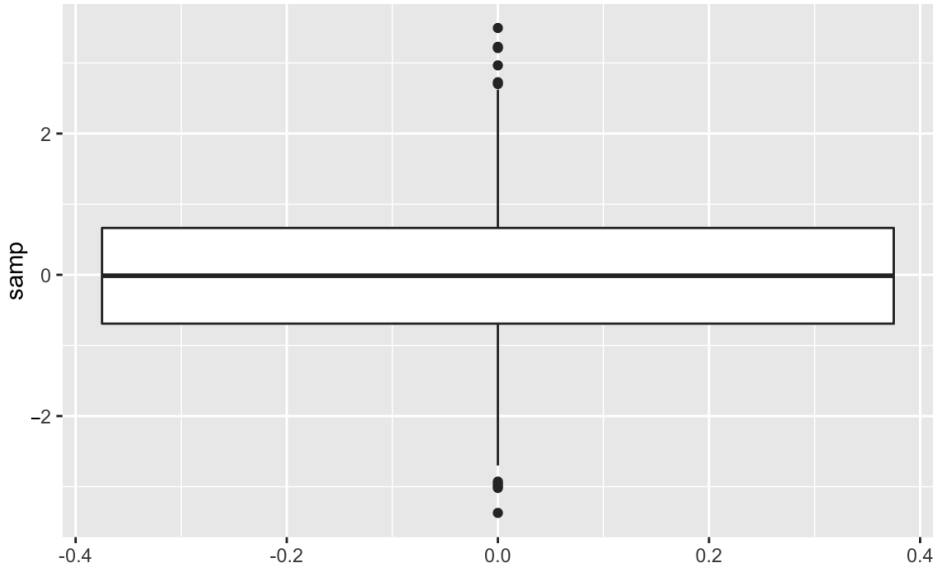


Рис. 10-41. Диаграмма размаха

Обсуждение

Диаграмма размаха обеспечивает быструю и простую визуальную сводку набора данных:

- толстая линия посередине – это медиана;
- рамка, окружающая медиану, обозначает первый и третий квартили; нижняя часть ящика – это первый квартиль (Q1), а верхняя часть – третий (Q3);
- «усы» над и под ящиком показывают диапазон данных, исключая выбросы;
- круги обозначают выбросы. По умолчанию выброс определяется как любое значение, которое больше 1,5 × IQR от ящика. (IQR – это *межквартильный диапазон*, или Q3–Q1.) В этом примере есть несколько выбросов на высокой стороне.

Можно вращать диаграмму, переворачивая координаты. В некоторых ситуациях это делает график более привлекательным, как показано на рис. 10-42.

```
ggplot(samp_df) +
 aes(y = samp) +
 geom_boxplot() +
 coord_flip()
```

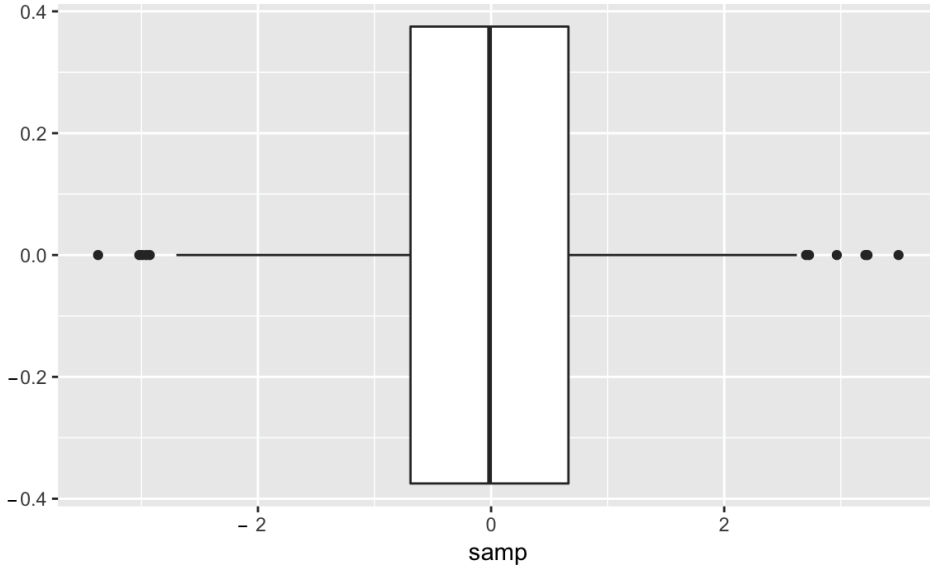


Рис. 10-42. Диаграмма размаха в перевернутом виде

См. также

Отдельная диаграмма размаха выглядит довольно скучно. *См.* рецепт 10.18, чтобы узнать, как создать несколько таких диаграмм.

10.18. Создание диаграммы размаха для каждого уровня фактора

Задача

Ваш набор данных содержит числовую переменную и фактор (или другой категориальный текст). Вы хотите создать несколько диаграмм размаха числовой переменной, разбитой по уровням.

Решение

С помощью `ggplot` мы передаем имя категориальной переменной параметру `x` в вызове `aes`. Полученная диаграмма будет сгруппирована по значениям в категориальной переменной:

```
ggplot(df) +
 aes(x = factor, y = values) +
 geom_boxplot()
```

Обсуждение

Данный рецепт – еще один отличный способ исследовать и проиллюстрировать связь между двумя переменными. В этом случае мы хотим знать, изменяется ли числовая переменная в соответствии с уровнем категории.

Набор данных UScereal из пакета MASS содержит множество переменных, касающихся хлопьев для завтрака. Одна переменная – это количество сахара на порцию, а другая – расположение полки (считая от пола). Производители зерновых могут договориться о расположении полки, размещая свою продукцию, чтобы обеспечить лучший потенциал продаж. Интересно: куда они кладут хлопья с высоким содержанием сахара? У нас может получиться рисунок, подобный рис. 10-43, чтобы исследовать этот вопрос, создавая по одной диаграмме размаха на полку:

```
data(UScereal, package = "MASS")

ggplot(UScereal) +
 aes(x = as.factor(shelf), y = sugars) +
 geom_boxplot() +
 labs(
   title = "Sugar Content by Shelf",
   x = "Shelf",
   y = "Sugar (grams per portion)"
 )
```

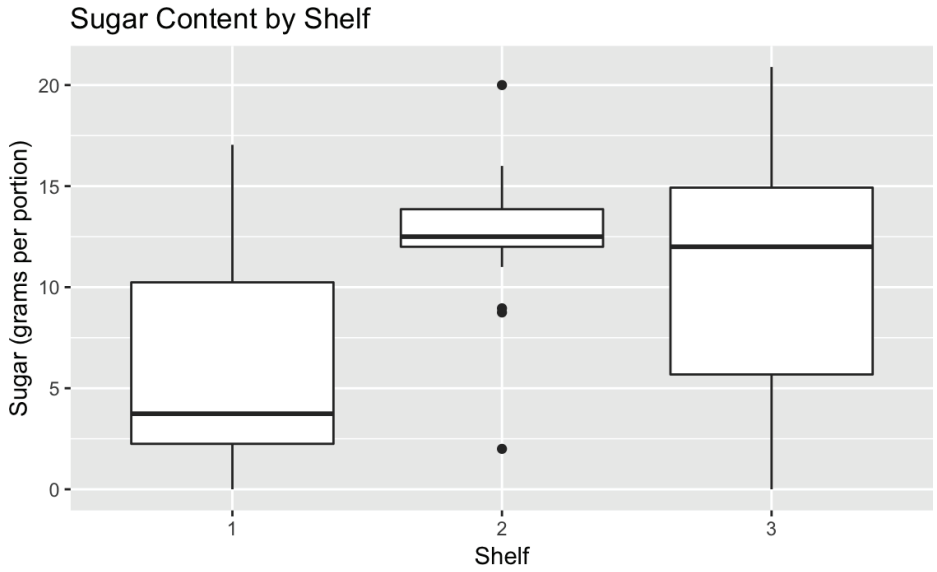


Рис. 10-43. Диаграммы размаха по номеру полки

Диаграммы показывают, что на полке № 2 содержится больше всего хлопьев с высоким содержанием сахара. Может ли быть так, что эта полка находится на уровне глаз для маленьких детей, которые могут повлиять на своих родителей, чтобы те сделали выбор в пользу этих хлопьев?

Обратите внимание на то, что в вызове aes мы должны были дать ggplot указание рассматривать число полок как фактор. В противном случае ggplot не отреагирует на полку как на группировку и будет выводить только одну диаграмму.

См. также

См. рецепт 10.17, чтобы узнать, как создать базовую диаграмму размаха.

10.19. Создание гистограммы

Задача

Вам необходимо создать гистограмму своих данных.

Решение

Используйте функцию `geom_histogram` и установите в качестве значения для `x` вектор числовых значений.

Обсуждение

На рис. 10-44 показана гистограмма столбца `MPG.city` из набора данных `Cars93`:

```
data(Cars93, package = "MASS")

ggplot(Cars93) +
 geom_histogram(aes(x = MPG.city))
#> 'stat_bin()' using 'bins = 30'. Pick better value with 'binwidth'.
```

Функция `geom_histogram` должна решить, сколько ячеек (отрезков диапазона) нужно создать для объединения данных. В этом примере алгоритм по умолчанию выбрал 30 отрезков. Если бы мы хотели меньше отрезков, мы бы включили сюда параметр `bins`, чтобы сообщить `geom_histogram`, сколько отрезков нам нужно:

```
ggplot(Cars93) +
 geom_histogram(aes(x = MPG.city), bins = 13)
```

На рис. 10-45 показана гистограмма с 13 отрезками.

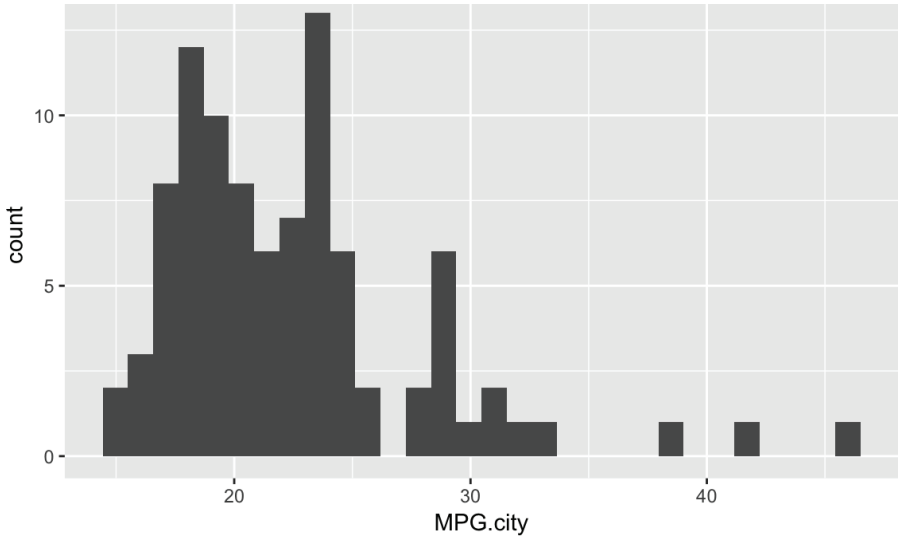


Рис. 10-44. Гистограмма столбца MPG.city из набора данных Cars93

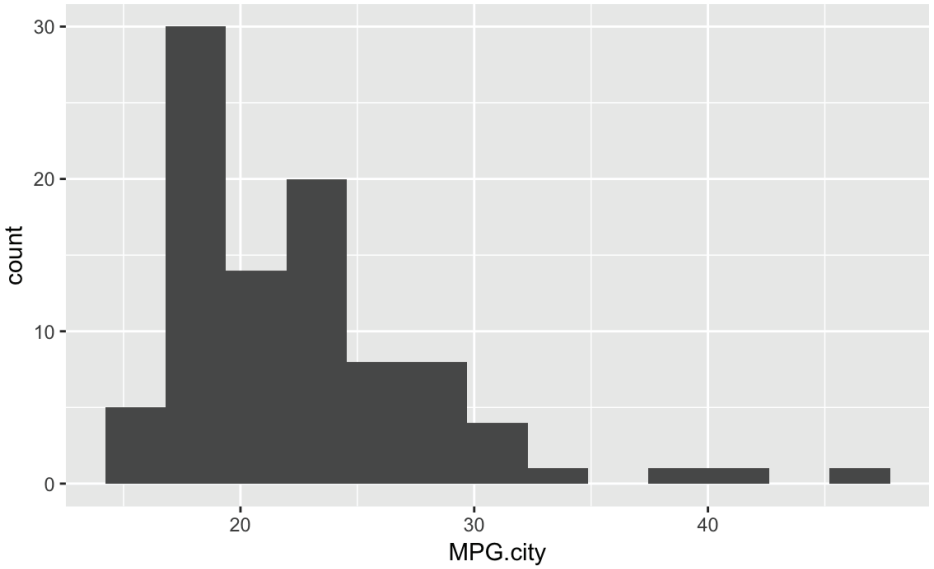


Рис. 10-45. Гистограмма с 13 отрезками

См. также

Функция `hist` из базовой версии R обеспечивает большую часть той же функциональности, что и функция `histogram` из пакета `lattice`.

10.20. Добавление оценки плотности к гистограмме

Задача

У вас есть гистограмма выборки данных, и вы хотите добавить кривую, чтобы проиллюстрировать кажущуюся плотность.

Решение

Используйте функцию `geom_density` для аппроксимации плотности выборки, как показано на рис. 10-46:

```
ggplot(Cars93) +
 aes(x = MPG.city) +
 geom_histogram(aes(y = ..density..), bins = 21) +
 geom_density()
```

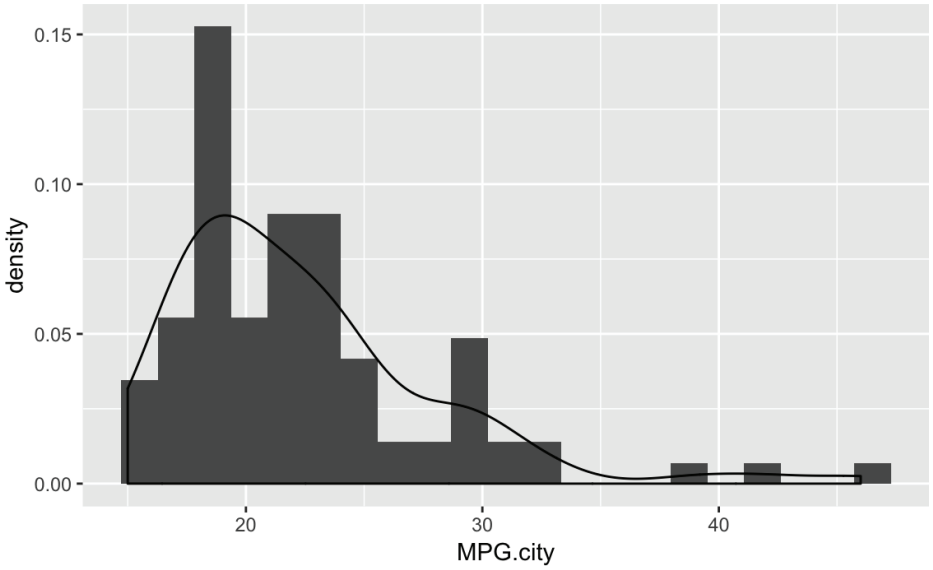


Рис. 10-46. Гистограмма с графиком плотности

Обсуждение

Гистограмма показывает функцию плотности ваших данных, но она грубая. Более точная оценка может помочь вам лучше визуализировать основное распределение. *Ядерная оценка плотности* – это более плавное представление одномерных данных.

В `ggplot` мы говорим функции `geom_histogram` использовать функцию `geom_density`, передавая ей `aes(y = ..density..)`.

В следующем примере мы берем выборку из гамма-распределения, а затем строим гистограмму и расчетную плотность, как показано на рис. 10-47:

```
samp <- rgamma(500, 2, 2)

ggplot() +
 aes(x = samp) +
 geom_histogram(aes(y = ..density..), bins = 10) +
 geom_density()
```

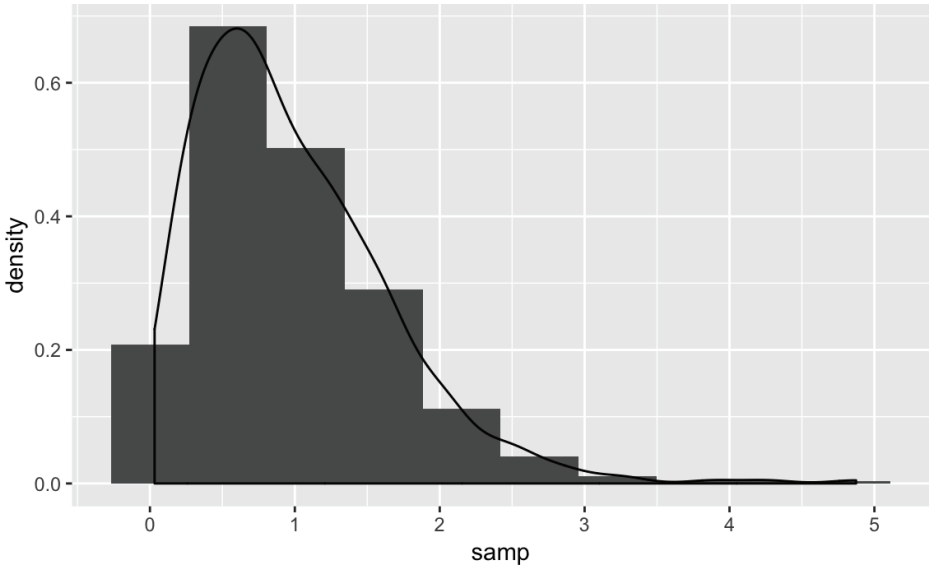


Рис. 10-47. Гистограмма и плотность: гамма-распределение

См. также

Функция `geom_density` аппроксимирует форму плотности непараметрически. Если вы знаете фактическое базовое распределение, используйте рецепт 8.11, чтобы построить функцию плотности.

10.21. Создание графика квантиль-квантиль

Задача

Вы хотите создать график квантиль-квантиль своих данных, обычно потому, что хотите знать, как эти данные отличаются от нормального распределения.

Решение

С помощью `ggplot` мы можем использовать функции `stat_qq` и `stat_qq_line` для создания графика квантиль-квантиль, который показывает наблюдаемые точки, а также линию К-К. На рис. 10-48 показан получившийся график:

```
df <- data.frame(x = rnorm(100))

ggplot(df, aes(sample = x)) +
  stat_qq() +
  stat_qq_line()
```

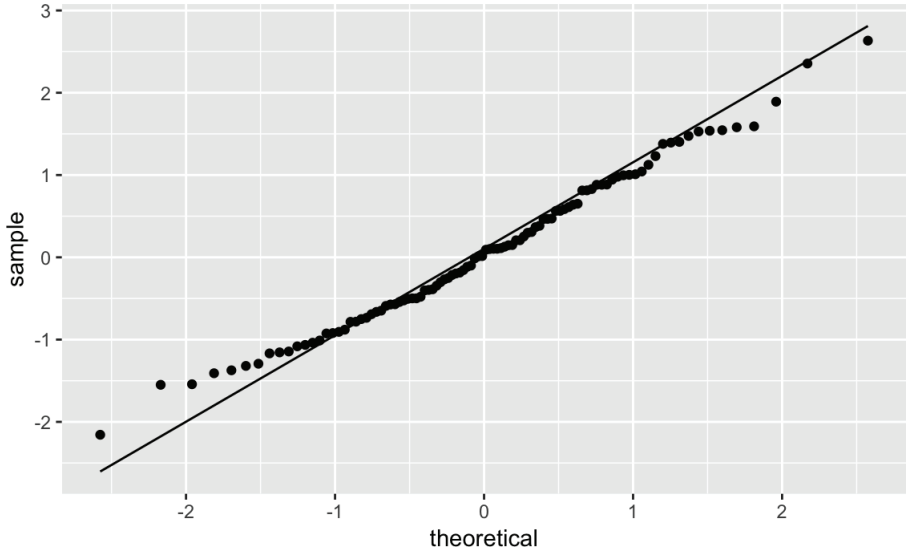


Рис. 10-48. График квантиль-квантиль

Обсуждение

Иногда важно знать, нормально ли распределены ваши данные. График квантиль-квантиль (К-К) хорошо подходит для этого.

Набор данных `Cars93` содержит столбец `Price`. Он нормально распределен? Этот фрагмент кода создает график `Price`, как показано на рис. 10-49:

```
ggplot(Cars93, aes(sample = Price)) +
 stat_qq() +
 stat_qq_line()
```

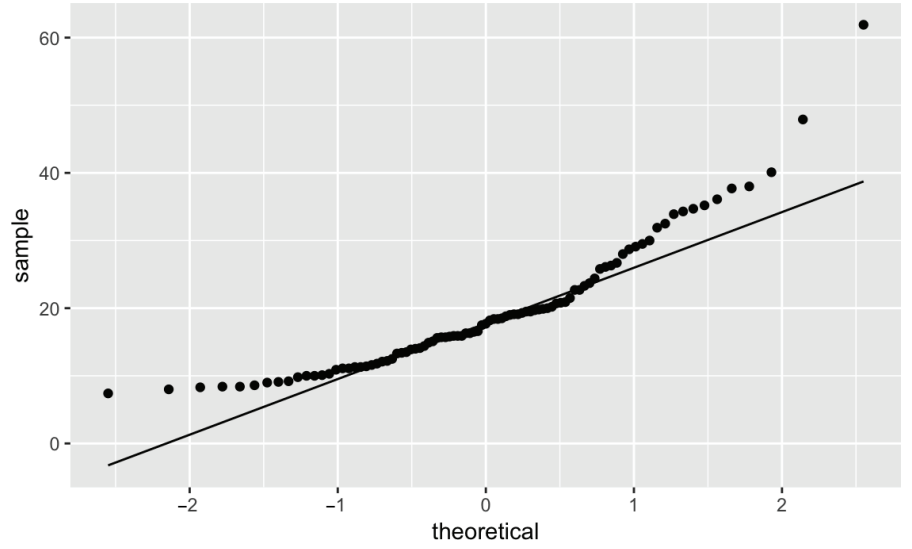


Рис. 10-49. График квантиль-квантиль цен на автомобили

Если бы данные имели идеальное нормальное распределение, то точки находились бы точно на диагональной линии. Многие точки расположены близко друг к другу, особенно в средней части, но точки в хвостах довольно удалены. Слишком много точек над линией, что указывает на общий перекос влево.

Перекос влево можно устранить логарифмическим преобразованием. Мы можем построить `log(Price)`. Результат показан на рис. 10-50:

```
ggplot(Cars93, aes(sample = log(Price))) +
 stat_qq() +
 stat_qq_line()
```

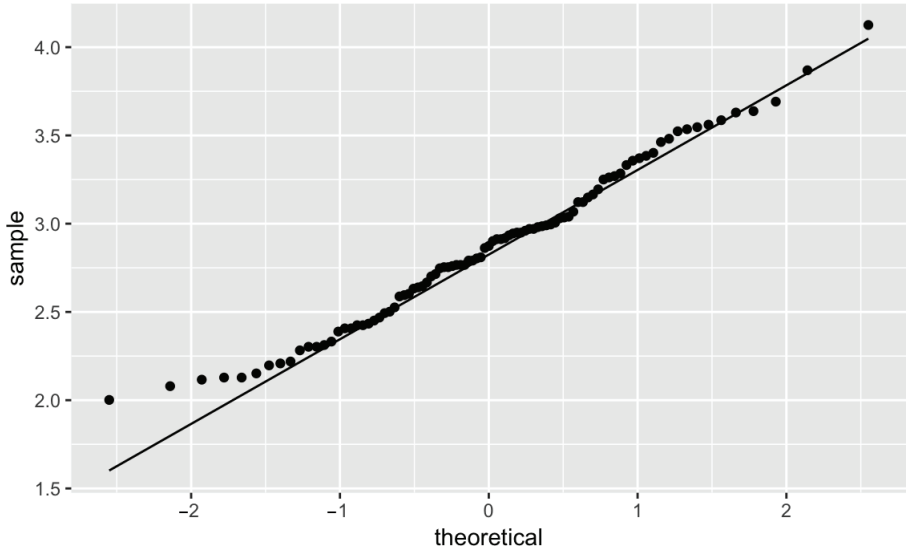


Рис. 10-50. График квантиль-квантиль log(Price)

Обратите внимание на то, что точки на новом графике ведут себя намного лучше, оставаясь близко к линии, за исключением крайнего левого хвоста. Похоже, что `log(Price)` приближенно нормальный.

См. также

См. рецепт 10.22, чтобы узнать, как создавать графики квантиль-квантиль для других распределений. *См.* рецепт 11.16, чтобы узнать, как применять эти графики для диагностики линейной регрессии.

10.22. Создание других графиков квантиль-квантиль

Задача

Вы хотите просмотреть график квантиль-квантиль своих данных, но эти данные не нормально распределены.

Решение

Конечно, здесь вы должны иметь представление о базовом распределении. Решение тут строится на следующих шагах.

1. Используйте функцию `ppoints`, чтобы сгенерировать последовательность точек от 0 до 1.
2. Преобразуйте эти точки в квантили, используя функцию квантиля для предполагаемого распределения.
3. Отсортируйте данные вашей выборки.
4. Разместите отсортированные данные по вычисленным квантилям.
5. Используйте функцию `abline`, чтобы построить диагональную линию.

Все это можно сделать с помощью двух строк кода. Вот пример, который предполагает, что ваши данные, `y`, имеют распределение Стьюдента с пятью степенями свободы. Напомним, что квантильной функцией для этого распределения является функция `qt`, а ее второй аргумент – степени свободы.

Сначала давайте создадим несколько примеров данных:

```
df_t <- data.frame(y = rt(100, 5))
```

Чтобы создать график квантиль-квантиль, нам нужно оценить параметры распределения, которое мы хотим построить. Поскольку это распределение Стьюдента, нам нужно оценить только один параметр – степени свободы. Конечно, мы знаем, что фактические степени свободы равны 5, но в большинстве ситуаций нам нужно будет рассчитать это значение. Поэтому мы будем использовать функцию `MASS::fitdistr` для оценки степеней свободы:

```
est_df <- as.list(MASS::fitdistr(df_t$y, "t")$estimate)[["df"]]
est_df
#> [1] 19.5
```

Как и ожидалось, это довольно близко к тому, что было использовано для генерации смоделированных данных, поэтому давайте передадим предполагаемые степени свободы функциям К-К и создадим то, что показано на рис. 10-51:

```
ggplot(df_t) +
 aes(sample = y) +
 geom_qq(distribution = qt, dparams = est_df) +
 stat_qq_line(distribution = qt, dparams = est_df)
```

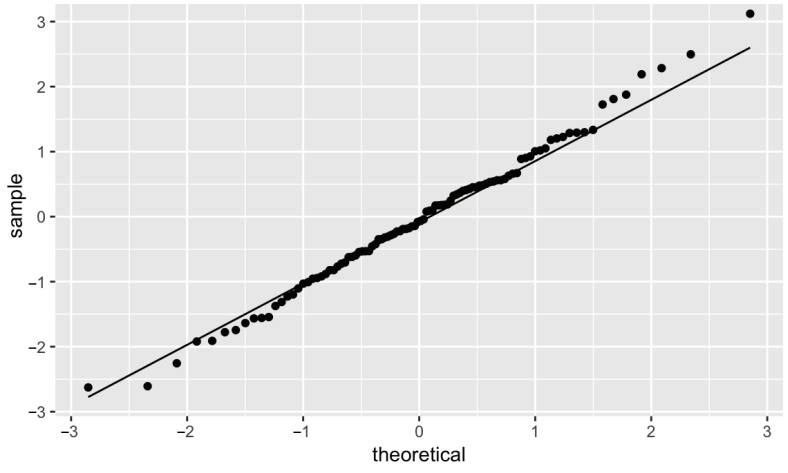


Рис. 10-51. График квантиль-квантиль распределения Стьюдента

Обсуждение

Это решение выглядит сложным, но суть его состоит в том, чтобы выбрать распределение, подобрать параметры, а затем передать эти параметры функциям К-К.

Мы можем проиллюстрировать этот рецепт, взяв случайную выборку из экспоненциального распределения со средним значением 10 (или, что эквивалентно, скоростью 1/10):

```
rate <- 1 / 10
n <- 1000
df_exp <- data.frame(y = rexp(n, rate = rate))

est_exp <- as.list(MASS::fitdistr(df_exp$y, "exponential")$estimate)[["rate"]]
est_exp
#> [1] 0.101
```

Обратите внимание, что в случае с экспоненциальным распределением оцениваемый нами параметр называется `rate`, а не `df`, который был параметром в распределении Стьюдента.

Квантильная функция экспоненциального распределения – это `qexp`. Она принимает аргумент `rate`. На рис. 10-52 показан получившийся график теоретического экспоненциального распределения:

```
ggplot(df_exp) +
 aes(sample = y) +
 geom_qq(distribution = qexp, dparams = est_exp) +
 stat_qq_line(distribution = qexp, dparams = est_exp)
```

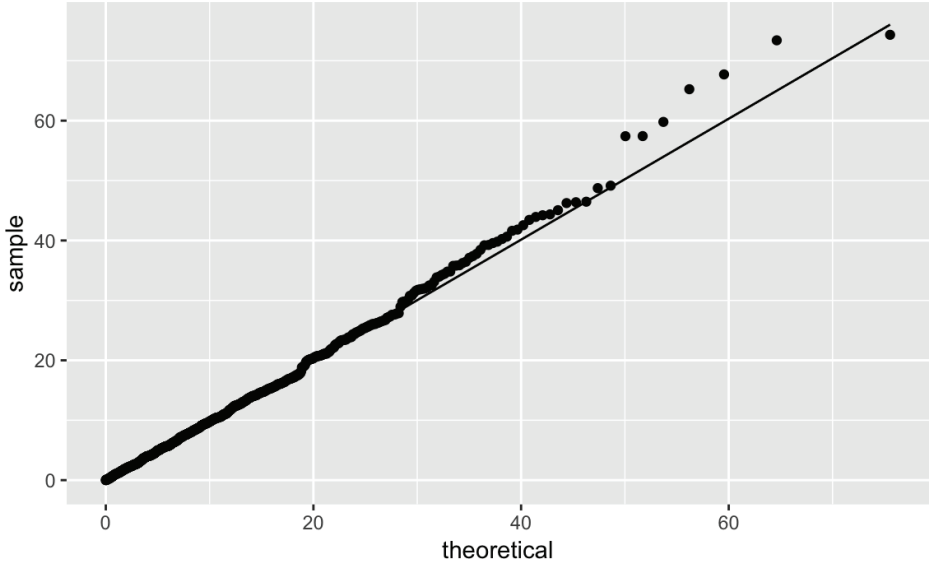


Рис. 10-52. График квантиль-квантиль экспоненциального распределения

10.23. Построение переменной в нескольких цветах

Задача

Вы хотите отобразить свои данные на графике в нескольких цветах, как правило, для того чтобы сделать график более информативным, читабельным или интересным.

Решение

Можно передать цвет функции `geom_` для получения вывода цвета (*см.* рис. 10-53):

```
df <- data.frame(x = rnorm(200), y = rnorm(200))

ggplot(df) +
 aes(x = x, y = y) +
 geom_point(color = "blue")
```

Если вы читаете это в распечатанном виде, то видите только черный цвет. Попробуйте сами, чтобы увидеть полноцветную версию графика.

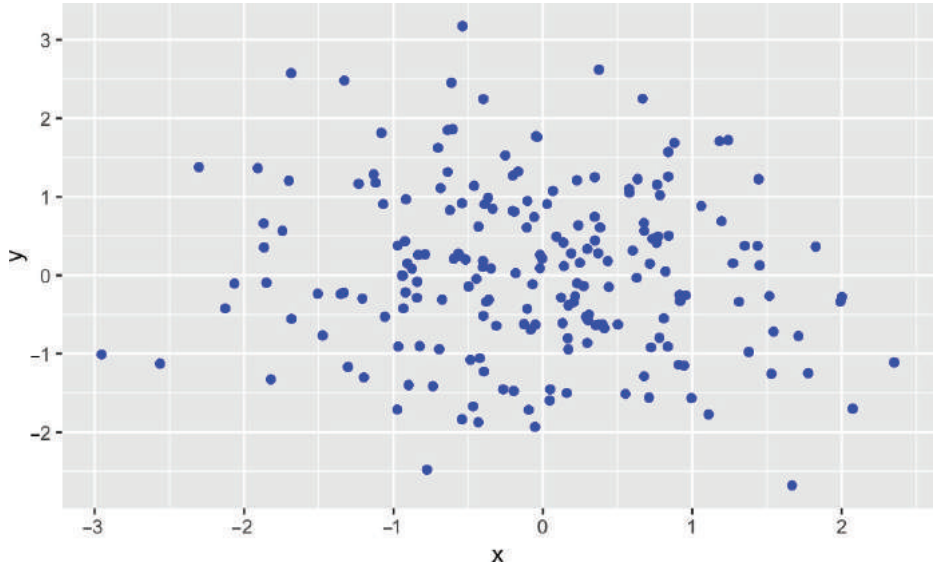


Рис. 10-53. Точечные данные в цвете

Значение `color` может представлять собой:

- один цвет, в этом случае все точки данных являются этим цветом;
- вектор цветов той же длины, что и `x`, и в этом случае каждое значение `x` окрашивается в соответствующий цвет;
- короткий вектор, в этом случае вектор цветов используется повторно.

Обсуждение

Цвет по умолчанию в `ggplot` – черный. Хотя и выглядит это не очень захватывающе, черный цвет контрастен, и практически любой может легко увидеть его.

Однако гораздо полезнее (и интереснее) менять цвет таким образом, чтобы подсвечивать данные. Давайте проиллюстрируем это на примере графика двумя способами: один график будет черно-белый, а другой с простым затенением.

Используя этот код, мы создаем базовый черно-белый график на рис. 10-54:

```
df <- data.frame(
 x = 1:100,
 y = rnorm(100)
)
ggplot(df) +
 aes(x, y) +
 geom_point()
```

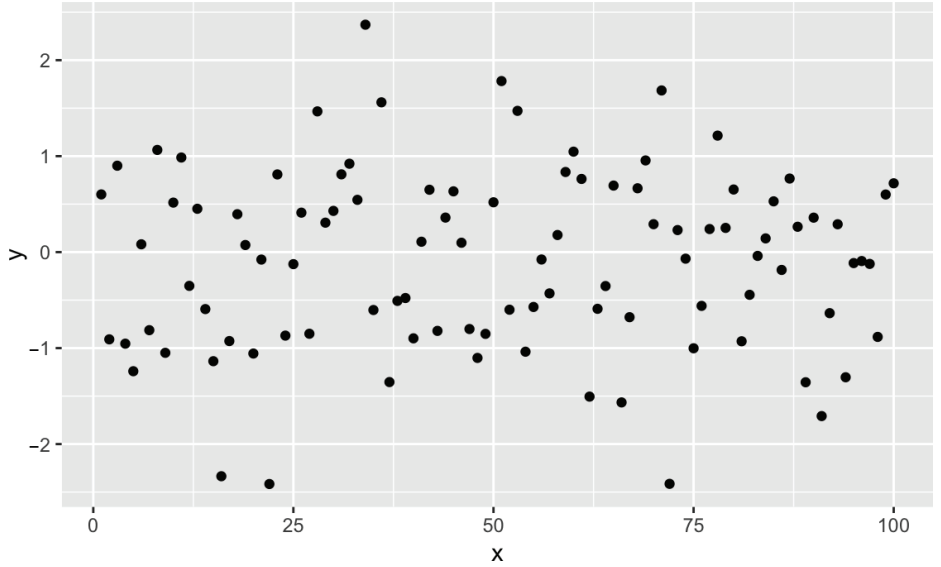


Рис. 10-54. Простой точечный график

Теперь мы можем сделать его более интересным, создав вектор значений `"gray"` и `"black"` в соответствии со знаком `x`, а затем построив график `x` с использованием этих цветов, как показано на рис. 10-55:

```
shade <- if_else(df$y >= 0, "black", "gray")

ggplot(df) +
 aes(x, y) +
 geom_point(color = shade)
```

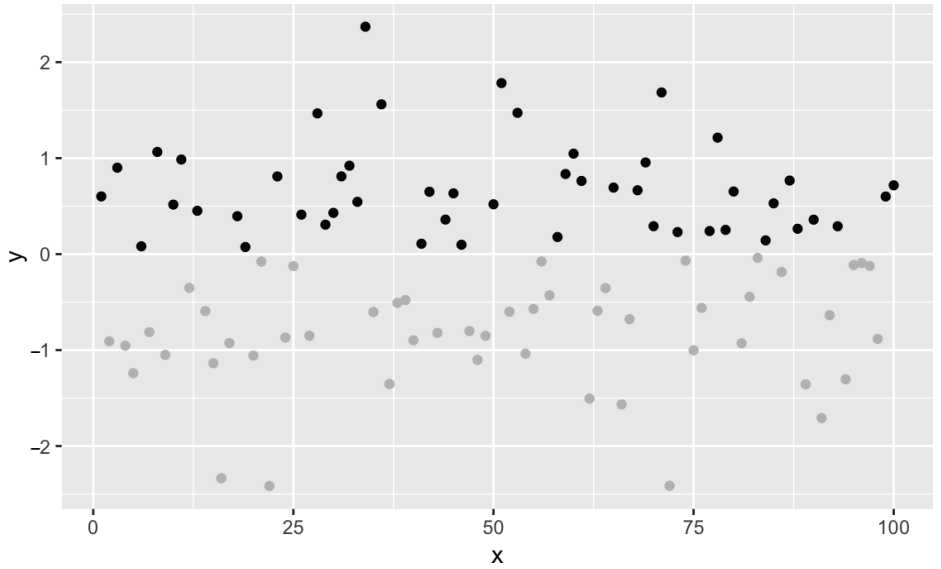


Рис. 10-55. Точечный график с цветными затенениями

Отрицательные значения теперь отображаются серым, поскольку соответствующий элемент `colors` – `"gray"`.

См. также

См. рецепт 5.3, где рассказывается о правиле повторного использования. Выполните функцию `colors`, чтобы увидеть список доступных цветов, и используйте функцию `geom_segment`, чтобы построить отрезки в нескольких цветах.

10.24. График функции

Задача

Вы хотите построить график значения функции.

Решение

Функция `stat_function` отобразит функцию в диапазоне. На рис. 10-56 изображена синусоида в диапазоне от –3 до 3:

```
ggplot(data.frame(x = c(-3, 3))) +
 aes(x) +
 stat_function(fun = sin)
```

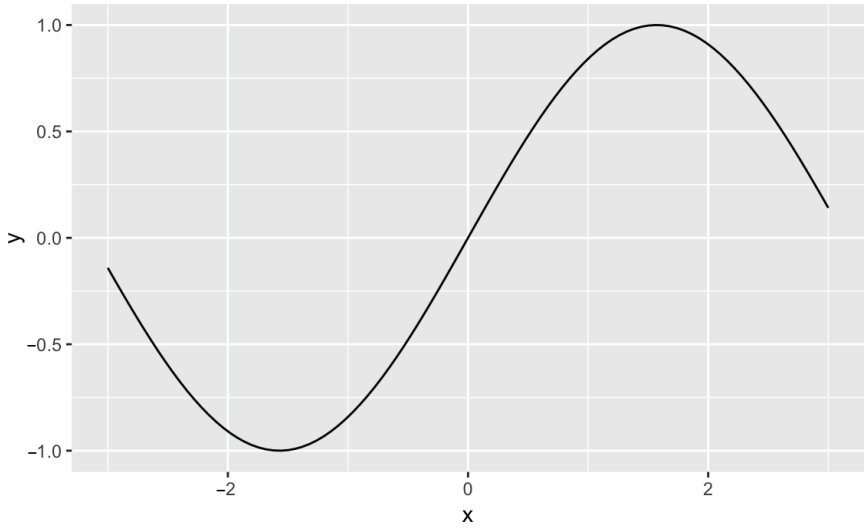


Рис. 10-56. График синусоиды

Обсуждение

Довольно часто возникает необходимость построить статистическую функцию, такую как нормальное распределение, по заданному диапазону. Функция `stat_function` из пакета `ggplot` позволяет сделать это. Нам нужно только предоставить таблицу данных с пределами значений `x`, а `stat_function` рассчитает значения `y` и отобразит результаты, как показано на рис. 10-57:

```
ggplot(data.frame(x = c(-3.5, 3.5))) +
 aes(x) +
 stat_function(fun = dnorm) +
 ggtitle("Standard Normal Density")
```

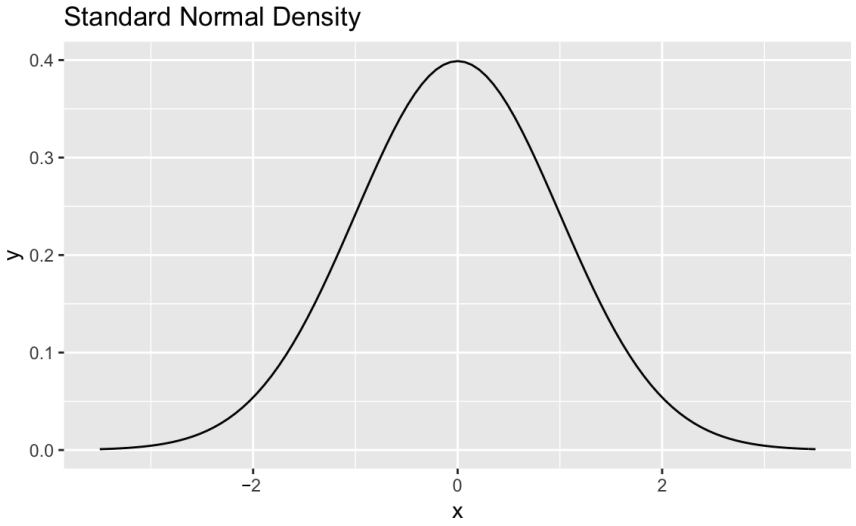


Рис. 10-57. Стандартный график нормальной плотности

Обратите внимание, что мы использовали функцию `ggtitle` для установки заголовка. Если мы устанавливаем несколько текстовых элементов в ggplot, то используем функцию `labs`, но когда мы просто добавляем заголовок, `ggtitle` более лаконична по сравнению с `labs(title='Standard Normal Density')`, хотя они делают одно и то же. *См.* `?labs`, где приводится более подробное обсуждение меток.

Функция `stat_function` может построить любую функцию, которая принимает один аргумент и возвращает одно значение. Давайте создадим функцию и затем построим ее. Наша функция – это затухающая синусоида, то есть синусоида, которая теряет амплитуду по мере удаления от 0:

```
f <- function(x) exp(-abs(x)) * sin(2 * pi * x)

ggplot(data.frame(x = c(-3.5, 3.5))) +
 aes(x) +
 stat_function(fun = f) +
 ggtitle("Dampened Sine Wave")
```

Полученный график показан на рис. 10-58.

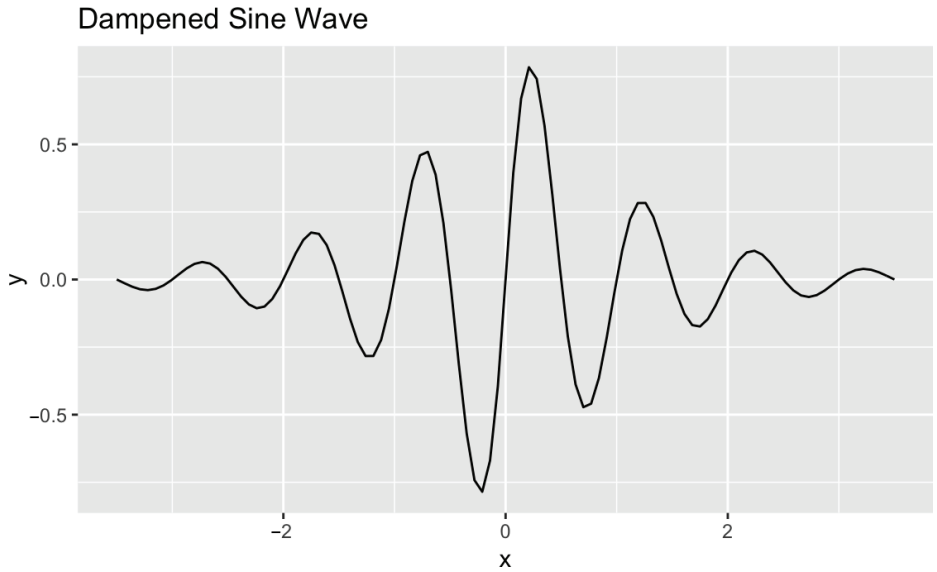


Рис. 10-58. График затухающей синусоиды

См. также

См. рецепт 15.3, чтобы узнать, как определить функцию.

10.25. Отображение нескольких графиков на одной странице

Задача

Вы хотите отобразить несколько графиков бок о бок на одной странице.

Решение

Существует несколько способов поместить графики `ggplot` в сетку, но одним из самых простых в использовании и понимании является `patchwork` от Томаса Лина Педерсена. В настоящее время он недоступен в CRAN, но его можно установить с сайта GitHub, используя пакет `devtools`:

```
devtools::install_github("thomasp85/patchwork")
```

После установки пакета его можно использовать для построения нескольких объектов `ggplot`, используя знак сложения + между объектами, а затем вызвать функцию `plot_layout`, чтобы расположить изображения в сетке, как показано на рис. 10-59. В этом примере мы видим четыре объекта `ggplot`:

```
library(patchwork)
p1 + p2 + p3 + p4
```

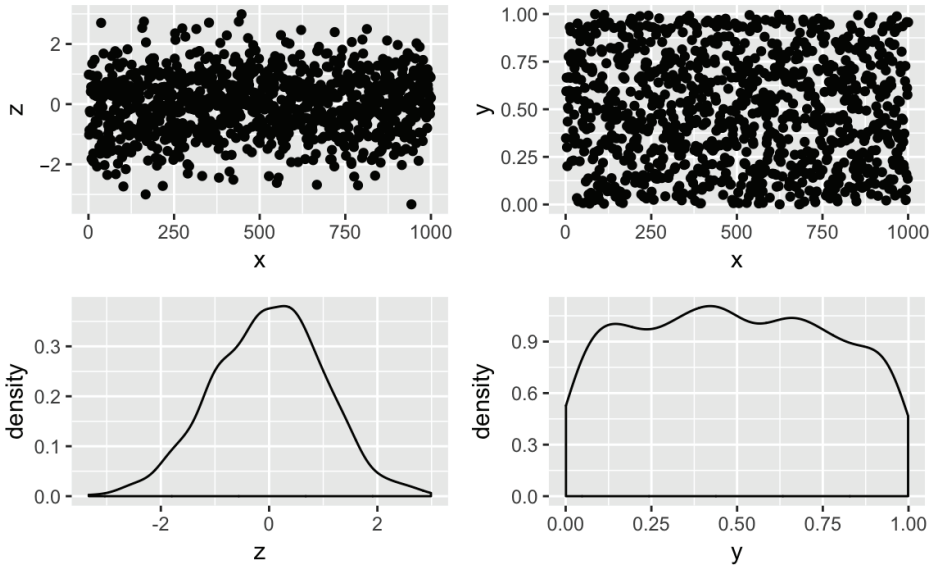


Рис. 10-59. График, построенный с помощью patchwork

`patchwork` поддерживает группировку с круглыми скобками и использование / для размещения групп под другими элементами, как показано на рис. 10-60:

```
p3 / (p1 + p2 + p4)
```

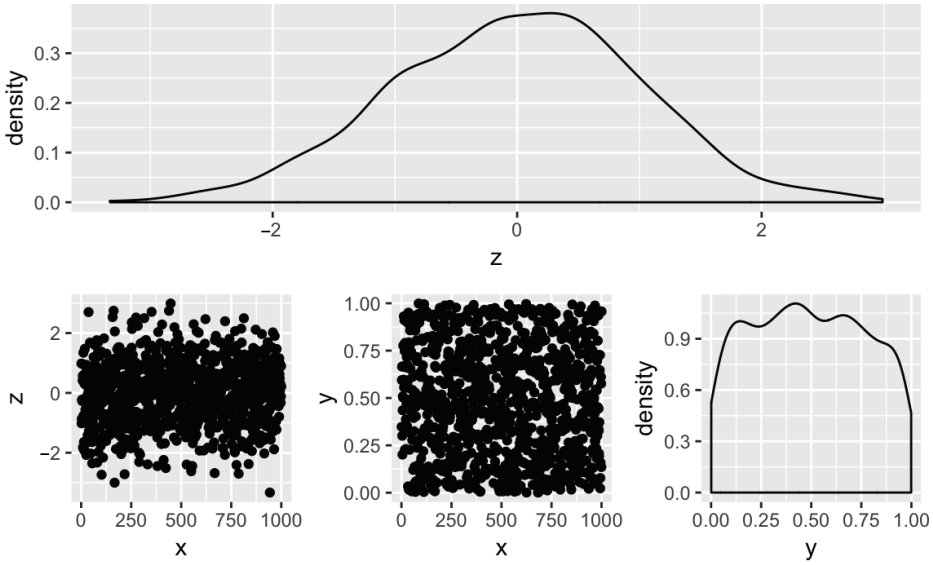


Рис. 10-60. График 1/2

Обсуждение

Давайте использовать такого рода график для отображения четырех разных бета-распределений. Используя функцию `ggplotand` и пакет `patchwork`, мы можем создать эффект макета 2×2, создав четыре графических объекта, а затем вывести их, используя знак +:

```
library(patchwork)

df <- data.frame(x = c(0, 1))

g1 <- ggplot(df) +
 aes(x) +
 stat_function(
    fun = function(x)
      dbeta(x, 2, 4)
 ) +
ggtitle("First")

g2 <- ggplot(df) +
 aes(x) +
 stat_function(
    fun = function(x)
      dbeta(x, 4, 1)
 ) +
ggtitle("Second")

g3 <- ggplot(df) +
 aes(x) +
 stat_function(
    fun = function(x)
```

```
      dbeta(x, 1, 1)
  ) +

ggtitle("Third")

g4 <- ggplot(df) +
 aes(x) +
 stat_function(
    fun = function(x)
      dbeta(x, .5, .5)
 ) +
ggtitle("Fourth")

g1 + g2 + g3 + g4 + plot_layout(ncol = 2, byrow = TRUE)
```

Вывод показан на рис. 10-61.

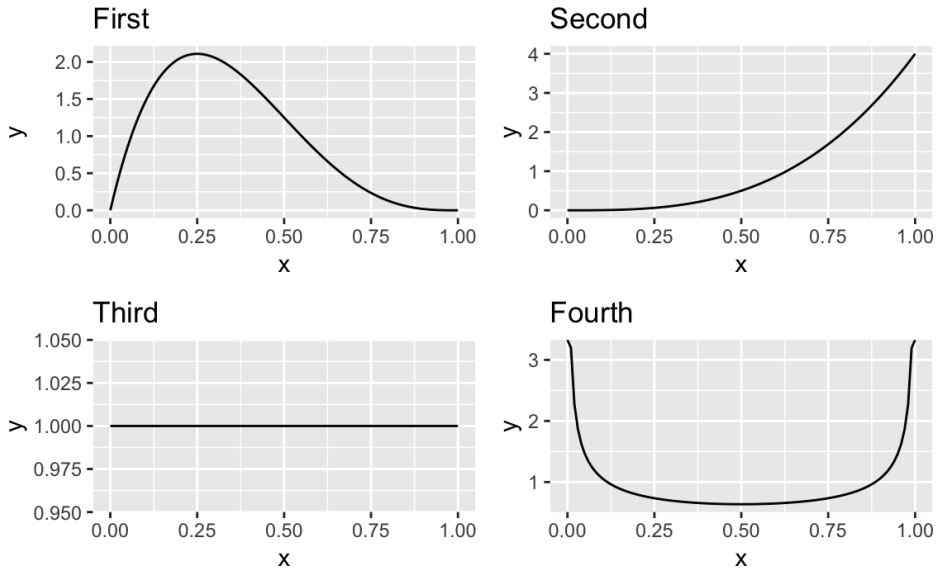


Рис. 10-61. Четыре графика с использованием patchwork

Чтобы расположить изображения в порядке следования столбцов, можно передать `byrow=FALSE` в функцию `plot_layout`:

```
g1 + g2 + g3 + g4 + plot_layout(ncol = 2, byrow = FALSE)
```

См. также

В рецепте 8.11 обсуждается построение функций плотности, как мы делаем это здесь.

В рецепте 10.9 показано, как создать матрицу графиков, используя функцию панели.

Пакеты `grid` и `lattice` содержат дополнительные инструменты для подобного рода макетов с базовой графической системой.

10.26. Запись графика в файл

Задача

Вы хотите сохранить свои графики в файле, таком как файл PNG, JPEG или PostScript.

Решение

Вы можете использовать функцию `ggsave`, чтобы сохранить отображаемый график в файл. Функция сделает некоторые предположения по умолчанию относительно размера и типа файла, позволяя вам указать только имя файла:

```
ggsave("filename.jpg")
```

Тип файла определяется по расширению, используемому в имени файла, которое вы передаете в функцию `ggsave`. Вы можете контролировать детали размера, типа файла и масштаба, передавая параметры в эту функцию. *См.* `?ggsave` для получения конкретных деталей.

Обсуждение

В RStudio нужно щелкнуть кнопкой мыши на надписи Export в окне **Plots** (Графики), а затем нажать на **Save as Image** (Сохранить как изображение), **Save as PDF** (Сохранить как PDF) или **Copy to Clipboard** (Скопировать в буфер обмена). Параметры сохранения предложат вам указать тип файла и имя файла перед записью файла. Параметр **Скопировать в буфер обмена** может быть полезен, если вы вручную копируете и вставляете свои графики в презентацию или текстовый процессор.

Помните, что файл будет записан в ваш текущий рабочий каталог (только если вы не используете абсолютный путь к файлу), поэтому убедитесь, что вы знаете, какой каталог является рабочим, прежде чем вызывать функцию `savePlot`.

В неинтерактивном сценарии, использующем `ggplot`, можно передавать объекты графика непосредственно в функцию `ggsave`, чтобы их не нужно было отображать перед сохранением. В предыдущем рецепте мы создали объект графика с именем `g1`. Мы можем сохранить его в файл:

```
ggsave("g1.png", plot = g1, units = "in", width = 5, height = 4)
```

Обратите внимание, что единицы `height` и `width` в функции `ggsave` указаны с помощью параметра `units`. В этом случае мы использовали слово `in` для обозначения дюймов, но `ggsave` также поддерживает миллиметры и сантиметры для более метрических наклонов.

См. также

См. рецепт 3.1 для получения дополнительной информации о текущем рабочем каталоге.

Глава 11

Линейная регрессия и дисперсионный анализ

В статистике моделирование – это то место, где мы приступаем к делу. Модели количественно определяют связи между нашими переменными. Модели позволяют делать прогнозы.

Простая линейная регрессия – это самая основная модель. Это всего лишь две переменные, которые моделируются как линейная связь с остаточным членом:

$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i$.

У нас есть данные для *x* и *y*. Наша миссия состоит в том, чтобы соответствовать модели, которая даст нам наилучшие оценки β_0 и β_1 (*см.* рецепт 11.1).

Это естественным образом распространяется на множественную линейную регрессию, где у нас есть несколько переменных в правой части связи (*см.* рецепт 11.2):

$y_i = \beta_0 + \beta_1 u_i + \beta_2 v_i + \beta_3 w_i + \varepsilon_i$.

Статистики называют *u*, *v* и *w* *независимыми переменными*, а *y* – *зависимой переменной*. Очевидно, что такая модель имеет смысл только в том случае, если между независимыми и зависимой переменными существует довольно линейная связь, но это требование гораздо менее ограничительно, чем вы думаете. В рецепте 11.12 обсуждается преобразование переменных в (более) линейную связь так, чтобы вы могли использовать хорошо развитый механизм линейной регрессии.

Прелесть R состоит в том, что каждый может создать эти линейные модели. Модели строятся с помощью функции `lm`, которая возвращает объект модели. Из этого объекта мы получаем коэффициенты (β_i) и статистику регрессии. Это просто. В самом деле!

Ужас R также заключается в том, что любой может построить эти модели. Ничто не требует от вас проверки того, что модель является разумной, а тем более статистически значимой. Прежде чем слепо поверить модели, проверьте ее! Бóльшая часть необходимой вам информации находится в сводке регрессии (*см.* рецепт 11.4).

Является ли эта модель статистически значимой?

Проверьте *F*-значение в нижней части сводки.

Значимы ли коэффициенты?

Проверьте *t*-статистику коэффициента и *p*-значения в сводке или их доверительные интервалы (*см.* рецепт 11.14).

Является ли эта модель полезной?

Проверьте коэффициент детерминации R^2 в нижней части сводки.

Хорошо ли эта модель соответствует данным?

Составьте график невязок и проверьте диагностику регрессии (*см.* рецепты 11.15 и 11.16).

Удовлетворяют ли данные предположениям, лежащим в основе линейной регрессии?

Проверьте, подтверждает ли диагностика, что линейная модель подходит для ваших данных (*см.* рецепт 11.14).

Дисперсионный анализ

Дисперсионный анализ[1] представляет собой мощный статистический метод. Студентам-первокурсникам, которые проходят статистику, преподают дисперсионный анализ практически сразу же, поскольку это важная тема как с теоретической, так и с практической точки зрения. Однако мы часто поражаемся тому, насколько люди за пределами области не знают о ее цели и ценности.

Регрессия создает модель, а дисперсионный анализ является одним из методов оценки таких моделей. Математика дисперсионного анализа переплетена с математикой регрессии, поэтому статистики обычно представляют их вместе; здесь и мы следуем этой традиции.

В действительности дисперсионный анализ – это семейство методов, которые связаны общим математическим анализом. В данной главе упоминается несколько приложений:

Однофакторный дисперсионный анализ

Это самое простое приложение. Предположим, у вас есть выборки данных из нескольких совокупностей, и вам интересно, разные ли у этих групп средние значения. Однофакторный дисперсионный анализ дает ответ на этот вопрос. Если совокупности имеют нормальные распределения, используйте функцию `oneway.test` (*см.* рецепт 11.21); в противном случае используйте непараметрическую версию, функцию `kruskal.test` (*см.* рецепт 11.24).

Сравнение моделей

Когда вы добавляете или удаляете независимую переменную в линейной регрессии, вы хотите знать, улучшило ли это изменение модели. Функция `anova` сравнивает две модели регрессии и вычисляет, значительно ли они отличаются (*см.* рецепт 11.25).

Таблица дисперсионного анализа

Функция `anova` также может построить таблицу дисперсионного анализа модели линейной регрессии, которая включает в себя F-значение, необходимое для измерения статистической значимости модели (*см.* рецепт 11.3). Эта важная таблица обсуждается почти в каждом учебнике по регрессии.

---

[1] В англоязычной литературе также встречается обозначение ANOVA (от англ. *ANalysis Of VAriance*).

Пример данных

Во многих примерах, приведенных в этой главе, мы начинаем с создания примера данных, используя возможности генерации псевдослучайных чисел, предоставляемые R. Поэтому в начале каждого рецепта вы можете увидеть нечто наподобие этого:

```
set.seed(42)
x <- rnorm(100)
e <- rnorm(100, mean=0, sd=5)
y <- 5 + 15 * x + e
```

Мы используем функцию `set.seed` для назначения начального числа при генерации случайных чисел, поэтому если вы запустите этот код на своем компьютере, то получите тот же ответ. В предыдущем примере `x` – это вектор из 100 отрисовок из стандартного нормального (`mean=0, sd=1`) распределения. Затем мы создаем небольшой случайный шум, `e`, из нормального распределения с `mean = 0` и `sd = 5`, а потом рассчитываем `y` как `5 + 15 * x + e`. Идея создания примера «игрушечных» данных вместо использования «реальных» данных заключается в том, что с помощью смоделированных данных можно изменить коэффициенты и параметры и увидеть, как это изменение повлияет на полученную модель. Например, можно увеличить стандартное отклонение `e` в примере данных и посмотреть, как это повлияет на *R^2* вашей модели.

См. также

Существует много хороших текстов по линейной регрессии. Один из наших фаворитов – четвертое издание книги *Applied Linear Regression Models* Майкла Катнера, Кристофера Нахтсхайма и Джона Нетера (издательство *McGraw-Hill/Irwin*). В этой главе мы обычно следуем их терминологии и соглашениям.

Нам также нравится книга *Linear Models with R* Джулиана Фаравея (издательство *Chapman & Hall / CRC*), потому что она иллюстрирует регрессию с использованием R и вполне читабельна. Более ранние версии работы Фаравея также доступны бесплатно по адресу https://cran.r-project.org/doc/contrib/Faraway-PRA.pdf.

11.1. Простая линейная регрессия

Задача

У вас есть два вектора, *x* и *y*, которые содержат зависимые наблюдения: (x_1, y_1), (x_2, y_2), …, (x_n, y_n). Вы полагаете, что между *x* и *y* существует линейная связь, и вы хотите создать регрессионную модель этой связи.

Решение

Функция `lm` выполняет линейную регрессию и вычисляет коэффициенты.

Если ваши данные в векторах:

```
lm(y ~ x)
```

Или если ваши данные находятся в столбцах таблицы данных:

```
lm(y ~ x, data = df)
```

Обсуждение

Простая линейная регрессия включает в себя две переменные: независимую переменную, которую часто называют *x*, и зависимую переменную, которую нередко называют *y*. Регрессия использует обычный алгоритм *наименьших квадратов* для соответствия линейной модели:

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i,$$

где β_0 и β_1 – коэффициенты регрессии, а ε_i – остаточные члены.

Функция `lm` может выполнять линейную регрессию. Основным аргументом является формула модели, например `y ~ x`. Слева от знака тильды (~) идет зависимая переменная, а справа – независимая. Функция вычисляет коэффициенты регрессии β_0 и β_1 как свободный член и коэффициент `x` соответственно:

```
set.seed(42)
x <- rnorm(100)
e <- rnorm(100, mean = 0, sd = 5)
y <- 5 + 15 * x + e

lm(y ~ x)
#>
#> Call:
#> lm(formula = y ~ x)
#>
#> Coefficients:
#> (Intercept) x
#> 4.56 15.14
```

В этом случае уравнение регрессии имеет вид:

$$y_i = 4.56 + 15.14xi + \varepsilon i.$$

Довольно часто данные фиксируются внутри таблицы данных, и в этом случае вам нужно выполнить регрессию между двумя столбцами таблицы данных. Здесь `x` и `y` – столбцы таблицы данных `dfrm`:

```
df <- data.frame(x, y)
head(df)
#>        x       y
#> 1   1.371   31.57
#> 2   -0.565  1.75
#> 3   0.363   5.43
#> 4   0.633   23.74
#> 5   0.404   7.73
#> 6   -0.106  3.94
```

Функция `lm` позволяет вам указать таблицу данных с помощью параметра `data`. Если вы это сделаете, функция возьмет переменные из таблицы данных, а не из вашей рабочей области:

```
lm(y ~ x, data = df)     # Берем x и y из df.
#>
#> Call:
#> lm(formula = y ~ x, data = df)
#>
```

```
#> Coefficients:
#> (Intercept)    x
#>      4.56    15.14
```

11.2. Множественная линейная регрессия

Задача

У вас есть несколько независимых переменных (например, *u*, *v* и *w*) и зависимая переменная *y*. Вы полагаете, что между независимыми и зависимой переменными существует линейная связь, и вы хотите выполнить линейную регрессию для данных.

Решение

Используйте функцию `lm`. Укажите несколько независимых переменных в правой части формулы, разделенных знаком плюс (+):

```
lm(y ~ u + v + w)
```

Обсуждение

Множественная линейная регрессия – это очевидное обобщение простой линейной регрессии. Она позволяет использовать несколько независимых переменных вместо одной и по-прежнему использует метод наименьших квадратов для вычисления коэффициентов линейного уравнения. Приведенная ниже регрессия с тремя переменными соответствует этой линейной модели:

$$y_i = \beta_0 + \beta_1 u_i + \beta_2 v_i + \beta_3 w_i + \varepsilon_i.$$

R использует функцию `lm` как для простой, так и для множественной линейной регрессии. Вы просто добавляете дополнительные переменные в правую часть формулы модели. Выходные данные затем показывают коэффициенты подогнанной модели. Давайте настроим несколько примеров случайных нормальных данных, используя функцию `rnorm`:

```
set.seed(42)

u <- rnorm(100)
v <- rnorm(100, mean = 3, sd = 2)
w <- rnorm(100, mean = -3, sd = 1)
e <- rnorm(100, mean = 0, sd = 3)
```

Затем мы можем создать уравнение, используя известные коэффициенты для вычисления нашей переменной *y*:

```
y <- 5 + 4 * u + 3 * v + 2 * w + e
```

Теперь если мы запустим линейную регрессию, то увидим, что R определяет коэффициенты и подходит очень близко к только что использованным фактическим значениям:

```
lm(y ~ u + v + w)
#>
#> Call:
#> lm(formula = y ~ u + v + w)
```

```
#>
#> Coefficients:
#> (Intercept)    u     v     w
#>        4.77 4.17  3.01  1.91
```

Параметр `lm` особенно полезен, когда число переменных увеличивается, поскольку гораздо проще хранить свои данные в одной таблице данных, чем во множестве отдельных переменных. Предположим, что ваши данные записаны в таблицу данных, например в показанную здесь переменную `df`:

```
df <- data.frame(y, u, v, w)
head(df)
#>      y     u       v     w
#> 1 16.67 1.371   5.402 -5.00
#> 2 14.96 -0.565  5.090 -2.67
#> 3 5.89  0.363   0.994 -1.83
#> 4 27.95 0.633   6.697 -0.94
#> 5 2.42  0.404   1.666 -4.38
#> 6 5.73  -0.106  3.211 -4.15
```

Когда вы предоставляете `df` параметру `lm`, R ищет регрессионные переменные в столбцах таблицы данных:

```
lm(y ~ u + v + w, data = df)
#>
#> Call:
#> lm(formula = y ~ u + v + w, data = df)
#>
#> Coefficients:
#> (Intercept)    u     v     w
#>        4.77 4.17  3.01  1.91
```

См. также

См. рецепт 11.1, где говорится о простой линейной регрессии.

11.3. Получение регрессионной статистики

Задача

Вам нужна критическая статистика и информация о вашей регрессии, такая как R^2, F-значение, доверительные интервалы коэффициентов, невязки, таблица дисперсионного анализа и т. д.

Решение

Сохраните регрессионную модель в переменной, например `m`:

```
m <- lm(y ~ u + v + w)
```

Затем используйте функции для извлечения регрессионной статистики и информации из модели:

```
anova(m)
```

Таблица дисперсионного анализа.

`coefficients(m)`

Коэффициенты модели.

`coef(m)`

То же, что и `coefficients(m)`.

`confint(m)`

Доверительные интервалы коэффициентов регрессии.

`deviance(m)`

Сумма квадратов невязок.

`effects(m)`

Вектор ортогональных эффектов.

`fitted(m)`

Вектор подогнанных значений y.

`residuals(m)`

Невязки модели.

`resid(m)`

То же, что и `residuals(m)`.

`summary(m)`

Основные статистические данные, такие как R^2, F-значение и стандартная ошибка невязки (σ).

`vcov(m)`

Дисперсионно-ковариационная матрица основных параметров.

Обсуждение

Когда мы начали использовать R, в документации говорилось, что для выполнения линейной регрессии используется функция `lm`. Поэтому мы сделали нечто наподобие этого, получив результат, показанный в рецепте 11.2:

```
lm(y ~ u + v + w)
#>
#> Call:
#> lm(formula = y ~ u + v + w)
#>
#> Coefficients:
#> (Intercept)      u      v    w
#>        4.77   4.17   3.01   1.91
```

Какое разочарование! Этот результат – ничто по сравнению с другими пакетами статистики, такими как SAS. Где коэффициент детерминации R^2? Где доверительные интервалы коэффициентов? Где F-статистика, ее p-значение и таблица дисперсионного анализа?

Конечно, вся эта информация доступна – нужно просто попросить. Другие статистические системы сбрасывают все и позволяют вам пройти через это. R более минималистский. Он выводит простой результат и позволяет делать запрос, если вам еще что-то нужно.

Функция `lm` возвращает объект модели, который вы можете присвоить переменной:

```
m <- lm(y ~ u + v + w)
```

Из объекта модели можно извлечь важную информацию, используя специальные функции. Наиболее важная – функция `summary`:

```
summary(m)
```

```
#>
#> Call:
#> lm(formula = y ~ u + v + w)
#>
#> Residuals:
#>    Min     1Q    Median     3Q    Max
#> -5.383  -1.760  -0.312  1.856 6.984
#>
#> Coefficients:
#>             Estimate Std. Error  t value Pr(>|t|)
#> (Intercept) 4.770       0.969    4.92  3.5e-06 ***
#> u           4.173       0.260    16.07 < 2e-16 ***
#> v           3.013       0.148    20.31 < 2e-16 ***
#> w ``````````1.905  ``````0.266  ``7.15``1.7e-10 ***
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 2.66 on 96 degrees of freedom
#> Multiple R-squared: 0.885, Adjusted R-squared: 0.882
#> F-statistic: 247 on 3 and 96 DF, p-value: <2e-16
```

Здесь показаны оценочные коэффициенты, критическая статистика (такие как коэффициент детерминации R^2 и F-статистика), а также оценка σ – стандартной ошибки невязки. Общая информация настолько важна, что существует целый рецепт, посвященный ей (рецепт 11.4).

Существуют специальные функции извлечения другой важной информации:

Модельные коэффициенты (точечные оценки)

```
coef(m)
#> (Intercept)    u     v     w
#>        4.77  4.17  3.01  1.91
```

Доверительные интервалы коэффициентов модели

```
confint(m)
#>             2.5 % `97.5 %
#> (Intercept) 2.85    6.69
#> u           3.66    4.69
#> v           2.72    3.31
#> w           1.38    2.43
```

Невязки модели

```
resid(m)
#>        1        2        3        4        5        6        7        8        9
#>  -0.5675   2.2880   0.0972   2.1474  -0.7169  -0.3617   1.0350   2.8040  -4.2496
#>       10       11       12       13       14       15       16       17       18
#>  -0.2048  -0.6467  -2.5772  -2.9339  -1.9330   1.7800  -1.4400  -2.3989   0.9245
#>       19       20       21       22       23       24       25       26       27
#>  -3.3663   2.6890  -1.4190   0.7871   0.0355  -0.3806   5.0459  -2.5011   3.4516
#>       28       29       30       31       32       33       34       35       36
#>   0.3371  -2.7099  -0.0761   2.0261  -1.3902  -2.7041   0.3953   2.7201  -0.0254
#>       37       38       39       40       41       42       43       44       45
#>  -3.9887  -3.9011  -1.9458  -1.7701  -0.2614   2.0977  -1.3986  -3.1910   1.8439
#>       46       47       48       49       50       51       52       53       54
#>   0.8218   3.6273  -5.3832   0.2905   3.7878   1.9194  -2.4106   1.6855  -2.7964
#>       55       56       57       58       59       60       61       62       63
#>  -1.3348   3.3549  -1.1525   2.4012  -0.5320  -4.9434  -2.4899  -3.2718  -1.6161
#>       64       65       66       67       68       69       70       71       72
#>  -1.5119  -0.4493  -0.9869   5.6273  -4.4626  -1.7568   0.8099   5.0320   0.1689
#>       73       74       75       76       77       78       79       80       81
#>   3.5761  -4.8668   4.2781  -2.1386  -0.9739  -3.6380   0.5788   5.5664   6.9840
#>       82       83       84       85       86       87       88       89       90
#>  -3.5119   1.2842   4.1445  -0.4630  -0.7867  -0.7565   1.6384   3.7578   1.8942
#>       91       92       93       94       95       96       97       98       99
#>   0.5542  -0.8662   1.2041  -1.7401  -0.7261   3.2701   1.4012   0.9476  -0.9140
#>      100
#>   2.4278
```

Остаточная сумма квадратов

```
deviance(m)
#> [1] 679
```

Таблица дисперсионного анализа

```
anova(m)
#> Analysis of Variance Table
#>
#> Response: y
#>             Df  Sum Sq  Mean Sq  F  value  Pr(>F)
#> u            1    1776     1776     251.0 < 2e-16  ***
#> v            1    3097     3097     437.7 < 2e-16  ***
#> w            1     362      362      51.1 1.7e-10  ***
#> Residuals   96     679        7
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Если вас раздражает сохранение модели в переменной, можно использовать один из таких вариантов:

```
summary(lm(y ~ u + v + w))
```

Или конвейеры `magrittr`:

```
lm(y ~ u + v + w) %>%
summary
```

См. также

См. рецепт 11.4 для получения более подробной информации о сводке регрессии. *См.* рецепт 11.17 для статистики регрессии, специфичной для диагностики модели.

11.4. Общая информация о регрессии

Задача

Вы создали модель линейной регрессии, `m`. Тем не менее вас смущает вывод функции `summary(m)`.

Обсуждение

Сводка модели важна, потому что она связывает вас с наиболее важной регрессионной статистикой. Вот сводка модели из рецепта 11.3:

```
summary(m)
#>
#> Call:
#> lm(formula = y ~ u + v + w)
#>
#> Residuals:
#>    Min     1Q  Median  3Q   Max
#> -5.383 -1.760 -0.312 1.856 6.984
#>
#> Coefficients:
#>             Estimate Std. Error t value  Pr(>|t|)
#>   (Intercept)   4.770   0.969   4.92   3.5e-06     ***
#>   u             4.173   0.260  16.07   < 2e-16     ***
#>   v             3.013   0.148  20.31   < 2e-16     ***
#>   w             1.905   0.266   7.15   1.7e-10     ***
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 2.66 on 96 degrees of freedom
#> Multiple R-squared: 0.885, Adjusted R-squared: 0.882
#> F-statistic: 247 on 3 and 96 DF, p-value: <2e-16
```

Давайте разберем эту сводку по разделам. Мы будем читать ее сверху вниз, хотя самая важная статистика (статистика F) появляется в конце:

Вызов

```
#> lm(formula = y ~ u + v + w)
```

Показывает, как была вызвана функция `lm` при создании модели, что важно для помещения этой сводки в соответствующий контекст.

Статистика невязок

```
#> Residuals:
#>      Min       1Q   Median       3Q      Max
#> -5.383   -1.760   -0.312    1.856   6.984
```

В идеале невязки регрессии должны иметь прекрасное нормальное распределение. Эти статистические данные помогут вам определить возможные отклонения от нормальности. Алгоритм наименьших квадратов математически гарантирует получение невязок со средним значением, равным нулю[1], следовательно, знак медианы указывает направление асимметрии, а величина медианы указывает на предел. В этом случае медиана отрицательна, что говорит о перекосе влево.

Если невязки имеют хорошее колоколообразное распределение, то первый квартиль (1Q) и третий квартиль (3Q) должны иметь примерно одинаковую величину. В этом примере большая величина 3Q против 1Q (1,856 против 1,76) указывает на небольшой перекос вправо в наших данных, хотя отрицательная медиана делает ситуацию менее четкой.

Невязки `Min` и `Max` предлагают быстрый способ обнаружения экстремальных выбросов в данных, поскольку экстремальные выбросы (в зависимой переменной) дают большие невязки.

Коэффициенты

```
#> Coefficients:
#>             Estimate Std. Error t  value   Pr(>|t|)
#> (Intercept) 4.770    0.969     4.92    3.5e-06   ***
#> u           4.173    0.260     16.07   < 2e-16   ***
#> v           3.013    0.148     20.31   < 2e-16   ***
#> w           1.905    0.266     7.15    1.7e-10   ***
```

Столбец `Estimate` содержит оцененные регрессионные коэффициенты, рассчитанные по методу наименьших квадратов.

Теоретически, если коэффициент переменной равен нулю, переменная ничего не стоит; она ничего не добавляет в модель. Тем не менее показанные здесь коэффициенты являются лишь оценочными, и они никогда не будут точно равны нулю. Поэтому мы спрашиваем: статистически говоря, насколько вероятно, что истинный коэффициент равен нулю? В этом и состоит цель *t*-статистики и *p*-значений, которые в сводке помечены (соответственно) как `t-value` и `Pr(>|t|)`.

p-значение – это вероятность. Оно измеряет вероятность того, что коэффициент не значителен, поэтому чем меньше, тем лучше. Большой – это плохо, потому что это указывает на высокую вероятность незначительности. В этом примере *p*-значение коэффициента `u` составляет всего 0,00106, поэтому он, вероятно, является значимым. Однако *p*-значение коэффициента `w` составляет 0,05744 – чуть больше нашего обычного предела 0,05, а это говорит о том, что `w`, вероятно, не является значимым[2]. Переменные с большими *p*-значениями – это кандидаты на исключение.

---

[1] Только если вы не выполняли линейную регрессию без свободного члена (см. рецепт 11.5).

[2] Уровень значимости $\alpha = 0{,}05$ – соглашение, соблюдаемое в этой книге. Вместо этого ваше приложение может использовать $\alpha = 0{,}10$, $\alpha = 0{,}01$ или другое значение. *См.* введение к главе 9.

Удобной особенностью является то, что R помечает значимые переменные для быстрой идентификации. Вы, наверное, обратили внимание на крайний правый столбец, содержащий тройные звездочки (*)?

Другие значения, которые можно увидеть в этом столбце, – двойная звездочка (**), звездочка (*) и точка (.). В этом столбце выделены значимые переменные. Строка, начинающаяся со слов `Signif.codes` в нижней части раздела «Коэффициенты», дает краткую справку по значениям флагов. Их можно интерпретировать следующим образом:

Индикация значимости	Значение
***	*p*-значение от 0 до 0.001
**	*p* -значение от 0.001 до 0.01
*	*p* -значение от 0.01 до 0.05
.	*p* -значение от 0.05 до 0.1
(пробел)	*p* -значение от 0.1 до 1.0

Колонка с надписью `Std. Error` – стандартная ошибка оцененного коэффициента.

Столбец, помеченный как `t value`, – *t*-статистика, из которой было рассчитано *p*-значение.

Стандартная ошибка невязки

```
# Residual standard error: 2.66 on 96 degrees of freedom
```

Вычисляет стандартную ошибку невязки (σ), то есть стандартное отклонение выборки ε.

R^2 (коэффициент детерминации)

```
# Multiple R-squared: 0.885,   Adjusted R-squared: 0.882
```

R^2 – это показатель качества модели. Чем больше, тем лучше. С математической точки зрения это доля дисперсии *y*, которая объясняется регрессионной моделью. Оставшуюся дисперсию нельзя объяснить с помощью модели, поэтому она должна быть обусловлена другими факторами (т. е. неизвестными переменными или изменчивостью выборки). В этом случае модель объясняет 0,4981 (49,81 %) дисперсии *y*, а оставшиеся 0,5019 (50,19 %) не объяснены.

При этом мы настоятельно рекомендуем использовать скорректированный, а не базовый коэффициент детерминации R^2.

Скорректированное значение учитывает количество переменных в вашей модели и поэтому представляет собой более реалистичную оценку его эффективности. В этом случае мы бы использовали 0,8815, а не 0,8851.

F-статистика

```
# F-statistic: 246.6 on 3 and 96 DF, p-value: < 2.2e-16
```

F-статистика сообщает вам, является данная модель значимой или нет. Модель является значимой, если любой из коэффициентов отличен от нуля (т. е. если $\beta_i \neq 0$ для некоего *i*), и незначимой, если все коэффициенты равны нулю ($\beta_1 = \beta_2 = \ldots = \beta_n = 0$).

Обычно *p*-значение меньше 0,05 указывает на то, что модель, скорее всего, является значимой (один или несколько коэффициентов β_i отличны от нуля), тогда как значения, превышающие 0,05, указывают на то, что модель, скорее всего, не значима. В данном случае вероятность того, что наша модель не является значимой, составляет всего лишь 2.2e – 16. Это хорошо.

Большинство сначала смотрит на статистику R^2. Специалист по статистике разумно начинает с F-статистики, потому что если модель не значима, остальное не имеет значения.

См. также

См. рецепт 11.3 для получения дополнительной информации по извлечению статистики и сведений из объекта модели.

11.5. Линейная регрессия без свободного члена

Задача

Вы хотите выполнить линейную регрессию, но хотите, чтобы свободный член уравнения был равен нулю.

Решение

Добавьте «+0» в правую часть формулы регрессии. Это заставит `lm` соответствовать модели со свободным членом, равным нулю:

```
lm(y ~ x + 0)
```

Соответствующее уравнение регрессии выглядит так:

$$y_i = \beta x_i + \varepsilon_i.$$

Обсуждение

Линейная регрессия обычно включает в себя свободный член уравнения, поэтому в R это значение по умолчанию. Однако в редких случаях вы можете захотеть подогнать данные, предполагая, что свободный член равен нулю. В этом случае вы делаете допущение, принимаемое при моделировании: когда *x* равен нулю, *y* тоже должен быть равен нулю.

Когда вы принудительно используете свободный член уравнения, равный нулю, вывод `lm` включает в себя коэффициент для *x*, но не свободный член для *y*, как показано здесь:

```
lm(y ~ x + 0)
#>
#> Call:
#> lm(formula = y ~ x + 0)
#>
#> Coefficients:
#> x
#> 4.3
```

Мы настоятельно рекомендуем вам проверить это допущение, прежде чем продолжить. Выполните регрессию со свободным членом, а затем посмотрите, может ли он действительно быть нулевым. Проверьте доверительный интервал свободного члена. В этом примере доверительный интервал равен (6,26, 8,84):

```
confint(lm(y ~ x))
#>              2.5 %   97.5 %
#> (Intercept)  6.26     8.84
#> x            2.82     5.31
```

Поскольку доверительный интервал не содержит ноль, статистически не правдоподобно, чтобы свободный член уравнения мог быть равен нулю. Таким образом, в этом случае нецелесообразно повторно запускать регрессию при принудительном использовании свободного члена с нулевым значением.

11.6. Регрессия только тех переменных, которые сильно коррелируют с вашей зависимой переменной

Задача

У вас есть таблица данных со множеством переменных, и вы хотите построить множественную линейную регрессию, используя только те переменные, которые сильно коррелируют с вашей зависимой переменной.

Решение

Если `df` – это наша таблица данных, в которой содержатся наша зависимая и все независимые переменные, а `dep_var` – наша зависимая переменная, мы можем определить наши наиболее подходящие независимые переменные и затем использовать их в линейной регрессии. Если нам нужны четыре верхние независимые переменные, можно использовать это:

```
best_pred <- df %>%
  select(-dep_var) %>%
  map_dbl(cor, y = df$dep_var) %>%
  sort(decreasing = TRUE) %>%
  .[1:4] %>%
  names %>%
  df[.]

mod <- lm(df$dep_var ~ as.matrix(best_pred))
```

Данный рецепт представляет собой сочетание множества различных частей логики, используемых в других местах этой книги. Здесь мы опишем каждый шаг, а затем пройдемся по ним в разделе «Обсуждение», используя примеры данных.

Сначала мы удаляем зависимую переменную из нашей конвейерной цепочки, чтобы в нашем потоке данных были только наши независимые переменные:

```
df %>%
  select(-dep_var)
```

Затем мы используем функцию `map_dbl` из пакета `purrr`, чтобы выполнить попарную корреляцию для каждого столбца относительно зависимой переменной:

```
map_dbl(cor, y = df$dep_var) %>%
```

Затем мы берем полученные корреляции и сортируем их в порядке убывания:

```
sort(decreasing = TRUE) %>%
```

Нам нужны только четыре верхние коррелированные переменные, поэтому мы выбираем четыре верхние записи в получившемся векторе:

```
.[1:4] %>%
```

И нам не нужны значения корреляции, а лишь имена строк, которые представляют собой имена переменных из нашей исходной таблицы данных, `df`:

```
names %>%
```

Затем мы можем передать эти имена в квадратные скобки, чтобы выбрать только те столбцы, имена которых соответствуют тем, которые нам нужны:

```
df[.]
```

Наша цепочка конвейеров присваивает получившуюся таблицу данных в `best_pred`. Затем мы можем использовать `best_pred` в качестве независимых переменных в нашей регрессии и `df$dep_var` в качестве зависимой переменной:

```
mod <- lm(df$dep_var ~ as.matrix(best_pred))
```

Обсуждение

Комбинируя функции отображения, которые мы обсуждали в рецепте 6.4, мы можем создать рецепт удаления переменных с низкой корреляцией из набора независимых переменных и использовать независимые переменные с высокой корреляцией в регрессии.

У нас есть пример таблицы данных, которая содержит шесть независимых переменных с именами от `pred1` до `pred6`. Зависимая переменная носит имя `resp`. Давайте проведем эту таблицу данных через нашу логику и посмотрим, как она работает.

Загрузить данные и исключить переменную `resp` довольно просто, поэтому давайте посмотрим на результат отображения функции `cor`:

```
# loads the pred data frame
load("./data/pred.rdata")

pred %>%
  select(-resp) %>%
  map_dbl(cor, y = pred$resp)
#> pred1 pred2 pred3 pred4 pred5 pred6
#> 0.573 0.279 0.753 0.799 0.322 0.607
```

Вывод представляет собой именованный вектор значений, где имена – это имена переменных, а значения – попарные корреляции между всеми независимыми переменными и `resp`, зависимой переменной.

Если мы отсортируем этот вектор, то получим корреляции в порядке убывания:

```
pred %>%
  select(-resp) %>%
  map_dbl(cor, y = pred$resp) %>%
  sort(decreasing = TRUE)
```

```
#> pred4 pred3 pred6 pred1 pred5 pred2
#> 0.799 0.753 0.607 0.573 0.322 0.279
```

С помощью процедуры выбора подвектора выбираем первые четыре записи. Точка (.) в начале — это тоже специальный оператор, который сообщает конвейеру, куда поместить результат предыдущего шага:

```
pred %>%
 select(-resp) %>%
 map_dbl(cor, y = pred$resp) %>%
 sort(decreasing = TRUE) %>%
 .[1:4]
#> pred4 pred3 pred6 pred1
#> 0.799 0.753 0.607 0.573
```

Затем мы используем функцию `names` для извлечения имен из нашего вектора. Это имена столбцов, которые мы в конечном итоге хотим использовать в качестве наших независимых переменных:

```
pred %>%
 select(-resp) %>%
 map_dbl(cor, y = pred$resp) %>%
 sort(decreasing = TRUE) %>%
 .[1:4] %>%
 names
#> [1] "pred4" "pred3" "pred6" "pred1"
```

Когда мы передаем вектор имен в `pred[.]`, имена используются для выбора столбцов из таблицы данных `pred`. Затем мы используем `head`, чтобы выбрать только первые шесть строк, чтобы вам было проще:

```
pred %>%
 select(-resp) %>%
 map_dbl(cor, y = pred$resp) %>%
 sort(decreasing = TRUE) %>%
 .[1:4] %>%
 names %>%
 pred[.] %>%
 head
#> pred4 pred3 pred6 pred1
#> 1 7.252 1.5127 0.560 0.206
#> 2 2.076 0.2579 -0.124 -0.361
#> 3 -0.649 0.0884 0.657 0.758
#> 4 1.365 -0.1209 0.122 -0.727
#> 5 -5.444 -1.1943 -0.391 -1.368
#> 6 2.554 0.6120 1.273 0.433
```

Теперь давайте соберем все вместе и передадим полученные данные в регрессию:

```
best_pred <- pred %>%
 select(-resp) %>%
 map_dbl(cor, y = pred$resp) %>%
 sort(decreasing = TRUE) %>%
 .[1:4] %>%
 names %>%
 pred[.]
```

```
mod <- lm(pred$resp ~ as.matrix(best_pred))
summary(mod)
#>
#> Call:
#> lm(formula = pred$resp ~ as.matrix(best_pred))
#>
#> Residuals:
#> Min 1Q Median 3Q Max
#> -1.485 -0.619 0.189 0.562 1.398
#>
#> Coefficients:
#>                       Estimate Std. Error t  value    Pr(>|t|)
#> (Intercept)                1.117    0.340    3.28     0.0051   **
#> as.matrix(best_pred)pred4  0.523    0.207    2.53     0.0231   *
#> as.matrix(best_pred)pred3  -0.693   0.870    -0.80    0.4382
#> as.matrix(best_pred)pred6  1.160    0.682    1.70     0.1095
#> as.matrix(best_pred)pred1  0.343    0.359    0.95     0.3549
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.927 on 15 degrees of freedom
#> Multiple R-squared: 0.838, Adjusted R-squared: 0.795
#> F-statistic: 19.4 on 4 and 15 DF, p-value: 8.59e-06
```

11.7. Линейная регрессия с эффектами взаимодействия

Задача

Вы хотите включить в свою регрессию эффект взаимодействия.

Решение

Синтаксис формул регрессии в языке R позволяет задавать эффекты взаимодействия. Чтобы указать взаимодействие двух переменных, `u` и `v`, мы разделяем их имена звездочкой (*):

```
lm(y ~ u * v)
```

Это соответствует модели $y_i = \beta_0 + \beta_1 u_i + \beta_2 v_i + \beta_3 u_i v_i + \varepsilon_i$, которая включает в себя эффект взаимодействия первого порядка $\beta_3 u_i vi$.

Обсуждение

В регрессии взаимодействие происходит, когда произведение двух независимых переменных также является значимой независимой переменной (т. е. в дополнение к самим независимым переменным).

Предположим, у нас есть две независимые переменные, u и v, и мы хотим включить их взаимодействие в регрессию. Это выражается следующим уравнением:

$$y_i = \beta_0 + \beta_1 u_i + \beta_2 v_i + \beta_3 u_i v_i + \varepsilon_i.$$

Здесь член произведения $\beta_3 u_i v_i$ называется *эффектом взаимодействия*. Формула этого уравнения:

```
y ~ u * v
```

Когда вы пишете `y ~ u * v`, R автоматически включает `u`, `v` и их произведение в модель. На то есть причина. Если модель включает в себя эффект взаимодействия, такой как $\beta_3 u_i v_i$, теория регрессии говорит нам, что модель также должна содержать составляющие переменные u_i и v_i.

Аналогично, если у вас есть три независимые переменные (`u`, `v` и `w`) и вы хотите включить все их взаимодействия, разделите их звездочками:

```
y ~ u * v * w
```

Это соответствует уравнению регрессии:

$$y_i = \beta_0 + \beta_1 u_i + \beta_2 v_i + \beta_3 w_i + \beta_4 u_i v_i + \beta_5 u_i w_i + \beta_6 v_i w_i + \beta_7 u_i v_i w_i + \varepsilon_i.$$

Теперь у нас есть все взаимодействия первого порядка и взаимодействие второго порядка ($\beta_7 u_i v_i w_i$).

Однако иногда вам могут быть не нужны всевозможные взаимодействия. Вы можете явно указать одно произведение, используя оператор двоеточия (:). Например, `u: v: w` обозначает член произведения $\beta u_i v_i w_i$, но безо всяких взаимодействий. Таким образом, формула

```
y ~ u + v + w + u:v:w
```

соответствует уравнению регрессии:

$$y_i = \beta_0 + \beta_1 u_i + \beta_2 v_i + \beta_3 w_i + \beta_4 u_i v_i w_i + \varepsilon_i.$$

Может показаться странным, что двоеточие (:) означает чистое умножение, в то время как звездочка (*) означает и умножение, и включение составляющих эффектов. Опять же, это связано с тем, что мы обычно включаем их, когда включаем их взаимодействие, поэтому использовать такой подход по умолчанию для * имеет смысл.

Существует дополнительный синтаксис для простого указания множества взаимодействий:

```
(u + v + ... + w) ^ 2
```

Включить все переменные (`u`, `v`,..., `w`) и все их взаимодействия первого порядка.

```
(u + v + ... + w) ^ 3
```

Включить все переменные, все их взаимодействия первого порядка и второго порядка.

```
(u + v + ... + w) ^ 4
```

И так далее.

Звездочка (*) и двоеточие (:) следуют «закону распределения», поэтому допускаются также следующие обозначения:

```
x *(u + v + ... + w)
```

То же, что и `x*u + x*v + ... + x*w` (то же самое, что `x + u + v + ... + w + x:u + x:v + ... + x:w`).

`x:(u + v + ... + w)`

То же, что и `x:u + x:v + ... + x:w`.

Весь этот синтаксис дает вам некоторую гибкость при написании вашей формулы. Например, эти три формулы эквивалентны:

```
y ~ u * v
y ~ u + v + u:v
y ~ (u + v) ^ 2
```

Все они определяют одно и то же уравнение регрессии: $y_i = \beta_0 + \beta_1 u_i + \beta_2 v_i + \beta_3 u_i v_i + \varepsilon_i$.

См. также

Полный синтаксис для формул более богат, чем описано здесь. См. книгу *R in a Nutshell* или посетите страницу по адресу https://cran.r-project.org/doc/manuals/R-lang.pdf для получения более подробной информации.

11.8. Выбор наиболее подходящих переменных регрессии

Задача

Вы создаете новую регрессионную модель или улучшаете существующую. Вы наслаждаетесь роскошью большого числа переменных регрессии и хотите выбрать наиболее подходящее подмножество этих переменных.

Решение

Функция `step` может выполнять прямую и обратную пошаговую регрессию.

Обратная пошаговая регрессия начинается со множества переменных и устраняет неподходящие:

```
full.model <- lm(y ~ x1 + x2 + x3 + x4)
reduced.model <- step(full.model, direction = "backward")
```

Прямая пошаговая регрессия начинается с нескольких переменных и добавляет новые для улучшения модели до тех пор, пока ее нельзя будет улучшать и далее:

```
min.model <- lm(y ~ 1)
fwd.model <-
  step(min.model,
       direction = "forward",
       scope = (~ x1 + x2 + x3 + x4))
```

Обсуждение

Когда у вас много независимых переменных, выбрать наиболее подходящее подмножество может быть довольно трудно. Добавление и удаление отдельных переменных влияет на общее сочетание, поэтому поиск «лучших» может стать утомительным.

Функция `step` автоматизирует этот поиск. Обратная пошаговая регрессия – самый простой подход. Начните с модели, которая включает в себя все независи-

мые переменные. Мы называем ее *полной моделью*. Краткое описание модели, приведенное ниже, показывает, что не все независимые переменные являются статистически значимыми:

```
# Пример данных.
set.seed(4)
n <- 150
x1 <- rnorm(n)
x2 <- rnorm(n, 1, 2)
x3 <- rnorm(n, 3, 1)
x4 <- rnorm(n,-2, 2)
e <- rnorm(n, 0, 3)
y <- 4 + x1 + 5 * x3 + e

# Создаем модель.
full.model <- lm(y ~ x1 + x2 + x3 + x4)
summary(full.model)
#>
#> Call:
#> lm(formula = y ~ x1 + x2 + x3 + x4)
#>
#> Residuals:
#>    Min     1Q Median  3Q    Max
#> -8.032 -1.774 0.158 2.032 6.626
#>
#> Coefficients:
#>          Estimate Std.  Error t value Pr(>|t|)
#> (Intercept)  3.40224    0.80767 4.21   4.4e-05 ***
#> x1           0.53937    0.25935 2.08   0.039    *
#> x2           0.16831    0.12291 1.37   0.173
#> x3           5.17410    0.23983 21.57 < 2e-16 ***
#> x4           -0.00982   0.12954 -0.08 0.940
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 2.92 on 145 degrees of freedom
#> Multiple R-squared: 0.77, Adjusted R-squared: 0.763
#> F-statistic: 121 on 4 and 145 DF, p-value: <2e-16
```

Мы хотим исключить незначимые переменные, поэтому используем функцию `step`. В результате получаем модель, которая носит название *уменьшенная модель*:

```
reduced.model <- step(full.model, direction="backward")
#> Start: AIC=327
#> y ~ x1 + x2 + x3 + x4
#>
#>      Df Sum of Sq   RSS   AIC
#> - x4 1         0   1240  325
#> - x2 1         16  1256  327
#> <none>             1240  327
#> - x1 1         37  1277  329
#> - x3 1       3979  5219  40
#>
#> Step: AIC=325
#> y ~ x1 + x2 + x3
#>
```

```
#>      Df Sum of Sq   RSS   AIC
#> - x2 1         16  1256  325
#> <none>             1240  325
#> - x1 1         37  1277  327
#> - x3 1       3988  5228  539
#>
#> Step: AIC=325
#> y ~ x1 + x3
#>
#>      Df Sum of Sq   RSS   AIC
#> <none> 1256 325
#> - x1 1         44  1300  328
#> - x3 1       3974  230   537
```

Вывод функции показывает последовательность моделей, которые она исследовала. В этом случае мы удалили x2 и x4 и оставили только x1 и x3 в окончательной (уменьшенной) модели. Ниже видно, что уменьшенная модель содержит лишь значимые независимые переменные:

```
summary(reduced.model)
#>
#> Call:
#> lm(formula = y ~ x1 + x3)
#>
#> Residuals:
#>    Min     1Q  Median     3Q  Max
#> -8.148 -1.850  -0.055  2.026 6.550
#>
#> Coefficients:
#>             Estimate Std. Error t value  Pr(>|t|)
#> (Intercept)  3.648      0.751    4.86     3e-06 ***
#> x1           0.582      0.255    2.28     0.024 *
#> x3           5.147      0.239    21.57    <2e-16 ***
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 2.92 on 147 degrees of freedom
#> Multiple R-squared: 0.767, Adjusted R-squared: 0.763
#> F-statistic: 241 on 2 and 147 DF, p-value: <2e-16
```

Обратная пошаговая регрессия проста, но иногда просто невозможно начать со «всего», потому что у вас слишком много переменных-кандидатов. В этом случае используйте прямую пошаговую регрессию, которая будет начинаться с нуля и постепенно добавлять переменные, чтобы улучшить регрессию. Когда дальнейшее улучшение уже невозможно, она останавливается.

Модель, которая «начинается с нуля», на первый взгляд может показаться странной:

```
min.model <- lm(y ~ 1)
```

Это модель с зависимой переменной (*y*), но у нее нет независимых переменных. (Все подогнанные значения для *y* – это просто ее среднее значение, о чем можно было бы догадаться, если бы не было доступных независимых переменных.)

Мы должны сообщить функции `step`, какие переменные-кандидаты доступны для включения в модель. Это задача аргумента `scope`. `scope` – это формула по левую сторону от тильды (~), а в правой части находятся переменные-кандидаты:

```
fwd.model <- step(
 min.model,
 direction = "forward",
 scope = (~ x1 + x2 + x3 + x4),
 trace = 0
)
```

Здесь видно, что x1, x2, x3 и x4 являются кандидатами на включение. (Мы также включили строку `trace = 0`, чтобы запретить объемный вывод из функции.) Получившаяся модель имеет две значимые независимые переменные, а незначимых переменных у нее нет:

```
summary(fwd.model)
#>
#> Call:
#> lm(formula = y ~ x3 + x1)
#>
#> Residuals:
#> Min 1Q Median 3Q Max
#> -8.148 -1.850 -0.055 2.026 6.550
#>
#> Coefficients:
#>             Estimate Std.  Error  t value  Pr(>|t|)
#> (Intercept)  3.648        0.751    4.86   3e-06 ***
#> x3           5.147        0.239    21.57 <2e-16 ***
#> x1           0.582        0.255    2.28   0.024 *
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 2.92 on 147 degrees of freedom
#> Multiple R-squared: 0.767, Adjusted R-squared: 0.763
#> F-statistic: 241 on 2 and 147 DF, p-value: <2e-16
```

Используя алгоритм прямой регрессии, мы получили ту же модель, что и в случае с обратной регрессией, включив x1 и x3 и исключив x2 и x4. Этот пример ненастоящий, поэтому это не удивительно. В реальных приложениях мы предлагаем опробовать как прямую, так и обратную регрессию, а затем сравнить результаты. Возможно, вы будете удивлены.

И наконец, не увлекайтесь пошаговой регрессией. Это не панацея, она не может превратить мусор в золото и, безусловно, не заменит тщательного и мудрого выбора независимых переменных. Вы, наверное, подумали: «О, боже! Я могу сгенерировать все возможные эффекты взаимодействия для своей модели, а затем пусть функция `step` выберет лучшие из них! Ну и модель у меня получится!» Вы будете думать о чем-то вроде того, чтобы начать со всех возможных взаимодействий, а затем попытаться уменьшить модель:

```
full.model <- lm(y ~ (x1 + x2 + x3 + x4) ^ 4)
reduced.model <- step(full.model, direction = "backward")
#> Start: AIC=337
#> y ~ (x1 + x2 + x3 + x4)^4
#>
#>               Df Sum of Sq RSS  AIC
#> - x1:x2:x3:x4 1 0.0321    1145 335
#> <none>                    1145 337
```

```
#>
#> Step: AIC=335
#>  y ~ x1 + x2 + x3 + x4 + x1:x2 + x1:x3 + x1:x4 + x2:x3 + x2:x4 +
#>    x3:x4 + x1:x2:x3 + x1:x2:x4 + x1:x3:x4 + x2:x3:x4
#>
#>            Df Sum of Sq   RSS AIC
#> - x2:x3:x4 1        0.76  1146 333
#> - x1:x3:x4 1        8.37  1154 334
#> <none>                    1145 335
#> - x1:x2:x4 1       20.95 1166 336
#> - x1:x2:x3 1       25.18 1170 336
#>
#> Step: AIC=333
#>  y ~ x1 + x2 + x3 + x4 + x1:x2 + x1:x3 + x1:x4 + x2:x3 + x2:x4 +
#>    x3:x4 + x1:x2:x3 + x1:x2:x4 + x1:x3:x4
#>
#>            Df Sum of Sq   RSS AIC
#> - x1:x3:x4 1        8.74  1155 332
#> <none>                    1146 333
#> - x1:x2:x4 1       21.72 1168 334
#> - x1:x2:x3 1       26.51 1172 334
#>
#> Step: AIC=332
#>  y ~ x1 + x2 + x3 + x4 + x1:x2 + x1:x3 + x1:x4 + x2:x3 + x2:x4 +
#>      x3:x4 + x1:x2:x3 + x1:x2:x4
#>
#>            Df Sum of Sq   RSS AIC
#> - x3:x4 1           0.29  1155 330
#> <none>                    1155 332
#> - x1:x2:x4 1       23.24 1178 333
#> - x1:x2:x3 1       31.11 1186 334
#>
#> Step: AIC=330
#>  y ~ x1 + x2 + x3 + x4 + x1:x2 + x1:x3 + x1:x4 + x2:x3 + x2:x4 +
#>      x1:x2:x3 + x1:x2:x4
#>
#>            Df Sum of Sq   RSS AIC
#> <none>                    1155 330
#> - x1:x2:x4 1       23.4   1178 331
#> - x1:x2:x3 1       31.5   1187 332
```

Такой вариант не подходит. Большинство эффектов взаимодействия не имеют смысла. Функция `step` будет перегружена, и у вас останется большое количество незначимых эффектов.

См. также

См. рецепт 11.25.

11.9. Регрессия для подмножества данных

Задача

Вы хотите подогнать линейную модель к подмножеству своих данных, а не ко всему набору данных.

Решение

У функции lm имеется параметр subset, который указывает, какие элементы данных следует использовать для подгонки. Значением параметра может быть любое индексное выражение, которое может индексировать ваши данные. Ниже приводится пример подгонки, где используются только первые 100 наблюдений:

```
lm(y ~ x1, subset=1:100) # Используем только x[1:100].
```

Обсуждение

У вас часто будет возникать необходимость выполнить регрессию только для подмножества своих данных. Это может произойти, например, когда вы используете данные в выборке для создания модели и данные вне выборки для тестирования.

У функции lm есть параметр subset, который выбирает наблюдения, используемые для подгонки.

Значение subset – это вектор. Это может быть вектор значений индекса, в этом случае lm выбирает только из ваших данных указанные наблюдения. Это также может быть логический вектор той же длины, что и ваши данные, и в этом случае lm выбирает наблюдения с соответствующим значением TRUE.

Предположим, у вас есть 1000 наблюдений пар (*x, y*), и вы хотите выполнить подгонку своей модели, используя только первую половину этих наблюдений. Используйте параметр subset 1: 500, который указывает на то, что функция lm должна использовать наблюдения от 1 до 500:

```
## Пример данных.
n <- 1000
x <- rnorm(n)
e <- rnorm(n, 0, .5)
y <- 3 + 2 * x + e
lm(y ~ x, subset = 1:500)
#>
#> Call:
#> lm(formula = y ~ x, subset = 1:500)
#>
#> Coefficients:
#> (Intercept)    x
#>           3   2
```

В целом можно использовать выражение 1:floor(length(x)/2), чтобы выбрать первую половину своих данных независимо от размера:

```
lm(y ~ x, subset = 1:floor(length(x) / 2))
#>
#> Call:
#> lm(formula = y ~ x, subset = 1:floor(length(x)/2))
#>
#> Coefficients:
#> (Intercept)    x
#>           3   2
```

Допустим, ваши данные были собраны в нескольких лабораториях, и у вас есть фактор, который определяет лабораторию происхождения. Вы можете ограничить регрессию наблюдениями, собранными в Нью-Джерси, используя логический вектор, который ИСТИНЕН только для этих наблюдений:

```
load('./data/lab_df.rdata')
lm(y ~ x, subset = (lab == "NJ"), data = lab_df)
#>
#> Call:
#> lm(formula = y ~ x, data = lab_df, subset = (lab == "NJ"))
#>
#> Coefficients:
#> (Intercept)       x
#>        2.58    5.03
```

11.10. Использование выражения в формуле регрессии

Задача

Вы хотите выполнить регрессию для расчетных значений, а не простых переменных, но синтаксис формулы регрессии, похоже, запрещает это.

Решение

Поместите выражения для расчетных значений внутри оператора `I(...)`. Это заставит R вычислить выражение и использовать расчетное значение для регрессии.

Обсуждение

Если вы хотите работать с суммой *u* и *v*, то ваше уравнение регрессии будет выглядеть так:

$$y_i = \beta_0 + \beta_1(u_i + v_i) + \varepsilon_i.$$

Как написать это уравнение в виде формулы регрессии? Этот вариант не сработает:

```
lm(y ~ u + v)        # Не совсем верно.
```

Здесь R будет интерпретировать `u` и `v` как две отдельные независимые переменные, у каждой из которых свой коэффициент регрессии. Аналогично, предположим, что ваше уравнение регрессии выглядит так:

$$y_i = \beta_0 + \beta_1 u_i + \beta_2 u_i^2 + \varepsilon_i.$$

Этот вариант не сработает:

```
lm(y ~ u + u ^ 2)      # Это взаимодействие, а не квадратичный член.
```

R будет интерпретировать `u^2` как эффект взаимодействия (*см.* рецепт 11.7), а не как квадрат `u`.

Решение состоит в том, чтобы окружить выражения оператором `I(...)`, который запрещает интерпретировать выражение как формулу регрессии. Вместо этого он заставляет R вычислить значение выражения, а затем включить это значение непосредственно в регрессию.

Таким образом, первый пример превращается в:

```
lm(y ~ I(u + v))
```

В ответ на эту команду R вычисляет `u + v`, а затем выполняет регрессию.

Для второго примера мы используем это:

```
lm(y ~ u + I(u ^ 2))
```

Здесь R вычисляет квадрат `u`, а потом выполняет регрессию.

Все основные двоичные операторы (+, -, *, /, ^) имеют специальные значения внутри формулы регрессии. По этой причине вы должны использовать оператор `I(...)` всякий раз, когда включаете расчетные значения в регрессию.

Прекрасным аспектом этих встроенных преобразований является тот факт, что R запоминает их и применяет, когда вы делаете прогнозы из модели. Рассмотрим квадратичную модель, описанную во втором примере. В ней используются `u` и `u^2`, но мы поставляем значение `u`, а R делает всю тяжелую работу. Нам не нужно рассчитывать квадрат `u` самостоятельно:

```
load('./data/df_squared.rdata')
m <- lm(y ~ u + I(u ^ 2), data = df_squared)
predict(m, newdata = data.frame(u = 13.4))
#> 1
#> 877
```

См. также

См. рецепт 11.11, в котором приводится особый случай полиномиальной регрессии. *См.* рецепт 11.12, чтобы узнать, как включить в регрессию другие преобразования данных.

11.11. Полиномиальная регрессия

Задача

Вы хотите регрессировать *y* на полиноме *x*.

Решение

Используйте функцию `poly(x, n)` в формуле регрессии, чтобы регрессировать на полиноме *x* *n*-й степени. В этом примере мы моделируем *y* как кубическую функцию *x*:

```
lm(y ~ poly(x, 3, raw = TRUE))
```

Показанная в примере формула соответствует приведенному ниже кубическому уравнению регрессии:

$$y_i = \beta_0 + \beta_1 x_i + \beta_2 x_i^2 + \beta_3 x_i^3 + \varepsilon_i.$$

Обсуждение

Когда кто-то использует модель полиномиальной регрессии в R впервые, то часто делает нечто неуклюжее вроде этого:

```
x_sq <- x ^ 2
x_cub <- x ^ 3
m <- lm(y ~ x + x_sq + x_cub)
```

Очевидно, что это очень раздражает и засоряет рабочее пространство дополнительными переменными.

Гораздо проще написать:

```
m <- lm(y ~ poly(x, 3, raw = TRUE))
```

Фрагмент `raw = TRUE` необходим. Без этого функция `poly` будет вычислять ортогональные полиномы вместо простых.

Помимо удобства, огромное преимущество состоит в том, что R будет вычислять все эти степени *x*, когда вы будете строить прогнозы с использованием данной модели (*см.* рецепт 11.19). Без этого вы застрянете, вычисляя x^2 и x^3 самостоятельно каждый раз, когда будете использовать модель.

А вот еще одна веская причина использовать функцию `poly`. Нельзя писать свою формулу регрессии так:

```
lm(y ~ x + x^2 + x^3) # Она делает не то, что вы думаете!
```

R будет интерпретировать `x^2` и `x^3` как эффекты взаимодействия, а не как степени *x*. Получившаяся модель представляет собой линейную регрессию с одним членом, совершенно не соответствующую вашим ожиданиям. Можно написать формулу регрессии следующим образом:

```
lm(y ~ x + I(x ^ 2) + I(x ^ 3))
```

Но она становится довольно длинной. Просто используйте функцию `poly`.

См. также

См. рецепт 11.7 для получения дополнительной информации об эффектах взаимодействия. *См.* рецепт 11.12, где рассказывается о том, как включить в регрессию другие преобразования данных.

11.12. Регрессия на преобразованных данных

Задача

Вы хотите построить регрессионную модель для *x* и *y*, но у них нет линейной зависимости.

Решение

Можно встроить необходимое преобразование в формулу регрессии. Например, если нужно преобразовать *y* в log (*y*), формула регрессии принимает следующий вид:

```
lm(log(y) ~ x)
```

Обсуждение

Важное предположение, лежащее в основе функции `lm`, состоит в том, что переменные имеют линейную зависимость. В случае если это предположение неверно, регрессия становится бессмысленной.

К счастью, многие наборы данных можно преобразовать в линейную зависимость, перед тем как применять функцию `lm`.

На рис. 11-1 показан пример затухания по экспоненте. На панели слева видны исходные данные, *z*. Пунктирная линия показывает линейную регрессию на исходных данных; ясно, что такое соответствие данным никуда не годится.

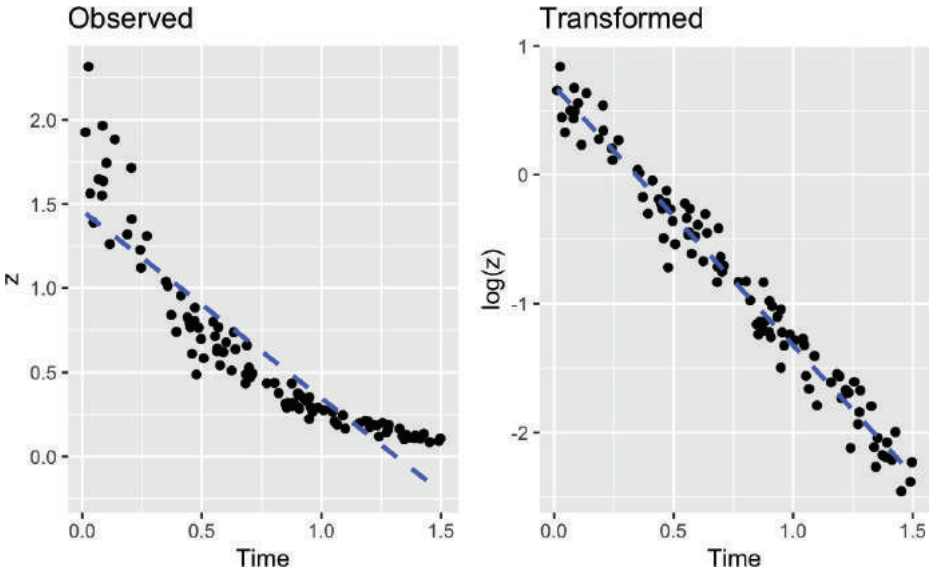


Рис. 11-1. Пример преобразования данных

Если данные действительно экспоненциальные, то возможная модель будет выглядеть так:

$$z = \exp[\beta_0 + \beta_1 t + \varepsilon],$$

где t – это время, а exp [] – экспоненциальная функция (e^x). Конечно, это не модель линейной регрессии, но мы можем линеаризовать это, взяв логарифмы:

$$\log(z) = \beta_0 + \beta_1 t + \varepsilon.$$

В R такая регрессия является простой, потому что мы можем встроить логарифм переменной непосредственно в формулу регрессии:

```
# read in our example data
load(file = './data/df_decay.rdata')
z <- df_decay$z
t <- df_decay$time

# transform and model
m <- lm(log(z) ~ t)
summary(m)
#>
#> Call:
#> lm(formula = log(z) ~ t)
#>
#> Residuals:
#>  Min       1Q   Median     3Q    Max
#> -0.4479 -0.0993 0.0049  0.0978  0.2802
```

```
#>
#> Coefficients:
#>             Estimate Std. Error t value  Pr(>|t|)
#> (Intercept)  0.6887     0.0306    22.5  <2e-16  ***
#> t           -2.0118     0.0351   -57.3 <2e-16  ***
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.148 on 98 degrees of freedom
#> Multiple R-squared: 0.971, Adjusted R-squared: 0.971
#> F-statistic: 3.28e+03 on 1 and 98 DF, p-value: <2e-16
```

На правой панели рис. 11-1 показан график зависимости log (*z*) от времени. Здесь же наложена линия регрессии. Соответствие данным, кажется, выглядит намного лучше; это подтверждается коэффициентом детерминации R^2 = 0,97 по сравнению с показателем 0,82 для линейной регрессии на исходных данных.

Вы можете встроить и другие функции в свою формулу. Если вы решили, что зависимость квадратичная, можно использовать преобразование квадратного корня:

```
lm(sqrt(y) ~ month)
```

Конечно, можно применять преобразования к переменным с обеих сторон формулы.

С помощью этой формулы мы строим регрессию y на квадратном корне x:

```
lm(y ~ sqrt(x))
```

Эта формула показывает двойную логарифмическую связь между x и y:

```
lm(log(y) ~ log(x))
```

См. также

См. рецепт 11.13.

11.13. Поиск наиболее подходящего степенного преобразования (тест Бокса–Кокса)

Задача

Вы хотите улучшить свою линейную модель, применяя степенное преобразование к зависимой переменной.

Решение

Используйте тест Бокса–Кокса, который реализуется функцией `boxcox` из пакета `MASS`. Тест определит степень, λ, таким образом, что преобразование y в y^{λ} улучшит соответствие данным вашей модели:

```
library(MASS)
m <- lm(y ~ x)
boxcox(m)
```

Обсуждение

Чтобы проиллюстрировать преобразование Бокса–Кокса, давайте создадим искусственные данные, используя уравнение $y^{-1.5} = x + \varepsilon$, где ε – остаточный член:

```
set.seed(9)
x <- 10:100
eps <- rnorm(length(x), sd = 5)
y <- (x + eps) ^ (-1 / 1.5)
```

Затем мы (по ошибке) смоделируем данные, применяя простую линейную регрессию, и получим скорректированное значение коэффициента детерминации R^2, равное 0,637:

```
m <- lm(y ~ x)
summary(m)
#>
#> Call:
#> lm(formula = y ~ x)
#>
#> Residuals:
#>      Min       1Q   Median      3Q     Max
#> -0.04032 -0.01633 -0.00792 0.00996 0.14516
#>
#> Coefficients:
#>             Estimate Std. Error t value     Pr(>|t|)
#> (Intercept)  0.166885   0.007078   23.6  <2e-16 * **
#> x -          0.001465   0.000116   -12.6 <2e-16   ***
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 0.0291 on 89 degrees of freedom
#> Multiple R-squared: 0.641, Adjusted R-squared: 0.637
#> F-statistic: 159 on 1 and 89 DF, p-value: <2e-16
```

При построении графика невязок и подогнанных значений мы получаем подсказку, что что-то не так. Можно получить диаграмму невязок `ggplot`, используя библиотеку `broom`. Функция `augment` из библиотеки `broom` поместит наши невязки (и другие вещи) в таблице данных, чтобы было проще. Затем мы можем использовать `ggplot` для построения графика:

```
library(broom)
augmented_m <- augment(m)

ggplot(augmented_m, aes(x = .fitted, y = .resid)) +
  geom_point()
```

Результат показан на рис. 11-2.

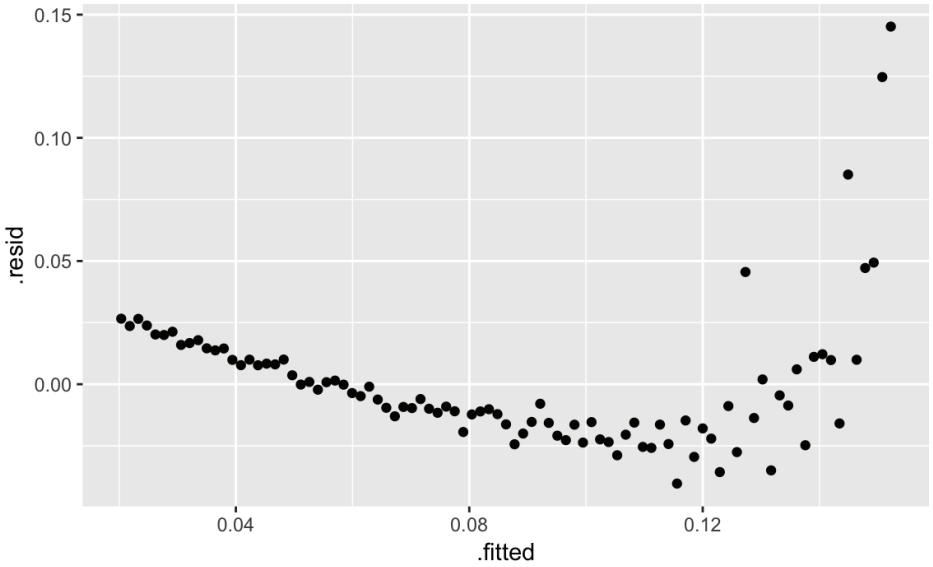


Рис. 11-2. Подогнанные значения и невязки

Если вам просто нужно быстро взглянуть на диаграмму невязок, не заботясь о том, чтобы получить результат в виде графика `ggplot`, можно использовать метод `plot` из базовой версии R для объекта модели, `m`:

```
plot(m, which = 1) # which = 1 plots only the fitted vs. Residuals
```

На рис. 11-2 видно, что этот график имеет четкую параболическую форму. Возможное исправление – степенное преобразование на y, поэтому мы запускаем тест Бокса–Кокса:

```
library(MASS)
#>
#> Attaching package: 'MASS'
#> The following object is masked from 'package:dplyr':
#>
#> select
bc <- boxcox(m)
```

Функция `boxcox` строит значения λ в зависимости от логарифмического правдоподобия полученной модели, как показано на рис. 11-3. Нам нужно максимизировать это логарифмическое правдоподобие, поэтому функция рисует линию с наилучшим значением, а также рисует линии в пределах своего доверительного интервала. В этом случае похоже, что наилучшее значение составляет около –1,5 с доверительным интервалом примерно (–1,75, –1,25).

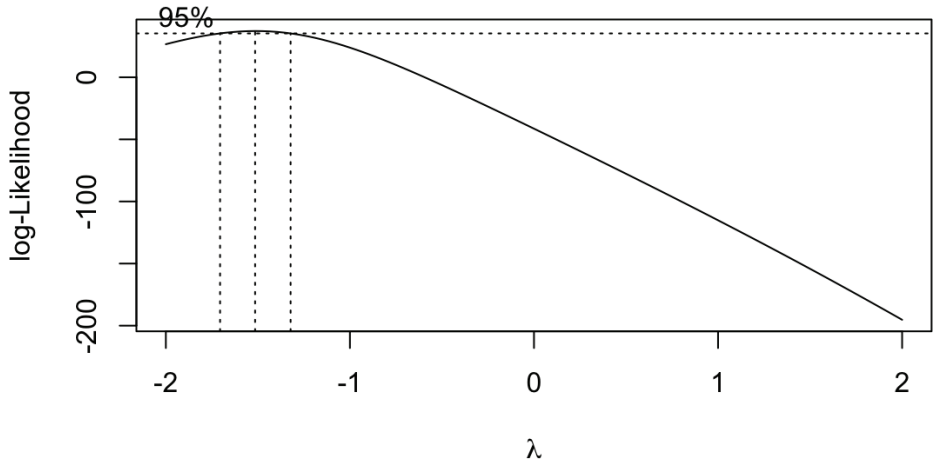


Рис. 11-3. Вывод функции boxcox для модели (m)

Как ни странно, функция boxcox не возвращает наилучшее значение λ. Скорее, она возвращает пары (x, y), отображаемые на графике. Найти значения λ, которые дают наибольшее логарифмическое правдоподобие, y, довольно легко. Мы используем функцию which.max:

```
which.max(bc$y)
#> [1] 13
```

Тогда это дает нам положение соответствующего значения λ:

```
lambda <- bc$x[which.max(bc$y)]
lambda
#> [1] -1.52
```

Функция вычисляет, что наилучшее значение λ – это 1,52. В реальном приложении мы настоятельно рекомендуем вам интерпретировать это число и выбирать степень, которая имеет для вас смысл, вместо того чтобы слепо принимать это «лучшее» значение. Используйте график, который поможет вам в этой интерпретации.

Здесь мы пойдем с –1,52.

Можно применить степенное преобразование к y, а затем подогнать пересмотренную модель; это дает гораздо лучший коэффициент детерминации R^2 в 0,967:

```
z <- y ^ lambda
m2 <- lm(z ~ x)
summary(m2)
#>
#> Call:
#> lm(formula = z ~ x)
#>
#> Residuals:
#>      Min       1Q   Median   3Q    Max
#>  -13.459 -3.711 -0.228  2.206 14.188
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
```

```
#> (Intercept) -0.6426     1.2517   -0.51    0.61
#> x            1.0514     0.0205   51.20    <2e-16 ***
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 5.15 on 89 degrees of freedom
#> Multiple R-squared: 0.967, Adjusted R-squared: 0.967
#> F-statistic: 2.62e+03 on 1 and 89 DF, p-value: <2e-16
```

Для тех, кто предпочитает однострочные сценарии, преобразование может быть встроено прямо в пересмотренную формулу регрессии:

```
m2 <- lm(I(y ^ lambda) ~ x)
```

По умолчанию функция `boxcox` ищет значения λ в диапазоне от −2 до +2. Вы можете изменить это с помощью аргумента `lambda`; *см.* страницу справки для получения дополнительной информации.

Мы предлагаем рассматривать результат теста Бокса–Кокса как отправную точку, а не как окончательный ответ. Если доверительный интервал для λ включает в себя 1,0, может оказаться, что никакое степенное преобразование на самом деле не помогает. Как всегда, проверьте невязки до и после преобразования. Они действительно улучшились?

Сравните рис. 11-4 (преобразованные данные) с рис. 11-2 (преобразование отсутствует).

```
augmented_m2 <- augment(m2)

ggplot(augmented_m2, aes(x = .fitted, y = .resid)) +
 geom_point()
```

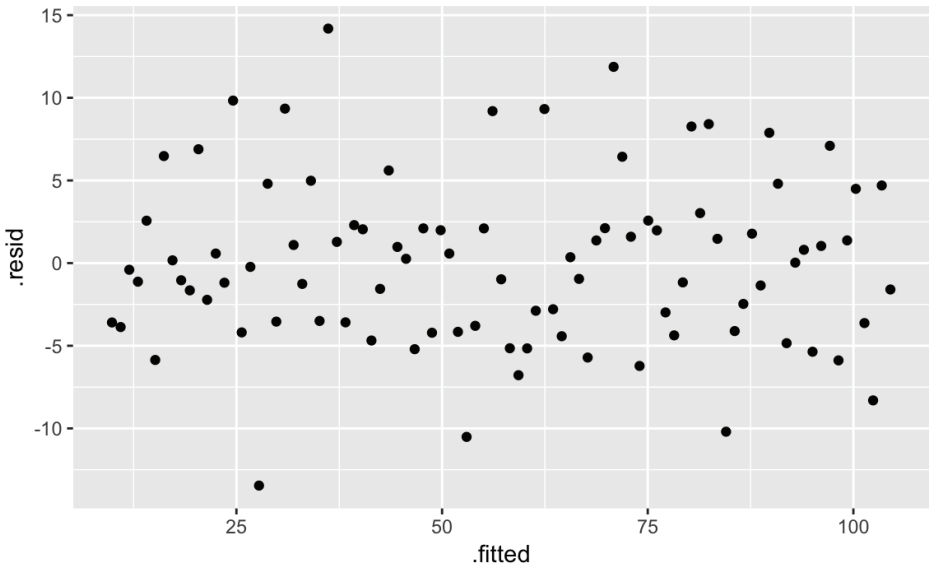


Рис. 11-4. Подогнанные значения и невязки: m2

См. также

См. рецепты 11.12 и 11.16.

11.14. Формирование доверительных интервалов для коэффициентов регрессии

Задача

Вы выполняете линейную регрессию, и вам нужны доверительные интервалы для коэффициентов регрессии.

Решение

Сохраните регрессионную модель в объекте; затем используйте функцию `confint` для извлечения доверительных интервалов:

```
load(file = './data/conf.rdata')
m <- lm(y ~ x1 + x2)
confint(m)
#>              2.5 %   97.5 %
#> (Intercept)  -3.90    6.47
#> x1           -2.58    6.24
#> x2            4.67    5.17
```

Обсуждение

В решении используется модель $y = \beta_0 + \beta_1 (x_{1)\,i} + \beta_2 (x_{2)\,i} + \varepsilon_i$. Функция `confint` возвращает доверительные интервалы для свободного члена (β_0), коэффициента x_1 (β_1) и коэффициента x_2 (β_2):

```
confint(m)
#>              2.5 %   97.5 %
#> (Intercept)  -3.90    6.47
#> x1           -2.58    6.24
#> x2            4.67    5.17
```

По умолчанию функция `confint` использует уровень доверия 95 %. Используйте параметр `level`, чтобы выбрать другой уровень:

```
confint(m, level = 0.99)
#>              0.5 %   99.5 %
#> (Intercept)  -5.72    8.28
#> x1            4.12    7.79
#> x2            4.58    5.26
```

См. также

Функция `coefplot` из пакета `arm` может строить доверительные интервалы для коэффициентов регрессии.

11.15. Построение невязок регрессии

Задача

Вам нужно визуальное отображение невязок регрессии.

Решение

Вы можете построить объект модели, используя `broom`, чтобы поместить результаты модели в таблицу данных, а затем построить график с помощью `ggplot`:

```
m <- lm(y ~ x1 + x2)

library(broom)
augmented_m <- augment(m)

ggplot(augmented_m, aes(x = .fitted, y = .resid)) +
 geom_point()
```

Обсуждение

Используя линейную модель `m` из предыдущего рецепта, можно создать простую диаграмму невязок:

```
library(broom)
augmented_m <- augment(m)

ggplot(augmented_m, aes(x = .fitted, y = .resid)) +
 geom_point()
```

Вывод показан на рис. 11-5.

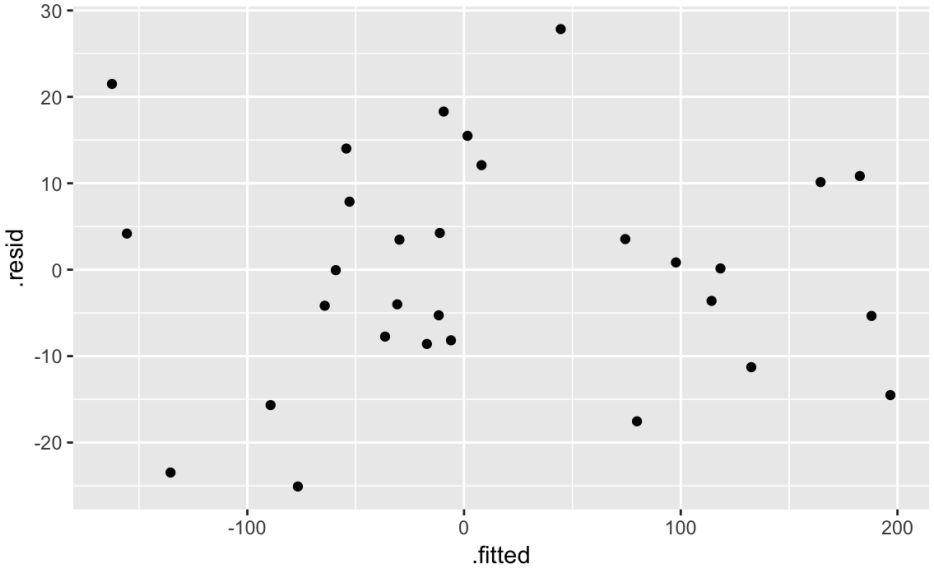


Рис. 11-5. Диаграмма невязок

Вы также можете использовать метод `plot` из базовой версии R для быстрого просмотра, но он будет давать графический вывод базовой версии вместо графика `ggplot`:

```
plot(m, which = 1)
```

См. также

См. рецепт 11.16, где приводятся примеры графиков невязок и других графиков диагностики.

11.16. Диагностика линейной регрессии

Задача

Вы построили линейную регрессию. Теперь вы хотите проверить качество модели, выполнив диагностические проверки.

Решение

Начните с построения объекта модели, который создаст несколько диагностических диаграмм с использованием базовой графической системы:

```
m <- lm(y ~ x1 + x2)
plot(m)
```

Затем определите возможные выбросы, посмотрев на диагностический график невязок или используя функцию `outlierTest` пакета `car`:

```
library(car)
outlierTest(m)
```

Наконец, выявите любые чрезмерно влиятельные наблюдения. *См.* рецепт 11.17.

Обсуждение

R создает впечатление, что линейная регрессия проста: просто используйте функцию `lm`. Однако подгонка данных – это только начало. Ваша задача – решить, действительно ли подогнанная модель работает и хорошо ли.

Прежде всего у вас должна быть статистически значимая модель. Проверьте F-статистику по сводке модели (рецепт 11.4) и убедитесь, что p-значение достаточно мало для ваших целей. Обычно оно должно быть меньше 0,05, иначе ваша модель, вероятно, будет не очень значимой.

Простое построение объекта модели дает несколько полезных диагностических графиков, показанных на рис. 11-6:

```
m <- lm(y ~ x1 + x2)
par(mfrow = (c(2, 2))) # Это дает нам график 2×2.
plot(m)
```

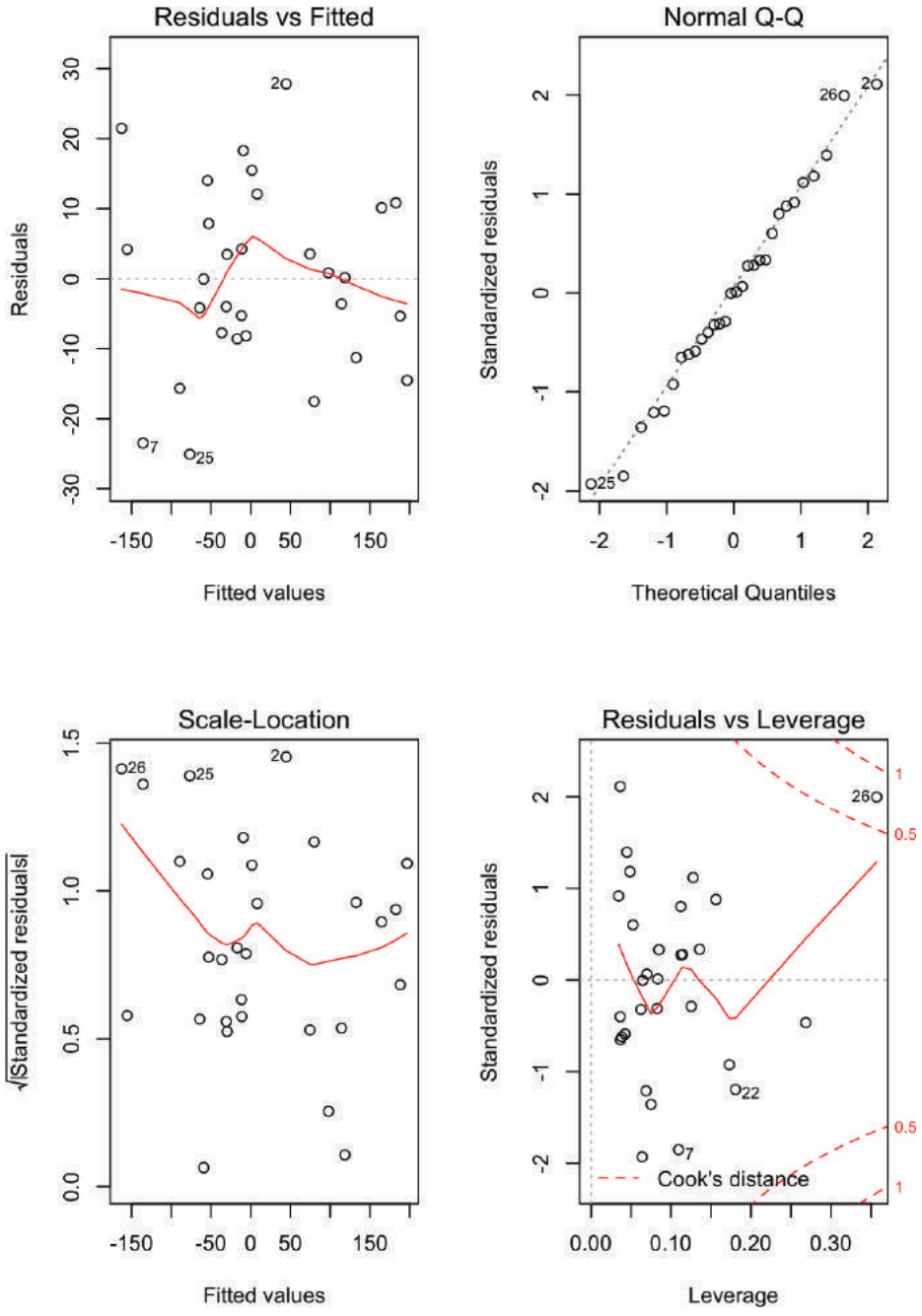


Рис. 11-6. Диагностические графики для хорошей регрессии

На рис. 11-6 показаны диагностические графики довольно хорошей регрессии:

- точки на графике «Невязки и подогнанные значения» разбросаны случайным образом без определенного шаблона;
- графическая проверка данных на нормальность (Normal Q-Q): здесь точки находятся более или менее на линии, а это указывает на то, что невязки соответствуют нормальному распределению;
- и на графике Scale – Location, и на графике зависимости невязок от «показателя напряженности» (Residuals vs Leverage) точки находятся в группе, не слишком далеко от центра.

Напротив, графики на рис. 11-7 показывают диагностику для не очень хорошей регрессии:

```
load(file = './data/bad.rdata')
m <- lm(y2 ~ x3 + x4)
par(mfrow = (c(2, 2)))   # Это дает нам диаграмму 2×2
plot(m)
```

Обратите внимание, что график Residuals vs Fitted имеет определенную параболическую форму. Это говорит нам о том, что эта модель неполная: отсутствует квадратичный фактор, который мог бы объяснить дополнительную модификацию *y*. Другие шаблоны в невязках могут указывать на дополнительные проблемы: например, форма конуса может указывать на непостоянную дисперсию *y*. Интерпретация этих структур – своего рода искусство, поэтому мы предлагаем просмотреть хорошую книгу по линейной регрессии при оценке графика невязок.

У этих графиков есть и другие проблемы. На графике Normal Q-Q больше точек вне линии, чем на графике для хорошей регрессии. На графиках Scale – Location и Residuals vs Leverage показаны точки, разбросанные от центра, а это говорит о том, что некоторые точки имеют чрезмерный показатель напряженности.

Еще один момент – это точка 28, выступающая на каждом графике. Это предупреждает нас, что в этом наблюдении есть что-то странное. Точка может быть выбросом, например. Можно проверить эту догадку с помощью функции `outlierTest` из пакета `car`:

```
library(car)
outlierTest(m)
#>     rstudent  unadjusted  p-value   Bonferonni p
#> 28      4.46             7.76e-05    0.0031
```

`outlierTest` определяет наиболее отклоняющееся наблюдение модели. В этом случае она идентифицировала наблюдение № 28 и таким образом подтвердила, что это может быть выброс.

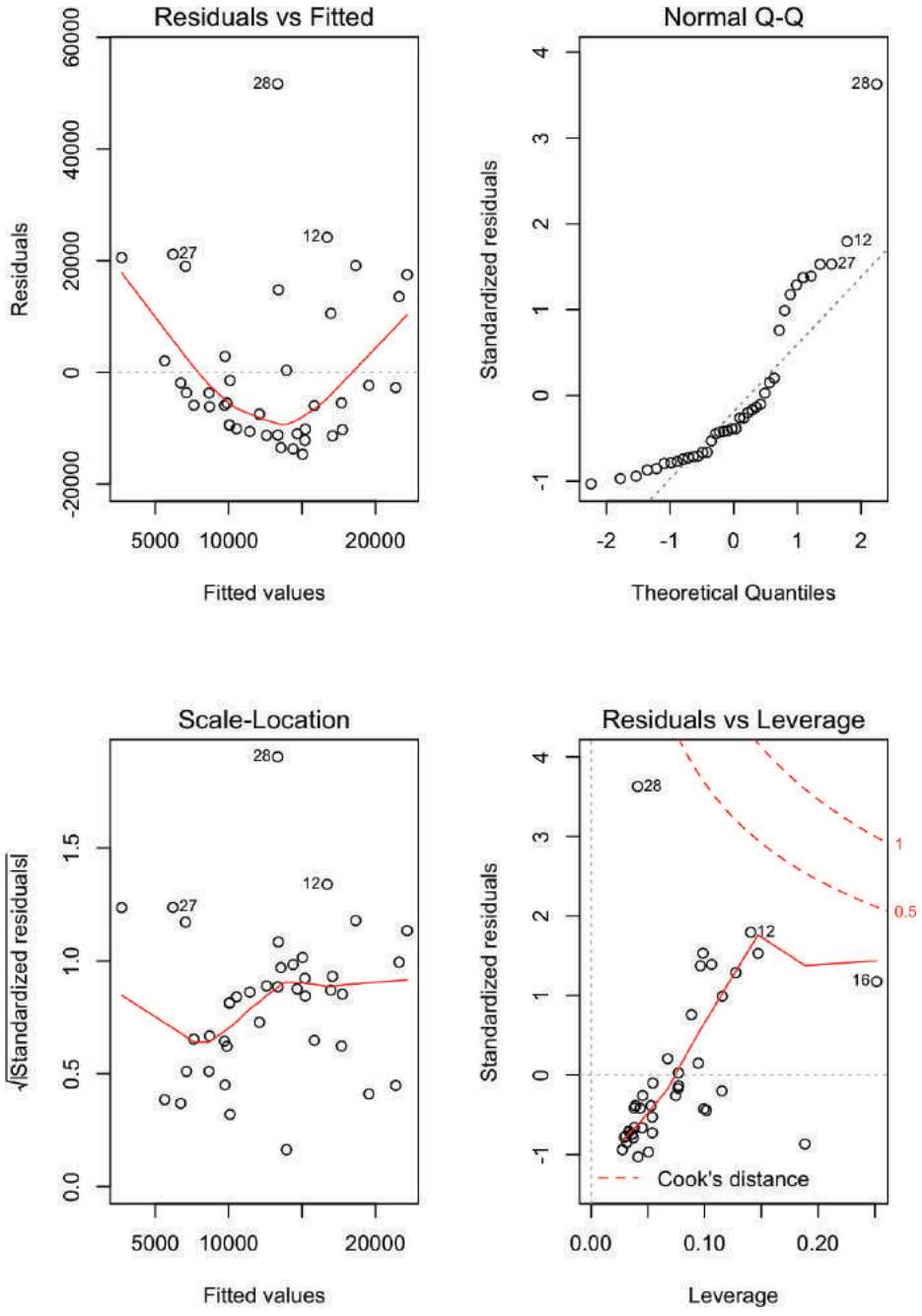


Рис. 11-7. Диагностические графики для плохой регрессии

См. также

См. рецепты 11.4 и 11.17. Пакет `car` не входит в стандартный дистрибутив R; *см.* рецепт 3.10, где показано, как его установить.

11.17. Обнаружение влиятельных наблюдений

Задача

Вы хотите найти наблюдения, которые оказывают наибольшее влияние на регрессионную модель. Это полезно для диагностики возможных проблем с данными.

Решение

Функция `influence.measures` вычисляет ряд полезных статистических данных для определения влиятельных наблюдений и помечает значимые из них звездочкой (*). Ее основным аргументом является объект модели из вашей регрессии:

```
influence.measures(m)
```

Обсуждение

Можно было бы назвать этот рецепт «Обнаружение *чрезмерно* влиятельных наблюдений», но это было бы излишне. Все наблюдения влияют на регрессионную модель, хотя бы незначительно. Когда специалист по статистике говорит, что наблюдение является влиятельным, это означает, что удаление наблюдения значительно изменит подогнанную модель регрессии. Мы хотим обнаружить эти наблюдения, потому что они могут быть выбросами, которые искажают нашу модель; мы обязаны сами исследовать их.

Функция `influence.measures` вычисляет ряд статистических данных: DFBETAS, DFFITS, коэффициент ковариации, расстояние Кука и значения матрицы проекции на пространство регрессоров. Если какая-либо из этих мер показывает, что наблюдение является влиятельным, функция помечает это наблюдение звездочкой (*) вдоль правой стороны:

```
influence.measures(m)
#> Influence measures of
#> lm(formula = y2 ~ x3 + x4) :
#>
#>          dfb.1_    dfb.x3    dfb.x4    dffit    cov.r    cook.d       hat    inf
#>     1  -0.18784   0.15174   0.07081 -0.22344    1.059  1.67e-02    0.0506
#>     2   0.27637  -0.04367  -0.39042  0.45416    1.027  6.71e-02    0.0964
#>     3  -0.01775  -0.02786   0.01088 -0.03876    1.175  5.15e-04    0.0772
#>     4   0.15922  -0.14322   0.25615  0.35766    1.133  4.27e-02    0.1156
#>     5  -0.10537   0.00814  -0.06368 -0.13175    1.078  5.87e-03    0.0335
#>     6   0.16942   0.07465   0.42467  0.48572    1.034  7.66e-02    0.1062
#>     7  -0.10128  -0.05936   0.01661 -0.13021    1.078  5.73e-03    0.0333
#>     8  -0.15696   0.04801   0.01441 -0.15827    1.038  8.38e-03    0.0276
#>     9  -0.04582  -0.12089  -0.01032 -0.14010    1.188  6.69e-03    0.0995
#>    10  -0.01901   0.00624   0.01740 -0.02416    1.147  2.00e-04    0.0544
#>    11  -0.06725  -0.01214   0.04382 -0.08174    1.113  2.28e-03    0.0381
#>    12   0.17580   0.35102   0.62952  0.74889    0.961  1.75e-01    0.1406
#>    13  -0.14288   0.06667   0.06786 -0.15451    1.071  8.04e-03    0.0372
```

```
#>    14   -0.02784    0.02366   -0.02727  -0.04790     1.173   7.85e-04    0.0767
#>    15    0.01934    0.03440   -0.01575   0.04729     1.197   7.66e-04    0.0944
#>    16    0.35521   -0.53827   -0.44441   0.68457     1.294   1.55e-01    0.2515       *
#>    17   -0.09184   -0.07199    0.01456  -0.13057     1.089   5.77e-03    0.0381
#>    18   -0.05807   -0.00534   -0.05725  -0.08825     1.119   2.66e-03    0.0433
#>    19    0.00288    0.00438    0.00511   0.00761     1.176   1.99e-05    0.0770
#>    20    0.08795    0.06854    0.19526   0.23490     1.136   1.86e-02    0.0884
#>    21    0.22148    0.42533   -0.33557   0.64699     1.047   1.34e-01    0.1471
#>    22    0.20974   -0.19946    0.36117   0.49631     1.085   8.06e-02    0.1275
#>    23   -0.03333   -0.05436    0.01568  -0.07316     1.167   1.83e-03    0.0747
#>    24   -0.04534   -0.12827   -0.03282  -0.14844     1.189   7.51e-03    0.1016
#>    25   -0.11334    0.00112   -0.05748  -0.13580     1.067   6.22e-03    0.0307
#>    26   -0.23215    0.37364    0.16153  -0.41638     1.258   5.82e-02    0.1883       *
#>    27    0.29815    0.01963   -0.43678   0.51616     0.990   8.55e-02    0.0986
#>    28    0.83069   -0.50577   -0.35404   0.92249     0.303   1.88e-01    0.0411       *
#>    29   -0.09920   -0.07828   -0.02499  -0.14292     1.077   6.89e-03    0.0361
#> # etc.
```

Это модель из рецепта 11.16, где у нас возникло подозрение, что наблюдение 28 было выбросом. Звездочка помечает это наблюдение, подтверждая его чрезмерное влияние.

С помощью данного рецепта можно обнаружить влиятельные наблюдения, но не стоит рефлексивно удалять их. Здесь нужно подумать. Эти наблюдения улучшают вашу модель или наносят ей вред?

См. также

См. рецепт 11.16. Используйте `help(effect.measures)`, чтобы получить список мер влияния и связанных с ними функций. Обратитесь к учебнику по регрессии, чтобы узнать об интерпретации различных мер воздействия.

11.18. Тестирование невязок на наличие автокорреляции (критерий Дарбина–Уотсона)

Задача

Вы построили линейную регрессию и хотите проверить невязки на наличие автокорреляции.

Решение

С помощью критерия Дарбина–Уотсона можно проверить невязки на наличие автокорреляции. Этот критерий реализуется функцией `dwtest` из пакета `lmtest`:

```
library(lmtest)
m <- lm(y ~ x)     # Создаем объект модели.
dwtest(m)          # Тестируем остатки модели.
```

Вывод содержит p-значение. Традиционно, если $p < 0{,}05$, то невязки значительно коррелируют, тогда как $p > 0{,}05$ не дает доказательств корреляции.

Можно выполнить визуальную проверку на наличие автокорреляции, построив график автокорреляционной функции (ACF) невязок:

```
acf(m)      # Строит график автокорреляционной функции невязок модели.
```

Обсуждение

Критерий Дарбина–Уотсона часто используется при анализе временных рядов, но первоначально он был создан для диагностики автокорреляции в невязках регрессии. Автокорреляция в невязках – это просто беда, поскольку она искажает регрессионную статистику, такую как *F*-статистика и *t*-статистика коэффициентов регрессии. Наличие автокорреляции говорит о том, что в вашей модели отсутствует полезная независимая переменная или что она должна включать в себя компонент временного ряда, такой как тренд или сезонный индикатор.

В первом примере мы строим простую регрессионную модель, а затем проверяем невязки на наличие автокорреляции. Тест возвращает *p*-значение значительно выше нуля, что указывает на отсутствие значимой автокорреляции:

```
library(lmtest)
load(file = './data/ac.rdata')
m <- lm(y1 ~ x)
dwtest(m)
#>
#> Durbin-Watson test
#>
#> data: m
#> DW = 2, p-value = 0.4
#> alternative hypothesis: true autocorrelation is greater than 0
```

Второй пример демонстрирует автокорреляцию в невязках. *p*-значение близко к нулю, поэтому автокорреляция, скорее всего, положительная:

```
m <- lm(y2 ~ x)
dwtest(m)
#>
#> Durbin-Watson test
#>
#> data: m
#> DW = 2, p-value = 0.01
#> alternative hypothesis: true autocorrelation is greater than 0
```

По умолчанию функция `dwtest` выполняет односторонний тест и отвечает на вопрос: автокорреляция невязок больше нуля? Если ваша модель может демонстрировать отрицательную автокорреляцию (да, такое возможно), вы должны использовать опцию `alternative` для выполнения двустороннего теста:

```
dwtest(m, alternative = "two.sided")
```

Критерий Дарбина–Уотсона также можно реализовать, используя функцию `durbinWatsonTest` из пакета `car`. Мы предложили функцию `dwtest` прежде всего потому, что полагаем, что ее вывод легче читать.

См. также

Ни пакет `lmtest`, ни пакет `car` не входят в стандартный дистрибутив R; *см.* рецепты 3.8 и 3.10, чтобы узнать, как получить доступ к их функциям и установить их. *См.* рецепты 14.13 и 14.16 для получения более подробной информации о тестах автокорреляции.

11.19. Предсказываем новые значения

Задача

Вы хотите предсказать новые значения на основе своей регрессионной модели.

Решение

Сохраните данные независимой переменной в таблице данных. Используйте функцию `predict`, задав для параметра `newdata` в качестве значения таблицу данных:

```
load(file = './data/pred2.rdata')

m <- lm(y ~ u + v + w)
preds <- data.frame(u = 3.1, v = 4.0, w = 5.5)
predict(m, newdata = preds)
#> 1
#> 45
```

Обсуждение

Когда у вас есть линейная модель, делать предсказания довольно легко, потому что функция `predict` выполняет всю тяжелую работу. Единственная неприятность – настройка таблицы данных, которая содержит ваши данные.

Функция `predict` возвращает вектор предсказанных значений с одним предсказанием для каждой строки в данных. Пример, использованный в решении, содержит одну строку, поэтому функция вернула одно значение.

Если данные независимой переменной содержат несколько строк, вы получите по одному прогнозу на строку:

```
preds <- data.frame(
  u = c(3.0, 3.1, 3.2, 3.3),
  v = c(3.9, 4.0, 4.1, 4.2),
  w = c(5.3, 5.5, 5.7, 5.9)
)
predict(m, newdata = preds)
#> 1 2 3 4
#> 43.8 45.0 46.3 47.5
```

В случае если это не очевидно: новые данные не должны содержать значения зависимых переменных, только независимые переменные. В конце концов, вы пытаетесь *рассчитать* ответ, поэтому было бы неразумно ожидать, что вы его предоставите.

См. также

Это лишь точечные оценки предсказаний. *См.* рецепт 11.20, где говорится о доверительных интервалах.

11.20. Формирование интервалов предсказаний

Задача

Вы делаете предсказания, используя модель линейной регрессии. Вам нужно знать интервалы предсказаний: диапазон распределения предсказания.

Решение

Используйте функцию `predict` и укажите `interval = "prediction"`:

```
predict(m, newdata = preds, interval = "prediction")
```

Обсуждение

Это продолжение рецепта 11.19, в котором описана упаковка ваших данных в таблице данных для функции `predict`. Мы добавляем `interval = "prediction"`, чтобы получить интервалы предсказания.

Вот пример из рецепта 11.19, теперь с интервалами предсказания. Новые столбцы `lwr` и `upr` имеют нижний и верхний пределы соответственно для интервала:

```
predict(m, newdata = preds, interval = "prediction")
#>    fit    lwr   upr
#> 1 43.8  38.2  49.4
#> 2 45.0  39.4  50.7
#> 3 46.3  40.6  51.9
#> 4 47.5  41.8  53.2
```

По умолчанию функция `predict` использует уровень достоверности 0,95. Его можно изменить с помощью аргумента `level`.

Предупреждение: эти интервалы предсказания чрезвычайно чувствительны к отклонениям от нормальности. Если вы подозреваете, что ваша зависимая переменная не распределена нормально, рассмотрите непараметрический метод, например бутстрэп (*см.* рецепт 13.8).

11.21. Однофакторный дисперсионный анализ

Задача

Ваши данные поделены на группы, а группы нормально распределены. Вы хотите знать, есть ли у групп существенно разные средние значения.

Решение

Используйте фактор для определения групп. Затем примените функцию `oneway.test`:

```
oneway.test(x ~ f)
```

Здесь x – вектор числовых значений, а f – фактор, идентифицирующий группы. Вывод включает в себя *p*-значение. Обычно *p*-значение меньше 0,05 указывает на то, что две или более групп имеют значительно разные средние значения, тогда как значение, превышающее 0,05, не дает таких доказательств.

Обсуждение

Сравнение средних значений групп – обычная задача. Однофакторный дисперсионный анализ выполняет это сравнение и вычисляет вероятность того, что они статистически идентичны. Небольшое *p*-значение указывает на то, что две или более групп, вероятно, имеют разные средние значения. (Это не означает, что у *всех* групп разные средние значения.)

Базовый тест ANOVA предполагает, что ваши данные имеют нормальное распределение или, по крайней мере, довольно близки к колоколообразным. Если это не так, используйте критерий Краскела–Уоллиса (*см.* рецепт 11.24).

Можно проиллюстрировать дисперсионный анализ с помощью исторических данных с фондового рынка. Является ли фондовый рынок более прибыльным в некоторые месяцы, чем в другие? Например, распространенный народный миф гласит, что октябрь – плохой месяц для инвесторов на фондовом рынке[1]. Мы исследовали этот вопрос, создав таблицу данных GSPC_df, содержащую два столбца r и mon. Фактор r – это ежедневная доходность в индексе Standard & Poor's 500 – широкий показатель эффективности фондового рынка. Коэффициент mon указывает на календарный месяц, в котором произошло это изменение: январь, февраль, март и т. д. Данные охватывают период с 1950 по 2009 год.

Однофакторный дисперсионный анализ показывает *p*-значение 0,03347:

```
load(file = './data/anova.rdata')
oneway.test(r ~ mon, data = GSPC_df)
#>
#> One-way analysis of means (not assuming equal variances)
#>
#> data: r and mon
#> F = 2, num df = 10, denom df = 7000, p-value = 0.03
```

Можно сделать вывод, что изменения на фондовом рынке значительно варьировались в зависимости от календарного месяца.

Однако, прежде чем бежать к своему брокеру и ежемесячно перетряхивать свой портфель, нужно кое-что проверить: изменилась ли модель в последнее время? Мы можем ограничить анализ последними данными, указав параметр subset, который подходит для функции oneway.test так же, как и для функции lm. В этом параметре содержатся индексы наблюдений для анализа; все остальные наблюдения игнорируются. Здесь мы даем индексы 2500 самых последних наблюдений, которые представляют собой данные примерно за 10 лет:

```
oneway.test(r ~ mon, data = GSPC_df, subset = tail(seq_along(r), 2500))
#>
#> One-way analysis of means (not assuming equal variances)
```

[1] Как говорил Марк Твен: «Октябрь – один из самых опасных месяцев в году для игры на бирже. Остальные опасные месяцы: июль, январь, сентябрь, апрель, ноябрь, май, март, июнь, декабрь, август и февраль».

```
#>
#> data: r and mon
#> F = 0.7, num df = 10, denom df = 1000, p-value = 0.8
```

Ой-ой! Эти ежемесячные различия испарились за последние 10 лет. Большое *p*-значение, 0,8, указывает на то, что в последнее время изменения не происходили в зависимости от календарного месяца. Видимо, эти различия ушли в прошлое.

Обратите внимание, что в выводе функции `oneway.test` сказано: `not assuming equal variances` (*не предполагая одинаковую дисперсию*). Если вы знаете, что у групп одинаковая дисперсия, то получите менее консервативный тест, указав для `var.equal` значение `TRUE`:

```
oneway.test(x ~ f, var.equal = TRUE)
```

Также можно выполнить однофакторный дисперсионный анализ с помощью функции `aov`:

```
m <- aov(x ~ f)
summary(m)
```

Однако функция `aov` всегда допускает одинаковую дисперсию и поэтому несколько менее гибкая, по сравнению с `oneway.test`.

См. также

Если средние значения значительно отличаются, используйте рецепт 11.23, чтобы увидеть реальные различия. Примените рецепт 11.24, если ваши данные не нормально распространены, как того требует дисперсионный анализ.

11.22. Создание диаграммы взаимодействия

Задача

Вы выполняете многофакторный дисперсионный анализ, используя две или более категориальных переменных в качестве независимых переменных. Вам нужна визуальная проверка возможного взаимодействия между независимыми переменными.

Решение

Используйте функцию `interaction.plot`:

```
interaction.plot(pred1, pred2, resp)
```

Здесь `pred1` и `pred2` – две категориальные независимые переменные, а `resp` – зависимая переменная.

Обсуждение

Дисперсионный анализ – это форма линейной регрессии, поэтому в идеале существует линейная связь между независимыми переменными и зависимой переменной. Одним из источников нелинейной связи является взаимодействие между двумя независимыми переменными: когда одна переменная меняет значение, другая меняет свою связь с зависимой переменной. Проверка взаимодействия между независимыми переменными является базовой диагностикой.

Пакет `faraway` содержит набор данных под названием `rats`. `treat` и `poison` здесь – это категориальные переменные, а `time` – зависимая переменная. При отображении зависимости переменной `poison` от переменной `time` мы ищем прямые, параллельные линии, которые указывают на линейную связь.

Тем не менее, используя функцию `interaction.plot`, мы получаем диаграмму, изображенную на рис. 11-8. Глядя на нее, видно, что что-то не так:

```
library(faraway)
data(rats)
interaction.plot(rats$poison, rats$treat, rats$time)
```

Каждая линяя на диаграмме отображает зависимость переменной `time` от переменной `poison`. Разница между линиями заключается в том, что каждая линия предназначена для разных значений переменной `treat`. Линии должны быть параллельными, однако верхние две не совсем параллельны. Очевидно, что различие в значении `treat` «исказило» линии, внося нелинейность в связь между `poison` и `time`.

Это сигнализирует о возможном взаимодействии, которое мы должны проверить. В случае с этими данными просто так получается, что да, взаимодействие есть, и нет, оно не является статистически значимым. Мораль ясна: визуальная проверка полезна, но не надежна. Выполните статистическую проверку.

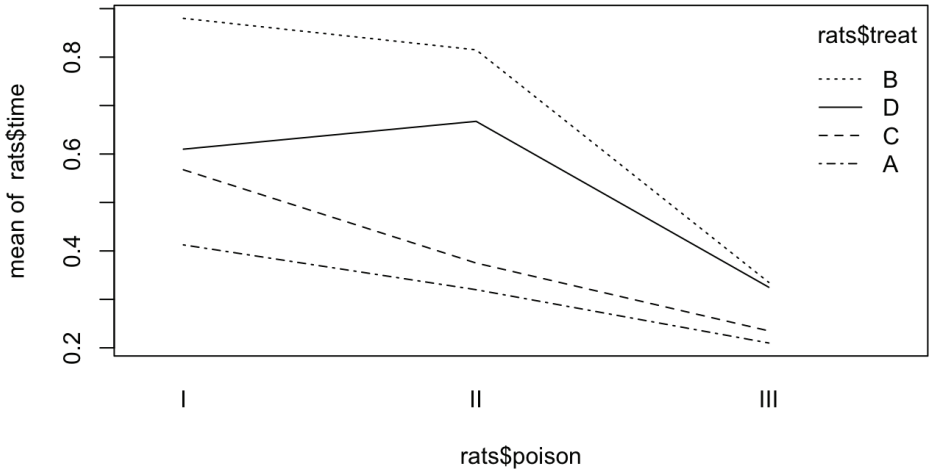


Рис. 11-8. Диаграмма взаимодействия

См. также

См. рецепт 11.7.

11.23. Находим различия между средними значениями групп

Задача

Ваши данные разделены на группы, и тест ANOVA показывает, что группы имеют существенно разные средние значения. Вы хотите знать, каковы различия между этими значениями для всех групп.

Решение

Выполните тест ANOVA, используя функцию aov, которая возвращает объект модели. Затем примените к объекту модели функцию TukeyHSD:

```
m <- aov(x ~ f)
TukeyHSD(m)
```

Здесь x – это ваши данные, а f – фактор группировки. Вы можете построить результат, полученный с помощью функции TukeyHSD, для получения графического отображения различий:

```
plot(TukeyHSD(m))
```

Обсуждение

Тест ANOVA важен, потому что он говорит вам, отличаются ли средние значения групп. Но он не определяет, *какие* группы отличаются, и не вычисляет их различия.

Функция TukeyHSD может рассчитать эти различия и помочь вам определить самые крупные из них. Она использует метод «честных значимых различий», изобретенный Джоном Тьюки.

Мы проиллюстрируем, как работает эта функция, продолжив пример из рецепта 11.21, в котором ежедневные изменения фондового рынка сгруппированы по месяцам. Здесь мы группируем их по дням недели, используя фактор wday, который определяет день недели (понедельник, ..., пятница), в который произошло изменение. Мы будем использовать первые 2500 наблюдений, которые примерно охватывают период с 1950 по 1960 год:

```
load(file = './data/anova.rdata')
oneway.test(r ~ wday, subset = 1:2500, data = GSPC_df)
#>
#> One-way analysis of means (not assuming equal variances)
#>
#> data: r and wday
#> F = 10, num df = 4, denom df = 1000, p-value = 5e-10
```

p-значение, по существу, равно нулю, что указывает на то, что средние изменения значительно варьируются в зависимости от дня недели. Чтобы использовать функцию TukeyHSD, мы сначала выполняем тест ANOVA с использованием функции aov, которая возвращает объект модели, а затем применяем функцию TukeyHSD к этому объекту:

```
m <- aov(r ~ wday, subset = 1:2500, data = GSPC_df)
TukeyHSD(m)
#> Tukey multiple comparisons of means
#> 95% family-wise confidence level
#>
#> Fit: aov(formula = r ~ wday, data = GSPC_df, subset = 1:2500)
#> $wday
#>                 diff          lwr          upr    p adj
#>    Mon-Fri -0.003153 -4.40e-03 -0.001911  0.000
#>    Thu-Fri -0.000934 -2.17e-03  0.000304  0.238
#>    Tue-Fri -0.001855 -3.09e-03 -0.000618  0.000
#>    Wed-Fri -0.000783 -2.01e-03  0.000448  0.412
#>    Thu-Mon  0.002219  9.79e-04  0.003460  0.000
#>    Tue-Mon  0.001299  5.85e-05  0.002538  0.035
#>    Wed-Mon  0.002370  1.14e-03  0.003605  0.000
```

```
#>    Tue-Thu     -0.000921    -2.16e-03     0.000314   0.249
#>    Wed-Thu      0.000151    -1.08e-03     0.001380   0.997
#>    Wed-Tue      0.001072    -1.57e-04     0.002300   0.121
```

Каждая строка в таблице вывода содержит разницу между средними значениями двух групп (`diff`), а также нижнюю и верхнюю границы доверительного интервала (`lwr` и `upr`). Например, в первой строке таблицы сравниваются группа Mon и группа Fri: разница их средних значений составляет 0,003 с доверительным интервалом (–0,0044, –0,0019).

Просматривая таблицу, мы видим, что сравнение Wed-Mon имело наибольшую разницу, которая составляла 0,00237.

Отличительной особенностью функции `TukeyHSD` является то, что она также может отображать эти различия визуально. Просто постройте возвращаемое значение функции, чтобы получить вывод, как показано на рис. 11-9:

```
plot(TukeyHSD(m))
```

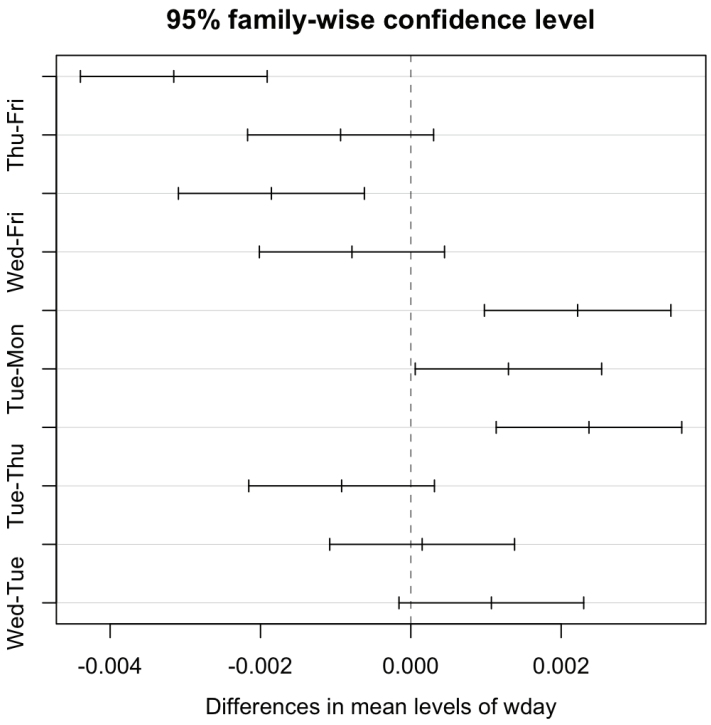


Рис. 11-9. Диаграмма функции TukeyHSD

Горизонтальные линии показывают доверительные интервалы для каждой пары. С помощью этого визуального представления можно быстро увидеть, что несколько доверительных интервалов пересекают ноль, указывая на то, что разница не обязательно значительна. Также видно, что пара Wed-Mon имеет наибольшую разницу, потому что их доверительный интервал больше всех сдвинут вправо.

См. также

См. рецепт 11.21.

11.24. Устойчивый дисперсионный анализ (критерий Краскела–Уоллиса)

Задача

Ваши данные поделены на группы. Группы не нормально распределены, но их распределения имеют схожие формы. Вы хотите выполнить тест, аналогичный тесту ANOVA, – вам нужно знать, значительно ли отличаются медианы групп.

Решение

Создайте фактор, который определяет группы ваших данных. Используйте функцию `kruskal.test`, которая реализует тест Краскела–Уоллиса. В отличие от теста ANOVA, этот тест не зависит от нормальности данных:

```
kruskal.test(x ~ f)
```

Здесь `x` – это вектор данных, а `f` – фактор группировки. Вывод включает в себя *p*-значение.

Традиционно, $p < 0{,}05$ указывает на то, что существует значительная разница между медианами двух или более групп, тогда как $p > 0{,}05$ не предоставляет таких доказательств.

Обсуждение

Обычный дисперсионный анализ предполагает, что ваши данные имеют нормальное распределение. Он может допускать некоторое отклонение от нормальности, но крайние отклонения приведут к бессмысленным *p*-значениям.

Критерий Краскела–Уоллиса представляет собой непараметрическую версию ANOVA. Это означает, что он не предполагает нормальности. Тем не менее он предполагает распределения одинаковой формы. Критерий Краскела–Уоллиса следует использовать, если распределение ваших данных ненормально или просто неизвестно.

Нулевая гипотеза состоит в том, что все группы имеют одинаковую медиану. Когда мы отвергаем нулевую гипотезу (когда $p < 0{,}05$), это не значит, что *все* группы разные, а предполагает, что две или более групп различны.

Однажды Пол преподавал бизнес-статистику 94 студентам. Занятия включали в себя промежуточный экзамен, а перед экзаменом было четыре домашних задания. Он хотел знать: какова связь между выполнением домашнего задания и успешной сдачей экзамена? Если связи нет, то домашнее задание не имеет значения и нуждается в переосмыслении.

Он создал вектор оценок, по одной на каждого учащегося, а также параллельный фактор, который отражал количество заданий, выполненных этим учеником. Данные находятся в таблице данных с именем `student_data`:

```
load(file = './data/student_data.rdata')
head(student_data)
```

```
#> # Tibble: 6 x 4
#>   att.fact hw.mean midterm hw
#> <fct>      <dbl>   <dbl>   <fct>
#> 1 3        0.808   0.818   4
#> 2 3        0.830   0.682   4
#> 3 3        0.444   0.511   2
#> 4 3        0.663   0.670   3
#> 5 2        0.9     0.682   4
#> 6 3        0.948   0.954   4
```

Обратите внимание, что переменная `hw` – хотя она выглядит числовой – на самом деле является фактором. Она назначает оценку за экзамен каждой из пяти групп в зависимости от того, сколько домашних заданий выполнил учащийся.

Распределение оценок за экзамен определенно не является нормальным: у учащихся широкий спектр математических навыков, поэтому существует необычное количество оценок «очень хорошо» и «очень плохо». Следовательно, обычный дисперсионный анализ не подходит. Вместо этого мы использовали критерий Краскела–Уоллиса и получили *p*-значение, по существу равное нулю (4×10^{-5}, или 0.00004):

```
kruskal.test(midterm ~ hw, data = student_data)
#>
#> Kruskal-Wallis rank sum test
#>
#> data: midterm by hw
#> Kruskal-Wallis chi-squared = 30, df = 4, p-value = 4e-05
```

Очевидно, что существует значительная разница в успеваемости между учащимися, которые выполняют домашнее задание, и теми, кто этого не делает. Но к какому выводу в действительности мог прийти Пол?

Сначала он был рад, что домашняя работа оказалась настолько эффективной. Затем до него дошло, что это классическая ошибка в статистических рассуждениях: он предположил, что корреляция подразумевает причинность. Конечно, это не так. Возможно, сильно мотивированные студенты хорошо справляются как с домашними заданиями, так и с экзаменами, а ленивые – нет. В этом случае причинным фактором является степень мотивации, а не блеск выбора домашней работы. В конце концов он мог сделать только один очень простой вывод: студенты, которые выполнили домашнее задание, скорее всего, преуспеют на промежуточном экзамене, – но он до сих пор не знает, почему.

11.25. Сравнение моделей с использованием функции anova

Задача

У вас есть две модели одних и тех же данных, и вы хотите знать, дают ли они разные результаты.

Решение

Функция `anova` может сравнивать две модели и сообщать, если они существенно различаются:

```
anova(m1, m2)
```

Здесь m1 и m2 – модельные объекты, возвращаемые функцией lm. Вывод функции anova включает в себя *p*-значение. Обычно *p*-значение менее 0,05 указывает на то, что модели значительно различаются, тогда как значение, превышающее 0,05, не дает таких доказательств.

Обсуждение

В рецепте 11.3 мы использовали функцию anova для вывода таблицы дисперсионного анализа для одной регрессионной модели. Теперь мы применяем вариант с двумя аргументами для сравнения двух моделей.

Функция anova предъявляет одно строгое требование при сравнении двух моделей: одна модель должна содержаться внутри другой. То есть все члены меньшей модели должны появляться в более крупной модели. В противном случае сравнение невозможно.

Дисперсионный анализ выполняет *F*-тест, который напоминает *F*-тест для линейной регрессии. Разница состоит в том, что этот тест проводится между двумя моделями, тогда как регрессионный F-тест проводится между использованием регрессионной модели и ее отсутствием. Предположим, что мы строим три модели y, добавляя члены по ходу дела:

```
load(file = './data/anova2.rdata')
m1 <- lm(y ~ u)
m2 <- lm(y ~ u + v)
m3 <- lm(y ~ u + v + w)
```

Действительно ли m2 отличается от m1? Можно использовать функцию anova для их сравнения, и в результате мы получим *p*-значение 0,0091:

```
anova(m1, m2)
#> Analysis of Variance Table
#>
#> Model 1: y ~ u
#> Model 2: y ~ u + v
#>   Res.Df RSS Df Sum of Sq F Pr(>F)
#> 1    18  197
#> 2    17  130 1 66.4 8.67 0.0091 **
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Небольшое *p*-значение указывает на то, что модели значительно отличаются. Однако, сравнивая m2 и m3, мы получаем p-значение 0,055:

```
anova(m2, m3)
#> Analysis of Variance Table
#>
#> Model 1: y ~ u + v
#> Model 2: y ~ u + v + w
#>     Res.Df  RSS Df Sum of Sq F Pr(>F)
#> 1       17  130
#> 2       16  103 1 27.5  4.27  0.055 .
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Прямо у края. Строго говоря, это не соответствует нашему требованию быть меньше 0,05; однако это достаточно близко для того, чтобы можно было судить, что модели «достаточно разные».

Это немного надуманный пример, поэтому он не показывает реальную силу функции `anova`. Мы используем эту функцию, когда, экспериментируя со сложными моделями, добавляя и удаляя несколько членов, мы должны знать, действительно ли новая модель отличается от исходной. Другими словами: если мы добавим члены, а новая модель практически не изменится, то дополнительные члены не будут стоить дополнительных сложностей.

Глава 12

Полезные хитрости

Рецепты, приведенные в этой главе, – это не изощрённые вычисления и не крутые статистические методы. Тем не менее это полезные функции и идиомы, которые вам, вероятно, понадобятся в тот или иной момент.

12.1. Просмотр данных

Задача

У вас много данных – слишком много, чтобы отображать их все сразу. Тем не менее вы хотите увидеть некоторые из них.

Решение

Используйте функцию `head` для просмотра первых нескольких значений данных или строк:

```
head(x)
```

Используйте функцию `tail` для просмотра последних нескольких значений данных или строк:

```
tail(x)
```

Либо можно просмотреть все это в интерактивном средстве просмотра в RStudio:

```
View(x)
```

Обсуждение

Вывод большого набора данных не имеет смысла, потому что все будет просто скатываться с экрана. Используйте функцию `head`, чтобы увидеть небольшую часть данных (по умолчанию шесть строк):

```
load(file = './data/lab_df.rdata')
head(lab_df)
#>        x   lab    y
#> 1 0.0761   NJ  1.621
#> 2 1.4149   KY  10.338
#> 3 2.5176   KY  14.284
#> 4 -0.3043 KY  0.599
#> 5 2.3916   KY  13.091
#> 6 2.0602   NJ  16.321
```

Используйте функцию `tail`, чтобы увидеть последние несколько строк и их число:

```
tail(lab_df)
#>          x   lab    y
#> 195 7.353   KY  38.880
#> 196 -0.742  KY  -0.298
#> 197 2.116   NJ  11.629
#> 198 1.606   KY  9.408
#> 199 -0.523  KY  -1.089
#> 200 0.675   KY  5.808
```

Обе эти функции позволяют передавать число в функцию, чтобы установить число возвращаемых строк:

```
tail(lab_df, 2)
#>          x   lab    y
#> 199 -0.523  KY  -1.09
#> 200 0.675   KY  5.81
```

RStudio поставляется со встроенным интерактивным средством просмотра. Его можно вызвать из консоли или сценария:

```
View(lab_df)
```

Или можно передать объект средству просмотра:

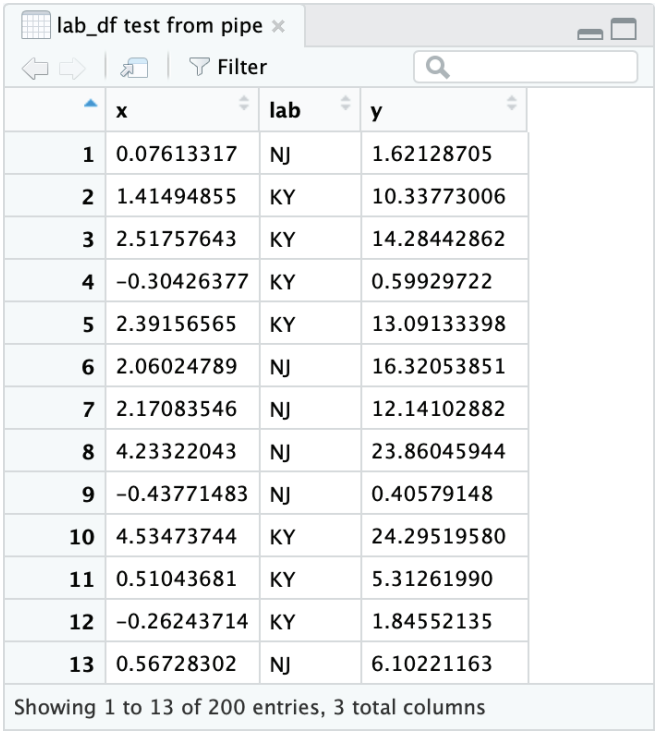
```
lab_df %>%

View()
```

При этом вы заметите, что средство просмотра называет вкладку **View** просто . (точка). Чтобы получить более информативное имя, можно поместить описательное имя в кавычки:

```
lab_df %>%
View("lab_df test from pipe")
```

Результат показан на рис. 12-1.


lab_df test from pipe

Filter

	x	lab	y
1	0.07613317	NJ	1.62128705
2	1.41494855	KY	10.33773006
3	2.51757643	KY	14.28442862
4	-0.30426377	KY	0.59929722
5	2.39156565	KY	13.09133398
6	2.06024789	NJ	16.32053851
7	2.17083546	NJ	12.14102882
8	4.23322043	NJ	23.86045944
9	-0.43771483	NJ	0.40579148
10	4.53473744	KY	24.29519580
11	0.51043681	KY	5.31261990
12	-0.26243714	KY	1.84552135
13	0.56728302	NJ	6.10221163

Showing 1 to 13 of 200 entries, 3 total columns

Рис. 12-1. Средство просмотра RStudio

См. также

См. рецепт 12.13, чтобы узнать, как увидеть структуру содержимого вашей переменной.

12.2. Вывод на экран результата присваивания

Задача

Вы присваиваете значение переменной и хотите увидеть его значение.

Решение

Просто поставьте круглые скобки вокруг присваивания:

```
x <- 1/pi       # Ничего не выводит.
(x <- 1/pi)     # Выводит присвоенное значение.
#> [1] 0.318
```

Обсуждение

Обычно R запрещает вывод на экран, когда видит, что вы вводите простое присваивание. Однако, когда вы окружаете присваивание круглыми скобками, оно перестает быть простым, поэтому R выводит значение. Это может быть очень удобно при быстрой отладке сценария.

См. также

См. рецепт 2.1, чтобы узнать, как осуществлять вывод.

12.3. Суммирование строк и столбцов

Задача

Вы хотите суммировать строки или столбцы матрицы либо таблицы данных.

Решение

Используйте функцию `rowSums` для суммирования строк:

```
rowSums(m)
```

Используйте функцию `colSums` для суммирования столбцов:

```
colSums(m)
```

Обсуждение

Это обычный рецепт, но он настолько распространен, что заслуживает упоминания. Мы используем его, например, при создании отчетов, которые включают в себя итоги столбцов. В этом примере `daily.prod` – это запись о фабричном производстве за эту неделю, и мы хотим получить итоги по продуктам и по дням:

```
load(file = './data/daily.prod.rdata')
daily.prod
#>      Widgets  Gadgets  Thingys
#> Mon    179      167      182
#> Tue    153      193      166
#> Wed    183      190      170
#> Thu    153      161      171
#> Fri    154      181      186
colSums(daily.prod)
#> Widgets  Gadgets  Thingys
#>    822      892      875
rowSums(daily.prod)
#> Mon Tue Wed Thu Fri
#> 528 512 543 485 521
```

Эти функции возвращают вектор. В случае сумм столбцов мы можем добавить вектор в матрицу и тем самым аккуратно вывести данные и итоги вместе:

```
rbind(daily.prod, Totals=colSums(daily.prod))
#>        Widgets  Gadgets  Thingys
#> Mon      179      167      182
#> Tue      153      193      166
#> Wed      183      190      170
#> Thu      153      161      171
#> Fri      154      181      186
#> Totals   822      892      875
```

12.4. Вывод данных в столбцах

Задача

У вас есть несколько параллельных векторов данных, и вы хотите вывести их в столбцах.

Решение

Используйте функцию `cbind`, чтобы сформировать данные в столбцы, а затем вывести результат.

Обсуждение

Когда у вас есть параллельные векторы, трудно увидеть их взаимосвязь, если вывести их отдельно:

```
load(file = './data/xy.rdata')
print(x)
#> [1] -0.626 0.184 -0.836 1.595 0.330 -0.820 0.487 0.738 0.576 -0.305
print(y)
#> [1] 1.5118 0.3898 -0.6212 -2.2147 1.1249 -0.0449 -0.0162 0.9438
#> [9] 0.8212 0.5939
```

Используйте функцию `cbind`, чтобы сформировать их в столбцы, которые при выводе показывают структуру данных:

```
print(cbind(x,y))
#>          x       y
#> [1,]  -0.626  1.5118
#> [2,]  0.184   0.3898
#> [3,]  -0.836  -0.6212
#> [4,]  1.595   -2.2147
#> [5,]  0.330   1.1249
#> [6,]  -0.820  -0.0449
#> [7,]  0.487   -0.0162
#> [8,]  0.738   0.9438
#> [9,]  0.576   0.8212
#> [10,] -0.305  0.5939
```

Вы также можете включить выражения в вывод. Используйте тег, чтобы дать им заголовок столбца:

```
print(cbind(x, y, Total = x + y))
#>          x       y      Total
#> [1,]  -0.626  1.5118  0.885
#> [2,]  0.184   0.3898  0.573
#> [3,]  -0.836  -0.6212 -1.457
#> [4,]  1.595   -2.2147 -0.619
#> [5,]  0.330   1.1249  1.454
#> [6,]  -0.820  -0.0449 -0.865
#> [7,]  0.487   -0.0162 0.471
#> [8,]  0.738   0.9438  1.682
#> [9,]  0.576   0.8212  1.397
#> [10,] -0.305  0.5939  0.289
```

12.5. Разбиение данных по бинам

Задача

У вас есть вектор, и вы хотите разбить данные на группы в соответствии с интервалами.

Статистики называют это *бинированием*.

Решение

Используйте функцию `cut`. Вы должны определить вектор, скажем, `breaks`, который дает диапазоны интервалов. Функция `cut` сгруппирует ваши данные в соответствии с этими интервалами. Она возвращает фактор, уровни (элементы) которого определяют группу для каждого значения:

```
f <- cut(x, breaks)
```

Обсуждение

В этом примере мы генерируем 1000 случайных чисел со стандартным нормальным распределением и разбиваем их на шесть групп, определяя интервалы в ±1, ±2 и ±3 стандартных отклонения:

```
x <- rnorm(1000)
breaks <- c(-3, -2, -1, 0, 1, 2, 3)
f <- cut(x, breaks)
```

Результатом является фактор `f`, который определяет группы. Функция `summary` показывает количество элементов по уровням. R создает имена для каждого уровня, используя математическое обозначение интервала:

```
summary(f)
#> (-3,-2] (-2,-1] (-1,0]  (0,1] (1,2] (2,3] NA's
#>      25     147     341     332   132    18    5
```

Результаты имеют форму колокола, чего мы и ожидаем от функции `rnorm`. Здесь есть пять значений `NA`, указывающих на то, что два значения в `x` выходят за пределы определенных интервалов.

Мы можем использовать параметр `labels`, чтобы дать отличные, предопределенные имена шести группам вместо причудливо синтезированных имен:

```
f <- cut(x, breaks, labels = c("Bottom", "Low", "Neg", "Pos", "High", "Top"))
```

Теперь функция `summary` использует наши имена:

```
summary(f)
#> Bottom   Low   Neg   Pos  High   Top  NA's
#>     25   147   341   332   132    18     5
```

Объединение полезно для таких сводок, как гистограммы. Но это приводит к потере информации, что может навредить моделированию. Рассмотрим крайний случай объединения непрерывной переменной в два значения: `high` и `low`. Объединяемые данные имеют только два возможных значения, поэтому вы заменили богатый источник информации *одним битом* информации.

В тех случаях, когда непрерывная переменная может быть мощной независимой переменной, объединяемая переменная может различать не более двух со-

стояний и, следовательно, будет иметь лишь небольшую часть первоначальной мощности. Прежде чем вы выполните объединение, мы предлагаем изучить другие преобразования с меньшими потерями.

12.6. Поиск положения определенного значения

Задача

У вас есть вектор. Вы знаете, что в содержимом встречается определенное значение, и вы хотите узнать его положение.

Решение

Функция `match` будет искать вектор определенного значения и возвращать положение:

```
vec <- c(100, 90, 80, 70, 60, 50, 40, 30, 20, 10)
match(80, vec)
#> [1] 3
```

Здесь `match` возвращает 3, это положение `80` в `vec`.

Обсуждение

Существуют специальные функции для определения местоположения минимального и максимального значений – `which.min` и `which.max` соответственно:

```
vec <- c(100,90,80,70,60,50,40,30,20,10)
which.min(vec)       # Положение наименьшего элемента.
#> [1] 10
which.max(vec)       # Положение наибольшего элемента.
#> [1] 1
```

См. также

Этот метод используется в рецепте 11.13.

12.7. Выбор каждого *N*-го элемента вектора

Задача

Вам нужно выбрать каждый *n*-й элемент вектора.

Решение

Создайте логический вектор индексации, который имеет значение `TRUE` для каждого *n*-го элемента. Один из подходов состоит в том, чтобы найти все индексы, которые равны нулю, когда взяты по модулю *n*:

```
v[seq_along(v) %% n == 0]
```

Обсуждение

Эта проблема возникает при систематической выборке: мы хотим выбрать набор данных, выбрав каждый *n*-й элемент. Функция `seq_along(v)` генерирует после-

довательность целых чисел, которая может индексировать v; это эквивалентно 1:length(v). Мы вычисляем каждое значение индекса по модулю *n* выражением:

```
v <- rnorm(10)
n <- 2
seq_along(v) %% n
#> [1] 1 0 1 0 1 0 1 0 1 0
```

Затем мы находим те значения, которые равны нулю:

```
seq_along(v) %% n == 0
#> [1] FALSE TRUE FALSE TRUE FALSE TRUE FALSE TRUE FALSE TRUE
```

Результатом является логический вектор той же длины, что и v, и значение TRUE для каждого *n*-го элемента, который может индексировать v для выбора нужных элементов:

```
v
#> [1] 2.325 0.524 0.971 0.377 -0.996 -0.597 0.165 -2.928 -0.848 0.799
v[ seq_along(v) %% n == 0 ]
#> [1] 0.524 0.377 -0.597 -2.928 0.799
```

Если вам нужно что-то простое, например каждый второй элемент, можно использовать правило повторного использования по-хитрому. Проиндексируйте v с двухэлементным логическим вектором, как здесь:

```
v[c(FALSE, TRUE)]
#> [1] 0.524 0.377 -0.597 -2.928 0.799
```

Если у v больше двух элементов, то вектор индексации слишком короткий. Следовательно, R вызовет правило повторного использования и расширит индексный вектор до длины v, «переиспользовав» его содержимое. Это дает индексный вектор, который имеет значение FALSE TRUE FALSE TRUE FALSE TRUE и т. д. Вуаля! Конечный результат – каждый второй элемент v.

См. также

См. рецепт 5.3 для получения дополнительной информации о правиле выравнивания.

12.8. Поиск минимумов или максимумов

Задача

У вас есть два вектора, v и w, и вы хотите найти минимумы или максимумы попарных элементов. То есть вы хотите рассчитать:

min(v1, w1), min(v2, w2), min(v3, w3), …

или:

max(v1, w1), max(v2, w2), max(v3, w3), …

Решение

R называет их *параллельным минимумом* и *параллельным максимумом*. Расчет выполняется pmin(v,w) и pmax(v,w) соответственно:

```
pmin(1:5, 5:1)        # Поиск поэлементного минимума.
#> [1] 1 2 3 2 1
pmax(1:5, 5:1)        # Поиск поэлементного максимума.
#> [1] 5 4 3 4 5
```

Обсуждение

Когда нужны попарные минимумы или максимумы, новички часто совершают ошибку, используя `min(v,w)` или `max(v,w)`. Это не парные операции: `min(v,w)` возвращает одно значение, минимум по всем `v` и `w`. Аналогично, `max(v,w)` возвращает одно значение из всех `v` и `w`.

Значения `pmin` и `pmax` сравнивают свои аргументы параллельно, выбирая минимум или максимум для каждого индекса. Они возвращают вектор, который соответствует длине входных данных.

Можно сочетать функции `pmin` и `pmax` с правилом повторного использования для выполнения полезных трюков.

Предположим, что вектор `v` содержит как положительные, так и отрицательные значения, и вы хотите переустановить отрицательные значения на ноль. Вот как это делается:

```
v <- c(-3:3)
v
#> [1] -3 -2 -1 0 1 2 3
v <- pmax(v, 0)
v
#> [1] 0 0 0 0 1 2 3
```

По правилу повторного использования R расширяет нулевой скаляр в вектор нулей, который имеет ту же длину, что и `v`. Затем функция `pmax` выполняет поэлементное сравнение, беря большее из нуля и каждого элемента из `v`.

На самом деле функции `pmin` и `pmax` более мощные, по сравнению с тем, что показано в решении. Они могут брать более двух векторов, сравнивая все векторы параллельно.

Нередко их используют для вычисления новой переменной в таблице данных на основе нескольких полей. Взглянем на этот пример:

```
df <- data.frame(a = c(1,5,8),
                 b = c(2,3,7),
                 c = c(0,4,9))

df %>%
mutate(max_val = pmax(a,b,c))
#>   a b c max_val
#> 1 1 2 0     2
#> 2 5 3 4     5
#> 3 8 7 9     9
```

Мы видим, что новый столбец `max_val` теперь содержит построчное максимальное значение из трех входных столбцов.

См. также

См. рецепт 5.3 для получения дополнительной информации о правиле выравнивания.

12.9. Генерация всех комбинаций нескольких переменных

Задача

У вас есть две или более переменных. Вы хотите сгенерировать все комбинации их уровней, также известные как их *декартово произведение*.

Решение

Используйте функцию `expand.grid`. В этом примере `f` и `g` – это векторы:

```
expand.grid(f, g)
```

Обсуждение

В данном фрагменте кода мы создаем два вектора: `sides` обозначает две стороны монеты, а `faces` – шесть граней игрального кубика (маленькие точки на кубике в английском языке называются *pips*):

```
sides <- c("Heads", "Tails")
faces <- c("1 pip", paste(2:6, "pips"))
```

Мы можем использовать функцию `expand.grid`, чтобы найти все комбинации одного броска кубика и одного броска монеты:

```
expand.grid(faces, sides)
#>      Var1  Var2
#> 1 1 pip   Heads
#> 2 2 pips  Heads
#> 3 3 pips  Heads
#> 4 4 pips  Heads
#> 5 5 pips  Heads
#> 6 6 pips  Heads
#> 7 1 pip   Tails
#> 8 2 pips  Tails
#> 9 3 pips  Tails
#> 104 pips  Tails
#> 115 pips  Tails
#> 126 pips  Tails
```

Точно так же можно было бы найти все комбинации из двух кубиков, но мы не будем здесь печатать этот вывод, поскольку он состоит из 36 строк:

```
expand.grid(faces, faces)
```

В результате мы получаем таблицу данных. R автоматически предоставляет имена строк и столбцов.

В решении и примере показано декартово произведение двух векторов, но функция `expand.grid` также может обрабатывать три или более факторов.

См. также

Если вы работаете со строками и вам нужно больше контроля над тем, как вы работаете с комбинациями, также можно использовать рецепт 7.6 для генерации комбинаций.

12.10. Преобразование таблицы данных

Задача

У вас есть таблица данных с числовыми значениями. Вы хотите обрабатывать все его элементы вместе, а не как отдельные столбцы – например, чтобы найти среднее значение для всех значений.

Решение

Преобразуйте таблицу данных в матрицу, а затем обработайте ее. В этом примере показано, как найти среднее значение всех элементов в таблице данных `dfrm`:

```
mean(as.matrix(dfrm))
```

Иногда необходимо преобразовать матрицу в вектор. В этом случае используйте `as.vector (as.matrix(dfrm))`.

Обсуждение

Предположим, у нас есть таблица данных, например данные о фабричном производстве из рецепта 12.3:

```
load(file = './data/daily.prod.rdata')
daily.prod
#>      Widgets  Gadgets  Thingys
#>  Mon    179      167      182
#>  Tue    153      193      166
#>  Wed    183      190      170
#>  Thu    153      161      171
#>  Fri    154      181      186
```

Предположим также, что мы хотим получить среднесуточное производство по всем дням и продуктам.

Этот вариант не сработает:

```
mean(daily.prod)
#> Warning in mean.default(daily.prod): argument is not numeric or logical:
#> returning NA
#> [1] NA
```

Функция `mean` не знает, что делать с таблицей данных, поэтому просто выдает ошибку. Если вы хотите получить среднее значение по всем значениям, сначала сверните таблицу данных до матрицы:

```
mean(as.matrix(daily.prod))
#> [1] 173
```

Данный рецепт подходит только для таблиц данных с полностью числовыми данными. Напомним, что преобразование таблицы данных со смешанными данными (числовые столбцы, смешанные с символьными столбцами или коэффициентами) в матрицу приводит к преобразованию всех столбцов в символы.

См. также

См. рецепт 5.29 для получения дополнительной информации о преобразовании между типами данных.

12.11. Сортировка таблицы данных

Задача

У вас есть таблица данных. Вы хотите отсортировать содержимое, используя один столбец в качестве ключа сортировки.

Решение

Используйте функцию `arrange` из пакета `dplyr`:

```
df <- arrange(df, key)
```

Здесь `df` – таблица данных, а `key` – столбец ключа сортировки.

Обсуждение

Функция `sort` отлично подходит для векторов, но неэффективна для таблиц данных. Предположим, у нас есть следующая таблица данных, и мы хотим отсортировать ее по месяцам:

```
load(file = './data/outcome.rdata')
print(df)
#>    month day outcome
#> 1    7   11    Win
#> 2    8   10    Lose
#> 3    8   25    Tie
#> 4    6   27    Tie
#> 5    7   22    Win
```

Функция `arrange` переставляет месяцы в порядке возрастания и возвращает всю таблицу данных:

```
library(dplyr)
arrange(df, month)
#>    month day outcome
#> 1    6   27    Tie
#> 2    7   11    Win
#> 3    7   22    Win
#> 4    8   10    Lose
#> 5    8   25    Tie
```

После перестановки таблицы данных столбец `month` находится в порядке возрастания, как мы и хотели. Если вы хотите отсортировать данные в порядке убывания, поместите перед столбцом, по которому вы хотите отсортировать данные, знак `-`:

```
arrange(df,-month)
#>    month day outcome
#> 1    8   10    Lose
#> 2    8   25    Tie
#> 3    7   11    Win
#> 4    7   22    Win
#> 5    6   27    Tie
```

Если вы хотите выполнить сортировку по нескольким столбцам, можно добавить их в функцию `arrange`.

В приведенном ниже примере мы выполняем сортировку сначала по месяцам, а затем по дням:

```
arrange(df, month, day)
#>   month day outcome
#> 1     6  27     Tie
#> 2     7  11     Win
#> 3     7  22     Win
#> 4     8  10    Lose
#> 5     8  25     Tie
```

В седьмом и восьмом месяцах дни теперь сортируются в порядке возрастания.

12.12. Удаление атрибутов из переменной

Задача

Переменная содержит старые атрибуты. Вы хотите удалить некоторые из них или их все.

Решение

Чтобы удалить все атрибуты, присвойте свойству переменной `attributes` значение `NULL`:

```
attributes(x) <- NULL
```

Чтобы удалить один атрибут, выберите его с помощью функции `attr` и установите для него значение `NULL`:

```
attr(x, "attributeName") <- NULL
```

Обсуждение

У любой переменной в R могут быть атрибуты. Атрибут – это просто пара «имя/значение», и у переменной их может быть много. Типичный пример – размеры матричной переменной, которые хранятся в атрибуте. Имя атрибута `dim`, а значением атрибута является двухэлементный вектор, задающий количество строк и столбцов.

Можно просмотреть атрибуты переменной `x` с помощью `attributes(x)` или `str(x)`.

Иногда вам нужно просто число, а R настаивает на присвоении ему атрибутов. Это может произойти, когда вы подгоняете простую линейную модель и извлекаете угловой коэффициент, который является вторым коэффициентом регрессии:

```
load(file = './data/conf.rdata')
m <- lm(y ~ x1)
slope <- coef(m)[2]
slope
#> x1
#> -11
```

Когда мы выводим `slope`, R также выводит `"x1"`. Это атрибут имени, который функция `lm` дала коэффициенту (потому что это коэффициент для переменной `x1`). Можно увидеть это более отчетливо, выведя внутреннюю часть угла наклона, который показывает атрибут `"names"`:

```
str(slope)
#> Named num -11
#> - attr(*, "names")= chr "x1"
```

Все атрибуты легко удалить, после чего значение наклона становится просто числом:

```
attributes(slope) <- NULL # Удаляем все атрибуты
str(slope) # Теперь атрибут "names" исчез
#> num -11

slope # И мы видим число без метки
#> [1] -11
```

В качестве альтернативы мы могли бы удалить единственный атрибут-нарушитель следующим образом:

```
attr(slope, "names") <- NULL
```

Помните, что матрица – это вектор (или список) с атрибутом `dim`. Если вы удалите все атрибуты из матрицы, размеры исчезнут, и она превратится в простой вектор (или список). Кроме того, удаление атрибутов из объекта (в частности, объект S3) может сделать его бесполезным. Поэтому удаляйте атрибуты осторожно.

См. также

См. рецепт 12.13, чтобы узнать больше об атрибутах.

12.13. Раскрываем структуру объекта

Задача

Вы вызвали функцию, которая что-то вернула. Теперь вы хотите заглянуть внутрь того, что она вернула, и узнать о нем больше.

Решение

Используйте функцию `class`, чтобы определить класс объекта вещи:

```
class(x)
```

Используйте режим, чтобы убрать объектно-ориентированные свойства и раскрыть основную структуру:

```
mode(x)
```

Используйте функцию `str`, чтобы показать внутреннюю структуру и содержание:

```
str(x)
```

Обсуждение

Мы регулярно удивляемся тому, как часто вызываем функцию и получаем что-то обратно: «Что это, черт возьми, такое?» Теоретически документация функции должна объяснять возвращаемое значение, но каким-то образом мы чувствуем

себя лучше, когда сами видим его структуру и содержимое. Это особенно актуально для объектов с вложенной структурой: объекты внутри объектов.

Давайте разберем значение, возвращаемое функцией `lm` (функцией линейного моделирования) в простейшем рецепте линейной регрессии, рецепте 11.1:

```
load(file = './data/conf.rdata')
m <- lm(y ~ x1)
print(m)
#>
#> Call:
#> lm(formula = y ~ x1)
#>
#> Coefficients:
#> (Intercept)      x1
#>        15.9   -11.0
```

Всегда начинайте с проверки класса. Класс указывает на то, что это такое: вектор, матрица, список, таблица данных или объект:

```
class(m)
#> [1] "lm"
```

Гм. Кажется, `m` – это объект класса `lm`. Это может ничего не значить для вас, но мы знаем, что все классы объектов построены на нативных структурах данных (вектор, матрица, список или таблица данных). Можно использовать функцию `mode`, чтобы удалить фасад объекта и раскрыть его базовую структуру:

```
mode(m)
#> [1] "list"
```

Ага! Кажется, `m` построен на структуре списка. Теперь мы можем использовать функции и операторы списка, чтобы покопаться в его содержимом. Сначала нам нужно узнать имена его элементов списка:

```
names(m)
#>  [1] "coefficients"  "residuals"  "effects"  "rank"
#>  [5] "fitted.values" "assign"     "qr"       "df.residual"
#>  [9] "xlevels"       "call"       "terms"    "model"
```

Первый элемент списка называется `"coefficients"`. Можно было бы предположить, что это коэффициенты регрессии. Давайте посмотрим:

```
m$coefficients
#> (Intercept)      x1
#>        15.9   -11.0
```

Да, именно они и есть. Мы узнаем эти значения.

Можно бы продолжить копаться в структуре списка `m`, но это было бы утомительно. Функция `str` хорошо показывает внутреннюю структуру любой переменной:

```
str(m)
#> List of 12
#> $ coefficients : Named num [1:2] 15.9 -11
#> ..- attr(*, "names")= chr [1:2] "(Intercept)" "x1"
#> $ residuals : Named num [1:30] 36.6 58.6 112.1 -35.2 -61.7 ...
#> ..- attr(*, "names")= chr [1:30] "1" "2" "3" "4" ...
```

```
#> $ effects : Named num [1:30] -73.1 69.3 93.9 -31.1 -66.3 ...
#> ..- attr(*, "names")= chr [1:30] "(Intercept)" "x1" "" "" ...
#> $ rank : int 2
#> $ fitted.values: Named num [1:30] 25.69 13.83 -1.55 28.25 16.74 ...
#> ..- attr(*, "names")= chr [1:30] "1" "2" "3" "4" ...
#> $ assign : int [1:2] 0 1
#> $ qr :List of 5
#> ..$ qr : num [1:30, 1:2] -5.477 0.183 0.183 0.183 0.183 ...
#> .. ..- attr(*, "dimnames")=List of 2
#> .. .. ..$ : chr [1:30] "1" "2" "3" "4" ...
#> .. .. ..$ : chr [1:2] "(Intercept)" "x1"
#> .. ..- attr(*, "assign")= int [1:2] 0 1
#> ..$ qraux: num [1:2] 1.18 1.02
#> ..$ pivot: int [1:2] 1 2
#> ..$ tol : num 1e-07
#> ..$ rank : int 2
#> ..- attr(*, "class")= chr "qr"
#> $ df.residual : int 28
#> $ xlevels : Named list()
#> $ call : language lm(formula = y ~ x1)
#> $ terms :Classes 'terms', 'formula' language y ~ x1
#> .. ..- attr(*, "variables")= language list(y, x1)
#> .. ..- attr(*, "factors")= int [1:2, 1] 0 1
#> .. .. ..- attr(*, "dimnames")=List of 2
#> .. .. .. ..$ : chr [1:2] "y" "x1"
#> .. .. .. ..$ : chr "x1"
#> .. ..- attr(*, "term.labels")= chr "x1"
#> .. ..- attr(*, "order")= int 1
#> .. ..- attr(*, "intercept")= int 1
#> .. ..- attr(*, "response")= int 1
#> .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
#> .. ..- attr(*, "predvars")= language list(y, x1)
#> .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
#> .. .. ..- attr(*, "names")= chr [1:2] "y" "x1"
#> $ model :'data.frame': 30 obs. of 2 variables:
#> ..$ y : num [1:30] 62.25 72.45 110.59 -6.94 -44.99 ...
#> ..$ x1: num [1:30] -0.8969 0.1848 1.5878 -1.1304 -0.0803 ...
#> ..- attr(*, "terms")=Classes 'terms', 'formula' language y ~ x1
#> .. .. ..- attr(*, "variables")= language list(y, x1)
#> .. .. ..- attr(*, "factors")= int [1:2, 1] 0 1
#> .. .. .. ..- attr(*, "dimnames")=List of 2
#> .. .. .. .. ..$ : chr [1:2] "y" "x1"
#> .. .. .. .. ..$ : chr "x1"
#> .. .. ..- attr(*, "term.labels")= chr "x1"
#> .. .. ..- attr(*, "order")= int 1
#> .. .. ..- attr(*, "intercept")= int 1
#> .. .. ..- attr(*, "response")= int 1
#> .. .. ..- attr(*, ".Environment")=<environment: R_GlobalEnv>
#> .. .. ..- attr(*, "predvars")= language list(y, x1)
#> .. .. ..- attr(*, "dataClasses")= Named chr [1:2] "numeric" "numeric"
#> .. .. .. ..- attr(*, "names")= chr [1:2] "y" "x1"
#> - attr(*, "class")= chr "lm"
```

Обратите внимание, что эта функция показывает все элементы `m`, а затем рекурсивно выводит содержимое и атрибуты каждого элемента. Длинные векторы и списки усекаются, чтобы выводом было легче управлять.

Изучение объекта в R – это искусство. Используйте функции `class`, `mode` и `str`, чтобы копаться во всём этом. Мы обнаружили, что часто функция `str` рассказывает все, что вы хотите знать… а иногда и многое другое!

12.14. Определяем время выполнения кода

Задача

Вы хотите знать, сколько времени требуется для запуска вашего кода. Это полезно, например, когда вы оптимизируете его и вам нужны цифры «до» и «после», чтобы оценить улучшения.

Решение

Пакет `tictoc` предоставляет очень простой способ определения времени выполнения и обозначения его фрагментов. Функция `tic` запускает таймер, а функция `toc` останавливает его и сообщает время выполнения:

```
library(tictoc)
tic('Optional helpful name here')
aLongRunningExpression()
toc()
```

Вывод представляет собой время выполнения в секундах.

Обсуждение

Предположим, мы хотим знать время, необходимое для того, чтобы сгенерировать 10 000 000 случайных нормальных чисел и сложить их:

```
library(tictoc)
tic('making big numbers')
total_val <- sum(rnorm(1e7))
toc()
#> making big numbers: 0.794 sec elapsed
```

Функция `toc` возвращает сообщение, установленное в `tic`, вместе со временем выполнения в секундах.

Если вы назначите результат `toc` объекту, то можно получить доступ к базовому времени начала, времени окончания и сообщению:

```
tic('two sums')
sum(rnorm(10000000))
#> [1] -84.1
sum(rnorm(10000000))
#> [1] -3899
toc_result <- toc()
#> two sums: 1.373 sec elapsed

print(toc_result)
#> $tic
#> elapsed
#> 2.64
#>
#> $toc
#> elapsed
```

```
#> 4.01
#>
#> $msg
#> [1] "two sums"
```

Если вам нужны результаты в минутах (или часах!), можно использовать элементы вывода на основном начальном и конечном времени:

```
print(paste('the code ran in',
            round((toc_result$toc - toc_result$tic) / 60, 4),
            'minutes'))
#> [1] "the code ran in 0.0229 minutes"
```

Вы можете сделать то же самое, используя только вызовы функции `Sys.time`, но вы будете лишены удобства обозначений и ясности синтаксиса, предоставляемых пакетом `tictoc`:

```
start <- Sys.time()
sum(rnorm(10000000))
#> [1] 3607
sum(rnorm(10000000))
#> [1] 1893
Sys.time() - start
#> Time difference of 1.37 secs
```

12.15. Избавляемся от предупреждений и сообщений об ошибках

Задача

Функция создает раздражающие сообщения об ошибках или предупреждения. Вы не хотите их видеть.

Решение

Окружите вызов функции с помощью `suppressMessage(…)` или `suppressWarnings(…)`:

```
suppressMessage(annoyingFunction())
suppressWarnings(annoyingFunction())
```

Обсуждение

Расширенный тест Дики–Фуллера, `adf.test`, представляет собой популярную функцию временных рядов. Тем не менее он генерирует раздражающее предупреждение, показанное здесь внизу вывода, если *p*-значение ниже 0,01:

```
library(tseries)
load(file = './data/adf.rdata')
results <- adf.test(x)
#> Warning in adf.test(x): p-value smaller than printed p-value
```

К счастью, мы можем приструнить функцию, вызвав ее внутри `suppressWarnings(…)`:

```
results <- suppressWarnings(adf.test(x))
```

Обратите внимание, что предупреждение исчезло. Сообщение потеряно не полностью, потому что R сохраняет его внутри. Мы можем получить сообщение, когда нам будет угодно, используя функцию `warnings`:

```
warnings()
```

Некоторые функции также создают «сообщения» (в терминологии R), которые даже более безопасны, чем предупреждения. Как правило, это чисто информативные сообщения, а не сигналы о проблемах. Если такое сообщение вас раздражает, можно заставить его исчезнуть, вызвав функцию внутри `suppressMessages(...)`.

См. также

Изучите функцию `options`, которая предоставляет другие способы управления отчетами об ошибках и предупреждениях.

12.16. Извлечение аргументов функции из списка

Задача

Ваши данные находятся в структуре списка. Вы хотите передать их в функцию, но функция не принимает список.

Решение

В простых случаях преобразуйте список в вектор. В более сложных случаях функция `do.call` может разбить список на отдельные аргументы и вызвать вашу функцию:

```
do.call(function, list)
```

Обсуждение

Если ваши данные в векторе, жизнь проста, и большинство функций R работают, как положено:

```
vec <- c(1, 3, 5, 7, 9)
mean(vec)
#> [1] 5
```

Если ваши данные заносятся в список, некоторые функции жалуются и возвращают бесполезный результат, например такой:

```
numbers <- list(1, 3, 5, 7, 9)
mean(numbers)
#> Warning in mean.default(numbers): argument is not numeric or logical:
#> returning NA
#> [1] NA
```

Список чисел – это простой одноуровневый список, поэтому мы можем просто преобразовать его в вектор и вызвать функцию:

```
mean(unlist(numbers))
#> [1] 5
```

Настоящая головная боль возникает, когда у вас есть структуры многоуровневых списков: списки в списках. Они могут встречаться в сложных структурах

данных. Вот список списков, в котором каждый подсписок представляет собой столбец данных:

```
my_lists <-
 list(col1 = list(7, 8),
      col2 = list(70, 80),
      col3 = list(700, 800))
my_lists
#> $col1
#> $col1[[1]]
#> [1] 7
#>
#> $col1[[2]]
#> [1] 8
#>
#>
#> $col2
#> $col2[[1]]
#> [1] 70
#>
#> $col2[[2]]
#> [1] 80
#>
#>
#> $col3
#> $col3[[1]]
#> [1] 700
#>
#> $col3[[2]]
#> [1] 800
```

Предположим, мы хотим сформировать эти данные в матрицу. Предполагается, что функция `cbind` создает столбцы данных, но структура списка сбивает ее с толку, и она возвращает что-то бесполезное:

```
cbind(my_lists)
#>      my_lists
#> col1 List,2
#> col2 List,2
#> col3 List,2
```

Если мы выводим данные из списка, то просто получаем один большой, длинный столбец, который нам не нужен:

```
cbind(unlist(my_lists))
#>        [,1]
#> col11 7
#> col12 8
#> col21 70
#> col22 80
#> col31 700
#> col32 800
```

Решение – использовать функцию `do.call`, которая разбивает список на отдельные элементы, а затем вызывает для них функцию `cbind`:

```
do.call(cbind, my_lists)
#> col1 col2 col3
```

```
#> [1,] 7 70 700
#> [2,] 8 80 800
```

Использование функции `do.call` таким образом функционально идентично вызову функции `cbind`:

```
cbind(my_lists[[1]], my_lists[[2]], my_lists[[3]])
#>      [,1]  [,2]  [,3]
#> [1,]  7    70    700
#> [2,]  8    80    800
```

Будьте осторожны, если у элементов списка есть имена. В этом случае функция `do.call` интерпретирует их как имена параметров функции, что может вызвать проблемы.

В этом рецепте мы показали самое базовое использование функции `do.call`. Это довольно мощная функция, и у нее есть множество других применений. *См.* страницу справки для получения более подробной информации.

См. также

См. рецепт 5.29, чтобы увидеть, как выполнять преобразования между типами данных.

12.17. Определение собственных бинарных операторов

Задача

Вы хотите определить собственные бинарные операторы, чтобы сделать код более удобным и читабельным.

Решение

R распознает любой текст между знаками процента (%...%) как бинарный оператор. Создайте и определите новый бинарный оператор, назначив ему функцию с двумя аргументами.

Обсуждение

У R есть интересная особенность, которая позволяет вам определять собственные бинарные операторы. Любой текст между двумя знаками процента (%...%) автоматически интерпретируется R как бинарный оператор. R предопределяет несколько таких операторов, например `%/%` для целочисленного деления, `%*%` для перемножения матриц и конвейер `%>%` из пакета `magrittr`.

Вы можете создать новый бинарный оператор, назначив ему функцию. В этом примере мы создаем оператор `%+-%`:

```
'%+-%' <- function(x, margin)
x + c(-1, +1) * margin
```

Выражение `x % + -% m` вычисляет `x ± m`. Здесь оно вычисляет 100 ± (1,96 × 15), диапазон двух стандартных отклонений стандартного теста IQ:

```
100 %+-% (1.96 * 15)
#> [1] 70.6 129.4
```

Обратите внимание, что мы берем двоичный оператор в кавычки, когда определяем его, но не тогда, когда используем.

Удовольствие от определения собственных операторов заключается в том, что вы можете заключать часто используемые операции в сжатый синтаксис. Если ваше приложение нередко объединяет две строки без промежуточного пробела, можно определить для этой цели бинарный оператор конкатенации:

```
'%+%' <- function(s1, s2)
 paste(s1, s2, sep = "")
"Hello" %+% "World"
#> [1] "HelloWorld"
"limit=" %+% round(qnorm(1 - 0.05 / 2), 2)
#> [1] "limit=1.96"
```

Однако опасность определения собственных операторов заключается в том, что код становится менее переносимым на другие среды. Храните определения вместе с кодом, в котором они используются; в противном случае R будет жаловаться на неопределенные операторы.

Все пользовательские операторы имеют одинаковый приоритет и перечислены в табл. 2-1 как `%any%`. Их приоритет довольно высок: выше, чем умножение и деление, но ниже, чем возведение в степень и создание последовательности. В результате легко выразить себя неправильно. Если мы опустим скобки в предыдущем примере, то получим неожиданный результат:

```
100 %+-% 1.96 * 15
#> [1] 1471 1529
```

R интерпретировал это выражение как (100% + -% 1,96) * 15.

См. также

См. рецепт 2.11 для получения дополнительной информации о приоритетах операторов и рецепт 15.3, чтобы узнать, как определить функцию.

12.18. Избавляемся от сообщения о запуске

Задача

Вы запускаете R из командной строки или сценария оболочки, и вам надоело видеть подробное сообщение о запуске.

Решение

Используйте параметр командной строки `--quiet` при запуске R из командной строки или сценария оболочки.

Обсуждение

Сообщение о запуске удобно для новичков, потому что оно содержит полезную информацию о проекте R и получении помощи. Но эта новизна довольно быстро надоедает, особенно если вы запускаете R из командной строки, чтобы использо-

вать его в качестве калькулятора в течение дня. Это не особенно полезно, если вы используете R только из RStudio.

Если вы запускаете R из командной строки, используйте параметр `--quiet`, чтобы скрыть сообщение о запуске:

```
R -quiet
```

В Linux или Mac можно использовать псевдоним R в командной строке, чтобы не видеть сообщение о запуске:

```
alias R="/usr/bin/R --quiet"
```

12.19. Получение и настройка переменных среды

Задача

Вы хотите увидеть значение переменной окружения или изменить ее значение.

Решение

Используйте функцию `Sys.getenv`, чтобы увидеть значения, и функцию `Sys.putenv`, чтобы изменить их:

```
Sys.setenv(DB_PASSWORD = "My_Password!")
Sys.getenv("DB_PASSWORD")
#> [1] "My_Password!"
```

Обсуждение

Переменные среды часто используются для настройки и управления программным обеспечением. У каждого процесса есть собственный набор переменных среды, которые наследуются от его родительского процесса.

Иногда вам нужно увидеть настройки переменных среды для своего процесса в R, чтобы понять его поведение. Кроме того, иногда вам нужно изменить эти настройки, чтобы изменить это поведение.

Распространенным вариантом использования является сохранение имени пользователя или пароля, чтобы использовать их при обращении к удаленной базе данных или облачной службе. Хранить пароли в виде простого текста в сценарии проекта – это действительно плохая идея. Один из способов избежать этого – установить переменную среды, содержащую ваш пароль при запуске R.

Чтобы ваш пароль и имя пользователя были доступны при каждом входе, можно добавить вызовы функции `Sys.setenv` в файле *.Rprofile* в вашем домашнем каталоге. *.Rprofile* – это скрипт R, который запускается каждый раз при запуске R.

Например, вы можете добавить в свой файл *.Rprofile* следующее:

```
Sys.setenv(DB_USERID = "Me")
Sys.setenv(DB_PASSWORD = "My_Password!")
```

Затем вы можете получить и использовать переменные среды в сценарии для входа в базу данных Amazon Redshift, например:

```
con <- DBI::dbConnect(
  RPostgreSQL::PostgreSQL(),
  dbname = "my_database",
  port = 5439,
```

```
  host = "my_database.amazonaws.com",
  user = Sys.getenv("DB_USERID"),
  password = Sys.getenv("DB_PASSWORD")
)
```

См. также

См. рецепт 3.16 для получения дополнительной информации об изменении конфигурации при запуске.

12.20. Разбиение кода на секции

Задача

У вас длинный сценарий, и вам трудно переходить от одного раздела кода к другому.

Решение

Разделы кода предоставляют разделители разделов в панели **Список** (Outline) на боковой панели вашего редактора.

Чтобы использовать разделы кода, просто начните комментарий со знака #, а затем завершите его с помощью ----, или ####, или ====:

```
# My First Section -----
x <- 1

# My Second Section ####
y <- 2

# My Third Section ====
z <- 3
```

В окне редактора RStudio видны разделы кода (*см.* рис. 12-2).

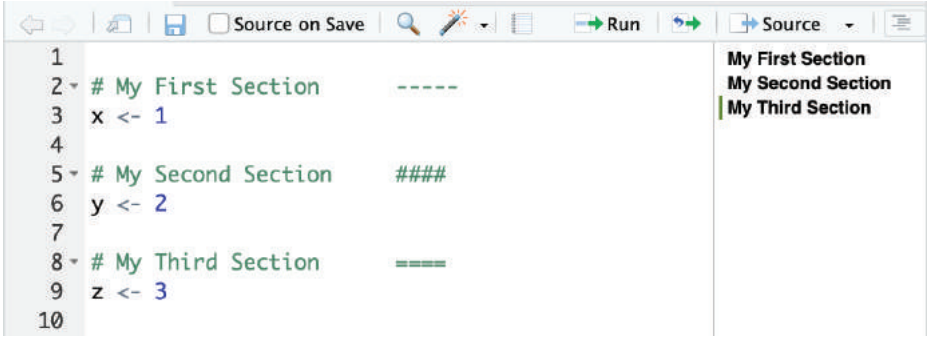


Рис. 12-2. Разделы кода

Обсуждение

Разделы кода – это просто специально отформатированный тип комментария R, поскольку они начинаются с символа #. Если вы откроете свой код в любом другом редакторе, кроме RStudio, они будут рассматриваться просто как комментарии к коду.

Но RStudio рассматривает эти специально отформатированные комментарии кода как заголовки разделов и создает полезную схему в боковой панели редактора.

При первом использовании разделов кода может потребоваться щелкнуть значок **Outline** справа от кнопки **Source** (Исходный код), чтобы отобразить его.

Если вы используете R Markdown вместо скрипта *.*R*, ваши заголовки и подзаголовки будут отображаться на панели структуры, что значительно облегчит навигацию по документу.

См. также

См. рецепт 16.4 для использования заголовков разделов в документах R Markdown.

12.21. Локальная параллелизация выполнения кода

Задача

У вас есть код, для запуска которого требуется время. Вы хотели бы ускорить его, используя больше ядер на вашем локальном компьютере.

Решение

Самое простое решение, чтобы начать работу, – использовать пакет `furrr`, который, в свою очередь, применяет пакет `future`, чтобы обеспечить параллельную обработку с помощью функций, аналогичных функциям `purrr`, за исключением того, что они работают параллельно.

Вам понадобится скачать последнюю версию с сайта GitHub, потому что пакет все еще находится в активной разработке на момент написания этого раздела:

```
devtools::install_github("DavisVaughan/furrr")
```

Чтобы использовать `furrr` для распараллеливания нашего кода, мы вызываем функцию `furrr::future_map` вместо функции `purrr::map`, которую обсуждали в рецепте 6.1. Но сначала мы должны сообщить `furrr`, как хотим распараллелить код. В этом случае нам нужен параллельный процесс `multiprocess`, который использует все наши локальные процессоры, поэтому мы устанавливаем его с помощью вызова `plan(multiprocess)`. Затем мы можем применить функцию к каждому элементу в нашем списке, используя функцию `future_map`:

```
library(furrr)

plan(multiprocess)

future_map(my_list, some_function)
```

Обсуждение

Давайте выполним пример моделирования, чтобы проиллюстрировать распараллеливание. Классическое стохастическое моделирование состоит в том, чтобы рисовать случайные точки внутри блока 2×2 и видеть, сколько точек выпадает

в одном блоке от центра блока. Соотношение точек внутри коробки / общее количество точек, умноженное на 4, является хорошей оценкой числа пи. Приведенная ниже функция принимает один вход, `n_iterations`, который представляет собой число случайных точек для моделирования. Затем она возвращает итоговую среднюю оценку числа пи:

```
simulate_pi <- function(n_iterations) {
  rand_draws <- matrix(runif(2 * n_iterations, -1, 1), ncol = 2)
  num_in <- sum(sqrt(rand_draws[, 1]**2 + rand_draws[, 2]**2) <= 1)
  pi_hat <- (num_in / n_iterations) * 4
  return(pi_hat)
}
simulate_pi(1000000)
#> [1] 3.14
```

Как видно, даже при 1 000 000 моделирований результат будет точным с точностью до пары десятичных знаков. Это не очень эффективный способ оценки числа пи, но для нашей иллюстрации он подходит.

Чтобы выполнить сравнения позже, давайте запустим 200 прогонов этого симулятора пи, где каждый прогон имеет 250 000 смоделированных точек. Мы будем делать это, создав список из 200 элементов, каждый из которых имеет значение 2 500 000, который передадим в `simulate_pi`. Мы рассчитаем код с помощью пакета `tictoc`:

```
library(purrr) # for 'map'
library(tictoc) # for timing our code

draw_list <- as.list(rep(5000000, 200))

tic("simulate pi - single process")
sims_list <- map(draw_list, simulate_pi)
toc()
#> simulate pi - single process: 90.772 sec elapsed

mean(unlist(sims_list))
#> [1] 3.14
```

На это требуется менее чем две минуты и дает оценку числа пи на основе миллиарда симуляций (5m × 200).

Теперь давайте возьмем точно такой же процесс, но запустим его с помощью функции `future_map`, чтобы запустить его параллельно:

```
library(furrr)
#> Loading required package: future
#>
#> Attaching package: 'future'
#> The following object is masked from 'package:tseries':
#>
#> value
plan(multiprocess)

tic("simulate pi - parallel")
sims_list <- future_map(draw_list, simulate_pi)
toc()
#> simulate pi - parallel: 26.33 sec elapsed
mean(unlist(sims_list))
#> [1] 3.14
```

Для запуска предыдущего примера использовался ноутбук MacBook Pro с четырьмя физическими и двумя виртуальными ядрами на физическое ядро. Когда вы выполняете код параллельно, идеальный вариант – когда время выполнения сокращается на 1 / (количество физических ядер). Когда у вас четыре физических ядра, вы видите, что параллельная среда выполнения намного быстрее, по сравнению с однопоточной версией, но это не совсем одна четверть времени выполнения однопоточной версии.

Перемещение данных всегда сопряжено с некоторыми затратами, поэтому вы никогда не столкнетесь с идеальным вариантом. И чем больше данных выдает каждая итерация, тем меньше скорости вы получите в ходе распараллеливания.

См. также

См. рецепт 12.22.

12.22. Удалённая параллелизация выполнения кода

Задача

У вас есть доступ к нескольким удаленным компьютерам, и вы хотели бы параллельно запускать свой код на них.

Решение

Параллельно запускать код на нескольких компьютерах поначалу может быть непросто.

Однако если мы начнем с нескольких ключевых предпосылок, вероятность успеха процесса будет значительно выше.

Начальные предпосылки:

- вы можете подключаться к каждому удаленному узлу по протоколу ssh со своего главного компьютера без пароля, используя ранее сгенерированные ключи SSH;
- на всех удаленных узлах установлен R (в идеале – одна и та же версия R);
- пути установлены так, что вы можете запустить `Rscript` из SSH;
- на удаленных узлах установлен пакет `furrr` (который, в свою очередь, устанавливает пакет `future`);
- на удаленных узлах уже установлены все пакеты, от которых зависит ваш распределенный код.

Как только у вас появятся рабочие узлы, которые настроены и готовы к работе, вы можете создать кластер, вызвав функцию `makeClusterPSOCK` из пакета `future`. Затем используйте полученный кластер с функцией `future_map` из пакета `furrr`:

```
library(furrr) # Загружаем пакет future в качестве зависимости.

workers <- c("node_1.domain.com", "node_2.domain.com")

cl <- makeClusterPSOCK(
worker = workers
)

plan(cluster, workers = cl)

future_map(my_list, some_function)
```

Обсуждение

Предположим, у нас есть две большие машины Linux с именами `von-neumann12` и `von-neumann15`, которые можно использовать для запуска численных моделей. Эти машины соответствуют только что перечисленным критериям, поэтому они являются хорошими кандидатами на роль нашего бэкенда для кластера `furrr/future`. Давайте выполним ту же симуляцию, что и в предыдущем рецепте, используя функцию `simulate_pi`:

```
library(tidyverse)
library(furrr)
library(tictoc)

my_workers <- c('von-neumann12','von-neumann15')

cl <- makeClusterPSOCK(
 workers = my_workers,
 rscript = '/home/anaconda2/bin/Rscript', # У вас это может выглядеть иначе.
 verbose=TRUE
)

draw_list <- as.list(rep(5000000, 200))

plan(cluster, workers = cl)

tic('simulate pi - parallel map')
sims_list_parallel <- draw_list %>%
 future_map(simulate_pi)
toc()
#> simulate pi - parallel map: 116.986 sec elapsed

mean(unlist(sims_list_parallel))
#> [1] 3.14167
```

Это ~ 8,5 млн симуляций в секунду.

Каждый из двух узлов в нашем специальном кластере имеет 32 процессора и 128 ГБ оперативной памяти.

Но если вы сравните время выполнения этого кода со временем выполнения кода из предыдущего рецепта на скромном MacBook Pro, то заметите, что на MacBook код был выполнен примерно за то же время, что и на многопроцессорном кластере Linux с 64 процессорами!

Этот неожиданный сюрприз вызван тем, что предыдущий код выполняется только на одном ЦП на каждый узел кластера. Таким образом, в результате он использует лишь два процессора, в то время как MacBook использует все четыре своих процессора.

Итак, как же запустить параллельный код в кластере и заставить каждый узел тоже работать параллельно на нескольких ядрах процессора? Для этого нам нужно внести три изменения в наш код:

1) создайте вложенный вызов функции параллелизации `plan`, который использует и `cluster`, и `multiprocess`;
2) создайте входной список, который является вложенным списком. Каждый кластерный компьютер получит из основного списка элемент, содержащий

элементы подсписка, которые он может обрабатывать параллельно на всех своих процессорах;

3) вызовите функцию future_map дважды, используя вложенный вызов. Внешний вызов future_map будет распараллеливать элементы по узлам кластера, а затем внутренний вызов выполнит распараллеливание между процессорами.

Чтобы создать вложенный вызов функции параллелизации plan, мы создадим многоэлементный план, передав список из двух планов функции plan:

```
plan(list(tweak(cluster, workers = cl), multiprocess))
```

Второе изменение заключается в создании вложенного списка для итерации. Это можно сделать, используя команду split и передав ей наш предыдущий список, за которым следует вектор 1:4:

```
split(draw_list, 1:4)
```

Так мы разобьем первоначальный список на четыре подсписка, поэтому в нашем получившемся списке будет четыре элемента.

Каждый подсписок будет иметь 50 входов для нашей последней функции simulate_pi.

Третье изменение в нашем коде заключается в создании вложенного вызова функции future_map, который будет передавать каждый из наших четырех элементов списка рабочим узлам, а они впоследствии будут перебирать элементы каждого подсписка. Создать эту вложенную функцию можно следующим образом:

```
future_map(draw_list, ~future_map(.,simulate_pi))
```

Знак ~ устанавливает R для ожидания анонимной функции в первом вызове future_map, а также дает R указание, куда поместить элемент списка. Анонимная функция в этом примере – это отдельный вызов future_map, который выполняется на каждом узле.

Вот все три изменения, интегрированных в код:

```
# вложенный вызов функции параллелизации plan - первая часть плана - это вызов кластера, за
которым следует мультипроцесс.
plan(list(tweak(cluster, workers = cl), multiprocess))

# Разбиваем draw_list на вложенный список с меньшим количеством элементов.
draw_list_nested <- split(draw_list, 1:4)

tic('simulate pi - parallel nested map')
sims_list_nested_parallel <- future_map(
 draw_list_nested, ~future_map(.,simulate_pi)
)
toc()
#> simulate pi - parallel nested map: 15.964 sec elapsed
mean(unlist(sims_list_nested_parallel))
#> [1] 3.14158
```

Видно, что время выполнения существенно уменьшилось по сравнению с предыдущим примером, хотя при наличии 32 процессоров на каждом узле мы не наблюдаем 32-кратного улучшения времени выполнения. Это связано с тем, что мы передаем только 50 наборов симуляций каждому узлу. Каждый узел выполняет 32 набора симуляций на первом проходе, но только 18 на втором проходе, оставляя половину процессоров бездействующими.

Давайте немного оживим процессоры, увеличив общее количество симуляций с 1 до 25 миллиардов. Затем мы разделим их на 500 рабочих блоков, которые будут распределены по двум рабочим узлам:

```
draw_list <- as.list(rep(5000000, 5000))
draw_list_nested <- split(draw_list, 1:50)

plan(list(tweak(cluster, workers = cl), multiprocess))

tic('simulate pi - parallel nested map')
sims_list_nested_parallel <- future_map(
 draw_list_nested, ~future_map(.,simulate_pi)
)
toc()
#> simulate pi - parallel nested map: 260.532 sec elapsed
mean(unlist(sims_list_nested_parallel))
#> [1] 3.14157
```

Это дает нам ~ 96 млн симуляций секунду.

См. также

В пакете `future` есть много хорошей сопроводительной документации. Чтобы лучше понять вложенный вызов функции `plan`, начните с `vignette('future-3-topologies',package = 'future')`.

Дополнительную информацию о пакете `furrr` можно найти на его странице на сайте GitHub.

Глава 13

За пределами основных цифр и статистики

В этой главе представлено несколько продвинутых методов, таких как те, с которыми вы можете столкнуться на первом или втором курсе аспирантуры по прикладной статистике.

В большинстве этих рецептов используются функции, доступные в базовом дистрибутиве. Посредством пакетов дополнений R предоставляет некоторые самые передовые статистические методы в мире.

Это связано с тем, что исследователи в области статистики теперь используют R в качестве языка общения, демонстрируя свои новейшие работы. Любой, кто ищет передовые статистические методы, должен воспользоваться CRAN и интернетом для поиска возможных реализаций.

13.1. Минимизация или максимизация однопараметрической функции

Задача

Имеется однопараметрическая функция `f`. Вы хотите найти точку, в которой эта функция достигает своего минимума или максимума.

Решение

Чтобы минимизировать однопараметрическую функцию, используйте функцию `optimize`. Укажите минимизируемую функцию и границы ее области (`x`):

```
optimize(f, lower = lowerBound, upper = upperBound)
```

Если вы хотите максимизировать функцию, укажите для `maximum` значение `TRUE`:

```
optimize(f,
         lower = lowerBound,
        upper = upperBound,
        maximum = TRUE)
```

Обсуждение

Функция `optimize` может обрабатывать функции из одного аргумента. Требуется верхняя и нижняя границы для `x`, которые разграничивают область поиска. В приведенном ниже примере мы находим минимум полинома, $3x^4 - 2x^3 + 3x^2 - 4x + 5$:

```
f <- function(x)
  3 * x ^ 4 - 2 * x ^ 3 + 3 * x ^ 2 - 4 * x + 5
optimize(f, lower = -20, upper = 20)
#> $minimum
#> [1] 0.597
#>
#> $objective
#> [1] 3.64
```

Возвращаемое значение представляет собой список с двумя элементами: `minimum`, значение `x`, которое минимизирует функцию; и `objective`, значение функции в этой точке.

Более узкий диапазон для `lower` и `upper` означает меньшую область поиска и, следовательно, более быструю оптимизацию. Однако если вы не уверены в соответствующих границах, используйте большие, но разумные значения, такие как lower = –1000 и upper = 1000. Только будьте осторожны, ваша функция не должна иметь нескольких минимумов в этом диапазоне! Функция `optimize` найдет и вернет только один такой минимум.

См. также

См. рецепт 13.2.

13.2. Минимизация или максимизация многопараметрической функции

Задача

Имеется многопараметрическая функция `f`. Вы хотите найти точку, в которой эта функция достигает своего минимума или максимума.

Решение

Чтобы минимизировать многопараметрическую функцию, используйте функцию `optim`. Вы должны указать начальную точку, которая является вектором начальных аргументов для `f`:

```
optim(startingPoint, f)
```

Чтобы, наоборот, максимизировать функцию, укажите этот параметр `control`:

```
optim(startingPoint, f, control = list(fnscale = -1))
```

Обсуждение

Функция `optim` является более общей по сравнению с функцией `optimize` (*см.* рецепт 13.1), потому что она обрабатывает многопараметрические функции. Чтобы вычислить вашу функцию в некой точке, `optim` упаковывает координаты точки

в вектор и вызывает вашу функцию с этим вектором. Функция должна возвращать скалярное значение. Функция optim начинает в вашей начальной точке, затем перемещается в пространстве параметров, разыскивая минимум функции.

Ниже приводится пример использования функции `optim` для подгонки нелинейной модели. Предположим, вы считаете, что зависимые наблюдения `z` и `x` связаны соотношением $z_i = (x_i + \alpha)^{\beta} + \varepsilon_i$, где α и β – неизвестные параметры, а ε_i – ненормальные шумовые составляющие. Давайте подгоним модель, минимизировав устойчивую метрику, сумму абсолютных отклонений:

$$\Sigma|z - (x + a)^b|.$$

Сначала мы определяем функцию, которая будет минимизирована. Обратите внимание, что функция имеет только один формальный параметр, двухэлементный вектор. Фактические параметры, которые должны быть рассчитаны, `a` и `b`, упакованы в вектор в местоположениях 1 и 2:

```
load(file = './data/opt.rdata') # Загружаем x, y, z.

f <-
 function(v) {
 a <- v[1]
 b <- v[2]                    # "Распаковка" v, которая дает нам a и b.
sum(abs(z - ((x + a) ^ b)))   # Выполняем расчет и возвращаем ошибку.
}
```

В этом коде мы выполняем вызов функции `optim`, начиная с (1, 1), и ищем минимальную точку `f`:

```
optim(c(1, 1), f)
#> $par
#> [1] 10.0 0.7
#>
#> $value
#> [1] 1.26
#>
#> $counts
#> function gradient
#>      485       NA
#>
#> $convergence
#> [1] 0
#>
#> $message
#> NULL
```

Возвращаемый список включает в себя `convergence`, показатель успеха или неудачи. Если этот показатель равен 0, это значит, что функция `optim` нашла минимум; в противном случае ничего не найдено. Очевидно, что этот показатель является наиболее важным возвращаемым значением, потому что другие значения не имеют смысла, если алгоритм не сходится.

Возвращаемый список также включает в себя `par`, параметры, которые минимизируют нашу функцию, и `value`, значение `f` в этой точке. В этом случае `optim` сходится и находит минимальную точку примерно при $a = 10{,}0$ и $b = 0{,}7$.

Для функции `optim` нет нижних и верхних границ, только начальная точка, которую вы предоставляете. Лучшее предположение для начальной точки означает более быструю минимизацию.

Функция `optim` поддерживает несколько различных алгоритмов минимизации, и вы можете выбрать тот, что нужен вам. Если алгоритм по умолчанию вам не подходит, *см.* страницу справки, посвященную альтернативам. Типичная проблема, связанная с многомерной минимизацией, состоит в том, что алгоритм застревает на локальном минимуме и не может найти более глубокий, глобальный минимум. Как правило, более мощные алгоритмы будут ошибаться с меньшей вероятностью. Однако за это приходится платить: они, как правило, работают медленнее.

См. также

Сообщество R реализовало множество инструментов для оптимизации. Посетите страницу по адресу https://cran.r-project.org/web/views/Optimization.html, чтобы получить информацию о дополнительных решениях.

13.3. Вычисление собственных значений и собственных векторов

Задача

Вы хотите вычислить собственные значения или собственные векторы матрицы.

Решение

Используйте функцию `eigen`. Она возвращает список с двумя элементами, `values` и `vectors`, которые содержат (соответственно) собственные значения и собственные векторы.

Обсуждение

Предположим, у нас есть матрица, например матрица Фибоначчи:

```
fibmat <- matrix(c(0, 1, 1, 1), 2, 2)
fibmat
#>      [,1] [,2]
#> [1,]  0    1
#> [2,]  1    1
```

Когда есть матрица, функция `eigen` вернет список ее собственных значений и собственных векторов:

```
eigen(fibmat)
#> eigen() decomposition
#> $values
#> [1] 1.618 -0.618
#>
#> $vectors
#>        [,1]      [,2]
#> [1,]   0.526   0.851
#> [2,]   0.851   0.526
```

Используйте `eigen(fibmat)$values` или `eigen(fibmat)$vectors`, чтобы выбрать нужное значение из списка.

13.4. Метод главных компонент

Задача

Вы хотите определить главные компоненты набора данных с несколькими переменными.

Решение

Используйте функцию `prcomp`. Первый аргумент – это формула, правая часть которой представляет собой набор переменных, разделенных знаком плюс (+). Левая сторона пуста:

```
r <- prcomp( ~ x + y + z)

summary(r)
#> Importance of components:
#>                           PC1     PC2     PC3
#> Standard deviation      1.894 0.11821 0.04459
#> Proportion of Variance  0.996 0.00388 0.00055
#> Cumulative Proportion   0.996 0.99945 1.00000
```

Обсуждение

Базовая версия R включает в себя две функции для выполнения анализа главных компонент, `prcomp` и `princomp`. В документации упоминается, что функция `prcomp` обладает лучшими числовыми свойствами, поэтому здесь представлена она.

Важным применением метода главных компонент является уменьшение размерности вашего набора данных. Предположим, что ваши данные содержат большое количество *N* переменных. В идеале все переменные более или менее независимы и вносят одинаковый вклад. Но если вы подозреваете, что некоторые переменные являются избыточными, метод главных компонент может сообщить вам количество источников дисперсии в ваших данных. Если это число около *N*, тогда все переменные полезны. Если число меньше *N*, тогда ваши данные могут быть уменьшены до набора данных с меньшей размерностью.

Метод главных компонент преобразует ваши данные в векторное пространство, где первое измерение фиксирует наибольшую дисперсию, второе измерение фиксирует второе наибольшее и т. д. Фактический вывод функции `prcomp` – это объект, который при печати дает необходимое вращение вектора:

```
load(file = './data/pca.rdata')
r <- prcomp(~ x + y)
print(r)
#> Standard deviations (1, .., p=2):
#> [1] 0.393 0.163
#>
#> Rotation (n x k) = (2 x 2):
#>      PC1   PC2
#> x -0.553   0.833
#> y -0.833 -0.553
```

Обычно мы находим сводку этого метода гораздо более полезной. Она показывает долю дисперсии, которая фиксируется каждым компонентом:

```
summary(r)
#> Importance of components:
#>                           PC1    PC2
#> Standard deviation      0.393 0.163
#> Proportion of Variance  0.853 0.147
#> Cumulative Proportion   0.853 1.000
```

В этом примере первый компонент зафиксировал 85 % дисперсии, а второй компонент только 15 %, поэтому мы знаем, что первый компонент зафиксировал большую часть.

После вызова функции `prcomp` используйте `plot(r)`, чтобы просмотреть гистограмму дисперсии основных компонентов, и `predict(r)`, чтобы повернуть ваши данные к основным компонентам.

См. также

См. рецепт 13.9, где приводится пример использования метода главных компонент. Дальнейшее применение этого метода в R обсуждается в книге *Modern Applied Statistics with S-Plus* У. Н. Венейблсома и Б. Д. Рипли.

13.5. Простая ортогональная регрессия

Задача

Вы хотите создать линейную модель, используя ортогональную регрессию, в которой дисперсии *x* и *y* обрабатываются симметрично.

Решение

Используйте функцию `prcomp` для выполнения анализа главных компонент для *x* и *y*. Из получившегося вращения вычислите угол наклона и свободный член:

```
r <- prcomp(~ x + y)
slope <- r$rotation[2, 1] / r$rotation[1, 1]
intercept <- r$center[2] - slope * r$center[1]
```

Обсуждение

Ортогональная регрессия также известна как *метод наименьших полных квадратов*. Метод наименьших квадратов (OLS) обладает странным свойством: он асимметричен. То есть вычисление `lm(y ~ x)` не является математической перестановкой вычисления `lm(x ~ y)`. Причина состоит в том, что OLS предполагает, что значения *x* являются константами, а значения *y* – случайными величинами, поэтому вся дисперсия присваивается *y*, а не *x*, что создает ситуацию с асимметрией.

Асимметрия проиллюстрирована на рис. 13-1, где в верхней левой панели отображено соответствие для `lm(y ~ x)`. Алгоритм OLS пытается минимизировать вертикальные расстояния, которые показаны пунктирными линиями. В верхней правой панели показан идентичный набор данных, но тут используется `lm(x ~ y)`, поэтому алгоритм минимизирует горизонтальные пунктирные линии.

Очевидно, вы получите другой результат в зависимости от того, какие расстояния минимизированы.

Нижняя панель на рис. 13-1 совсем другая. Здесь используется метод главных компонент для реализации ортогональной регрессии. Теперь минимизируемые расстояния – это ортогональные расстояния от точек данных до линии регрессии. Это симметричная ситуация: изменение ролей *x* и *y* не меняет расстояния, которые нужно минимизировать.

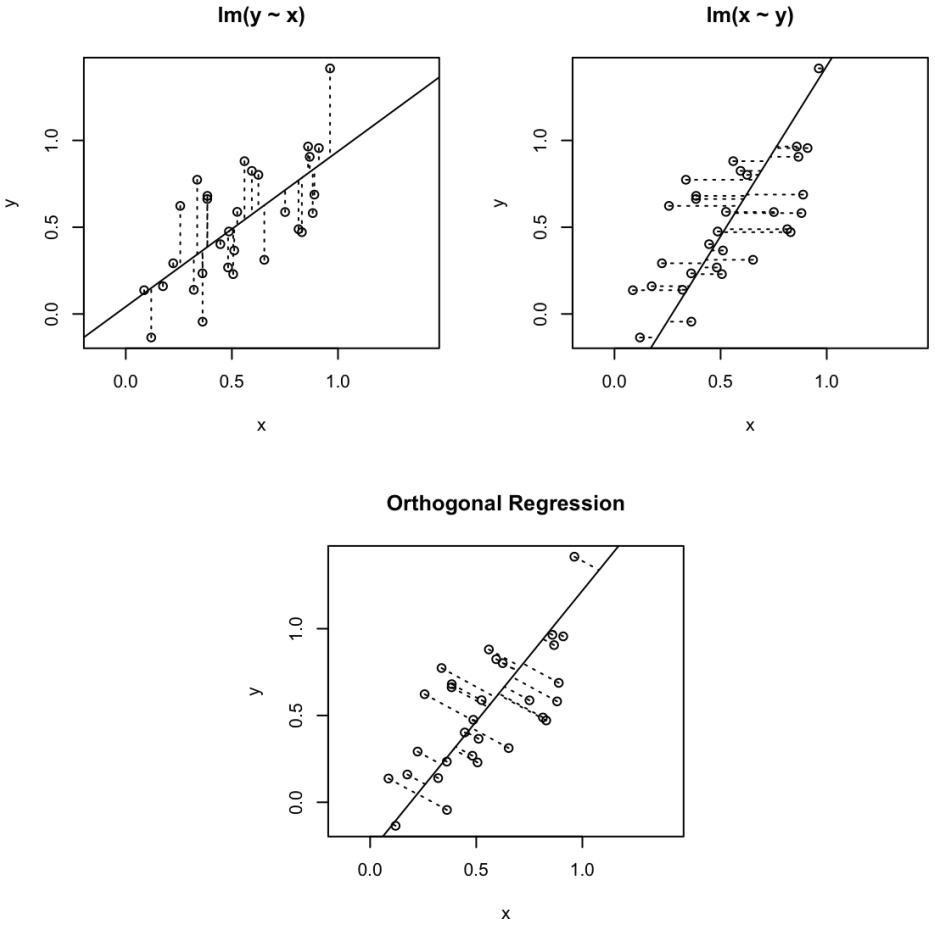


Рис. 13-1. Метод наименьших квадратов в сравнении с ортогональной регрессией

Реализация базовой ортогональной регрессии в R довольно проста. Сначала выполните анализ главных компонент:

```
load(file = './data/pca.rdata')
r <- prcomp(~ x + y)
```

Далее используйте вращения, чтобы вычислить угол наклона:

```
slope <- r$rotation[2, 1] / r$rotation[1, 1]
```

А затем вычислите свободный член:

```
intercept <- r$center[2] - slope * r$center[1]
```

Мы называем это «базовой» регрессией, поскольку она дает лишь точечные оценки для угла наклона и свободного члена, а не доверительные интервалы. Очевидно, что мы также хотели бы иметь регрессионную статистику. В рецепте 13.8 показан один из способов оценки доверительных интервалов с использованием алгоритма бутстрэпа.

См. также

Метод главных компонент описан в рецепте 13.4. Приведенные там графики были вдохновлены работой Винсента Зоонекенда и его учебным курсом по регрессии (http://zoonek2.free.fr/UNIX/48_R/09.html).

13.6. Поиск кластеров в данных

Задача

Вы полагаете, что ваши данные содержат кластеры: группы точек, которые «находятся рядом» друг с другом. Вы хотите идентифицировать эти кластеры.

Решение

Ваш набор данных `x` может быть вектором, таблицей данных или матрицей. Предположим, что `n` – это количество нужных вам кластеров:

```
d <- dist(x)              # Вычисляем расстояния между наблюдениями.
hc <- hclust(d)           # Формируем иерархические кластеры.
clust <- cutree(hc, k=n)  # Организуем их в n самых крупных кластеров.
```

Результат, `clust`, представляет собой вектор чисел от 1 до `n`, по одному для каждого наблюдения в `x`. Каждое число классифицирует свое соответствующее наблюдение в один из *n* кластеров.

Обсуждение

Функция `dist` вычисляет расстояния между всеми наблюдениями. По умолчанию используется евклидово расстояние, которое хорошо подходит для многих приложений, но также доступны и другие меры расстояния.

Функция `hclust` использует эти расстояния для формирования наблюдений в иерархическое дерево кластеров. Можно построить результат этой функции для создания визуализации иерархии под названием *дендрограмма*, как показано на рис. 13-2.

Наконец, функция `cutree` извлекает кластеры из этого дерева. Вы должны указать, сколько кластеров вам нужно, или высоту, на которой дерево должно быть срублено. Часто количество кластеров неизвестно, и в этом случае вам нужно будет изучить набор данных для кластеризации, который имеет смысл в вашем приложении.

Мы проиллюстрируем кластеризацию синтетического набора данных и начнем с генерации 99 нормальных переменных, каждая из которых имеет случайно выбранное среднее значение –3, 0 или +3:

```
means <- sample(c(-3, 0, +3), 99, replace = TRUE)
x <- rnorm(99, mean = means)
```

Ради собственного любопытства мы можем вычислить истинные средние значения исходных кластеров. (В реальной ситуации у нас бы не было фактора `means` и мы бы не смогли выполнить это вычисление.) Мы можем подтвердить, что средние значения групп довольно близки к –3, 0 и +3:

```
tapply(x, factor(means), mean)
#> -3 0 3
#> -3.015 -0.224 2.760
```

Чтобы «обнаружить» кластеры, мы сначала вычислим расстояния между всеми точками:

```
d <- dist(x)
```

Затем мы создаем иерархические кластеры:

```
hc <- hclust(d)
```

И мы можем построить иерархическую дендрограмму кластеров, вызвав функцию `plot` для объекта `hc` (рис. 13-2):

```
plot(hc,
     sub = "",
     labels = FALSE)
```

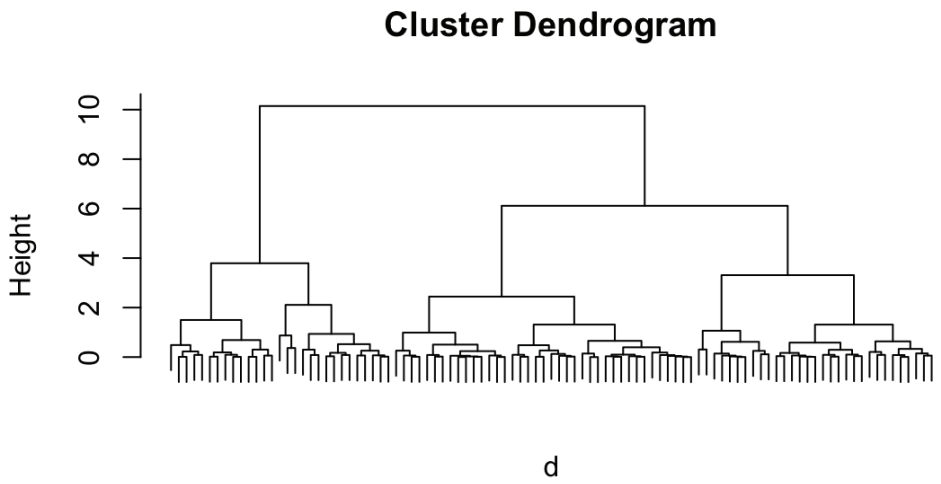


Рис. 13-2. Иерархическая дендрограмма кластеров

Теперь мы можем извлечь три самых крупных кластера:

```
clust <- cutree(hc, k=3)
```

Очевидно, у нас есть огромное преимущество, потому что мы знаем истинное количество кластеров. В реальной жизни редко все бывает так просто. Однако, даже если мы еще не знали, что имеем дело с тремя кластерами, рассмотрение дендрограммы дает нам хорошее представление о том, что в данных есть три больших кластера.

`clust` – это вектор целых чисел от 1 до 3, по одному целому числу на каждое наблюдение в выборке, который присваивает каждое наблюдение кластеру. Вот первые 20 присваиваний:

```
head(clust, 20)
#> [1] 1 2 2 2 1 2 3 3 2 3 1 3 2 3 2 1 2 1 1 3
```

Рассматривая число кластеров как фактор, мы можем вычислить среднее значение каждого статистического кластера (*см.* рецепт 6.6):

```
tapply(x, clust, mean)
#> 1 2 3
#> 3.190 -2.699 0.236
```

R хорошо поработал над разбиением данных на кластеры: средние значения кажутся различными: одно около –2,7, одно около 0,27 и одно около +3,2. (Конечно же, порядок извлеченных средних значений не обязательно соответствует порядку исходных групп.) Извлеченные средние значения аналогичны, но не идентичны исходным. Диаграммы размаха, расположенные рядом друг с другом, могут показать, почему (*см.* рис. 13-3):

```
library(patchwork)

df_cluster <- data.frame(x,
                         means = factor(means),
                         clust = factor(clust))

g1 <- ggplot(df_cluster) +
geom_boxplot(aes(means, x)) +
labs(title = "Original Clusters", x = "Cluster Mean")

g2 <- ggplot(df_cluster) +
geom_boxplot(aes(clust, x)) +
labs(title = "Identified Clusters", x = "Cluster Number")

g1 + g2
```

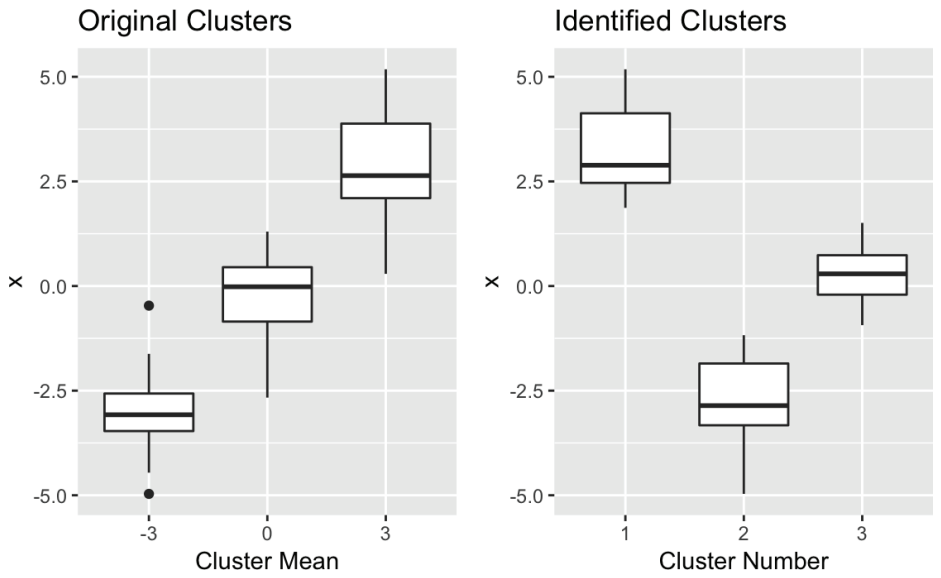


Рис. 13-3. Кластерные диаграммы размаха

Алгоритм кластеризации отлично разделил данные на неперекрывающиеся группы. Исходные кластеры перекрываются, тогда как обнаруженные кластеры – нет.

В этой иллюстрации использовались одномерные данные, но функция `dist` одинаково хорошо работает с многомерными данными, хранящимися в таблице данных или матрице. Каждая строка в таблице данных или матрице рассматривается как одно наблюдение в многомерном пространстве, а функция `dist` вычисляет расстояния между этими наблюдениями.

См. также

Эта демонстрация основана на свойствах кластеризации базового пакета. Существуют и другие пакеты, такие как `mclust`, предлагающие альтернативные механизмы кластеризации.

13.7. Прогнозирование бинарной переменной (логистическая регрессия)

Задача

Вы хотите выполнить логистическую регрессию, модель регрессии, которая предсказывает вероятность возникновения бинарного события.

Решение

Вызовите функцию `glm` с `family = binomial` для выполнения логистической регрессии. В результате получается модельный объект:

```
m <- glm(b ~ x1 + x2 + x3, family = binomial)
```

Здесь `b` – это фактор с двумя уровнями (например, `TRUE` и `FALSE`, 0 и 1), в то время как `x1`, `x2` и `x3` являются независимыми переменными.

Используйте объект модели `m` и функцию `predict` для прогнозирования вероятности на основе новых данных:

```
df <- data.frame(x1 = value, x2 = value, x3 = value)
predict(m, type = "response", newdata = dfrm)
```

Обсуждение

Прогнозирование бинарного результата является распространенной проблемой в моделировании. Будет лечение эффективным или нет? Будут цены расти или падать? Кто победит в игре, команда А или команда Б? Логистическая регрессия полезна для моделирования таких ситуаций. В истинном духе статистики она не просто дает ответ в стиле «большой палец вверх» или «большой палец вниз»; она скорее вычисляет вероятность для каждого из двух возможных результатов.

При вызове функции `predict` мы устанавливаем для `type` значение `"response"`, чтобы функция возвращала вероятность. В противном случае она возвращает логарифмы отношения шансов, которые большинство из нас не находит интуитивно понятными.

В своей неопубликованной книге *Practical Regression and ANOVA Using R* (https://cran.r-project.org/doc/contrib/Faraway-PRA.pdf) ее автор Джулиан Фарауэй приводит пример прогнозирования бинарной переменной: тест из набора данных `pima` ве-

рен, если у пациента положительный тест на диабет. Независимые переменные – это диастолическое артериальное давление и индекс массы тела (ИМТ). Фарауэй использует линейную регрессию, поэтому давайте попробуем вместо нее логистическую регрессию:

```
data(pima, package = "faraway")
b <- factor(pima$test)
m <- glm(b ~ diastolic + bmi, family = binomial, data = pima)
```

Сводка получившийся модели `m` показывает, что соответствующие *p*-значения переменных `diastolic` и `bmi` равны 0,8 и (по существу) 0. Следовательно, можно сделать вывод, что значима только переменная `bmi`:

```
summary(m)
#>
#> Call:
#> glm(formula = b ~ diastolic + bmi, family = binomial, data = pima)
#>
#> Deviance Residuals:
#>  Min       1Q   Median   3Q    Max
#> -1.913 -0.918 -0.685 1.234 2.742
#>
#> Coefficients:
#>             Estimate S td. Error z value Pr(>|z|)
#> (Intercept)  -3.62955   0.46818    -7.75    9.0e-15 ***
#> diastolic    -0.00110   0.00443    0.25   0.8
#> bmi          0.09413    0.01230    7.65   1.9e-14    ***
#> ---
#> Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> (Dispersion parameter for binomial family taken to be 1)
#>
#>     Null deviance: 993.48 on 767 degrees of freedom
#> Residual deviance: 920.65 on 765 degrees of freedom
#> AIC: 926.7
#>
#> Number of Fisher Scoring iterations: 4
```

Поскольку значима только переменная `bmi`, мы можем создать сокращенную модель следующим образом:

```
m.red <- glm(b ~ bmi, family = binomial, data = pima)
```

Давайте используем эту модель для расчета вероятности того, что у человека со средним ИМТ (32,0) будет положительный результат на диабет:

```
newdata <- data.frame(bmi = 32.0)
predict(m.red, type = "response", newdata = newdata)
#> 1
#> 0.333
```

Согласно этой модели, такая вероятность составляет около 33,3 %. Такой же расчет для человека в 90-м процентиле дает вероятность 54,9 %:

```
newdata <- data.frame(bmi = quantile(pima$bmi, .90))
predict(m.red, type = "response", newdata = newdata)
#> 90%
#> 0.549
```

См. также

Использование логистической регрессии включает в себя интерпретацию отклонения, чтобы судить о значимости модели. Мы предлагаем вам почитать о логистической регрессии, прежде чем пытаться получить какие-либо выводы из своей регрессии.

13.8. Бутстрэппинг

Задача

У вас есть набор данных и функция для расчета статистики из этого набора данных. Вы хотите вычислить доверительный интервал статистики.

Решение

Используйте пакет `boot`. Примените функцию `boot`, чтобы вычислить бутстрэп-выборки статистики:

```
library(boot)
bootfun <- function(data, indices) {
  # . . . Расчет статистики с использованием data[indices]. . .
  return(statistic)
}

reps <- boot(data, bootfun, R = 999)
```

Здесь `data` – это ваш исходный набор данных, который может храниться либо в векторе, либо в таблице данных. Статистическая функция (в данном случае `bootfun`) должна ожидать двух аргументов: `data` и `indices`, вектор целых чисел, который выбирает бутстрэп-выборку из данных.

Затем используйте функцию `boot.ci`, чтобы рассчитать доверительный интервал из выборок:

```
boot.ci(reps, type = c("perc", "bca"))
```

Обсуждение

Любой может рассчитать статистику, но это лишь точечная оценка. Мы хотим поднять ее на следующий уровень: что такое доверительный интервал? Для какой-то статистики его можно рассчитать аналитически. Например, доверительный интервал среднего значения рассчитывается с помощью функции `t.test`. К сожалению, это исключение, а не правило. В большинстве случаев расчеты слишком извилисты либо просто неизвестны, и для доверительного интервала не существует известного вычисления в замкнутой форме.

Бутстрэп-алгоритм может вычислить доверительный интервал, даже если вычисление в замкнутой форме не доступно. Вот как это работает. Алгоритм предполагает, что у вас есть выборка размера N и функция для вычисления статистики, и выполняет следующие шаги:

1. Произвольно выбираем N элементов из выборки, выборка с заменой. Этот набор элементов называется *бутстрэп-выборкой*.
2. Применяем функцию к бутстрэп-выборке, чтобы рассчитать статистику. Это значение называется *бутстрэп-репликацией*.

3. Повторяем шаги 1 и 2 много раз, чтобы получить много (обычно тысячи) таких значений.
4. Из этих значений вычисляем доверительный интервал.

Последний шаг может показаться загадочным, но существует несколько алгоритмов для вычисления доверительного интервала. Простой алгоритм использует процентили репликаций, например берет 2,5 процентиля и 97,5 процентиля для формирования 95 % ДИ.

Мы большие поклонники бутстрэпа, потому что ежедневно работаем с неясными статистическими данными. Нам важно знать их доверительные интервалы, и, для того чтобы их получить, известной формулы точно не существует. Бутстрэп дает нам хорошую аппроксимацию.

Давайте проработаем один пример. В рецепте 13.4 мы рассчитали угол наклона линии, используя ортогональную регрессию. Это дало нам точечную оценку, но как найти доверительный интервал? Сначала мы инкапсулируем вычисление угла наклона внутри функции:

```
stat <- function(data, indices) {
 r <- prcomp(~ x + y, data = data, subset = indices)
 slope <- r$rotation[2, 1] / r$rotation[1, 1]
 return(slope)
}
```

Обратите внимание, что функция тщательно выбирает подмножество, определенное `indices`, и вычисляет угол наклона из этого подмножества.

Далее мы рассчитываем 999 репликаций угла наклона. Напомним, что у нас было два вектора, *x* и *y*, в первоначальном рецепте; здесь мы объединяем их в таблицу данных:

```
load(file = './data/pca.rdata')
library(boot)
set.seed(3) # Для воспроизводимости.
boot.data <- data.frame(x = x, y = y)
reps <- boot(boot.data, stat, R = 999)
```

999 репликаций – хорошая отправная точка. Вы всегда можете использовать бутстрэп еще раз и посмотреть, значительно ли изменятся результаты.

Функция `boot.ci` может вычислить доверительный интервал из репликаций. Она реализует несколько различных алгоритмов, а аргумент `type` выбирает, какие алгоритмы будут выполняться. Для каждого выбранного алгоритма `boot.ci` вернет получившуюся оценку:

```
boot.ci(reps, type = c("perc", "bca"))
#> BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
#> Based on 999 bootstrap replicates
#>
#> CALL :
#> boot.ci(boot.out = reps, type = c("perc", "bca"))
#>
#> Intervals :
#> Level Percentile        BCa
#> 95% ( 1.07, 1.99 ) ( 1.09, 2.05 )
#> Calculations and Intervals on Original Scale
```

Здесь мы выбрали два алгоритма, процентиль и BCa (Bias-corrected and accelerated), установив для аргумента `type` значение `c("perc","bca")`. Две полученные оценки отображены внизу под их именами. Доступны и другие алгоритмы; *см.* страницу справки по функции `boot.ci`.

Вы заметите, что два доверительных интервала немного отличаются: (1,068, 1,992) против (1,086, 2,050). Это неудобный, но неизбежный результат использования двух разных алгоритмов. Мы не знаем метода, позволяющего решить, какой из них лучше. Если выбор критически важен, вам необходимо изучить справочную информацию и понять различия. Тем временем наш лучший совет – быть консервативным и использовать минимальную нижнюю и максимальную верхнюю границы; в этом случае получится: (1,068, 2,050).

По умолчанию функция `boot.ci` вычисляет 95%-ный доверительный интервал. Вы можете изменить это с помощью аргумента `conf`, например так:

```
boot.ci(reps, type = c("perc", "bca"), conf = 0.90)
```

См. также

См. рецепт 13.4, чтобы узнать, как рассчитать угол наклона. Хорошим руководством и справочником по бутстрэпу является книга *An Introduction to the Bootstrap* Брэдли Эфрона и Роберта Тибширани (издательство *Chapman & Hall / CRC*).

13.9. Факторный анализ

Задача

Вы хотите выполнить факторный анализ своего набора данных в основном для того, чтобы выяснить, что общего у ваших переменных.

Решение

Используйте функцию `factanal`, которая требует ваш набор данных и вашу оценку количества факторов:

```
factanal(data, factors = n)
```

Вывод включает в себя *n* факторов, показывая загрузки каждой входной переменной для каждого фактора, а также *p*-значение. Обычно *p*-значение меньше 0,05 указывает на то, что число факторов слишком мало и не отражает полную размерность набора данных; *p*-значение, превышающее 0,05, указывает на то, что факторов достаточно (или более чем достаточно).

Обсуждение

Факторный анализ создает линейные комбинации ваших переменных, называемые факторами, которые абстрагируют основную общность переменных. Если ваши *n* переменных абсолютно независимы, то у них нет ничего общего, и для их описания требуется *n* факторов.

Но в той степени, в которой переменные имеют основную общность, меньшее количество факторов учитывает большую часть дисперсии, и поэтому требуется меньше, чем *n* факторов.

Для каждого фактора и переменной мы вычисляем корреляцию между ними, известную как *нагрузка*. Переменные с высокой нагрузкой хорошо объясняются

фактором. Нагрузку можно выровнять, чтобы узнать, какая доля общей дисперсии переменной объясняется им.

Факторный анализ полезен, когда показывает, что несколько факторов отражают большую часть дисперсии ваших переменных. Таким образом, он предупреждает вас об избыточности ваших данных. В этом случае вы можете уменьшить свой набор данных, комбинируя тесно связанные переменные или устраняя избыточные переменные.

Более тонкое применение факторного анализа – интерпретация факторов с целью найти взаимосвязь между вашими переменными. Если две переменные имеют большие нагрузки для одного и того же фактора, вы знаете, что у них есть что-то общее. Что это? На этот вопрос нельзя ответить машинально. Вам нужно изучить данные и их значение.

В факторном анализе есть два хитрых аспекта. Первый – это выбор числа факторов. К счастью, можно использовать метод главных компонент, чтобы получить хорошую начальную оценку числа факторов. Второй аспект – интерпретация самих факторов.

Давайте продемонстрируем, что такое факторный анализ, используя цены на акции, или, точнее, изменения в них. Набор данных содержит шесть месяцев, в течение которых менялись цены на акции 12 компаний. Каждая компания занимается нефтяной промышленностью. Цены на их акции, вероятно, движутся вместе, поскольку они подвержены сходным экономическим и рыночным силам. Можно было бы задать вопрос: сколько факторов требуется, чтобы объяснить их изменения? Если требуется только один фактор, то все акции одинаковы, и одна компания так же хороша, как и другая. Если требуется много факторов, мы знаем, что владение несколькими из них обеспечивает диверсификацию.

Начнем с того, что выполним анализ главных компонент для `diffs`, таблицы данных изменений цен. График результатов анализа показывает дисперсию, отражаемую компонентами (рис. 13-4):

```
load(file = './data/diffs.rdata')
plot(prcomp(diffs))
```

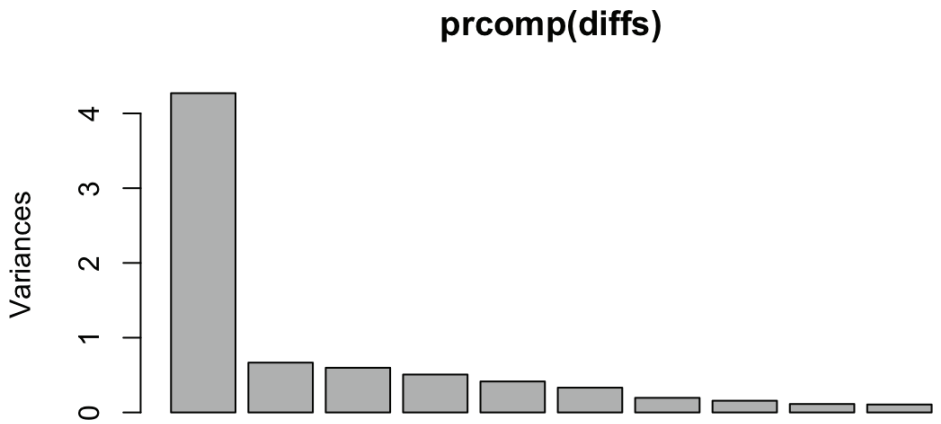


Рис. 13-4. График результатов анализа главных компонент

На рис. 13-4 видно, что первые два компонента отражают большую часть дисперсии, но, возможно, требуется третий. Таким образом, мы выполняем начальный факторный анализ, предполагая, что необходимы два фактора:

```
factanal(diffs, factors = 2)
#>
#> Call:
#> factanal(x = diffs, factors = 2)
#>
#> Uniquenesses:
#>    APC    BP    BRY    CVX    HES    MRO    NBL    OXY    ETP    VLO   XOM
#>  0.307 0.652 0.997 0.308 0.440 0.358 0.363 0.556 0.902 0.786 0.285
#>
#> Loadings:
#> Factor1 Factor2
#> APC 0.773 0.309
#> BP 0.317 0.497
#> BRY
#> CVX 0.439 0.707
#> HES 0.640 0.389
#> MRO 0.707 0.377
#> NBL 0.749 0.276
#> OXY 0.562 0.358
#> ETP 0.283 0.134
#> VLO 0.303 0.350
#> XOM 0.355 0.767
#>
#> Factor1 Factor2
#> SS loadings 2.98 2.072
#> Proportion Var 0.27 0.188
#> Cumulative Var 0.27 0.459
#>
#> Test of the hypothesis that 2 factors are sufficient.
#> The chi square statistic is 62.9 on 34 degrees of freedom.
#> The p-value is 0.00184
```

Можно игнорировать большую часть выходных данных, потому что *p*-значение в нижней части очень близко к нулю (.00184). Небольшое *p*-значение указывает на то, что двух факторов недостаточно, поэтому анализ не очень хороший. Требуется больше, поэтому мы попробуем еще раз с тремя факторами:

```
factanal(diffs, factors = 3)

#>
#> Call:
#> factanal(x = diffs, factors = 3)
#>
#> Uniquenesses:
#>    APC    BP  BRY    CVX    HES    MRO    NBL    OXY    ETP  VLO  XOM
#> 0.316 0.650 0.984 0.315 0.374 0.355 0.346 0.521 0.723 0.605 0.271
#>
#> Loadings:
#> Factor1 Factor2 Factor3
#> APC 0.747 0.270 0.230
#> BP 0.298 0.459 0.224
#> BRY 0.123
#> CVX 0.442 0.672 0.197
```

```
#> HES 0.589 0.299 0.434
#> MRO 0.703 0.350 0.167
#> NBL 0.760 0.249 0.124
#> OXY 0.592 0.357
#> ETP 0.194 0.489
#> VLO 0.198 0.264 0.535
#> XOM 0.355 0.753 0.190
#>
#>                Factor1 Factor2 Factor3
#> SS loadings      2.814   1.774   0.951
#> Proportion Var   0.256   0.161   0.086
#> Cumulative Var   0.256   0.417   0.504
#>
#> Test of the hypothesis that 3 factors are sufficient.
#> The chi square statistic is 30.2 on 25 degrees of freedom.
#> The p-value is 0.218
```

Большое *p*-значение (0,218) подтверждает, что достаточно трех факторов, поэтому мы можем использовать этот анализ.

Вывод включает в себя таблицу объясненной дисперсии, показанную здесь:

```
                Factor1  Factor2  Factor3
SS loadings       2.814    1.774    0.951
Proportion Var    0.256    0.161    0.086
Cumulative Var    0.256    0.417    0.504
```

Эта таблица показывает, что доля дисперсии, объясняемой каждым фактором, составляет 0,256, 0,161 и 0,086 соответственно. В совокупности они объясняют 0,50 дисперсии, что оставляет 1 – 0,504 = 0,496 без объяснений.

Далее мы хотим интерпретировать факторы, которые больше похожи на вуду, чем на науку. Давайте посмотрим на эти нагрузки:

```
Loadings:
    Factor1 Factor2  Factor3
APC   0.747   0.270   0.230
BP    0.298   0.459   0.224
BRY                   0.123
CVX   0.442   0.672   0.197
HES   0.589   0.299   0.434
MRO   0.703   0.350   0.167
NBL   0.760   0.249   0.124
OXY   0.592   0.357
ETP   0.194           0.489
VLO   0.198   0.264   0.535
XOM   0.355   0.753   0.190
```

Каждая строка помечена именем переменной (символом акции): APC, BP, BRY и т. д. У первого фактора много больших нагрузок, а это указывает на то, что он объясняет дисперсию многих акций. Это обычное явление в факторном анализе. Мы часто смотрим на связанные переменные, а первый фактор отражает их базовую связь. В этом примере мы имеем дело с акциями, и большинство акций движутся вместе с оживленным рынком. Вероятно, это отражено в первом факторе.

Второй фактор более тонкий. Обратите внимание, что нагрузки для CVX (0,67) и XOM (0,75) являются доминирующими, при этом BP не сильно отстает (0,46),

но у всех остальных акций заметно меньшие нагрузки. Это указывает на связь между CVX, XOM и BP. Возможно, они работают вместе на общем рынке (например, транснациональные энергоресурсы) и поэтому склонны двигаться вместе.

У третьего фактора также есть три доминирующие нагрузки: VLO, ETP и HES. Это не такие крупные компании, как те мировые гиганты, которых мы видели во втором факторе. Возможно, эти трое делят схожие рынки или риски, поэтому их акции также имеют тенденцию двигаться вместе.

Таким образом, похоже, здесь есть три группы акций:

- CVX, XOM, BP;
- VLO, ETP, HES;
- все остальные.

Факторный анализ – это искусство и наука. Предлагаем вам прочитать хорошую книгу по многомерному анализу, прежде чем использовать его.

См. также

См. рецепт 13.4 для получения дополнительной информации о методе главных компонент.

Глава 14

Анализ временных рядов

Анализ временных рядов стал горячей темой с ростом количественного финансирования и автоматической торговли ценными бумагами. Многие из средств, описанных в этой главе, были изобретены практиками и исследователями в области финансов, торговли ценными бумагами и управления портфелями.

Прежде чем вы начнете анализ временных рядов в R, ключевым решением будет выбор представления данных (класс объекта). Это особенно важно в объектно-ориентированном языке, таком как R, потому что выбор не просто влияет на то, как хранятся данные; он также определяет, какие функции (методы) будут доступны для загрузки, обработки, анализа, вывода и построения ваших данных. Когда многие начинают использовать R, они просто сохраняют данные временных рядов в векторах. Это кажется естественным. Однако они быстро обнаруживают, что ни одна из самых крутых аналитик для анализа временных рядов не работает с простыми векторами. Мы обнаружили, что когда пользователи переключаются на использование класса объектов, предназначенного для данных временных рядов, анализ становится проще, открывая путь к полезным функциям и аналитике.

Первый рецепт этой главы рекомендует использовать пакеты `zoo` или `xts` для представления данных временных рядов. Они носят общий характер и должны отвечать потребностям большинства пользователей. Почти каждый последующий рецепт предполагает, что вы используете одно из этих двух представлений.

Реализация `xts` – это расширенный набор `zoo`, поэтому `xts` может делать все то, что делает `zoo`. В этой главе всякий раз, когда рецепт подходит для объекта `zoo`, можно смело предполагать (если не указано иное), что он также подходит и для объекта `xts`.

Другие представления

Во вселенной R доступны и другие представления данных временных рядов, включая:

- пакет `fts`;
- класс `irts` из пакета `tseries%`;
- пакет `timeSeries`;
- класс `ts` в базовом дистрибутиве;
- `tsibble`, пакет в стиле tidyverse для временных рядов.

На самом деле существует целый набор инструментов, tsbox, просто для преобразования между представлениями.

Два представления заслуживают особого упоминания.

ts (базовый дистрибутив)

Базовый дистрибутива R включает в себя класс временных рядов, ts. Мы не рекомендуем это представление для общего использования, потому что сама реализация слишком ограничена.

Однако базовый дистрибутив включает в себя некоторые важные аналитики временных рядов, которые зависят от ts, такие как функция автокорреляции (acf) и функция взаимной корреляции (ccf). Чтобы использовать эти базовые функции для данных xts, применяйте функцию to.ts для «понижения» своих данных до представления ts перед тем, как вызвать функцию. Например, если x – объект xts, его автокорреляцию можно вычислить следующим образом:

```
acf(as.ts(x))
```

Пакет tsibble

Пакет tsibble – недавнее расширение tidyverse, специально разработанное для работы с данными временных рядов внутри tidyverse. Мы находим это полезным для *перекрестных* данных, то есть данных, для которых наблюдения сгруппированы по дате, и вы хотите выполнять аналитику внутри дат, а не по датам.

Дата против времени и даты

Каждое наблюдение во временном ряду имеет связанную дату или время. Классы объектов, используемые в этой главе, zoo и xts, дают вам возможность выбора либо даты, либо даты и времени для обозначения временного компонента данных. Конечно, вы будете использовать даты для обозначения ежедневных, а также еженедельных, ежемесячных или даже годовых данных; в этих случаях дата дает день, когда произошло наблюдение. Дату и время следует использовать для суточных данных, когда необходимы дата и время наблюдения.

При описании рецептов, приведенных в этой главе, мы посчитали, что будет довольно громоздко все время писать «дата или дата-время». Поэтому мы упростили стиль, предполагая, что ваши данные являются ежедневными и, таким образом, используют целые даты. Пожалуйста, имейте в виду, конечно, что вы свободны в выборе и можете использовать временные метки ниже разрешения календарной даты.

См. также

В R имеется множество полезных функций и пакетов для анализа временных рядов. Вы найдете указатели на них на странице https://cran.r-project.org/web/views/TimeSeries.html.

14.1. Представление данных временного ряда

Задача

Вам нужна структура данных R, которая может представлять данные временного ряда.

Решение

Мы рекомендуем пакеты zoo и xts. Они определяют структуру данных временных рядов и содержат множество полезных функций для работы с данными временных рядов. Создайте объект zoo таким образом, где x – это вектор, матрица или таблица данных, а dt – вектор соответствующих дат или даты и времени:

```
library(zoo)
ts <- zoo(x, dt)
```

Создайте объект xts:

```
library(xts)
ts <- xts(x, dt)
```

Выполните преобразование между представлениями данных временного ряда, используя функции as.zoo и as.xts:

`as.zoo(ts)`

Преобразует ts в объект zoo.

`as.xts(ts)`

Преобразует ts в объект xts.

Обсуждение

В R есть по меньшей мере восемь различных реализаций структур данных для представления временных рядов. Мы не пробовали их все, но можем сказать, что zoo и xts являются отличными пакетами для работы с данными временных рядов и они лучше, чем те, что мы пробовали.

Эти представления предполагают, что у вас есть два вектора: вектор наблюдений (данные) и вектор дат или времени этих наблюдений. Функция zoo объединяет их в объект zoo:

```
library(zoo)
#>
#> Attaching package: 'zoo'
#> The following objects are masked from 'package:base':
#>
#> as.Date, as.Date.numeric
x <- c(3, 4, 1, 4, 8)
dt <- seq(as.Date("2018-01-01"), as.Date("2018-01-05"), by = "days")
ts <- zoo(x, dt)
print(ts)
#> 2018-01-01 2018-01-02 2018-01-03 2018-01-04 2018-01-05
#>          3          4          1          4          8
```

Функция xts действует похожим образом, возвращая объект xts:

```
library(xts)
#>
#> Attaching package: 'xts'
#> The following objects are masked from 'package:dplyr':
#>
#>     first, last
```

```
ts <- xts(x, dt)
print(ts)
#>            [,1]
#> 2018-01-01   3
#> 2018-01-02   4
#> 2018-01-03   1
#> 2018-01-04   4
#> 2018-01-05   8
```

Данные, `x`, должны быть числовыми. Вектор дат или дат и времени, `dt`, называется *индекс*. Допустимые индексы отличаются в зависимости от пакета:

`zoo`

Индексом могут быть любые упорядоченные значения, такие как объекты `Date`, объекты `POSIXct`, целые числа или даже значения с плавающей запятой.

`xts`

Индекс должен быть поддерживаемым классом даты или времени. Сюда входят объекты `Date`, `POSIXct` и `chron`. Их должно быть достаточно для большинства приложений, но вы также можете использовать объекты `yearmon`, `yearqtr` и `dateTime`. Пакет `xts` более строг, чем `zoo`, поскольку он реализует мощные операции, которые требуют индекса на базе времени.

В следующем примере создается объект `zoo`, который содержит цену акций IBM за первые пять дней 2010 года; в нем используются объекты `Date` для индекса:

```
prices <- c(132.45, 130.85, 130.00, 129.55, 130.85)
dates <- as.Date(c(
"2010-01-04", "2010-01-05", "2010-01-06",
"2010-01-07", "2010-01-08"
))
ibm.daily <- zoo(prices, dates)
print(ibm.daily)
#> 2010-01-04 2010-01-05 2010-01-06 2010-01-07 2010-01-08
#>        132        131        130        130        131
```

В следующем примере, напротив, показана цена на акции IBM с интервалом в одну секунду. Он представляет время по количеству часов после полуночи, начиная с 9:30 утра. (1 секунда = 0,00027778 часа, более или менее):

```
prices <- c(131.18, 131.20, 131.17, 131.15, 131.17)
seconds <- c(9.5, 9.500278, 9.500556, 9.500833, 9.501111)
ibm.sec <- zoo(prices, seconds)
print(ibm.sec)
#>  10   10   10   10   10
#> 131 131 131 131 131
```

В этих двух примерах использовался один временной ряд, в котором данные были взяты из вектора. И `zoo`, и `xts` могут обрабатывать несколько параллельных временных рядов. Для этого запишите несколько временных рядов в матрицу или таблицу данных, а затем создайте многомерный временной ряд, вызвав функцию `zoo` (или `xts`):

```
ts <- zoo(df, dt) # OR: ts <- xts(dfrm, dt)
```

Второй аргумент – это вектор дат (или дат и времени) каждого наблюдения. Существует только один вектор дат для всех временных рядов; другими словами, все наблюдения в каждой строке матрицы или таблицы данных должны иметь одну и ту же дату. Если у ваших данных даты не совпадают, *см.* рецепт 14.5.

Как только данные захвачены внутри объекта `zoo` или `xts`, вы можете извлечь чистые данные с помощью функции `coredata`, которая возвращает простой вектор (или матрицу):

```
coredata(ibm.daily)
#> [1] 132 131 130 130 131
```

Извлечь дату или время можно с помощью функции `index`:

```
index(ibm.daily)
#> [1] "2010-01-04" "2010-01-05" "2010-01-06" "2010-01-07" "2010-01-08"
```

Пакет `xts` очень похож на `zoo`. Он оптимизирован по скорости, поэтому особенно хорошо подходит для обработки больших объемов данных, а также хорошо справляется с преобразованием в другие представления временного ряда и обратно.

Одним из больших преимуществ фиксирования данных внутри объекта `zoo` или `xts` является то, что вы получаете доступ к специальным функциям вывода, построения графиков, вычисления разностей, слияния, периодической выборки, а также таким функциям, как `rollapply`, и другим полезным операциям. Существует даже функция `read.zoo`, предназначенная для чтения данных временных рядов из файлов ASCII.

Помните, что пакет `xts` может делать все то, что может делать пакет `zoo`, поэтому везде, где в этой главе говорится об объектах `zoo`, вы также можете использовать объекты `xts`.

Если вы серьезный пользователь данных временных рядов, мы настоятельно рекомендуем изучить документацию к этим пакетам, чтобы узнать, как они могут улучшить вашу жизнь. Это мощные пакеты со множеством полезных функций.

См. также

См. сайт CRAN, где содержится документация по `zoo` и `xts`, включая справочные руководства, сопроводительную документацию и краткие справочные карты. Если эти пакеты уже установлены на вашем компьютере, просмотрите их документацию с помощью функции `vignette`:

```
vignette("zoo")
vignette("xts")
```

Пакет `timeSeries` – еще одна хорошая реализация объекта временных рядов. Он является частью проекта по обучению информатике финансов Rmetrics.

14.2. Построение данных временных рядов

Задача

Вы хотите построить один или несколько временных рядов.

Решение

Используйте функцию `plot(x)`, которая работает с объектами `zoo` и `xts`, содержащими один или несколько временных рядов.

Для простого вектора `v` наблюдений временного ряда можно использовать `plot(v,type ="l")` или `plot.ts(v)`.

Обсуждение

У функции `plot` есть версия для объектов `zoo` и `xts`. Она может строить объекты, которые содержат один или несколько временных рядов. В последнем случае она может построить каждый ряд на отдельном графике или все ряды вместе на одном.

Предположим, что `ibm.infl` – это объект zoo, который содержит два временных ряда. Один показывает котировочную цену акций IBM с января 2000 года по декабрь 2017 года, а другой – ту же самую цену с поправкой на инфляцию. Если вы нанесете объект на график, R построит два временных ряда вместе на одном графике, как показано на рис. 14-1:

```
load(file = "./data/ibm.rdata")
library(xts)

main <- "IBM: Historical vs. Inflation-Adjusted"
lty <- c("dotted", "solid")

# Plot the xts object
plot(ibm.infl,
 lty = lty, main = main,
 legend.loc = "left"
)
```

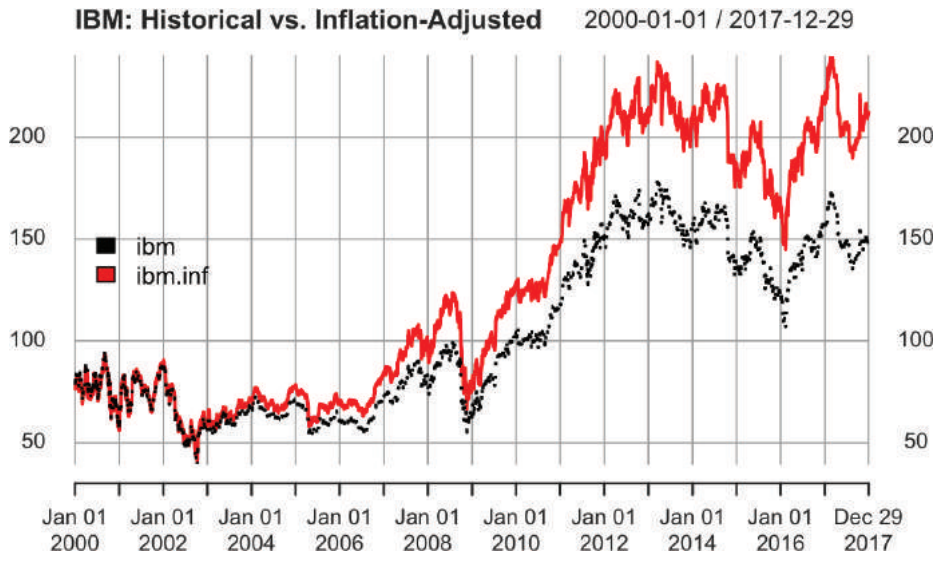


Рис. 14-1. Пример графика `xts`

Функция plot для xts предоставляет заголовок по умолчанию просто как имя объекта xts. Как показано здесь, обычно основной параметр имеет более значимый заголовок.

Код определяет два типа линий (lty), поэтому обе линии нарисованы в двух разных стилях, чтобы их было легче различать.

См. также

Для работы с финансовыми данными пакет quantmod содержит специальные функции построения графиков, которые создают красивые стилизованные диаграммы.

14.3. Извлечение самых старых или самых последних наблюдений

Задача

Вы хотите увидеть только самые старые или самые последние наблюдения вашего временного ряда.

Решение

Используйте функцию head, чтобы просмотреть самые старые наблюдения:

```
head(ts)
```

Используйте функцию tail, чтобы просмотреть самые последние наблюдения:

```
tail(ts)
```

Обсуждение

Функции head и tail являются общими, поэтому они будут работать независимо от того, хранятся ли ваши данные в простом векторе, объекте zoo или объекте xts.

Предположим, у вас есть объект xts с многолетней историей цены акций IBM, такой же, как в предыдущем рецепте. Нельзя отобразить весь набор данных, потому что он будет выходить за пределы экрана. Но вы можете просмотреть начальные наблюдения:

```
ibm <- ibm.infl$ibm  # Берем одну колонку в качестве иллюстрации.
head(ibm)
#>              ibm
#> 2000-01-01 78.6
#> 2000-01-03 82.0
#> 2000-01-04 79.2
#> 2000-01-05 82.0
#> 2000-01-06 80.6
#> 2000-01-07 80.2
```

И просмотреть финальные:

```
tail(ibm)
#>             ibm
#> 2017-12-21 148
#> 2017-12-22 149
#> 2017-12-26 150
```

```
#> 2017-12-27 150
#> 2017-12-28 151
#> 2017-12-29 150
```

По умолчанию `head` и `tail` показывают (соответственно) шесть самых старых и шесть самых последних наблюдений.

Можно увидеть больше наблюдений, предоставив второй аргумент, например `tail(ibm, 20)`.

Пакет `xts` также включает в себя функции `first` и `last`, которые используют календарные периоды вместо количества наблюдений. Можно использовать эти функции для выбора данных по количеству дней, недель, месяцев или даже лет:

```
first(ibm, "2 week")
#>             ibm
#> 2000-01-01 78.6
#> 2000-01-03 82.0
#> 2000-01-04 79.2
#> 2000-01-05 82.0
#> 2000-01-06 80.6
#> 2000-01-07 80.2
```

На первый взгляд этот вывод может сбить с толку. Мы попросили `"2 week"`, а `xts` вернул шесть дней. Это может показаться странным, пока мы не посмотрим на календарь января 2000 года (рис. 14-2).

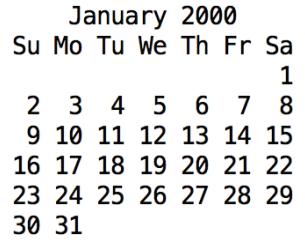


Рис. 14-2. Календарь на январь 2000 года

По календарю видно, что в первой неделе января 2000 года есть только один день, суббота – 1-е число. Затем идет вторая неделя со 2-е по 8-е число. У наших данных нет значения для 8-го числа, поэтому, когда мы используем функцию `first`, чтобы запросить первые две недели, она возвращает все значения за первые две календарные недели. В нашем примере первые две календарные недели содержат только шесть значений.

Точно так же можно попросить функцию `last` дать нам данные за последний месяц:

```
last(ibm, "month")
#>             ibm
#> 2017-12-01 152
#> 2017-12-04 153
#> 2017-12-05 152
#> 2017-12-06 151
#> 2017-12-07 150
#> 2017-12-08 152
#> 2017-12-11 152
```

```
#> 2017-12-12 154
#> 2017-12-13 151
#> 2017-12-14 151
#> 2017-12-15 149
#> 2017-12-18 150
#> 2017-12-19 150
#> 2017-12-20 150
#> 2017-12-21 148
#> 2017-12-22 149
#> 2017-12-26 150
#> 2017-12-27 150
#> 2017-12-28 151
#> 2017-12-29 150
```

Если бы мы использовали здесь объекты `zoo`, нам нужно было бы преобразовать их в объекты `xts`, прежде чем передавать их в функции `first` и `last`, поскольку это функции `xts`.

См. также

См. `help(first.xts)` и `help(last.xts)` для получения подробной информации о функциях `first` и `last` соответственно.

В пакете tidyverse `dplyr` также есть функции `first` и `last`. Если ваш рабочий процесс включает в себя загрузку пакетов `xts` и `dplyr`, убедитесь, что вы явно указали, какую функцию вызываете, используя нотацию `package::function` (например, `xts::first`).

14.4. Выбор элементов из временного ряда

Задача

Вы хотите выбрать один или несколько элементов из временного ряда.

Решение

Можно проиндексировать объект `zoo` или `xts` по положению. Используйте один или два индекса, в зависимости от того, содержит объект один временной ряд или несколько:

`ts[_i-]`

Выбирает *i*-е наблюдение из одного временного ряда.

`ts[j,i]`

Выбирает *i*-е наблюдение *j*-го временного ряда нескольких временных рядов.

Вы можете проиндексировать временной ряд по дате. Используйте тот же тип объекта, что и индекс вашего временного ряда. В этом примере предполагается, что индекс содержит объекты `Date`:

```
ts[as.Date("yyyy-mm-dd")]
```

Можно проиндексировать временной ряд по последовательности дат:

```
dates <- seq(startdate, enddate, increment)
ts[dates]
```

Функция window может выбрать диапазон по дате начала и окончания:

```
window(ts, start = startdate, end = enddate)
```

Обсуждение

Вспомним наш объект xts, представляющий собой выборку скорректированных с учетом инфляции цен на акции IBM из предыдущего рецепта:

```
head(ibm)
#>              ibm
#> 2000-01-01 78.6
#> 2000-01-03 82.0
#> 2000-01-04 79.2
#> 2000-01-05 82.0
#> 2000-01-06 80.6
#> 2000-01-07 80.2
```

Мы можем выбрать наблюдение по положению точно так же, как выбирали элементы из вектора (*см.* рецепт 2.9):

```
ibm[2]
#>            ibm
#> 2000-01-03 82
```

Мы также можем выбрать несколько наблюдений по положению:

```
ibm[2:4]
#>              ibm
#> 2000-01-03 82.0
#> 2000-01-04 79.2
#> 2000-01-05 82.0
```

Иногда удобнее выбирать по дате. Просто используйте саму дату в качестве индекса:

```
ibm[as.Date("2010-01-05")]
#>             ibm
#> 2010-01-05 103
```

Наши данные ibm – это объект xts, поэтому мы также можем использовать сегментацию типа даты (объект zoo не дает подобной гибкости):

```
ibm['2010-01-05']

ibm['20100105']
```

Мы также можем выбрать вектор объектов Date:

```
dates <- seq(as.Date("2010-01-04"), as.Date("2010-01-08"), by = 2)
ibm[dates]
#>             ibm
#> 2010-01-04 104
#> 2010-01-06 102
#> 2010-01-08 103
```

Функция window проще для выбора диапазона последовательных дат:

```
window(ibm, start = as.Date("2010-01-05"), end = as.Date("2010-01-07"))
#>            ibm
#> 2010-01-05 103
```

```
#> 2010-01-06 102
#> 2010-01-07 102
```

Мы можем выбрать комбинацию год/месяц, используя разделение типа *ггггмм*:

```
ibm['201001'] # Январь 2010 года.
```

Выберите диапазоны года, используя наклонную черту /:

```
ibm['2009/2011'] # Все года с 2009 по 2011.
```

Или используйте наклонную черту / для выбора диапазонов, включая месяцы:

```
ibm['2009/201001']    # Все месяцы 2009 года и январь 2010 года.
ibm['2009/201001']    # Все месяцы 2009 года и январь 2010 года.
ibm['200906/201005']  # С июня 2009 по май 2010.
```

См. также

Пакет `xts` предоставляет множество других умных способов индексирования временных рядов. *См.* документацию к нему.

14.5. Объединение нескольких временных рядов

Задача

У вас есть два или более временных рядов. Вы хотите объединить их в единый объект временного ряда.

Решение

Используйте объект `zoo` или `xts` для представления временных рядов, а затем примените функцию `merge`, чтобы объединить их:

```
merge(ts1, ts2)
```

Обсуждение

Объединение двух временных рядов – невероятная головная боль, когда два ряда имеют разные временные метки. Рассмотрим приведенные ниже два временных ряда с ежедневной ценой акций IBM с 1999 по 2017 год и месячный индекс потребительских цен (ИПЦ) за тот же период:

```
load(file = "./data/ibm.rdata")
head(ibm)
#>             ibm
#> 1999-01-04 64.2
#> 1999-01-05 66.5
#> 1999-01-06 66.2
#> 1999-01-07 66.7
#> 1999-01-08 65.8
#> 1999-01-11 66.4
head(cpi)
#>              cpi
#> 1999-01-01 0.938
#> 1999-02-01 0.938
#> 1999-03-01 0.938
```

```
#> 1999-04-01 0.945
#> 1999-05-01 0.945
#> 1999-06-01 0.945
```

Очевидно, что два временных ряда имеют разные временные метки, потому что один из них – это ежедневные данные, а другой – ежемесячные. Хуже того, скачанные данные ИПЦ имеют метку времени для первого дня каждого месяца, даже если этот день – праздничный или выходной (например, Новый год).

Слава богу, у нас есть функция `merge`, которая обрабатывает беспорядочные детали согласования разных дат:

```
head(merge(ibm, cpi))
#>             ibm cpi
#> 1999-01-01 NA 0.938
#> 1999-01-04 64.2 NA
#> 1999-01-05 66.5 NA
#> 1999-01-06 66.2 NA
#> 1999-01-07 66.7 NA
#> 1999-01-08 65.8 NA
```

По умолчанию эта функция находит *объединение* всех дат: выходные данные содержат все даты обоих вводов, а отсутствующие наблюдения заполняются значениями `NA`. Можно заменить значения `NA` самыми последними наблюдениями, используя функцию `na.locf` из пакета `zoo`:

```
head(na.locf(merge(ibm, cpi)))
#>               ibm cpi
#> 1999-01-01   NA 0.938
#> 1999-01-04 64.2 0.938
#> 1999-01-05 66.5 0.938
#> 1999-01-06 66.2 0.938
#> 1999-01-07 66.7 0.938
#> 1999-01-08 65.8 0.938
```

(Здесь `locf` означает «последнее наблюдение перенесено вперед» (*last observation carried forward*).) Обратите внимание, что значения NA были заменены. Однако функция `na.locf` оставила значение `NA` в первом наблюдении (1999-01-01), потому что на тот день цены на акции IBM нет.

Можно получить *пересечение* всех дат, установив для `all` значение `FALSE`:

```
head(merge(ibm, cpi, all = FALSE))
#>               ibm cpi
#> 1999-02-01 63.1 0.938
#> 1999-03-01 59.2 0.938
#> 1999-04-01 62.3 0.945
#> 1999-06-01 79.0 0.945
#> 1999-07-01 92.4 0.949
#> 1999-09-01 89.8 0.956
```

Теперь вывод ограничен наблюдениями, *общими* для обоих файлов.

Однако обратите внимание, что пересечение начинается 1 февраля, а не 1 января. Причина в том, что 1 января – выходной день, поэтому на эту дату цена на акции IBM отсутствует, а, следовательно, пересечения с данными ИПЦ нет. Чтобы это исправить, *см.* рецепт 14.6.

14.6. Заполнение временного ряда

Задача

В данных вашего временного ряда отсутствуют наблюдения. Вы хотите заполнить данные отсутствующими датами / временем.

Решение

Создайте объект `zoo` или `xts` нулевой ширины (без данных) с отсутствующими датами / временем. Затем объедините свои данные с объектом нулевой ширины, взяв объединение всех дат:

```
empty <- zoo(, dates) # 'dates' - это вектор отсутствующих дат.
merge(ts, empty, all = TRUE)
```

Обсуждение

Пакет `zoo` включает в себя удобное свойство в конструкторе объектов `zoo`: вы можете опустить данные и создать объект нулевой ширины. Объект не содержит данных, только даты. Мы можем использовать эти объекты для выполнения таких операций, как заполнение других объектов временных рядов.

Предположим, вы скачиваете данные ИПЦ за месяц, использованные в последнем рецепте. Данные содержат временную отметку в виде первого дня каждого месяца:

```
head(cpi)
#>              cpi
#> 1999-01-01 0.938
#> 1999-02-01 0.938
#> 1999-03-01 0.938
#> 1999-04-01 0.945
#> 1999-05-01 0.945
#> 1999-06-01 0.945
```

Насколько R известно, у нас нет наблюдений для других дней. Однако мы знаем, что каждое значение ИПЦ применяется к последующим дням в течение месяца.

Итак, сначала мы создаем объект нулевой ширины с каждым днем десятилетия, но без данных:

```
dates <- seq(from = min(index(cpi)), to = max(index(cpi)), by = 1)
empty <- zoo(, dates)
```

Мы используем `min(index(cpi))` и `max(index (cpi))`, чтобы получить минимальное и максимальное индексные значения из наших данных `cpi`. Таким образом, получившийся у нас объект `empty` – это просто индекс ежедневных дат с тем же диапазоном, что и наши данные `cpi`.

Затем мы берем объединение данных CPI и объекта нулевой ширины, получая набор данных, заполненный значениями `NA`:

```
filled.cpi <- merge(cpi, empty, all = TRUE)
head(filled.cpi)
#>              cpi
#> 1999-01-01 0.938
#> 1999-01-02 NA
```

```
#> 1999-01-03 NA
#> 1999-01-04 NA
#> 1999-01-05 NA
#> 1999-01-06 NA
```

Получившийся временной ряд содержит каждый календарный день со значением NA там, где не было наблюдения. Возможно, это то, что вам нужно. Однако более распространенным требованием является замена каждого значения NA на самое последнее наблюдение на эту дату. Функция na.locf из пакета zoo делает именно это:

```
filled.cpi <- na.locf(merge(cpi, empty, all = TRUE))
head(filled.cpi)
#>              cpi
#> 1999-01-01 0.938
#> 1999-01-02 0.938
#> 1999-01-03 0.938
#> 1999-01-04 0.938
#> 1999-01-05 0.938
#> 1999-01-06 0.938
```

Значение 1 за январь переносится на 1 февраля, после чего оно заменяется значением за февраль. Теперь у каждого дня есть самое последнее значение ИПЦ на эту дату. Обратите внимание, что в этом наборе данных ИПЦ основан на 1 января 1999 г. = 100 %, и все значения ИПЦ относятся к значению на эту дату:

```
tail(filled.cpi)
#>             cpi
#> 2017-11-26 1.41
#> 2017-11-27 1.41
#> 2017-11-28 1.41
#> 2017-11-29 1.41
#> 2017-11-30 1.41
#> 2017-12-01 1.41
```

Мы можем использовать данный рецепт, чтобы исправить проблему, упомянутую в рецепте 14.5. Там дневная цена на акции IBM и месячные данные ИПЦ не имели пересечений в определенные дни. Это можно исправить, используя несколько разных методов. Один из способов – добавить данные IBM для включения дат ИПЦ, а затем взять пересечение (напомним, что index(cpi) возвращает все даты во временном ряду ИПЦ):

```
filled.ibm <- na.locf(merge(ibm, zoo(, index(cpi))))
head(merge(filled.ibm, cpi, all = FALSE))
#>               ibm cpi
#> 1999-01-01   NA 0.938
#> 1999-02-01 63.1 0.938
#> 1999-03-01 59.2 0.938
#> 1999-04-01 62.3 0.945
#> 1999-05-01 73.6 0.945
#> 1999-06-01 79.0 0.945
```

Это дает ежемесячные наблюдения. Другой способ – заполнить данные CPI (как было описано ранее), а затем взять пересечение с данными IBM. Это дает ежедневные наблюдения, а именно:

```
filled_data <- merge(ibm, filled.cpi, all = FALSE)
head(filled_data)
#>             ibm  cpi
#> 1999-01-04 64.2 0.938
#> 1999-01-05 66.5 0.938
#> 1999-01-06 66.2 0.938
#> 1999-01-07 66.7 0.938
#> 1999-01-08 65.8 0.938
#> 1999-01-11 66.4 0.938
```

Еще одна распространенная техника заполнения пропущенных значений использует метод *кубического сплайна*, который интерполирует сглаженные промежуточные значения из известных данных. Мы можем использовать функцию `na.approx`, чтобы заполнить наши пропущенные значения с помощью кубического сплайна:

```
combined_data <- merge(ibm, cpi, all = TRUE)
head(combined_data)
#>            ibm cpi
#> 1999-01-01 NA  0.938
#> 1999-01-04 64.2 NA
#> 1999-01-05 66.5 NA
#> 1999-01-06 66.2 NA
#> 1999-01-07 66.7 NA
#> 1999-01-08 65.8 NA

combined_spline <- na.spline(combined_data)
head(combined_spline)
#>              ibm cpi
#> 1999-01-01 4.59 0.938
#> 1999-01-04 64.19 0.938
#> 1999-01-05 66.52 0.938
#> 1999-01-06 66.21 0.938
#> 1999-01-07 66.71 0.938
#> 1999-01-08 65.79 0.938
```

Обратите внимание, что оба пропущенных значения для `cpi` и `ibm` были заполнены. Однако значение, указанное для столбца `ibm` на 1 января 1999 г., не соответствует наблюдению 4 января. Это демонстрирует одну из проблем, связанных с кубическими сплайнами: они могут стать весьма нестабильными, если интерполируемое значение находится в самом начале или в самом конце ряда. Чтобы обойти эту нестабильность, можно получить точки данных до 1 января 1999 года, а затем выполнить интерполяцию с помощью функции `na.spline`, или можно было бы просто выбрать другой метод интерполяции.

14.7. Смещение временного ряда

Задача

Вы хотите сместить временной ряд во времени либо вперед, либо назад.

Решение

Используйте функцию `lag`. Второй аргумент, `k`, – это количество периодов для смещения данных:

```
lag(ts, k)
```

Используйте положительное значение k для сдвига данных во времени (завтрашние данные становятся сегодняшними). Используйте отрицательное k, чтобы сместить данные назад во времени (вчерашние данные становятся сегодняшними).

Обсуждение

Вспомните объект zoo, содержащий пять дней цены на акции IBM из рецепта 14.1:

```
ibm.daily
#> 2010-01-04 2010-01-05 2010-01-06 2010-01-07 2010-01-08
#>        132        131        130 1        30         131
```

Чтобы сдвинуть данные на один день вперед, мы используем k = +1:

```
lag(ibm.daily, k = +1, na.pad = TRUE)
#> 2010-01-04 2010-01-05 2010-01-06 2010-01-07 2010-01-08
#>         NA         132        131        130         130
```

Мы также устанавливаем для na.pad значение TRUE, чтобы заполнить конечные даты значением NA. В противном случае они будут просто отброшены, что приведет к сокращению временных рядов.

Чтобы сдвинуть данные назад на один день, мы используем k = -1 и снова устанавливаем для na.pad значение TRUE, дабы поставить в начале значение NA:

```
lag(ibm.daily, k = -1, na.pad = TRUE)
#> 2010-01-04 2010-01-05 2010-01-06 2010-01-07 2010-01-08
#>         NA         132        131        130         130
```

Если соглашение о знаках для k кажется вам странным, вы не одиноки.

Функция носит имя lag, но положительный k на самом деле генерирует *опережающие* данные, а не отстающие. Чтобы получить отстающие данные, используйте отрицательное k. Да, это странно. Возможно, функцию следовало бы назвать lead.

Еще одна вещь, с которой следует быть осторожным при использовании функции lag, – это тот факт, что пакет dplyr также содержит функцию с именем lag. Аргументы dplyr::lag не совсем такие же, как и для функции lag из базового дистрибутива R. В частности, dplyr использует n вместо k:

```
dplyr::lag(ibm.daily, n = 1)
#> 2010-01-04 2010-01-05 2010-01-06 2010-01-07 2010-01-08
#>         NA         132        131        130         130
```

Если вы хотите загрузить dplyr, следует использовать пространство имен, чтобы явно указать, какую функцию lag вы используете. Если речь идет о базовом дистрибутиве, то это функция stats::lag, тогда как в случае с dplyr это, естественно, функция dplyr::lag.

14.8. Вычисление последовательных различий

Задача

Имеется временной ряд, *x*. Вы хотите вычислить разницу между последовательными наблюдениями: (*x2 – x1*), (*x3 – x2*), (*x4 – x3*), ….

Решение

Используйте функцию `diff`:

```
diff(x)
```

Обсуждение

Функция `diff` является общей, поэтому она работает с простыми векторами, объектами `xts` и `zoo`. Прелесть различий в объектах `zoo` или `xts` заключается в том, что результатом будет тот же тип объекта, с которого вы начали, *и* различия будут иметь правильные даты. Здесь мы вычисляем разницу для последовательных цен на акции IBM:

```
ibm.daily
#> 2010-01-04 2010-01-05 2010-01-06 2010-01-07 2010-01-08
#>        132         131        130        130         131
diff(ibm.daily)
#> 2010-01-05 2010-01-06 2010-01-07 2010-01-08
#>      -1.60      -0.85      -0.45       1.30
```

Различие, отмеченное 2010-01-05, является изменением по сравнению с предыдущим днем (2010-01-04), а это обычно то, что вам нужно. Разностные ряды короче по сравнению с исходными на один элемент, поскольку, конечно, R не может вычислить изменение по состоянию на 2010-01-04.

По умолчанию функция `diff` вычисляет последовательные различия. Можно вычислить различия, которые расположены более широко, используя параметр `lag`. Предположим, у вас есть данные ИПЦ за месяц, и вы хотите рассчитать изменение за предыдущие 12 месяцев с учетом изменения по сравнению с прошлым годом. Укажите для параметра `lag` значение 12:

```
head(cpi, 24)
#>              cpi
#> 1999-01-01 0.938
#> 1999-02-01 0.938
#> 1999-03-01 0.938
#> 1999-04-01 0.945
#> 1999-05-01 0.945
#> 1999-06-01 0.945
#> 1999-07-01 0.949
#> 1999-08-01 0.952
#> 1999-09-01 0.956
#> 1999-10-01 0.957
#> 1999-11-01 0.959
#> 1999-12-01 0.961
#> 2000-01-01 0.964
#> 2000-02-01 0.968
#> 2000-03-01 0.974
```

```
#> 2000-04-01 0.973
#> 2000-05-01 0.975
#> 2000-06-01 0.981
#> 2000-07-01 0.983
#> 2000-08-01 0.983
#> 2000-09-01 0.989
#> 2000-10-01 0.990
#> 2000-11-01 0.992
#> 2000-12-01 0.994
head(diff(cpi, lag = 12), 24) # Вычисляем изменение в годовом исчислении.
#>            cpi
#> 1999-01-01 NA
#> 1999-02-01 NA
#> 1999-03-01 NA
#> 1999-04-01 NA
#> 1999-05-01 NA
#> 1999-06-01 NA
#> 1999-07-01 NA
#> 1999-08-01 NA
#> 1999-09-01 NA
#> 1999-10-01 NA
#> 1999-11-01 NA
#> 1999-12-01 NA
#> 2000-01-01 0.0262
#> 2000-02-01 0.0302
#> 2000-03-01 0.0353
#> 2000-04-01 0.0285
#> 2000-05-01 0.0296
#> 2000-06-01 0.0353
#> 2000-07-01 0.0342
#> 2000-08-01 0.0319
#> 2000-09-01 0.0330
#> 2000-10-01 0.0330
#> 2000-11-01 0.0330
#> 2000-12-01 0.0330
```

14.9. Выполнение расчетов по временным рядам

Задача

Вы хотите использовать арифметические и обычные функции для данных временных рядов.

Решение

Нет проблем. R довольно умен в том, что касается операций с объектами `zoo` и `xts`. Вы можете использовать арифметические операторы (+, -, *, / и т. д.), а также обычные функции (`sqrt`, `log` и т. д.) и обычно получать то, чего ожидаете.

Обсуждение

Когда вы выполняете вычисления для объектов `zoo` или `xts`, R выравнивает объекты по дате, чтобы результаты имели смысл. Предположим, мы хотим рассчитать процентное изменение акций IBM. Нам необходимо разделить дневное изменение на цену, но эти два временных ряда не выровнены естественным образом –

у них разное время начала и разная длина. Вот как это выглядит при использовании объекта zoo:

```
ibm.daily
#> 2010-01-04 2010-01-05 2010-01-06 2010-01-07 2010-01-08
#>        132        131         130          130         131
diff(ibm.daily)
#> 2010-01-05 2010-01-06 2010-01-07 2010-01-08
#>      -1.60      -0.85      -0.45       1.30
```

К счастью, когда мы делим один ряд на другой, R выравнивает ряд за нас и возвращает объект zoo:

```
diff(ibm.daily) / ibm.daily
#> 2010-01-05 2010-01-06 2010-01-07 2010-01-08
#>   -0.01223   -0.00654   -0.00347    0.00994
```

Мы можем масштабировать результат на 100, чтобы вычислить процентное изменение, и в результате получаем еще один объект zoo:

```
100 * (diff(ibm.daily) / ibm.daily)
#> 2010-01-05 2010-01-06 2010-01-07 2010-01-08
#>     -1.223     -0.654     -0.347     0.994
```

Функции работают просто прекрасно. Если мы вычислим логарифм или квадратный корень объекта zoo, то в результате получим объект zoo с сохраненными временными метками:

```
log(ibm.daily)
#> 2010-01-04 2010-01-05 2010-01-06 2010-01-07 2010-01-08
#>       4.89       4.87       4.87       4.86       4.87
```

В управлении инвестициями вычисление разницы логарифмов цен довольно распространено. Для R это проще простого:

```
diff(log(ibm.daily))
#> 2010-01-05 2010-01-06 2010-01-07 2010-01-08
#>   -0.01215   -0.00652   -0.00347    0.00998
```

См. также

См. рецепт 14.8, где приводится особый случай вычисления разницы между последовательными значениями.

14.10. Вычисление скользящей средней

Задача

Вы хотите вычислить скользящую среднюю временного ряда.

Решение

Используйте функцию `rollmean` пакета `zoo` для расчета скользящей средней *k*-периода:

```
library(zoo)
ma <- rollmean(ts, k)
```

Здесь ts – это данные временного ряда в объекте zoo, а k – количество периодов.

В большинстве финансовых приложений вам необходимо, чтобы функция rollmean рассчитывала среднее значение, используя только исторические данные; то есть для каждого дня вы используете только те данные, которые доступны в этот день. Для этого укажите для align значение right. В противном случае функция будет «жульничать» и использовать будущие данные, которые в то время были фактически недоступны:

```
ma <- rollmean(ts, k, align = "right")
```

Обсуждение

Трейдеры любят скользящую среднюю для сглаживания колебаний цен. Скользящую среднюю можно вычислить, как описано в рецепте 14.12, сочетая функции rollapply и mean, но функция rollmean намного быстрее.

Помимо скорости, прелесть rollmean состоит в том, что она возвращает тот же тип объекта временного ряда, для которого он был вызван (то есть xts или zoo). Для каждого элемента объекта его дата – это дата «по состоянию на» для вычисленного среднего значения. Поскольку результатом является объект временного ряда, можно легко объединить исходные данные и скользящую среднюю, а затем построить их вместе, как на рис. 14-3:

```
ibm_year <- ibm["2016"]
ma_ibm <- rollmean(ibm_year, 7, align = "right")
ma_ibm <- merge(ma_ibm, ibm_year)
plot(ma_ibm)
```

Рис. 14-3. График скользящей средней

На выходе обычно не хватает нескольких исходных точек данных, поскольку для вычисления среднего значения функции rollmean требуется полное k наблюдений. Следовательно, вывод короче, чем ввод. Если для вас это проблема, ука-

жите для na.pad значение TRUE. Тогда rollmean будет дополнять начальные выводы со значениями NA.

См. также

См. рецепт 14.12 для получения более подробной информации о параметре align.

Описанная здесь скользящая средняя представляет собой простую скользящую среднюю. Пакеты quantmod, TTR и fTrading содержат функции для вычисления и построения множества видов скользящих средних, в том числе простых.

14.11. Применение функции по календарному периоду

Задача

Имеется временной ряд. Вы хотите сгруппировать содержимое по календарному периоду (например, за неделю, месяц или год), а затем применить некую функцию к каждой группе.

Решение

В пакет xts входят функции для обработки временных рядов по дням, неделям, месяцам, кварталам или годам:

```
apply.daily(ts, f)
apply.weekly(ts, f)
apply.monthly(ts, f)
apply.quarterly(ts, f)
apply.yearly(ts, f)
```

Здесь ts – это временной ряд xts, а f – функция, применяемая к каждому дню, неделе, месяцу, кварталу или году.

Если ваш временной ряд – объект zoo, сначала преобразуйте его в объект xts, чтобы иметь возможность обращаться к этим функциям, например:

```
apply.monthly(as.xts(ts), f)
```

Обсуждение

Обычно данные временного ряда обрабатываются в соответствии с календарным периодом. Но подсчет календарных периодов в лучшем случае утомителен, а в худшем – причудлив. Пусть эти функции делают всю тяжелую работу.

Предположим, у нас есть пятилетняя история цен на акции IBM, которая хранится в объекте xts:

```
ibm_5 <- ibm["2012/2017"]
head(ibm_5)
#>            ibm
#> 2012-01-03 152
#> 2012-01-04 151
#> 2012-01-05 150
#> 2012-01-06 149
#> 2012-01-09 148
#> 2012-01-10 148
```

Мы можем рассчитать среднюю цену по месяцам, если будем использовать функции `apply.monthly` и `mean` вместе:

```
ibm_mm <- apply.monthly(ibm_5, mean)
head(ibm_mm)
#>             ibm
#> 2012-01-31 151
#> 2012-02-29 158
#> 2012-03-30 166
#> 2012-04-30 167
#> 2012-05-31 164
#> 2012-06-29 159
```

Обратите внимание, что данные IBM с самого начала находятся в объекте `xts`. Если бы они находились в объекте `zoo`, нам бы пришлось преобразовать их в `xts`, используя функцию `as.xts`.

Более интересное приложение – это расчет волатильности по календарному месяцу, где волатильность измеряется как стандартное отклонение дневного дохода. Ежедневные отчеты о доходах рассчитываются следующим образом:

```
diff(log(ibm_5))
```

Мы рассчитываем их стандартное отклонение, месяц за месяцем, следующим образом:

```
apply.monthly(as.xts(diff(log(ibm_5))), sd)
```

Мы можем масштабировать дневное число для оценки годовой волатильности, как показано на рис. 14-4:

```
ibm_vol <- sqrt(251) * apply.monthly(as.xts(diff(log(ibm_5))), sd)
plot(ibm_vol,
  main = "IBM: Monthly Volatility"
)
```

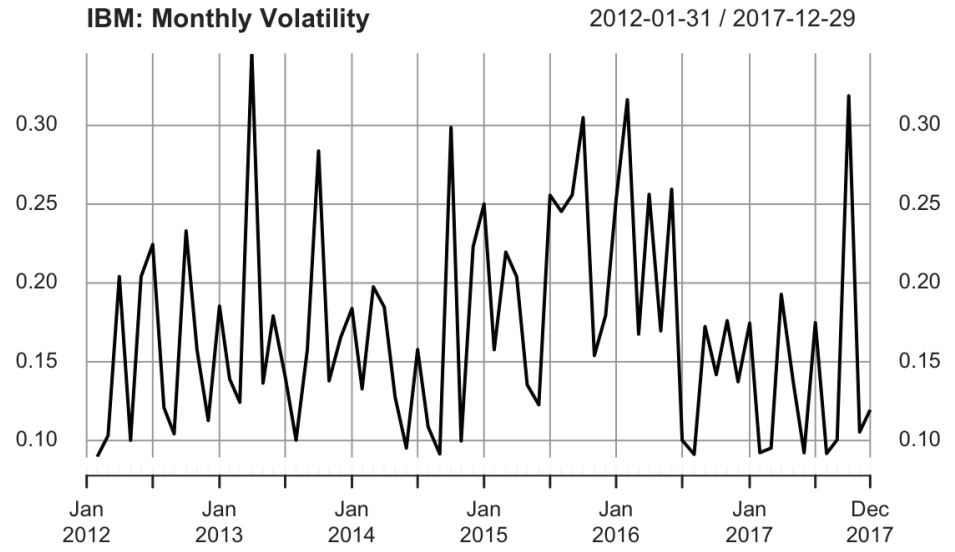


Рис. 14-4. График волатильности акций IBM

14.12. Применение функции rollapply

Задача

Вы хотите применить функцию к временному ряду в скользящей манере: вычислить функцию в точке данных, используя некое временное окно вокруг этой точки, перейти к следующей точке данных, вычислить функцию вокруг нее, перейти к следующей точке и т. д.

Решение

Используйте функцию `rollapply` из пакета `zoo`. Параметр `width` определяет, сколько точек данных из временного ряда (`ts`) должно быть обработано функцией (`f`) в каждой точке:

```
library(zoo)
rollapply(ts, width, f)
```

В случае с большим количеством приложений вы, скорее всего, установите для параметра `align` значение `"right"`, чтобы избежать вычисления `f`, используя исторические данные, которые в то время были недоступны:

```
rollapply(ts, width, f, align = "right")
```

Обсуждение

Функция `rollapply` извлекает «окно» данных из вашего временного ряда, вызывает вашу функцию с этими данными, сохраняет результат и переходит к следующему окну – и повторяет этот шаблон для всего ввода. В качестве иллюстрации рассмотрим вызов функции `rollapply` с шириной 21:

```
rollapply(ts, 21, f)
```

`rollapply` будет неоднократно вызывать функцию `f` со скользящим окном данных, например так:

1. `f(ts[1:21])`
2. `f(ts[2:22])`
3. `f(ts[3:23])`
4. `… etc. …`

Обратите внимание, что функция должна ожидать один аргумент, который представляет собой вектор значений. `rollapply` сохранит возвращаемые значения перед упаковкой их в объект `zoo` вместе с временной меткой для каждого значения. Выбор метки зависит от параметра `align`, заданного для функции:

`align="right"`

Временная метка берется из крайнего правого значения.

`align="left"`

Временная метка берется из крайнего левого значения.

`align="center"` (по умолчанию)

Временная метка берется из среднего значения.

По умолчанию пересчитает функцию в последовательных точках данных. Вместо этого у вас может возникнуть желание вычислить функцию в каждой *n*-й точке данных. Используйте параметр `by = n`, чтобы после каждого вызова функция `rollapply` сдвигалась вперед на *n* точек. Например, когда мы вычисляем скользящее стандартное отклонение временного ряда, мы обычно хотим, чтобы каждое окно данных было отдельным, а не перекрывающимся, поэтому устанавливаем значение `by`, равное размеру окна:

```
ibm_sds <- rollapply(ibm_5, width = 30, FUN = sd, by = 30, align = "right")
ibm_sds <- na.omit(ibm_sds)
head(ibm_sds)
```

Функция `rollapply` по умолчанию возвращает объект с количеством наблюдений, равным входным данным, с отсутствующими значениями, заполненными значениями `NA`. В предыдущем примере мы используем функцию `na.omit` для удаления значений `NA`, чтобы у получившегося объекта были записи только для дат, для которых у нас есть значения.

14.13. Построение функции автокорреляции

Задача

Вы хотите построить функцию автокорреляции (ACF) своего временного ряда.

Решение

Используйте функцию `acf`:

```
acf(ts)
```

Обсуждение

Функция автокорреляции – важный инструмент для выявления взаимосвязей во временном ряду. Это набор корреляций ρ_k для $k = 1, 2, 3, \ldots$, где ρ_k – корреляция между всеми парами точек данных, которые находятся на расстоянии k шагов друг от друга.

Визуализация автокорреляций гораздо полезнее, чем их перечисление, поэтому функция `acf` отображает их для каждого значения k. В следующем примере показаны функции автокорреляции для двух временных рядов: один с автокорреляциями (рис. 14-5) и другой без (рис. 14-6). Пунктирная линия разграничивает значимые и незначимые корреляции: значения над линией – значимые (высота линии определяется количеством данных). Можно построить их следующим образом:

```
load(file = "./data/ts_acf.rdata")

acf(ts1, main = "Significant Autocorrelations")

acf(ts2, main = "Insignificant Autocorrelations")
```

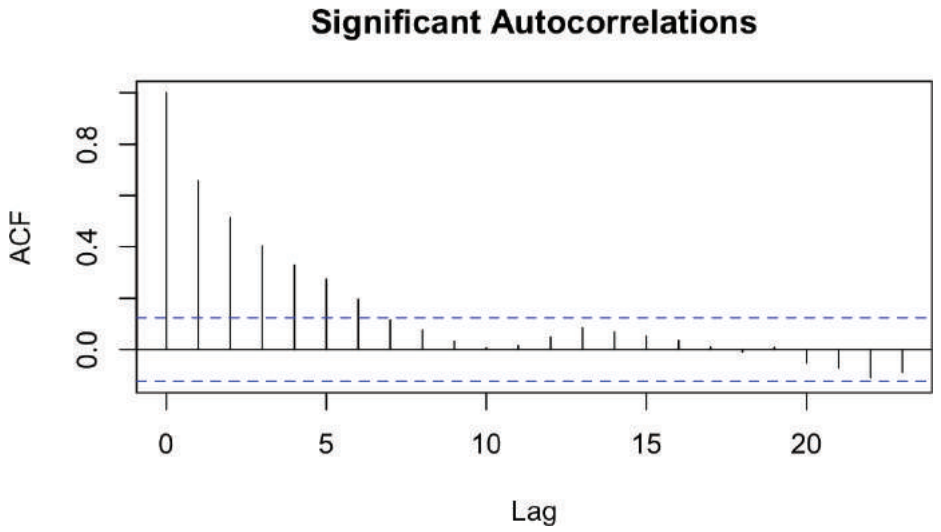


Рис. 14-5. Автокорреляции при каждом интервале: `ts1`

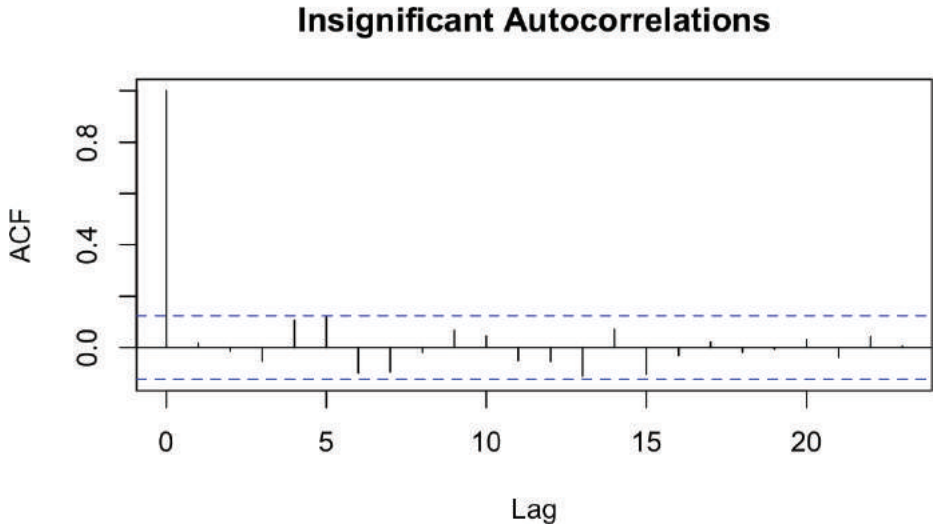


Рис. 14-6. Автокорреляции при каждом интервале: `ts2`

Наличие автокорреляций является одним из признаков того, что авторегрессионная интегрированная модель скользящего среднего (ARIMA) может моделировать временные ряды. Используя автокорреляционную функцию, можно посчитать количество значимых автокорреляций, что является полезной оценкой числа коэффициентов скользящего среднего в модели. Например, на рис. 14-5 показано семь значимых автокорреляций, поэтому мы оцениваем, что для ее модели ARIMA потребуется семь коэффициентов скользящего среднего (MA (7)). Однако эта оценка является лишь отправной точкой и должна быть проверена путем подгонки и диагностики модели.

14.14. Тестирование временного ряда на наличие автокорреляций

Задача

Вы хотите проверить свой временной ряд на наличие автокорреляций.

Решение

Используйте функцию `Box.test`, которая реализует критерий Бокса–Пирса для автокорреляции:

```
Box.test(ts)
```

Вывод включает в себя *p*-значение. Традиционно, *p*-значение меньше 0,05 указывает на то, что данные содержат значительные автокорреляции, тогда как *p*-значение, превышающее 0,05, не дает таких доказательств.

Обсуждение

График функции автокорреляции полезен для того, чтобы покопаться в своих данных. Однако иногда вам просто нужно знать, автокоррелированы ли данные. Статистический тест, такой как тест Бокса–Пирса, может дать ответ на этот вопрос.

Мы можем применить этот тест к данным, автокорреляционную функцию которых мы построили в рецепте 14.13. Тест показывает *p*-значения двух временных рядов, которые составляют почти 0 и 0,79 соответственно:

```
Box.test(ts1)
#>
#> Box-Pierce test
#>
#> data: ts1
#> X-squared = 100, df = 1, p-value <2e-16

Box.test(ts2)
#>
#> Box-Pierce test
#>
#> data: ts2
#> X-squared = 0.07, df = 1, p-value = 0.8
```

p-значение около 0 указывает на то, что первый временной ряд имеет значимые автокорреляции. (Мы не знаем, какие автокорреляции значимы; мы просто знаем, что они существуют.) *p*-значение 0,8 указывает на то, что в ходе теста автокорреляции во втором временном ряду не обнаружено.

Функция `Box.test` также может выполнять тест Льюнга–Бокса, который лучше подходит для небольших выборок. Этот тест вычисляет *p*-значение, интерпретация которого та же, что и для *p*-значения в тесте Бокса–Пирса:

```
Box.test(ts, type = "Ljung-Box")
```

См. также

См. рецепт 14.13, чтобы узнать, как построить функцию автокорреляции для визуальной проверки автокорреляции.

14.15. Построение функции частичной автокорреляции

Задача

Вы хотите построить график функции частичной автокорреляции (PACF) для своего временного ряда.

Решение

Используйте функцию `pacf`:

```
pacf(ts)
```

Обсуждение

Частичная автокорреляционная функция является еще одним инструментом для выявления взаимосвязей во временных рядах. Однако ее интерпретация намного менее интуитивна по сравнению с интерпретацией функции автокорреляции. Мы оставим математическое определение частичной корреляции для учебников по статистике. Здесь мы просто скажем, что частичная корреляция между двумя случайными переменными, X и Y, – это корреляция, которая остается после учета корреляции, показанной X и Y, со всеми другими переменными. В случае временных рядов частичная автокорреляция на интервале k – это корреляция между всеми точками данных, которые находятся на расстоянии ровно k шагов, после учета их корреляции с данными между этими k шагами.

Практическая ценность частичной автокорреляционной функции состоит в том, что она помогает вам определить число авторегрессионных коэффициентов в авторегрессионной интегрированной модели скользящего среднего. В следующем примере показана частичная автокорреляционная функция для двух временных рядов, использованных в рецепте 14.13. Один из этих рядов имеет частичные автокорреляции, а другой – нет. Значения интервалов, линии которых пересекают пунктирную линию, являются статистически значимыми. В первом временном ряду (рис. 14-7) есть два таких значения, при $k = 1$ и $k = 2$, поэтому у нашей начальной модели ARIMA будет два авторегрессионных коэффициента (AR (2)). Однако, как и в случае с автокорреляцией, это лишь начальная оценка, и ее нужно проверить путем подгонки и диагностики модели. Второй временной ряд (рис. 14-8) не показывает такой автокорреляции. Мы можем построить их следующим образом:

```
pacf(ts1, main = "Significant Partial Autocorrelations")
pacf(ts2, main = "Insignificant Partial Autocorrelations")
```

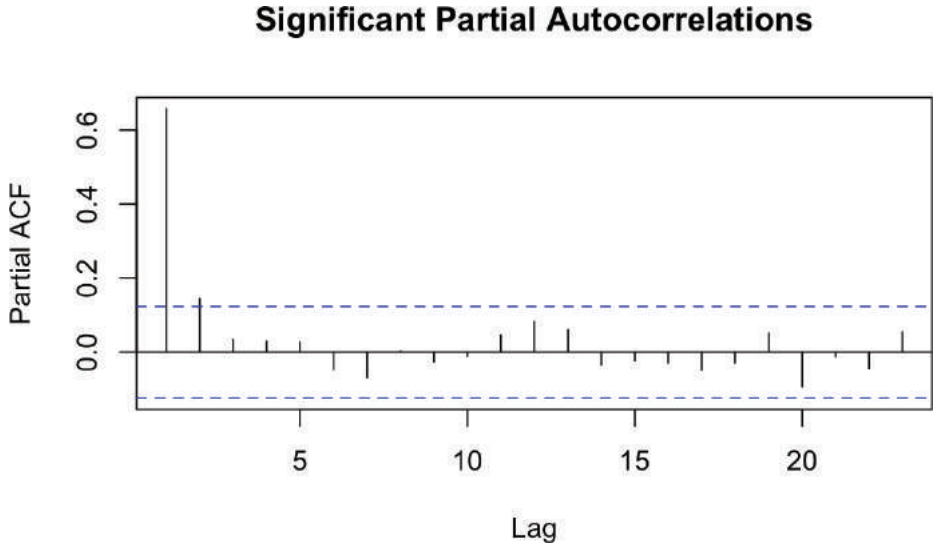


Рис. 14-7. Автокорреляции на каждом интервале: `ts1`

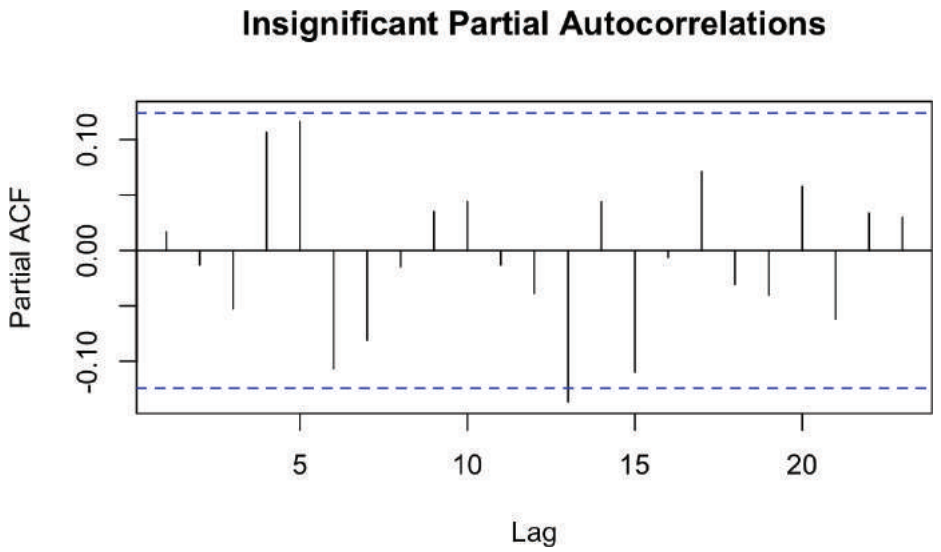


Рис. 14-8. Автокорреляции на каждом интервале: `ts2`

См. также

См. рецепт 14.13.

14.16. Поиск корреляций с временным лагом между двумя временными рядами

Задача

У вас есть два временных ряда, и вам интересно, есть ли между ними корреляция с временным лагом.

Решение

Используйте функцию `Ccf` из пакета `forecast`, чтобы построить взаимно корреляционную функцию, которая покажет корреляции с временным лагом:

```
library(forecast)
Ccf(ts1, ts2)
```

Обсуждение

Функция взаимной корреляции помогает обнаруживать корреляции с временным лагом между двумя временными рядами. Такой тип корреляции возникает, когда сегодняшнее значение в одном временном ряду соотносится с будущим или прошлым значением в другом временном ряду.

Рассмотрим взаимосвязь между ценами на товары и ценами на облигации. Некоторые аналитики полагают, что эти цены связаны, потому что изменения цен на сырьевые товары являются барометром инфляции, одним из ключевых факторов при ценообразовании облигаций. Можно ли обнаружить связь между ними?

На рис. 14-9 показана функция взаимной корреляции, полученная из ежедневных изменений цен облигаций и индекса цен на товары[1].

```
library(forecast)
load(file = "./data/bnd_cmty.Rdata")
b <- coredata(bonds)[, 1]
c <- coredata(cmdtys)[, 1]

Ccf(b, c, main = "Bonds vs. Commodities")
```

[1] В частности, переменная `bonds` – это логарифмическая доходность инвестиционного фонда Vanguard Long-Term Bond Index Fund (VBLTX), а переменная `cmdtys` – логарифмическая доходность фонда Invesco DB Commodity Tracking Fund (DBC). Данные были взяты с периода 2007-01-01 по 2017-12-31.

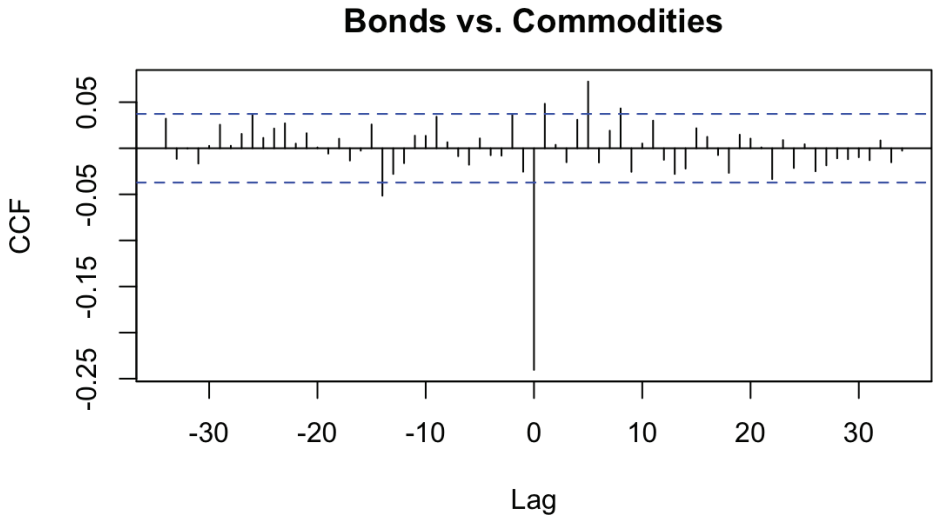


Рис. 14-9. Функция взаимной корреляции

Обратите внимание, что поскольку объекты, с которых мы начинаем, `bond` и `cmdtys`, являются объектами `xts`, мы извлекаем из каждого вектора данных, используя `coredata()[1]`, потому что функция `Ccf` ожидает, что входные данные будут простыми векторами. Каждая вертикальная линия показывает корреляцию между двумя временными рядами с некоторой задержкой, как показано вдоль оси *x*. Если корреляция распространяется выше или ниже пунктирных линий, она является статистически значимой.

Обратите внимание, что корреляция при лаге 0 равна –0,24. Это простая корреляция между переменными:

```
cor(b, c)
#> [1] -0.24
```

Гораздо интереснее корреляции в лагах 1, 5 и 8, которые являются статистически значимыми. Очевидно, что в ежедневных ценах на облигации и товары есть некий «волновой эффект», потому что изменения сегодняшнего дня коррелируют с изменениями дня завтрашнего. Обнаружение такой связи полезно для тех, кто занимается краткосрочными прогнозами, например рыночных аналитиков и торговцев облигациями.

14.17. Удаление тренда из временного ряда

Задача

Данные вашего временного ряда содержат тренд, который вы хотите удалить.

Решение

Используйте линейную регрессию для определения компонента тренда, а затем вычтите этот компонент из исходного временного ряда. Эти две строки показывают, как удалить тренд из объекта `zoo` ts и поместить результат в `detr`:

```
m <- lm(coredata(ts) ~ index(ts))
detr <- zoo(resid(m), index(ts))
```

Обсуждение

Некоторые данные временного ряда содержат тренды, а это означает, что они постепенно наклоняются вверх или вниз с течением времени. Предположим, что наш объект временного ряда (в данном случае это объект `zoo`), `yield`, содержит тренд, как показано на рис. 14-10.

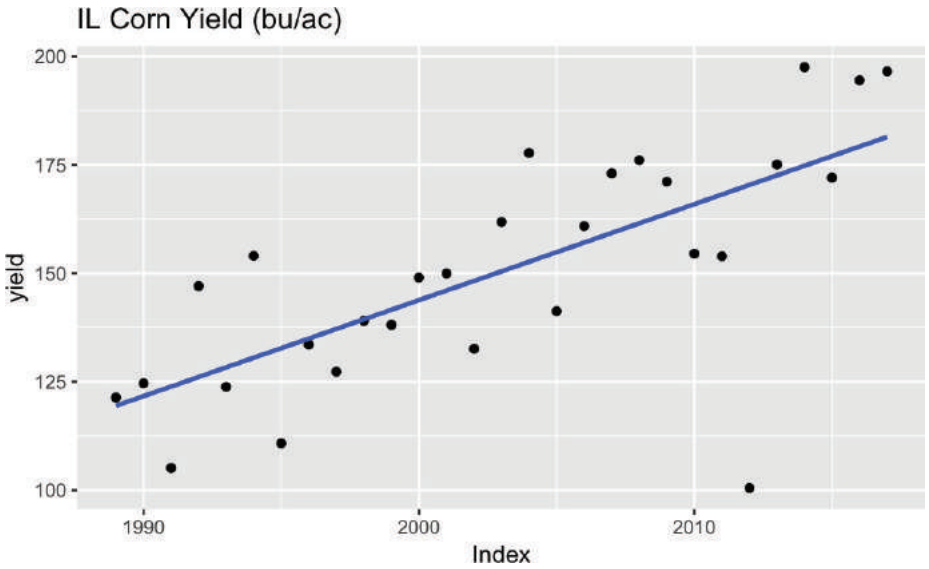


Рис. 14-10. Временные ряды с трендом

Можно удалить компонент тренда в два этапа. Сначала мы определяем общий тренд с помощью функции линейной модели `lm`. Модель должна использовать индекс временного ряда для переменной *x* и данные временного ряда для переменной *y*:

```
m <- lm(coredata(yield) ~ index(yield))
```

После этого мы удаляем линейный тренд из исходных данных, вычитая прямую, которую нашла функция `lm`. Это легко, потому что у нас есть доступ к невязкам линейной модели, которые определяются разницей между исходными данными и подогнанной линией:

$$r_i = y_i - \beta_1 x_i - \beta_0,$$

где r_i – i-я невязка, а β_1 и β_0 – угловой коэффициент и свободный член модели соответственно. Мы можем извлечь невязки из линейной модели, используя функцию `resid`, а затем встроить невязки в объект `zoo`:

```
detr <- zoo(resid(m), index(yield))
```

Обратите внимание, что мы используем тот же временной индекс, что и исходные данные. Когда мы строим график `detr`, совершенно очевидно, что у него нет тренда, как видно на рис. 14-11:

```
autoplot(detr)
```

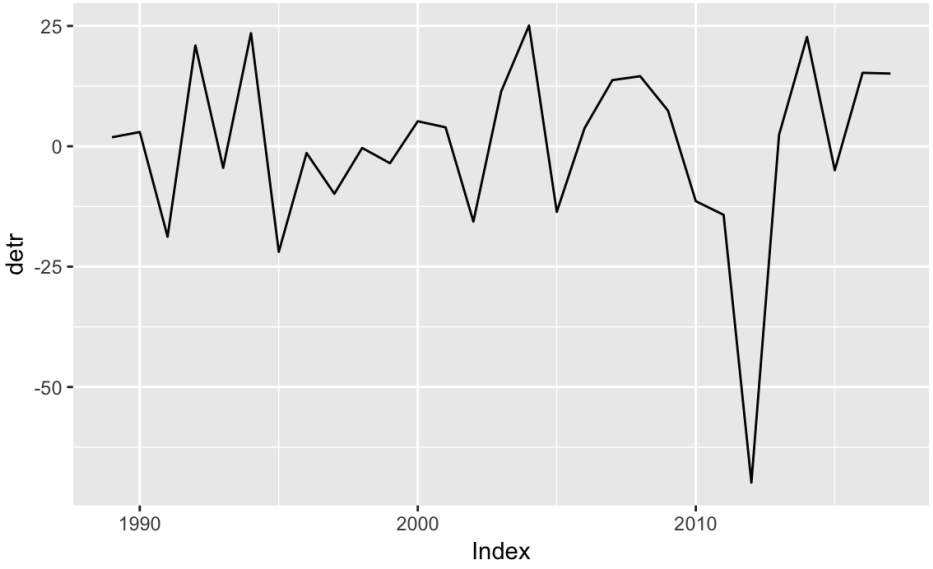


Рис. 14-11. Диаграмма невязок

Эти данные представляют собой среднюю урожайность кукурузы в штате Иллинойс в бушелях на акр (бу/акр), поэтому `detr` – это разница между фактической урожайностью и трендом. Иногда при удалении тренда можно определить процентное отклонение от тренда. В этом случае можно разделить на начальную меру (*см.* рис. 14-12):

```
library(patchwork)
# y <- autoplot(yield) +
# labs(x='Year', y='Yield (bu/ac)', title='IL Corn Yield')
d <- autoplot(detr, geom = "point") +
 labs(
   x = "Year", y = "Yield Dev (bu/ac)",
   title = "IL Corn Yield Deviation from Trend (bu/ac)"
 )
dp <- autoplot(detr / yield, geom = "point") +
 labs(
   x = "Year", y = "Yield Dev (%)",
   title = "IL Corn Yield Deviation from Trend (%)"
 )

d / dp
```

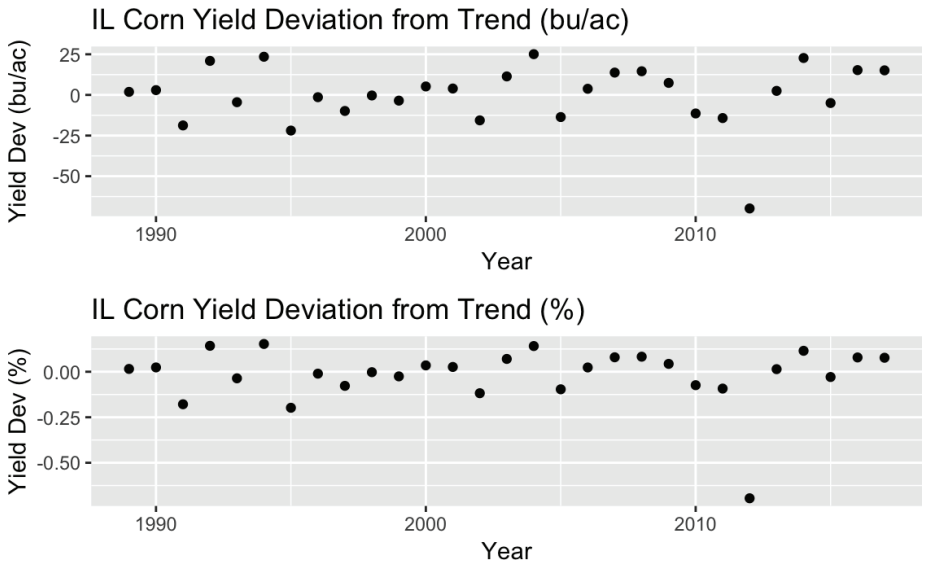


Рис. 14-12. Диаграммы с удаленными трендами

Верхняя диаграмма на рис. 14-12 показывает отклонение доходности от тренда в бушелях на акр (исходные единицы), тогда как нижняя показывает процентное отклонение от тренда.

14.18. Подгонка модели ARIMA

Задача

Вы хотите подогнать модель ARIMA к вашим данным временных рядов.

Решение

Функция `auto.arima` в пакете `forecast` может выбрать правильный порядок модели и подогнать модель под ваши данные:

```
library(forecast)
auto.arima(x)
```

Если вы уже знаете порядок модели (*p, d, q*), то функция `arima` может подогнать модель напрямую:

```
arima(x, order = c(p, d, q))
```

Обсуждение

Создание модели ARIMA включает в себя три этапа:

1) определение порядка модели;
2) подгонку модели к данным с указанием коэффициентов;
3) применение диагностических мер для проверки модели.

Порядок модели обычно обозначается тремя целыми числами (*p, d, q*), где *p* – число коэффициентов авторегрессии, *d* – степень дифференцирования, а *q* – число коэффициентов скользящего среднего.

Когда большинство из нас строит модель ARIMA, мы, как правило, не имеем понятия о соответствующем порядке. Вместо утомительного поиска лучшей комбинации p, d и q мы обычно используем функцию `auto.arima`, которая выполняет поиск за нас:

```
library(forecast)
library(fpp2)

auto.arima(ausbeer)
#> Series: ausbeer
#> ARIMA(1,1,2)(0,1,1)[4]
#>
#> Coefficients:
#>          ar1   ma1     ma2   sma1
#>        0.050 -1.009  0.375 -0.743
#> s.e.   0.196 0.183   0.153 0.050
#>
#> sigma^2 estimated as 241: log likelihood=-886
#> AIC=1783   AICc=1783   BIC=1800
```

В этом случае функция `auto.arima` решила, что наилучшим порядком был (1, 1, 2), а это означает, что она один раз дифференцировала данные ($d = 1$) перед выбором модели с одним авторегрессионным коэффициентом ($p = 1$) и двумя коэффициентами скользящего среднего ($q = 2$). Кроме того, функция определила, что наши данные имеют сезонность, и включила сезонные слагаемые $P = 0$, $D = 1$, $Q = 1$ и период $m = 4$. Слагаемые сезонности аналогичны несезонным слагаемым ARIMA, но относятся к компоненту сезонности. Слагаемое m сообщает нам периодичность сезонности, которая в данном случае является квартальной. Нам будет легче это увидеть, если мы нанесем данные `ausbeer` на график, как показано на рис. 14-13:

```
autoplot(ausbeer)
```

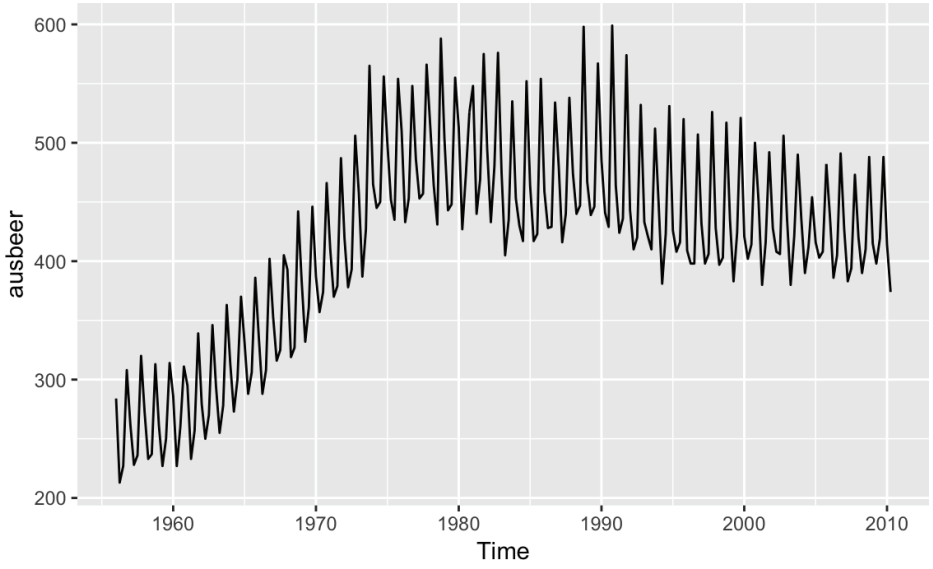


Рис. 14-13. Потребление австралийского пива

По умолчанию функция `auto.arima` ограничивает p и q диапазоном $0 \leqslant p \leqslant 5$ и $0 \leqslant q \leqslant 5$. Если вы уверены, что вашей модели требуется менее пяти коэффициентов, используйте параметры `max.p` и `max.q`, чтобы ограничить поиск дальше; это сделает его быстрее. Аналогично, если вы считаете, что вашей модели нужно больше коэффициентов, используйте параметры `max.p` и `max.q` для расширения пределов поиска.

Если вы хотите отключить компонент сезонности в `auto.arima`, можете установить для параметра `seasonal` значение `FALSE`:

```
auto.arima(ausbeer, seasonal = FALSE)
#> Series: ausbeer
#> ARIMA(3,2,2)
#>
#> Coefficients:
#>          ar1      ar2      ar3      ma1    ma2
#>       -0.957  -0.987  -0.925  -1.043  0.142
#> s.e.  0.026   0.018   0.024   0.062   0.062
#>
#> sigma^2 estimated as 327: log likelihood=-935
#> AIC=1882 AICc=1882 BIC=1902
```

Но обратите внимание, что поскольку модель соответствует несезонной модели, коэффициенты отличаются от сезонной модели.

Если вы уже знаете порядок своей модели ARIMA, функция `arima` может быстро подогнать модель под ваши данные:

```
arima(ausbeer, order = c(3, 2, 2))
#>
#> Call:
#> arima(x = ausbeer, order = c(3, 2, 2))
#>
#> Coefficients:
#>         ar1    ar2      ar3    ma1    ma2
#>      -0.957 -0.987 -0.925 -1.043 0.142
#> s.e. 0.026   0.018 0.024  0.062 0.062
#>
#> sigma^2 estimated as 319: log likelihood = -935, aic = 1882
```

Вывод выглядит идентично выводу функции `auto.arima` с параметром `seasonal`, значение которого установлено как `FALSE`. Здесь не видно, что функция `arima` выполняется намного быстрее.

Вывод обеих функций включает в себя подогнанные коэффициенты и стандартную ошибку (`s.e.`) для каждого коэффициента:

```
Coefficients:
        ar1     ar2     ar3     ma1     ma2
     -0.9569 -0.9872 -0.9247 -1.0425 0.1416
s.e.  0.0257  0.0184  0.0242  0.0619  0.0623
```

Можно найти доверительные интервалы коэффициентов, зафиксировав модель ARIMA в объект, а затем использовать функцию `confint`:

```
m <- arima(x = ausbeer, order = c(3, 2, 2))
confint(m)
#>        2.5 %   97.5 %
```

```
#> ar1 -1.0072  -0.907
#> ar2 -1.0232  -0.951
#> ar3 -0.9721 -0.877
#> ma1 -1.1639 -0.921
#> ma2 0.0195   0.264
```

Этот вывод иллюстрирует главную проблему, которая возникает при моделировании ARIMA: не все коэффициенты обязательно являются значимыми. Если один из интервалов содержит ноль, истинный коэффициент может быть нулевым сам по себе, и в этом случае составляющая не нужна.

Если вы обнаружите, что ваша модель содержит незначимые коэффициенты, используйте рецепт 14.19, чтобы удалить их.

Функции `auto.arima` и `arima` содержат полезные свойства для подгонки наиболее подходящей модели. Например, вы можете заставить их включить или исключить компонент тренда. *См.* справочные страницы для получения дополнительной информации.

И последнее замечание: опасность функции `auto.arima` состоит в том, что она делает моделирование ARIMA простым. Моделировать ARIMA *не* просто. Это больше искусство, чем наука, и автоматически сгенерированная модель является лишь отправной точкой. Мы настоятельно рекомендуем вам прочитать подходящую книгу о моделировании ARIMA, прежде чем остановиться на окончательной модели.

См. также

См. рецепт 14.20, чтобы узнать, как выполнять диагностические тесты для модели ARIMA.

В качестве учебника по прогнозированию временных рядов мы настоятельно рекомендуем 2-е издание книги *Forecasting principles and practice*, Роба Дж. Хиндмана и Джорджа Атанасопулоса, которое свободно доступно в интернете.

14.19. Удаление незначимых коэффициентов ARIMA

Задача

Один или несколько коэффициентов в вашей модели ARIMA статистически не значимы. Вы хотите удалить их.

Решение

Функция `arima` включает в себя параметр `fixed`, который является вектором. Вектор должен содержать один элемент для каждого коэффициента в модели, включая член дрейфа (если таковой есть). Каждый элемент имеет значение `NA` или `0`. Используйте значение `NA` для сохраняемых коэффициентов и `0` для коэффициентов, которые нужно удалить. В этом примере показана модель ARIMA (2, 1, 2) с первым авторегрессионным коэффициентом и первым коэффициентом скользящего среднего, равным `0`:

```
arima(x, order = c(2, 1, 2), fixed = c(0, NA, 0, NA))
```

Обсуждение

Пакет `fpp2` содержит набор данных под названием `euretail`, который является квартальным розничным индексом зоны евро. Давайте выполним функцию `auto.arima` для данных и посмотрим на 98%-ные доверительные интервалы:

```
m <- auto.arima(euretail)
m
#> Series: euretail
#> ARIMA(0,1,3)(0,1,1)[4]
#>
#> Coefficients:
#>         ma1  ma2    ma3    sma1
#>       0.263 0.369 0.420 -0.664
#> s.e.  0.124 0.126 0.129  0.155
#>
#> sigma^2 estimated as 0.156: log likelihood=-28.6
#> AIC=67.3 AICc=68.4 BIC=77.7
confint(m, level = .98)
#>           1 %     99%
#>  ma1   -0.0246  0.551
#>  ma2    0.0774  0.661
#>  ma3    0.1190  0.721
#>  sma1  -1.0231 -0.304
```

В этом примере видно, что 98%-ный доверительный интервал для параметра `ma1` содержит 0, и мы можем разумно заключить, что этот параметр незначим на этом уровне доверия. Мы можем установить этот параметр на 0, используя параметр `fixed`:

```
m <- arima(euretail,
                   order = c(0, 1, 3),
                   seasonal = c(0, 1, 1),
                   fixed = c(0, NA, NA, NA)).
m
#>
#> Call:
#> arima(x = euretail,
                   order = c(0, 1, 3),
                   seasonal = c(0, 1, 1),
                   fixed = c(0,
#> NA, NA, NA))
#>
#> Coefficients:
#>       ma1    ma2    ma3    sma1
#>         0 0.404  0.293 -0.700
#>  s.e.   0 0.129  0.107  0.135
#>
#> sigma^2 estimated as 0.156: log likelihood = -30.8, aic = 69.5
```

Заметьте, что коэффициент `ma1` теперь равен `0`. Остальные коэффициенты (`ma2`, `ma3`, `sma1`) все еще значимы, как показывают их доверительные интервалы, поэтому у нас есть разумная модель:

```
confint(m, level = .98)
#>         1 %    99 %
#> ma1     NA     NA
```

```
#> ma2  0.1049   0.703
#> ma3  0.0438   0.542
#> sma1 -1.0140  -0.386
```

14.20. Выполнение диагностики для модели ARIMA

Задача

Вы создали модель ARIMA с использованием пакета `forecast` и хотите запустить диагностические тесты для проверки модели.

Решение

Используйте функцию `checkresiduals`. В этом примере мы подгоняем модель ARIMA с использованием функции `auto.arima`, помещаем результаты в `m`, а затем запускаем диагностику для модели:

```
m <- auto.arima(x)
checkresiduals(m)
```

Обсуждение

Результатом работы функции `checkresiduals` является набор из трех графиков, как показано на рис. 14-14. Хорошая модель должна давать такие результаты:

```
#>
#> Ljung-Box test
#>
#> data: Residuals from ARIMA(1,1,2)(0,1,1)[4]
#> Q* = 5, df = 4, p-value = 0.3
#>
#> Model df: 4. Total lags used: 8
```

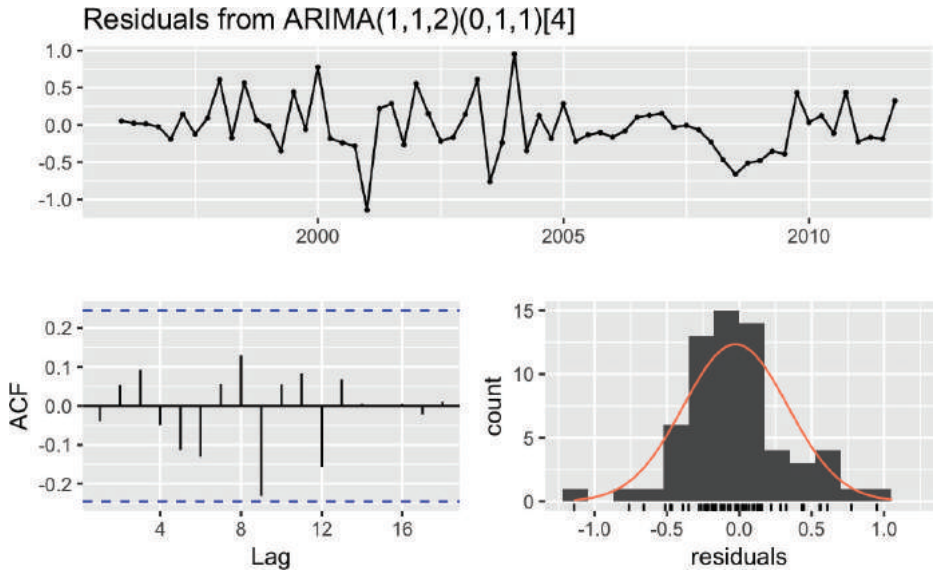


Рис. 14-14. Графики невязок: хорошая модель

Вот что хорошо в этих графиках:

- стандартизированные невязки не показывают кластеры волатильности;
- автокорреляционная функция (ACF) не показывает значимой автокорреляции между невязками;
- невязки имеют форму колокола, что говорит о том, что они достаточно симметричны;
- *p*-значение в тесте Льюнга–Бокса велико, а это указывает на то, что невязки не имеют шаблонов, то есть вся информация была извлечена моделью и остался только шум.

Для сравнения, на рис. 14-15 показаны диагностические диаграммы с проблемами:

```
#>
#> Ljung-Box test
#>
#> data: Residuals from ARIMA(1,1,1)(0,0,1)[4]
#> Q* = 20, df = 5, p-value = 5e-04
#>
#> Model df: 3. Total lags used: 8
```

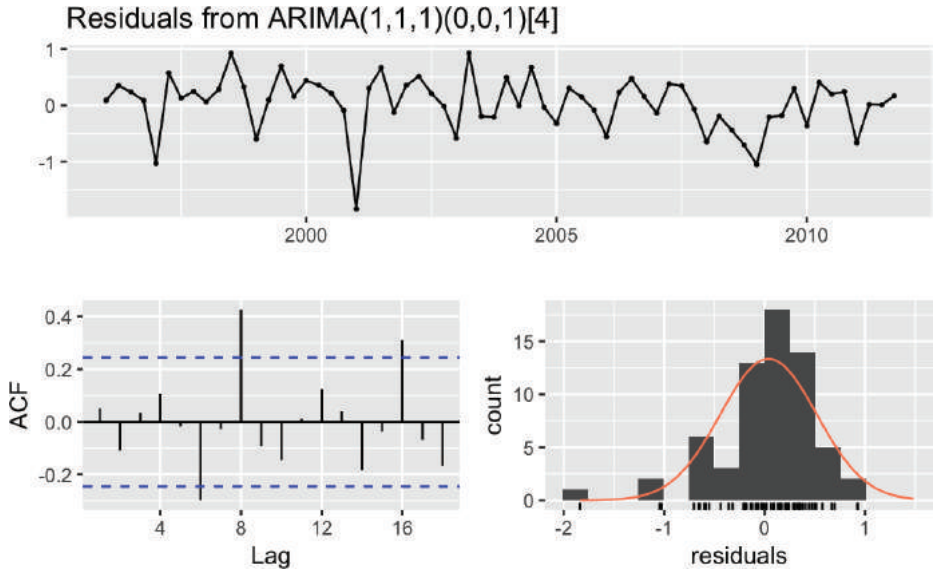


Рис. 14-15. Графики невязок: проблемная модель

Проблемы здесь следующие:

- автокорреляционная функция показывает значительные автокорреляции между невязками;
- *p*-значения в тесте Льюнга–Бокса невелики, что указывает на наличие некоторой закономерности в невязках (т. е. еще есть информация, которую необходимо извлечь из данных);
- невязки выглядят асимметрично.

Это базовая диагностика, но начало хорошее. Найдите подходящую книгу по моделированию ARIMA и выполните рекомендуемые диагностические тесты, прежде чем сделать вывод, что ваша модель исправна. Дополнительные проверки невязок могут включать в себя:

- тесты на нормальность;
- график квантиль-квантиль (Q-Q);
- точечную диаграмму для подогнанных значений.

14.21. Прогнозирование по модели ARIMA

Задача

У вас есть модель ARIMA для вашего временного ряда, которую вы построили с помощью пакета `forecast`. Вы хотите прогнозировать следующие несколько наблюдений в ряде.

Решение

Сохраните модель в объекте, а затем примените к объекту функцию `forecast`. В этом примере мы сохраняем модель из рецепта 14.19 и прогнозируем следующие восемь наблюдений:

```
m <- arima(euretail, order = c(0, 1, 3), seasonal = c(0, 1, 1),
fixed = c(0, NA, NA, NA))
forecast(m)
#>         Point Forecast  Lo80  Hi80  Lo95  Hi95
#> 2012Q1     95.1        94.6  95.6  94.3  95.9
#> 2012Q2     95.2        94.5  95.9  94.1  96.3
#> 2012Q3     95.2        94.2  96.3  93.7  96.8
#> 2012Q4     95.3        93.9  96.6  93.2  97.3
#> 2013Q1     94.5        92.8  96.1  91.9  97.0
#> 2013Q2     94.5        92.6  96.5  91.5  97.5
#> 2013Q3     94.5        92.3  96.7  91.1  97.9
#> 2013Q4     94.5        92.0  97.0  90.7  98.3
```

Обсуждение

Функция `forecast` рассчитает следующие несколько наблюдений и их стандартные ошибки в соответствии с моделью. Она возвращает список из 10 элементов. Когда мы выводим модель, как мы только что это сделали, функция возвращает точки временного ряда, которые она прогнозирует, прогноз и две пары доверительных интервалов: максимум/минимум 80 % и максимум/минимум 95 %.

Если мы хотим извлечь только прогноз, мы можем сделать это, назначив результаты объекту, а затем вытянув элемент списка с именем `mean`:

```
fc_m <- forecast(m)
fc_m$mean
#>      Qtr1 Qtr2 Qtr3 Qtr4
#> 2012 95.1 95.2 95.2 95.3
#> 2013 94.5 94.5 94.5 94.5
```

14.22. Построение прогноза

Задача

Вы создали прогноз временных рядов с помощью пакета `forecast` и хотели бы построить его.

Решение

У моделей временных рядов, созданных с помощью пакета `forecast`, есть метод построения графиков, который использует `ggplot2` для простого создания графиков, как показано на рис. 14-16:

```
fc_m <- forecast(m)
autoplot(fc_m)
```

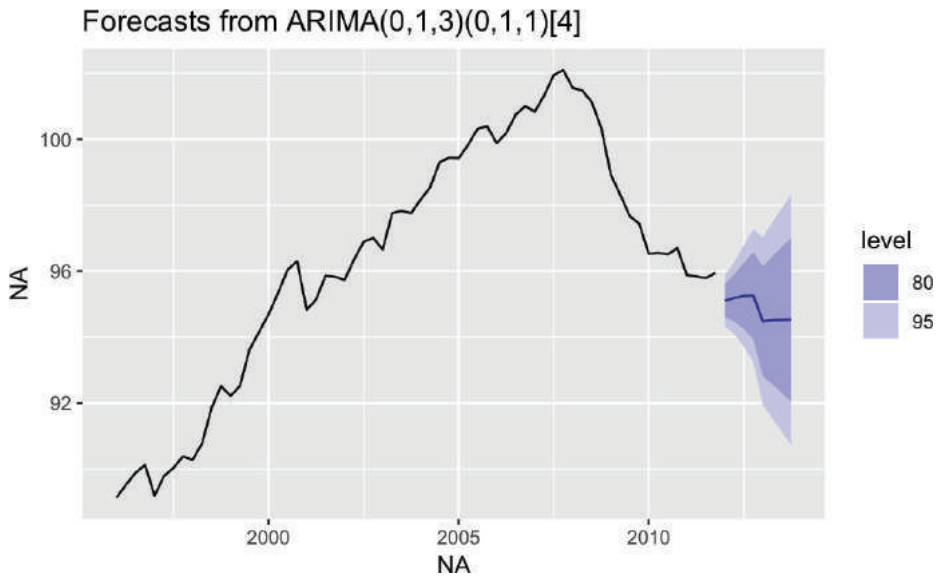


Рис. 14-16. Конус неопределенности: по умолчанию

Обсуждение

Функция `autoplot` дает очень разумный график, как показано на рис. 14-16. Поскольку получившийся график является объектом `ggplot`, мы можем настроить параметры построения так же, как и для любого другого объекта `ggplot`. Здесь мы добавляем подписи и заголовок и меняем тему, как показано на рис. 14-17:

```
autoplot(fc_m) +
 ylab("Euro Index") +
 xlab("Year/Quarter") +
 ggtitle("Forecasted Retail Index") +
 theme_bw()
```

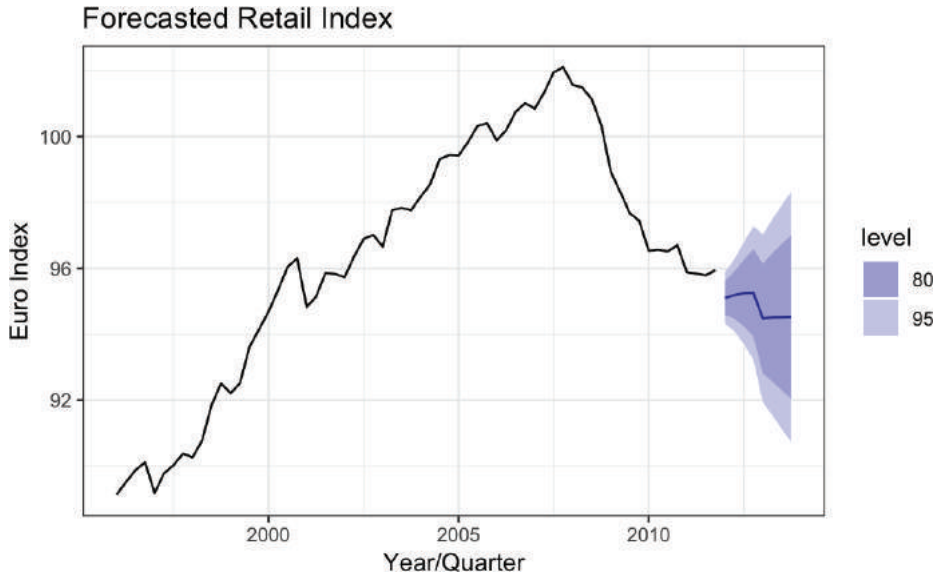


Рис. 14-17. Конус неопределенности: с подписями

См. также

См. главу 10 для получения дополнительной информации о работе с графиками `ggplot`.

14.23. Тестирование на наличие возвращения к среднему

Задача

Вы хотите знать, возвращается ли ваш временной ряд к среднему значению (является ли он стационарным).

Решение

Распространенным тестом на наличие возвращения к среднему является дополненный тест Дики–Фуллера (ADF), который реализуется функцией `adf.test` из пакета `tseries`:

```
library(tseries)
adf.test(ts)
```

Вывод функции `adf.test` включает в себя p-значение. Традиционно, если $p < 0{,}05$, временной ряд, вероятно, возвращается к среднему значению, тогда как $p > 0{,}05$ не дает таких доказательств.

Обсуждение

Когда временной ряд возвращается к среднему, он стремится вернуться к своему долгосрочному среднему значению. Значение может «гулять», но в конце концов

вернётся. Если временной ряд не является возвращающимся к среднему, он может сместиться, так и не вернувшись к среднему значению.

На рис. 14-18 кажется, что он блуждает вверх и не возвращается. Большое *p*-значение из `adf.test` подтверждает, что это не возврат к среднему:

```
library(tseries)
library(fpp2)
autoplot(goog200)
adf.test(goog200)
#>
#> Augmented Dickey-Fuller Test
#>
#> data: goog200
#> Dickey-Fuller = -2, Lag order = 5, p-value = 0.7
#> alternative hypothesis: stationary
```

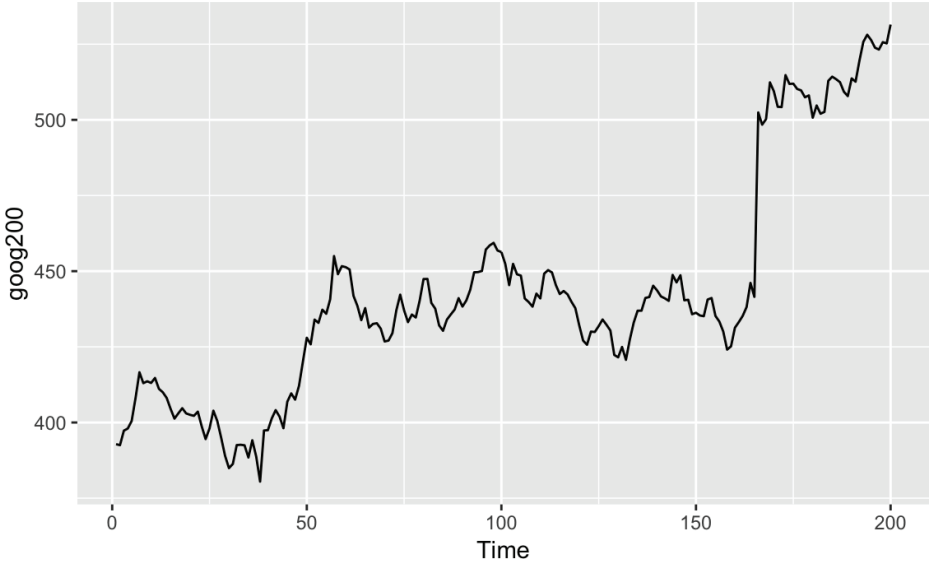


Рис. 14-18. Временной ряд без возврата к среднему значению

Временные ряды на рис. 14-19, однако, просто отскакивают от своего среднего значения. Небольшое *p*-значение (0,01) подтверждает, что это возврат к среднему:

```
autoplot(hsales)
adf.test(hsales)
#>
#> Augmented Dickey-Fuller Test
#>
#> data: hsales
#> Dickey-Fuller = -4, Lag order = 6, p-value = 0.01
#> alternative hypothesis: stationary
```

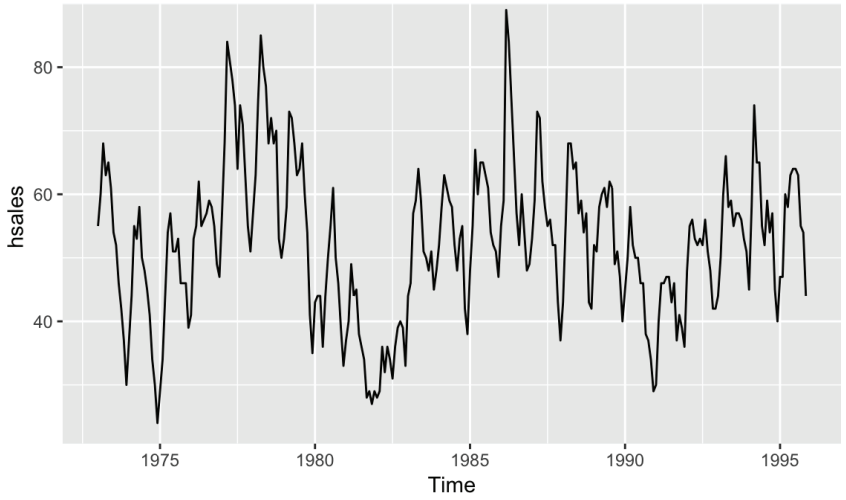


Рис. 14-19. Временной ряд с возвратом к среднему значению

Приведенные здесь примеры данных взяты из пакета `fpp2` и содержат все типы объектов `Time-Series`. Если бы ваши данные находились в объекте `zoo` или `xts`, вам нужно было бы вызвать функцию `coredata` для извлечения необработанных данных из объекта, прежде чем передавать их в функцию `adf.test`:

```
library(xts)
data(sample_matrix)
xts_obj <- as.xts(sample_matrix, dateFormat = "Date")[, "Close"] # Вектор данных.
adf.test(coredata(xts_obj))
#>
#> Augmented Dickey-Fuller Test
#>
#> data: coredata(xts_obj)
#> Dickey-Fuller = -3, Lag order = 5, p-value = 0.3
#> alternative hypothesis: stationary
```

Функция `adf.test` уплотняет ваши данные перед выполнением теста Дики–Фуллера. Сначала она автоматически удаляет тренд из данных, а затем повторно центрирует данные, давая им среднее значение нуля.

Если подобного рода операции нежелательны для вашего приложения, используйте вместо этого функцию `adfTest` из пакета `fUnitRoots`:

```
library(fUnitRoots)
adfTest(coredata(ts1), type = "nc")
```

Когда для `type` установлено значение `"nc"`, функция не удаляет тренд и не центрирует повторно данные. Когда для `type` установлено значение `"c"`, функция повторно центрирует их, но не удаляет тренд.

Функции `adf.test` и `adfTest` позволяют указать значение сдвига (лага), контролирующее точную статистику, которую они рассчитывают. Эти функции обеспечивают разумные значения по умолчанию, но серьезным пользователям следует изучить описание теста Дики–Фуллера из учебника, чтобы определить соответствующий сдвиг для своего приложения.

См. также

Пакеты `urca` и `CADFtest` также реализуют тесты единичного корня. Это тест на наличие возвращения к среднему. Однако будьте осторожны при сравнении тестов из нескольких пакетов. Каждый пакет может делать несколько разные предположения, что может привести к удивительно разным результатам.

14.24. Сглаживание временного ряда

Задача

У вас есть шумный временной ряд. Вы хотите сгладить данные, чтобы устранить шум.

Решение

Пакет `KernSmooth` содержит функции сглаживания. Используйте функцию `dpill` для выбора параметра начальной ширины окна (bandwidth), а затем функцию `locpoly` для сглаживания данных:

```
library(KernSmooth)

gridsize <- length(y)
bw <- dpill(t, y, gridsize = gridsize)
lp <- locpoly(x = t, y = y, bandwidth = bw, gridsize = gridsize)
smooth <- lp$y
```

Здесь `t` – переменная времени, а `y` – временной ряд.

Обсуждение

Пакет `KernSmooth` является стандартной частью дистрибутива R. Он включает в себя функцию `locpoly`, которая создает вокруг каждой точки данных полином, подогнанный к соседним точкам данных. Они носят название *локальные полиномы*. Локальные полиномы соединены вместе, чтобы создать сглаженную версию исходного ряда данных.

Данный алгоритм требует параметра ширины окна для управления степенью сглаживания. Небольшая ширина окна означает меньшее сглаживание, и в этом случае результат будет более точно соответствовать исходным данным. Большая ширина означает бóльшее сглаживание, поэтому результат содержит меньше шума. Сложность заключается в выборе правильной ширины окна: она должна быть не слишком маленькой и не слишком большой.

К счастью, в пакете `KernSmooth` также есть функция `dpill` для вычисления подходящей ширины окна, и работает она довольно хорошо. Мы рекомендуем начать со значения `dpill`, а затем поэкспериментировать со значениями выше и ниже этой начальной точки.

Здесь нет волшебной формулы. Вам нужно решить, какой уровень сглаживания лучше всего подходит для вашего приложения.

Ниже приводится пример сглаживания. Мы создадим несколько примеров данных, которые представляют собой сумму простой синусоиды и нормально распределенного «шума»:

```
t <- seq(from = -10, to = 10, length.out = 201)
noise <- rnorm(201)
y <- sin(t) + noise
```

И dpill, и locpoly требуется размер сетки, иными словами, количество точек, для которых строится локальный полином. Мы часто используем размер сетки, равный количеству точек данных, что дает хорошее разрешение. Получившийся временной ряд очень плавный. Вы можете использовать меньший размер сетки, если вам нужно более грубое разрешение или если у вас очень большой набор данных:

```
library(KernSmooth)
gridsize <- length(y)
bw <- dpill(t, y, gridsize = gridsize)
```

Функция locpoly выполняет сглаживание и возвращает список. Элемент y из этого списка – это сглаженные данные:

```
lp <- locpoly(x = t, y = y, bandwidth = bw, gridsize = gridsize)
smooth <- lp$y

ggplot() +
 geom_line(aes(x = t, y = y)) +
 geom_line(aes(x = t, y = smooth), linetype = 2)
```

На рис. 14-20 сглаженные данные показаны пунктирной линией, а сплошная линия – это наши исходные данные. На рисунке видно, что функция locpoly проделала отличную работу по извлечению исходной синусоиды.

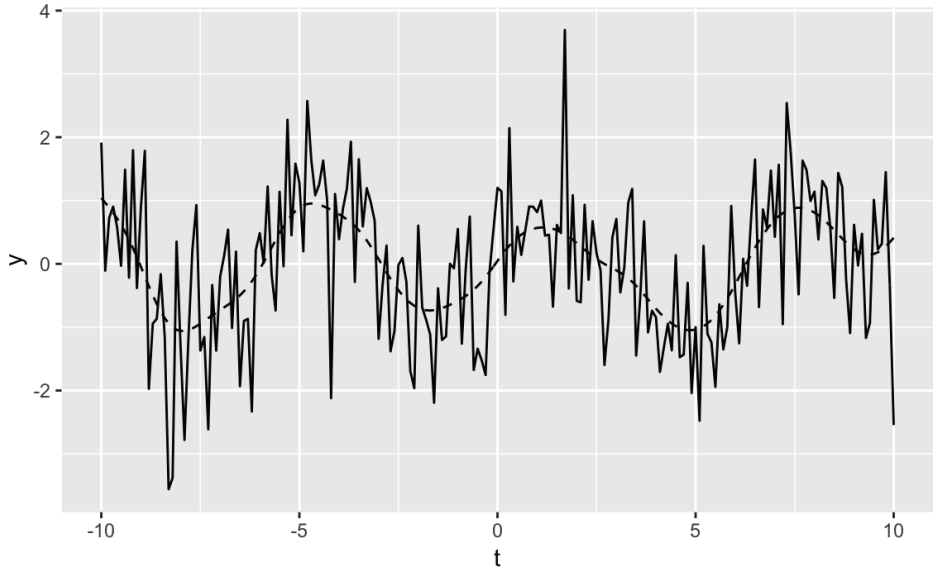


Рис. 14-20. Пример диаграммы временного ряда

См. также

Функции ksmooth, lowess и HoltWinters в базовом распределении также могут выполнять сглаживание. Пакет expsmooth реализует экспоненциальное сглаживание.

Глава 15

Простое программирование

R позволяет достичь многого, при этом не требуя от вас знания программирования. Однако программирование открывает двери для достижения большего, и большинство серьезных пользователей в конечном итоге приобретают некоторый уровень программирования, начиная с простого и, возможно, становясь достаточно опытными пользователями. Хотя это и не книга по программированию, в данной главе изложены некоторые рецепты, которые пользователи R обычно находят полезными, чтобы начать свое путешествие.

Если вы уже знакомы с программированием и языками программирования, некоторые приведенные здесь примечания могут помочь вам быстро адаптироваться. (Если эти термины вам незнакомы, можете пропустить данный раздел.) Вот некоторые технические детали R, о которых следует знать:

Переменные без типов

Переменные в R не имеют фиксированного типа, такого как целое число или символ, в отличие от типизированных языков, таких как C и Java. Переменная сначала может содержать число, а в следующее мгновение – таблицу данных.

Возвращаемые значения

Все функции возвращают значение. Обычно функция возвращает значение последнего выражения в своем теле. Вы также можете использовать `return(expr)` в любом месте функции.

Параметры вызова по значению

Параметры функции – это «вызов по значению»; другими словами, параметры – это строго локальные переменные, и изменения этих переменных не влияют на значение вызывающей функции.

Локальные переменные

Вы создаете локальную переменную, просто присваивая ей значение. Явное объявление не требуется. Когда функция завершает работу, локальные переменные теряются.

Глобальные переменные

Глобальные переменные хранятся в рабочей области пользователя. Внутри функции вы можете изменить глобальную переменную с помощью оператора присваивания << -, но делать это не рекомендуется.

Условные операторы

Синтаксис R включает в себя оператор `if`. *См.* `help(Control)` для получения дополнительной информации.

Циклы

Синтаксис R также включает в себя циклы `for`, `while` и `loop`. Для получения дополнительной информации *см.* `help(Control)`.

Операторы case или switch

Специальная функция под названием `switch` предоставляет базовый оператор `case`. Однако такая семантика может показаться вам странной. *См.* `help(switch)` для получения дополнительной информации.

Ленивые вычисления

R не сразу вычисляет аргументы функции при ее вызове. Скорее, он ждет, пока аргумент фактически будет использоваться внутри функции, а затем вычисляет его. Это дает языку особенно богатую и мощную семантику. В большинстве случаев это незаметно, но иногда приводит к ситуациям, которые ставят в тупик программистов, знакомых только с «энергичным» вычислением, когда аргументы вычисляются при вызове функции.

Функциональная семантика

Функции – это «граждане первого сорта», и с ними можно делать то же, что и с другими объектами: назначать переменным, передавать в функции, выводить на экран, проверять и т. д.

Объектная ориентированность

R поддерживает объектно-ориентированное программирование. На самом деле существует несколько различных парадигм объектной ориентированности, что является благом, если вам нравится, чтобы у вас был выбор или, в противном случае, сбивать с толку.

15.1. Выбор из двух альтернатив: if/else

Задача

Вы хотите написать условие для проверки, которое будет выбирать из двух путей на основе простого теста.

Решение

Блок `if` может реализовать условную логику, протестировав простое условие:

```
if (condition) {
  ## Выполняется, если условие истинно.
} else {
## Выполняется, если условие ложно.
}
```

Обратите внимание на круглые скобки вокруг требуемого условия и фигурные скобки вокруг последующих двух блоков кода.

Обсуждение

Конструкция `if` позволяет выбирать из двух альтернативных путей, проверяя некое условие, такое как `x == 0` или `y> 1`, и затем следуя по одному или другому пути соответственно. Здесь, например, мы выполняем проверку на предмет наличия отрицательных чисел, перед тем как вычислить квадратный корень:

```
if (x >= 0) {
  print(sqrt(x))              # Выполняется, если x >= 0.
} else {
  print("negative number")    # В противном случае делаем это.
}
```

Вы можете связать последовательность конструкций `if/else`, чтобы принять ряд решений. Предположим, мы хотим, чтобы значение было ограничено 0 (без отрицательных значений) и ограничено 1. Это можно было бы написать следующим образом:

```
x <- -0.3

if (x < 0) {
   x <- 0
} else if (x > 1) {
   x <- 1
}

print(x)
#> [1] 0
```

Важно, чтобы проверка условия (выражение после `if`) была *простой*; то есть она должна возвращать одно логическое значение `TRUE` или `FALSE`. Часто по ошибке используют *вектор* логических значений, как в этом примере:

```
x <- c(-2, -1, 0, 1, 2)

if (x < 0) {
 print("values are negative")
}
#> Warning in if (x < 0) {: the condition has length > 1 and only the first
#> element will be used
#> [1] "values are negative"
```

Проблема возникает из-за того, что условие `x <0` неоднозначно, когда `x` является вектором: проверяете ли вы, чтобы все значения были отрицательными, или проверяете только некоторые из них? R предоставляет вспомогательные функции `all` и `any`, чтобы разрешить эту ситуацию. Они берут вектор логических значений и уменьшают их до одного-единственного значения:

```
x <- c(-2, -1, 0, 1, 2)

if (all(x < 0)) {
 print("all are negative")
}

if (any(x < 0)) {
 print("some are negative")
}
#> [1] "some are negative"
```

См. также

Представленная здесь конструкция `if` предназначена для программирования. Существует также функция `ifelse`, которая реализует векторизованную конструкцию `if/else`, полезную для преобразования целых векторов. *См.* `help(ifelse)`.

15.2. Итерация с помощью цикла

Задача

Вам нужно перебрать элементы вектора или списка.

Решение

Для итерации обычно используется конструкция цикла `for`. Если `v` – это вектор или список, цикл `for` выбирает каждый элемент `v` один за другим, присваивает элемент `x` и что-то с ним делает:

```
for (x in v) {
 # Что-то делаем с x.
}
```

Обсуждение

Программисты, работающие с C и Python, распознают циклы. В R они менее распространены, но тем не менее иногда полезны.

В качестве иллюстрации этот цикл `for` выводит первые пять целых чисел и их квадраты. Он последовательно устанавливает для `x` значения 1, 2, 3, 4 и 5, каждый раз выполняя тело цикла:

```
for (x in 1:5) {
 cat(x, x^2, "\n")
}
#> 1 1
#> 2 4
#> 3 9
#> 4 16
#> 5 25
```

Мы также можем перебирать *индексы* вектора или списка, что полезно для обновления данных на месте. Здесь мы инициализируем *v* с вектором 1:5, затем обновляем его элементы, возводя каждый из них в квадрат:

```
v <- 1:5
for (i in 1:5) {
 v[[i]] <- v[[i]] ^ 2
}
print(v)
#> [1] 1 4 9 16 25
```

Но, честно говоря, это также иллюстрирует одну из причин, почему циклы менее распространены в R, чем в других языках программирования. Векторизованные операции в R бывают быстрыми и легкими, часто полностью устраняя необходимость в использовании циклов. Вот векторизованная версия предыдущего примера:

```
v <- 1:5
v <- v^2
print(v)
#> [1] 1 4 9 16 25
```

См. также

Еще одна причина, по которой циклы встречаются редко, заключается в том, что `map` и аналогичные функции могут обрабатывать целые векторы и списки одновременно, обычно делая это быстрее и проще по сравнению с циклом. *См.* рецепт 6.1 для получения подробной информации об использовании пакета `purrr`, чтобы узнать, как применять функции к спискам.

15.3. Определение функции

Задача

Вы хотите определить новую функцию в R.

Решение

Создайте функцию, используя ключевое слово `function`, за которым следует список имен параметров, а затем тело функции:

```
name <- function(param1, ..., paramN) {
          expr1
          .
          .
          .
          exprM
        }
```

Поставьте круглые скобки вокруг имен параметров и фигурные скобки вокруг тела функции, которое представляет собой последовательность из одного или нескольких выражений. R вычислит каждое выражение по порядку и вернет значение последнего, обозначенное здесь как `exprM`.

Обсуждение

Определения функций – это то, как вы говорите R: «Вот как *это* вычислить». Например, в R нет встроенной функции для вычисления коэффициента вариации, но мы можем создать такую функцию, дав ей имя `cv`:

```
cv <- function(x) {
 sd(x) / mean(x)
}
```

У этой функции один параметр, `x`, а `sd(x) / mean(x)` – это тело функции.

Когда мы вызываем функцию с аргументом, R устанавливает для параметра `x` это значение, а затем вычисляет тело функции:

```
cv(1:10)        # Устанавливаем для x значение 1:10 и вычисляем sd(x)/mean(x).
#> [1] 0.550482
```

Обратите внимание на то, что параметр `x` отличается от любой другой переменной с именем `x`. Например, если в вашей рабочей области есть глобальная перемен-

ная x, то тот x будет отличаться от этого x и не будет зависеть от cv. Кроме того, параметр x существует только во время выполнения функции cv и после этого исчезает.

У функции может быть более одного аргумента. У этой функции два аргумента, оба из которых – целые числа. Она реализует алгоритм Евклида для вычисления их наибольшего общего делителя:

```
gcd <- function(a, b) {
 if (b == 0) {
   a # Возвращаем a вызывающей стороне.
 } else {
    gcd(b, a %% b) # Рекурсивно вызываем сами себя.
 }
}

# Каков наибольший одинаковый знаменатель 14 и 21?
gcd(14, 21)
#> [1] 7
```

(Это определение функции является *рекурсивным*, потому что она вызывает саму себя, когда b отлично от нуля.)

Обычно функция возвращает значение последнего выражения в теле функции. Однако можно вернуть значение раньше, написав `return(expr)`, заставляя функцию остановиться и немедленно вернуть expr вызывающей стороне. Мы можем проиллюстрировать это, кодируя gcd несколько иным способом, используя явный возврат:

```
gcd <- function(a, b) {
 if (b == 0) {
    return(a) # Останавливаемся и возвращаем a.
 }
 gcd(b, a %% b)
}
```

Когда параметр b равен 0, gcd выполняет `return(a)`, немедленно возвращая это значение вызывающей стороне.

См. также

Функции являются центральным компонентом программирования на R, поэтому они хорошо освещены в таких книгах, как *R for Data Science* Хедли Уикхэма и Гаррета Гролемунда (O'Reilly) и *The Art of R Programming* Нормана Мэтлоффа (No Starch Press).

15.4. Создание локальной переменной

Задача

Вы хотите создать переменную, *локальную* для функции, то есть переменную, которая создается внутри функции, используется внутри нее и удаляется по завершении работы функции.

Решение

Внутри функции просто присвойте значение имени. Имя автоматически становится локальной переменной и будет удалено после завершения работы функции.

Обсуждение

Приведенная ниже функция отобразит вектор x в единичный отрезок. Ей требуются два промежуточных значения, low и high:

```
unitInt <- function(x) {
 low <- min(x)
 high <- max(x)
 (x - low) / (high - low)
}
```

Значения low и high автоматически создаются операторами присваивания. Поскольку присваивание происходит в теле функции, эти переменные являются *локальными* для функции. Это дает два важных преимущества.

Во-первых, локальные переменные с именами low и high отличаются от любых глобальных переменных с именами low и high в вашем рабочем пространстве. Поскольку они отличаются, здесь нет «столкновения»: изменения в локальных переменных не изменяют глобальные переменные.

Во-вторых, локальные переменные исчезают, когда функция завершает работу. Это предотвращает беспорядок и автоматически освобождает пространство, которое они использовали.

15.5. Выбор из нескольких альтернатив: функция switch

Задача

Переменная может принимать несколько разных значений. Вы хотите, чтобы ваша программа обрабатывала каждый случай отдельно, в соответствии со значением.

Решение

Функция switch будет создавать ветку значений, позволяя вам выбрать способ обработки каждого случая.

Обсуждение

Первый аргумент функции – это значение, которое R должен рассмотреть. Остальные аргументы показывают, как обрабатывать каждое возможное значение. Например, в этом вызове функции switch мы рассматриваем значение who, а затем возвращаем один из трех возможных результатов:

```
hair_type = switch(who,
                   Moe = "long",
                   Larry = "fuzzy",
                   Curly = "none")
```

Обратите внимание, что каждое выражение после who, которое идет вначале, помечено его возможным значением. Если who – это Moe, функция switch возвращает значение "long"; если это Larry, возвращается значение "fuzzy"; если это Curly, возвращается значение "none".

Очень часто нельзя ожидать, что будут рассмотрены все возможные значения, поэтому функция switch позволяет вам определить значение по умолчанию для си-

туации, когда ни одна метка не совпадает. Говоря проще, последний по умолчанию без метки. В этом примере мы преобразуем содержимое s из "one", "two" или "three" в соответствующее целое число и возвращаем NA для любого другого значения:

```
num <- switch(s,
              one = 1,
              two = 2,
              three = 3,
              NA)
```

Раздражающая особенность этой функции проявляется, когда метки являются целыми числами. Например, этот код не будет делать то, что вы ожидаете:

```
switch(i,           # Работает не так, как вы ожидали.
       10 = "ten",
       20 = "twenty",
       30 = "thirty",
       "other")
```

Но есть обходной путь – преобразуйте целое число в строку символов, а затем используйте строки символов для меток:

```
switch(as.character(i),
       "10" = "ten",
       "20" = "twenty",
       "30" = "thirty",
       "other")
```

См. также

См. help(switch) для получения более подробной информации.

Такого рода особенность довольно распространена в других языках программирования, где ее обычно называют оператором *switch* или *case*.

Функция switch работает только со скалярами. Использование ее с таблицей данных – вещь более сложная. Обратите внимание на функцию case_when из пакета dplyr, которая предоставляет мощный механизм, чтобы справиться с этой ситуацией.

15.6. Определение значений по умолчанию для параметров функции

Задача

Вы хотите определить параметры по умолчанию для функции, то есть значения, которые будут использоваться, когда вызывающая сторона не предоставляет явных аргументов.

Решение

R позволяет устанавливать значения по умолчанию для параметров, включив их в определение function:

```
my_fun <- function(param = default_value) {
  ...
}
```

Обсуждение

Давайте создадим небольшую функцию, которая приветствует кого-то по имени:

```
greet <- function(name) {
 cat("Hello,", name, "\n")
}

greet("Fred")
#> Hello, Fred
```

Если мы вызовем `greet` без аргумента `name`, то получим вот такую ошибку:

```
greet()
#> Error in cat("Hello,", name, "\n") :
#> argument "name" is missing, with no default
```

Однако можно изменить определение функции, чтобы определить имя по умолчанию. В этом случае мы по умолчанию используем общее имя `world`:

```
greet <- function(name = "world") {
 cat("Hello,", name, "\n")
}
```

Теперь, если мы опускаем аргумент, R предоставляет значение по умолчанию:

```
greet()
#> Hello, world
```

Это удобный механизм для параметров по умолчанию. Тем не менее мы рекомендуем использовать его разумно. Мы видели слишком много случаев, когда *создатель* функции определял значения по умолчанию, а *вызывающая сторона* принимала эти значения без особого раздумывания, что приводило к сомнительным результатам. Например, если вы используете алгоритм *k*-ближайших соседей, выбор *k* имеет решающее значение и предоставление значения по умолчанию не имеет смысла. Иногда лучше заставить вызывающую сторону сделать выбор.

15.7. Подать сигнал с помощью сообщения об ошибке

Задача

Когда ваш код сталкивается с серьезной проблемой, вы хотите остановить и предупредить пользователя.

Решение

Вызовите функцию `stop`, которая выведет ваше сообщение и прекратит всю обработку.

Обсуждение

Крайне важно остановить обработку, когда ваш код сталкивается с фатальными ошибками, как, например, проверка того, что в учетной записи все еще присутствует положительный баланс:

```
if (balance < 0) {
  stop("Funds exhausted.")
}
```

В результате этого вызова функции `stop` будет отображено сообщение, обработка будет завершена, а пользователь вернется в командную строку консоли:

```
#> Error in eval(expr, envir, enclos): Funds exhausted
```

Проблемы возникают по разным причинам: неверные данные, ошибка пользователя, сбои в сети и недочеты в коде, среди прочего. Этот список бесконечен. Важно, чтобы вы предвидели потенциальные проблемы и писали код соответствующим образом:

Обнаружение

Как минимум, обнаружение возможных ошибок. Остановитесь, если дальнейшая обработка невозможна. Необнаруженные ошибки являются основным источником программных сбоев.

Отчет

Если вам нужно остановиться, дайте пользователям разумное объяснение, почему. Это поможет им диагностировать и устранить проблему.

Восстановление

В некоторых случаях код может сам исправить ситуацию и продолжить работу. Однако мы рекомендуем предупредить пользователя о том, что ваш код столкнулся с проблемой и она была исправлена.

Обработка ошибок – это часть *безопасного программирования*, практики создания надежного кода.

См. также

Альтернативной функцией `stop` является функция `warning`, которая выводит свое сообщение и продолжает работу без остановки. Однако убедитесь в том, что продолжать на самом деле будет разумно.

15.8. Защита от ошибок

Задача

Вы предвидите возможность фатальных ошибок и хотите устранить их, вместо того чтобы останавливать их.

Решение

Используйте функцию `possibly`, чтобы «обернуть» проблемный код. Она будет перехватывать ошибки и позволять вам на них реагировать.

Обсуждение

Пакет `purrr` содержит функцию под названием `possibly`, которая принимает два параметра. Первый параметр – это функция, и в этой функции `possibly` будет осуществлять защиту от сбоев. Второй параметр – значение под названием `otherwise`.

Здесь будет полезен конкретный пример. Функция `read.csv` пытается прочитать содержимое файла, но просто останавливается, если файл не существует. Это может быть нежелательно. Возможно, мы, наоборот, хотим продолжить.

Можно «обернуть» функцию `read.csv` в защитный слой следующим образом:

```
library(purrr)
safe_read <- possibly(read.csv, otherwise=NULL)
```

Это может показаться странным, но функция `possibly` возвращает *новую функцию*. Новая функция, которая здесь носит имя `safe_read`, ведет себя точно так же, как и старая функция `read.csv`, но с одним очень важным отличием. Когда `read.csv` завершится с ошибкой и остановится, `safe_read` вернет значение `otherwise` (`NULL`) и позволит вам продолжить. (Если все прошло успешно, вы получите свой обычный результат: таблицу данных.)

Можно использовать функцию `safe_read` для обработки необязательных файлов:

```
details = safe_read("details.csv") # Попытка прочитать содержимое файла details.csv.
if (is.null(details)) { # NULL означает неудачу.
  cat("Details are not available\n")
} else {
  print(details) # У нас есть содержимое!
}
```

Если файл *details.csv* существует, функция `safe_read` возвращает содержимое, и этот код выведет его. Если такого файла не существует, `read.csv` завершается ошибкой, `safe_read` возвращает значение `NULL`, и этот код выводит сообщение.

Значение `otherwise` в этом случае равно `NULL`, но оно может быть любым. Например, это может быть таблица данных, которая обеспечивает значение по умолчанию. В этом случае, когда файл *details.csv* недоступен, `safe_read` вернет это значение по умолчанию.

См. также

Пакет `purrr` содержит другие функции для защиты от ошибок. Обратите внимание на функции `safely` и `quietly`.

Если вам нужны еще более мощные инструменты, воспользуйтесь `help(tryCatch)`, чтобы увидеть механизм, который стоит за функцией `possibly`. Он обладает сложными приемами для обработки ошибок и предупреждений и отражает знакомую парадигму `try/catch` других языков программирования.

15.9. Создание анонимной функции

Задача

Вы используете функции tidyverse, такие как `map` или `discard`, которые в качестве параметра требуют функцию. Вам нужен ярлык для простого определения требуемой функции.

Решение

Используйте ключевое слово `function`, чтобы определить функцию с параметрами и телом, но, вместо того чтобы дать функции имя, просто определите её по месту.

Обсуждение

Создание функции без имени может показаться странным, но это может быть удобно. В рецепте 15.3 мы определили функцию `is_na_or_null` и использовали ее для удаления элементов `NA` и `NULL` из списка:

```
is_na_or_null <- function(x) {
 is.na(x) || is.null(x)
}

lst %>%
 discard(is_na_or_null)
```

Иногда написание крошечной одноразовой функции, такой как `is_na_or_null`, раздражает. Этой проблемы можно избежать, напрямую используя определение функции, не давая ей ее имени:

```
lst %>%
discard(function(x) is.na(x) || is.null(x))
```

Такой вид функции называется *анонимной функцией* по очевидной причине: у нее нет имени.

См. также

Определения функций описаны в рецепте 15.3.

15.10. Создание коллекции многократно используемых функций

Задача

Вы хотите повторно использовать одну или несколько функций в нескольких сценариях.

Решение

Сохраните функции в локальном файле, скажем *myLibrary.R*, а затем используйте функцию `source`, чтобы загрузить эти функции в свой сценарий:

```
source("myLibrary.R")
```

Обсуждение

Довольно часто вы будете писать функции, которые будут полезны в нескольких сценариях. Например, у вас может быть одна функция, которая загружает, проверяет и очищает ваши данные; теперь вы хотите повторно использовать эту функцию в каждом сценарии, который нуждается в данных.

Большинство новичков просто вырезают и повторно вставляют используемую функцию в каждый сценарий, дублируя код. Это создает серьезную проблему. Что, если вы обнаружите ошибку в этом дублированном коде? Или что, если вы должны изменить код, чтобы приспособиться к новым обстоятельствам? Вы будете вынуждены отслеживать каждую копию и вносить одинаковые изменения повсюду. Это раздражает, и такой код подвержен ошибкам.

Вместо этого создайте файл, скажем *myLibrary.R*, и сохраните определение функции там. Содержимое этого файла может выглядеть так:

```
loadMyData <- function() {
   # Здесь идет код для загрузки, проверки и очистки данных.
}
```

Затем, внутри каждого сценария, используйте функцию `source` для чтения кода из файла:

```
source("myLibrary.R")
```

Когда вы запускаете этот скрипт, функция `source` читает указанный файл, как если бы вы набирали содержимое файла в этом месте в сценарии. Это лучше, чем вырезание и вставка, потому что вы изолировали определение функции в одном известном месте.

В этом примере в исходном файле только одна функция, но файл, конечно же, может содержать несколько функций. Мы предлагаем собрать связанные функции в собственный файл, создав группу связанных многоразовых функций.

См. также

Этот рецепт представляет собой очень простой метод повторного использования кода, который подходит для небольших проектов. Более мощный подход заключается в создании собственного пакета функций R, который особенно полезен при сотрудничестве с другими людьми. Создание пакета – большая тема, но начать довольно легко. Предлагаем вам прочитать отличную книгу *R Packages* Хэдли Уикхема (O'Reilly), которая доступна в печатном виде или онлайн (http://r-pkgs.had.co.nz).

15.11. Автоматическое форматирование кода

Задача

Вы хотите переформатировать свой код так, чтобы все отступы были выставлены правильно.

Решение

Для последовательного отступа блока кода выделите текст в RStudio, затем нажмите сочетание клавиш **Ctrl-I** (в Windows или Linux) или **Cmd-I** (в Mac).

Обсуждение

Одна из множества особенностей интегрированной среды разработки RStudio заключается в том, что она помогает в повседневном обслуживании кода, например переформатировании. Когда вы редактируете код, легко получить отступы, которые непоследовательны и немного сбивают с толку. Интегрированная среда разработки может это исправить.

Возьмем, к примеру, этот код:

```
for (i in 1:5) {
    if (i >= 3) {
  print(i**2)
} else {
  print(i * 3)
}

  }
```

Несмотря на то что это правильный код, его может быть сложно читать из-за странного отступа. Если мы выделим текст в RStudio и нажмем клавиши **Ctrl-I** (или **Cmd-I** в Mac), то наш код получит последовательный отступ:

```
for (i in 1:5) {
 if (i >= 3) {
   print(i**2)
 }
 else {
   print(i * 3)
 }
}
```

См. также

В RStudio есть несколько полезных функций для редактирования кода. Вы можете получить доступ к шпаргалкам, нажав **Help** → **Cheatsheets** (Справка → Шпаргалки), или перейдите непосредственно по адресу https://www.rstudio.com/resources/cheatsheets/.

Глава 16

R Markdown и публикации

Хотя R сам по себе является невероятно мощным инструментом для анализа и визуализации данных, почти всем нам после проведения анализа потребуется сообщить результаты другим. Это можно сделать с помощью публикаций, сообщений в блогах, презентаций в PowerPoint или книг. R Markdown – это инструмент, который помогает нам перейти от анализа и визуализации в R к публикуемым документам.

R Markdown – это пакет (а также экосистема инструментов), который позволяет нам добавлять код R в текстовый файл с форматированием языка разметки Markdown. Затем этот документ может быть представлен во многих различных выходных форматах, включая PDF, HTML, Microsoft Word и Microsoft PowerPoint. При рендеринге, также называемом *вязанием*, выполняется код R, а полученный результат и графики помещаются в итоговый документ.

В этой главе мы дадим вам рецепты, которые помогут вам приступить к созданию документов R Markdown. После того как вы ознакомитесь с этими рецептами, один из лучших способов узнать больше о R Markdown – это просмотреть исходные файлы и окончательный вывод работы R Markdown у других людей. Книга, которую вы читаете, была написана на R Markdown. Исходник этой книги можно увидеть на сайте GitHub (https://github.com/CerebralMastication/R-Cookbook).

Кроме того, Йихуэ Цзе, Дж. Дж. Аллер и Гаррет Гроулмунд, авторы книги *R Markdown: The Definition Guide* (https://bookdown.org/yihui/rmarkdown/) (Chapman & Hall / CRC), также выложили свой исходник на сайте GitHub (https://github.com/rstudio/rmarkdown-book).

Множество других книг, написанных с помощью R Markdown, есть в свободном доступе в интернете (https://bookdown.org).

Мы упомянули, что R Markdown – это не только пакет, но и экосистема. Существуют специализированные пакеты для расширения R Markdown для ведения блогов (`blogdown`), для книг (`book down`) и для создания панелей с сеткой (`flexdashboard`). Первоначальный пакет в экосистеме называется `knitr`, а мы по-прежнему называем процесс превращения R Markdown в окончательный формат «связыванием» документа. Экосистема R Markdown поддерживает множество выходных форматов, и рассказывать обо всех них было бы неразумно. В этой книге мы в основном будем придерживаться четырех распространенных форматов вывода: HTML, LaTeX, Microsoft Word и Microsoft PowerPoint.

Интегрированная среда разработки RStudio содержит множество полезных функций для создания и редактирования документов R Markdown. Хотя мы будем использовать эти функции в последующих рецептах, R Markdown не зависит от RStudio, чтобы быть полезным. Можно редактировать простые текстовые файлы R Markdown в своем любимом текстовом редакторе, а затем связать документ с помощью интерфейса командной строки R. Однако инструменты RStudio настолько полезны, что мы расскажем о них подробно.

16.1. Создание нового документа

Задача

Вы хотите создать новый документ R Markdown, чтобы рассказать историю ваших данных.

Решение

Самым простым способом создания нового документа R Markdown является использование меню **File** → **New File** → **R Markdown**... (Файл → Новый файл → R Markdown...) в интегрированной среде разработки RStudio (*см.* рис. 16-1).

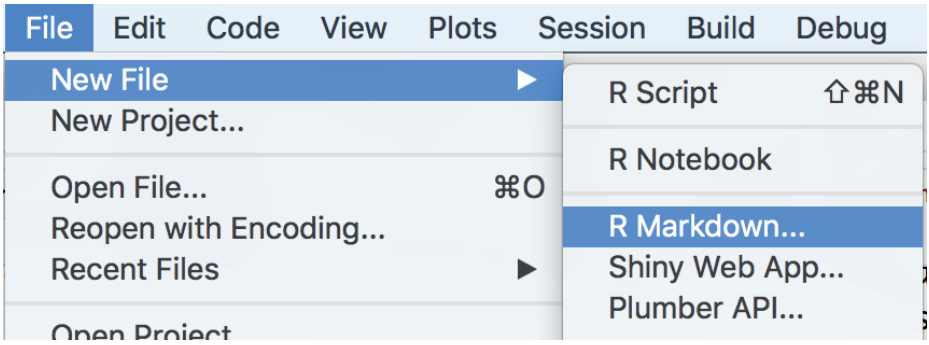


Рис. 16-1. Создание нового документа R Markdown

После выбора «R Markdown...» откроется новое диалоговое окно **New** R Markdown, где можно выбрать тип выходного документа, который вы хотите создать (*см.* рис. 16-2). Параметр по умолчанию – HTML, который является хорошим вариантом, если вы хотите опубликовать свою работу в интернете или сообщении электронной почты, или если вы еще не решили, в каком виде хотите вывести финальный документ. Позже перейти на другой формат обычно так же просто, как связать одну строку текста в документе или сделать несколько кликов в среде разработки.

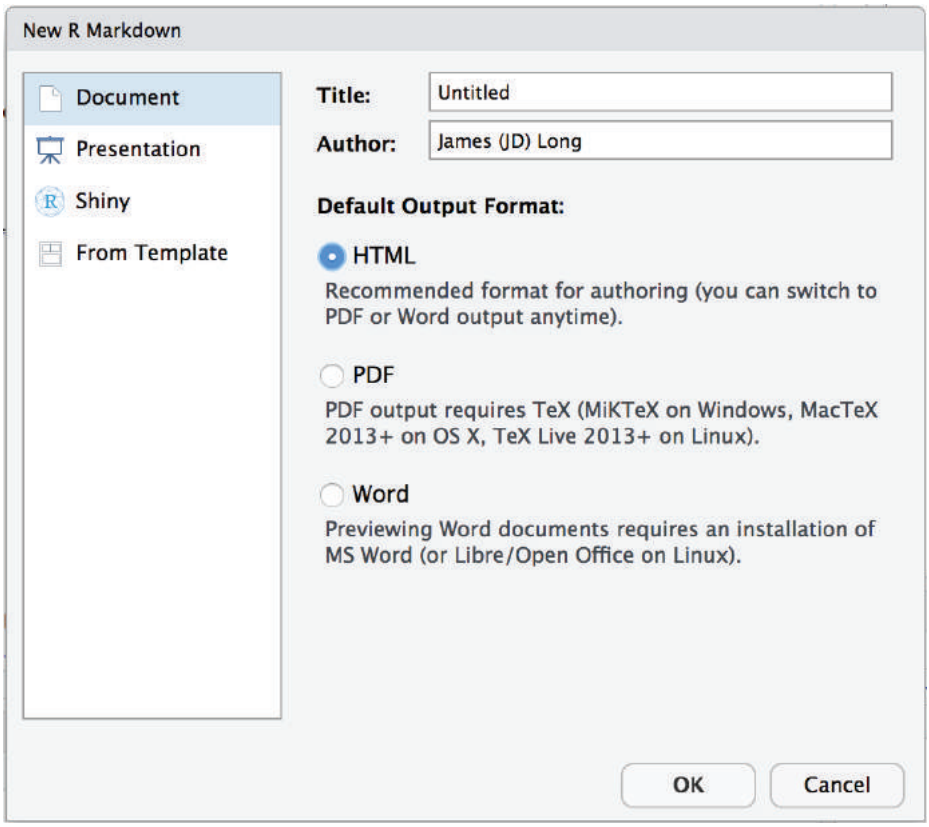


Рис. 16-2. Параметры нового документа R Markdown

После того как вы сделаете выбор и нажмете **OK**, вы получите шаблон R Markdown с метаданными и примером текста (*см.* рис. 16-3).

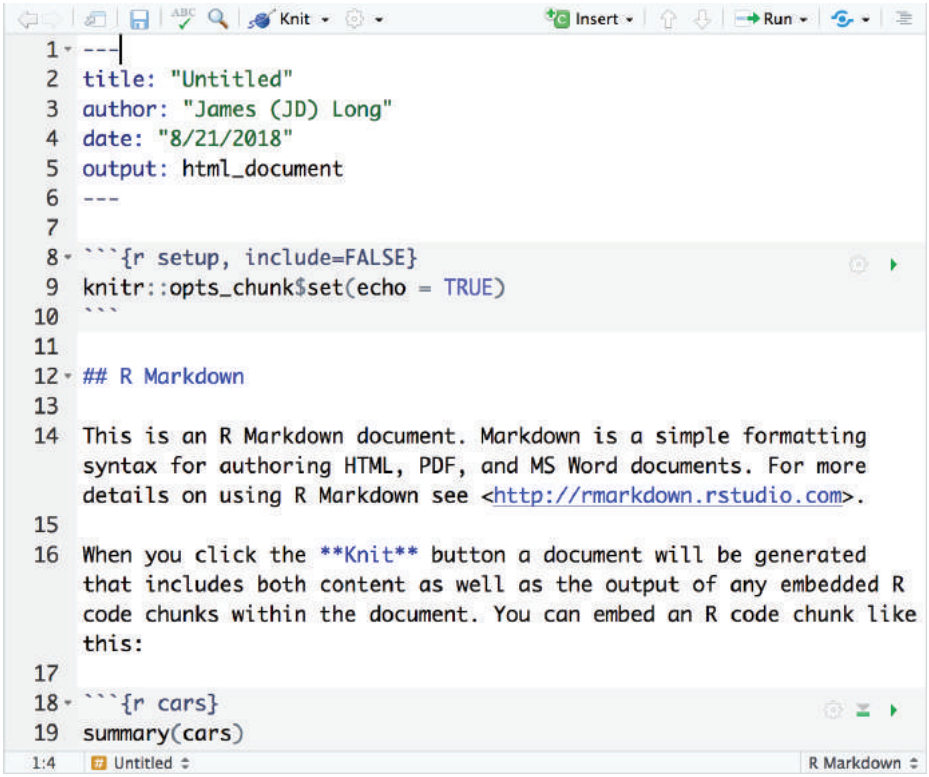


Рис. 16-3. Новый документ R Markdown

Обсуждение

Документы R Markdown представляют собой текстовые файлы. Только что обозначенный ярлык – это самый быстрый способ получить шаблон для создания нового текстового документа R Markdown. Когда у вас есть шаблон, вы можете редактировать текст, изменять код R и менять все, что хотите. Другие рецепты в этой главе посвящены тем вещам, которые вы, вероятно, захотите сделать в своем документе R Markdown, но если вы просто хотите посмотреть, как выглядят выходные данные, нажмите кнопку **Knit** в среде разработки RStudio – и вы увидите документ R Markdown в желаемом выходном формате.

16.2. Добавление заголовка, автора или даты

Задача

Вы хотите изменить название, автора или дату вашего документа.

Решение

В верхней части документа R Markdown находится блок специально отформатированного текста, который начинается и заканчивается знаком ---. Этот блок содержит важные метаданные о вашем документе.

В этом блоке вы можете установить заголовок, автора и дату:

```
---
title: "Your Title Here"
author: "Your Name Here"
date: "12/31/9999"
output: html_document
---
```

Вы также можете установить выходной формат (например, `output: html_document`). Мы обсудим различные форматы вывода позже в рецептах, которые охватывают определенные форматы.

Обсуждение

Когда вы свяжете свой документ R Markdown для создания выходных данных, R будет запускать каждый блок, создавать документ Markdown (а не R Markdown) для каждого вывода блока и передавать весь документ в Pandoc. Pandoc – это программа, которая создает окончательный выходной документ из промежуточного документа Markdown. В большинстве случаев вам даже не нужно думать об этих шагах, если только у вас нет проблем с вязанием документа.

Текст в верхней части документа R Markdown между знаками --- находится в формате YAML (это еще один язык разметки). Этот блок используется для передачи метаданных в программное обеспечение Pandoc, которое создает ваш выходной документ. Pandoc считывает поля `title`, `author` и `date`, и они вставляются вверху в большинстве форматов выходных документов.

То, как эти значения форматируются и вставляются в выходной документ, является функцией шаблона, используемого для вывода. Шаблоны по умолчанию для HTML, PDF и Microsoft Word форматируют поля заголовка, автора и даты одинаково (*см.* рис. 16-4).

Рис. 16-4. Иллюстрация заголовка

Вы можете добавить другие пары «ключ/значение» в заголовок YAML, но если ваш шаблон не настроен на использование этих значений, они будут проигнорированы.

См. также

Для получения информации о создании собственных шаблонов *см.* главу 17 «Шаблоны документов» в книге *R Markdown: The Definitive Guide* (https://bookdown.org/yihui/rmarkdown/document-templates.html).

16.3. Форматирование текста документа

Задача

Вы хотите отформатировать текст документа, например выделив его курсивом или жирным шрифтом.

Решение

Тело документа R Markdown представляет собой простой текст. Его можно форматировать с использованием нотации Markdown. Скорее всего, вы захотите добавить форматирование, например выделение текста **полужирным шрифтом** или *курсивом*. Вы также захотите добавить заголовки разделов, списки и таблицы, которые будут рассмотрены в следующих рецептах. Все это можно сделать с помощью Markdown.

В табл. 16-1 приводится краткое изложение примеров наиболее распространенного синтаксиса форматирования.

Таблица 16-1. Синтаксис форматирования Markdown

Markdown	Вывод
`plain text`	обычный текст
`*italics*`	*курсив*
`**bold**`	**полужирный шрифт**
`'code'`	код
`sub~script~`	_{нижний} индекс
`super^script^`	^{верхний} индекс
`~~strikethrough~~`	~~зачёркивание~~
`endash: --`	короткое тире: –
`emdash: ---`	длинное тире: —

См. также

RStudio публикует удобное справочное руководство (https://rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf).

См. также рецепты, чтобы узнать, как вставлять различные структуры, например рецепты 16.4, 16.5 и 16.9.

16.4. Вставка заголовков документов

Задача

Ваш документ R Markdown нуждается в заголовках разделов.

Решение

Вы можете вставить заголовки разделов, начав строку с символа # (хеш). Используйте один хеш-символ для верхнего уровня, два для второго уровня и т. д.:

```
# Level 1 Heading
## Level 2 Heading
```

```
### Level 3 Heading
#### Level 4 Heading
##### Level 5 Heading
###### Level 6 Heading
```

Обсуждение

Markdown и HTML поддерживают до шести уровней заголовков, так что это то, что поддерживается в R Markdown. В R Markdown (и Markdown в целом) форматирование не включает в себя конкретные детали шрифта; оно только сообщает, какой класс форматирования применить к тексту. Специфика каждого класса определяется выходным форматом и шаблоном, используемым каждым таким форматом.

16.5. Вставка списка

Задача

Вы хотите включить в свой документ маркированный или нумерованный список.

Решение

Чтобы создать маркированный список, начинайте каждую строку со знака звездочки (*), например:

```
* first item
* second item
* third item
```

Чтобы создать нумерованный список, начинайте каждую строку с 1.:

```
1. first item
1. second item
1. third item
```

R Markdown заменит префиксы 1. последовательностью 1., 2., 3. и т. д.
Правила для списков несколько строги:

- *перед* списком должна быть пустая строка;
- *после* списка должна быть пустая строка;
- после ведущей звездочки должен быть символ пробела.

Обсуждение

Синтаксис списков прост, но следите за правилами, приведенными в решении. Если вы нарушите хотя бы одно из них, на выходе получится абракадабра.

Важной особенностью списков является то, что они допускают наличие подсписков. У этого маркированного списка три подпункта:

```
* first item
 * first subitem
 * second subitem
 * third subitem
* second item
```

Что дает вот такой вывод:

- первый пункт
  - первый подпункт
  - второй подпункт
  - третий подпункт
- второй пункт

Опять же, есть важное правило: подсписки должны иметь отступ в два, три или четыре пробела относительно уровня выше. Не больше и не меньше, иначе наступит хаос.

В решении мы рекомендуем использовать префикс 1. для идентификации нумерованных списков. Вы также можете использовать `a.` и `i.`, что приведет к появлению строчных букв и последовательности римских цифр соответственно. Это удобно для форматирования подсписков:

```
1. first item
1. second item
    a. subitem 1
    a. subitem 2
      i. sub-subitem 1
      i. sub-subitem 2
    a. subitem 2
1. third item
```

Это дает следующее:

1. первый пункт
2. второй пункт
   a. подпункт 1
   b. подпункт 2
      i. подподпункт 1
      ii. подподпункт 2
   c. подпункт 2
3. третий пункт

См. также

Синтаксис списков является более гибким и функциональным, чем описано здесь. *См.* подробности в справочных материалах, например здесь: https://pandoc.org/MANUAL.html#pandocs-markdown.

16.6. Вывод результатов из кода R

Задача

Вы хотите выполнить некий код и показать результаты в выходном документе.

Решение

Можно вставить код R в документ R Markdown. Он будет выполнен, а выходные данные будут включены в итоговый документ.

Есть два способа вставить код. Если это маленькие фрагменты кода, вставьте их между двумя отметками (``), как показано здесь:

```
The square root of pi is 'r sqrt(pi)'.
```

что приведет к такому результату:

```
The square root of pi is 1.772.
```

Для больших блоков кода определите *кусок кода*, поместив блок между соответствующими тройными отметками (```).

```
'''{r}
# Здесь идет блок кода.
'''
```

Обратите внимание на знак {r} после первой тройной отметки: он предупреждает R Markdown, что мы хотим, чтобы он выполнил код.

Обсуждение

Внедрение кода R в ваш документ – самая мощная функция R Markdown. Фактически без этого R Markdown был бы просто старым языком разметки Markdown. Inline R, описанный в начале решения, полезен для ввода небольших фрагментов информации непосредственно в текст отчета – это такая информация, как даты, время или результаты небольших вычислений.

Куски кода предназначены для выполнения тяжелой работы. По умолчанию фрагмент кода отображается в тексте, а результаты отображаются непосредственно под кодом. Результату предшествует префикс, который по умолчанию имеет двойной хештег: ##.

Если бы у нас был такой фрагмент кода в исходном документе R Markdown:

```
'''{r}
sqrt(pi)
sqrt(1:5)
'''
```

он бы дал этот вывод:

```
sqrt(pi)
## [1] 1.77
sqrt(1:5)
## [1] 1.00 1.41 1.73 2.00 2.24
```

Удобно, когда результаты, начинающиеся с ##, позволяют читателю вставлять код и результаты в собственный сеанс R и выполнить код. R будет игнорировать эти результаты, потому что они выглядят как комментарии.

Знак {r} после штрихов важен, поскольку R Markdown допускает наличие блоков кода из других языков, таких как Python или SQL. Если вы работаете в мультиязычной среде, это очень мощная функция. *См.* подробности в документации R Markdown.

См. также

См. рецепт 16.7, чтобы узнать, как контролировать то, что показано в выводе.

Для получения подробной информации о доступных языковых движках *см.* https://bookdown.org/yihui/rmarkdown/language-engines.html.

16.7. Контролируем, какой код и результаты отображаются

Задача

Ваш документ содержит куски кода R. Вы хотите контролировать то, что показано в итоговом документе: только результаты, только код либо ни то, ни другое.

Решение

Блоки кода поддерживают несколько параметров, которые управляют тем, что появляется в конечном документе. Установите параметры в верхней части блока. Например, в этом блоке для `echo` установлено значение `FALSE`:

```
'''{r echo=FALSE}
# . . . идущий здесь код не появится в выводе . . .
'''
```

См. обсуждение таблицы доступных вариантов.

Обсуждение

Есть множество параметров отображения, таких как `echo`, который контролирует, появляется ли сам код в конечном выводе, и `eval`, который контролирует, будет ли код вычисляться (выполняться).

Некоторые из наиболее популярных вариантов перечислены в табл. 16-2.

Таблица 16-2. Параметры, управляющие тем, что показано в итоговом документе

Параметр	Выполняет код	Показывает код	Показывает выходной текст	Показывает цифры
`results='hide'`	X	X		X
`include=FALSE`	X			
`echo=FALSE`	X		X	X
`fig.show='hide'`	X	X	X	
`eval=FALSE`		X		

Вы можете смешивать и сочетать комбинации параметров, чтобы получить нужные вам результаты. Вот некоторые распространенные случаи использования:

- вы хотите, чтобы появился вывод кода, но не сам код: `echo=FALSE`;
- вы хотите, чтобы код появлялся, но не выполнялся: `eval=FALSE`;
- вы хотите выполнить код для побочных эффектов (например, загрузка пакетов или загрузка данных), но ни код, ни какой-либо случайный вывод не должны появиться: `include=FALSE`.

Мы часто используем параметр `include=FALSE` для первого фрагмента кода документа R Markdown, где мы вызываем функцию `library`, инициализируем переменные и выполняем другие вспомогательные задачи, случайный вывод которых просто раздражает.

В дополнение к только что описанным опциям вывода есть несколько опций, которые управляют обработкой сообщений об ошибках, предупреждений и информационных сообщений, генерируемых вашим кодом:

- `error=TRUE` позволяет полностью создать документ, даже если в блоке кода есть ошибка. Это полезно, когда вы создаете документ, в котором вы конкретно хотите увидеть ошибку в выводе. По умолчанию это `error=FALSE`;
- `warning=FALSE` подавляет предупреждения. По умолчанию это `warning=TRUE`;
- `message=FALSE` подавляет информационные сообщения. Это удобно, когда ваш код использует болтливые пакеты, которые выдают сообщения при загрузке. По умолчанию это `message=TRUE`.

См. также

Шпаргалка R Markdown от RStudio содержит множество доступных опций. Автор `knitr`, Йихуэй Цзи, задокументировал параметры на своем веб-сайте (https://yihui.name/knitr/options/).

16.8. Вставка графика

Задача

Вы хотите вставить график в свой выходной документ.

Решение

Просто создайте фрагмент кода, который создает график, и вставьте этот фрагмент в документ R Markdown. R Markdown зафиксирует график и вставит его в ваш выходной документ.

Обсуждение

Вот фрагмент кода R Markdown, который создает график `ggplot` с именем `gg`, а затем «распечатывает» его:

```
'''{r}
library(ggplot2)
gg <- ggplot(airquality, aes(Wind, Temp)) + geom_point()
print(gg)
'''
```

Напомним, что строка `print(gg)` отображает график. Если мы вставим этот фрагмент кода в документ R Markdown, R Markdown зафиксирует результат и вставит его в вывод, который выглядит примерно так:

```
library(ggplot2)
gg <- ggplot(airquality, aes(Wind, Temp)) + geom_point()
print(gg)
```

Получившийся график показан на рис. 16-5.

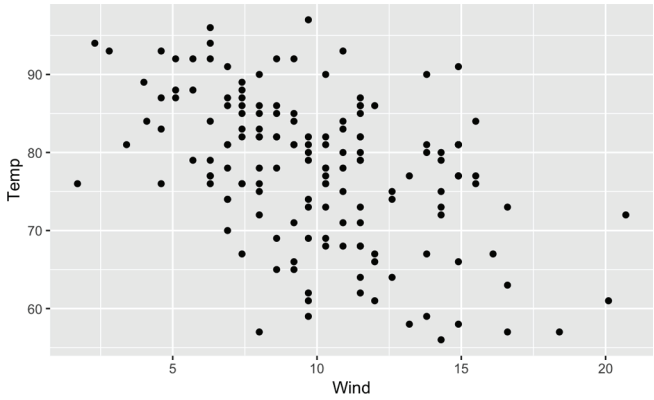


Рис. 16-5. Пример диаграммы ggplot в R Markdown

Почти любой график, который мы можем создать в R, может быть визуализирован в выходной документ. У нас есть некоторый контроль над отображаемыми результатами с использованием параметров в блоке кода, таких как установка размера, разрешения и формата вывода. Давайте посмотрим на несколько примеров, используя объект `gg plot`, который мы только что создали.

Мы можем уменьшить вывод, используя `out.width`:

```
'''{r out.width='30%'}
print(gg)
'''
```

В результате чего получается график, изображенный на рис. 16-6:

```
print(gg)
```

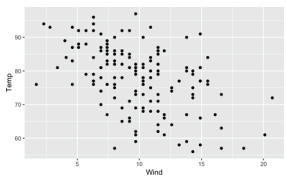


Рис. 16-6. График малой ширины

Или можно увеличить вывод на всю ширину страницы:

```
'''{r out.width='100%'}
print(gg)
'''
```

В результате чего получится график, изображенный на рис. 16-7:

```
print(gg)
```

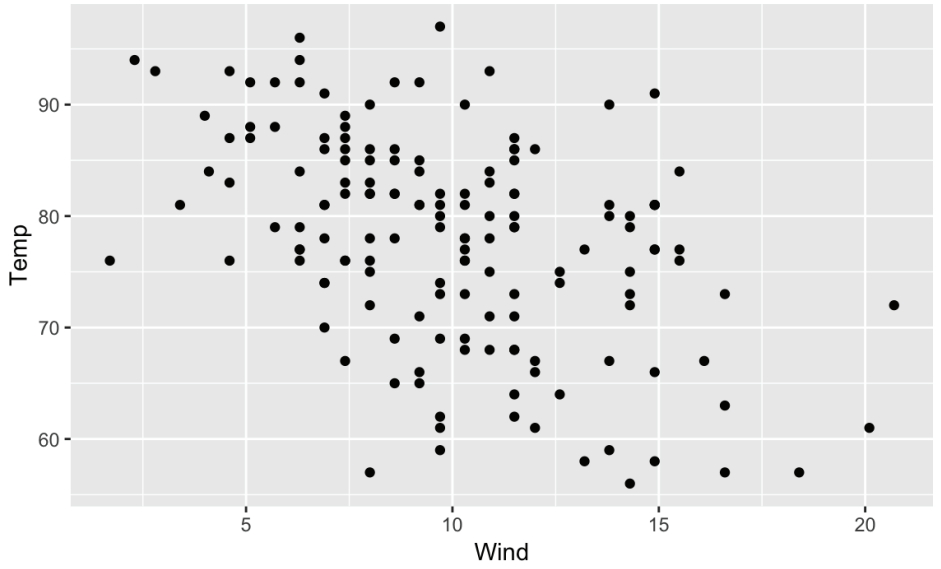


Рис. 16-7. График большой ширины

Вот некоторые распространенные настройки вывода для использования с графиками:

`out.width` и `out.height`

Размер выходного рисунка в процентах от размера страницы.

`dev`

Графическое устройство R, используемое для создания рисунка. По умолчанию это вывод в формате `png` для HTML и в формате `pdf` для LaTeX. Также можно использовать, например, `'jpg'` или `'svg'`.

`fig.cap`

Подпись к рисунку.

`fig.align`

Выравнивание графика: `'left'`, `'center'` или `'right'`.

Давайте воспользуемся этими настройками для создания графика с шириной 50 %, высотой 20 %, подписью и выравниванием по левому краю:

```
'''{r out.width='50%',
out.height='20%',
fig.cap='Temperature versus wind speed',
fig.align='left'}
print(gg)
```

В результате получится график, изображенный на рис. 16-8:

```
print(gg)
```

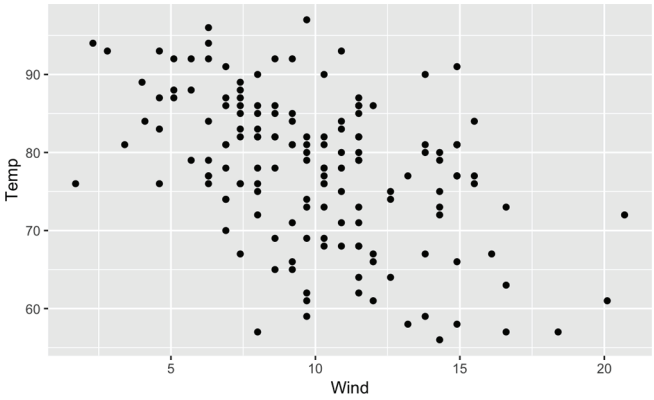


Рис. 16-8. Температура против скорости ветра

16.9. Вставка таблицы

Задача

Вы хотите вставить красиво отформатированную таблицу в ваш документ.

Решение

Разложите содержимое в текстовой таблице, используя символ вертикальной черты (|) для разделения столбцов. Используйте тире, чтобы «подчеркнуть» заголовки столбцов. R Markdown отформатирует все это в привлекательный вывод. Например, это:

```
| Stooge | Year | Hair?           |
|--------|------|-----------------|
| Moe    | 1887 | Yes             |
| Larry  | 1902 | Yes             |
| Curly  | 1903 | No (ironically) |
```

превратится в это:

Stooge	Year	Hair?
Moe	1887	Yes
Larry	1902	Yes
Curly	1903	No (ironically)

Вы *должны* поместить пустую строку до и после таблицы.

Обсуждение

Синтаксис таблиц позволяет «рисовать» таблицу, используя символы ASCII. «Подчеркивание», созданное штрихами, является сигналом для R Markdown, что строка над ним содержит заголовки столбцов. Без этого «подчеркивания» R Markdown будет интерпретировать первую строку как содержимое, а не заголовки.

Форматирование таблицы немного более гибкое, чем может предложить это решение. Этот (некрасивый) ввод, например, будет давать такой же (красивый) вывод, как показано в решении:

```
| Stooge | Year | Hair?  |
|--------|------|----------------|
| Moe  | 1887 | Yes  |
| Larry  | 1902 | Yes  |
| Curly  | 1903 | No (ironically) |
```

Компьютер волнуют только символы вертикальной черты (|) и тире. Наличие пробелов необязательно. Используйте их, чтобы вам легче было читать ввод.

Удобной функцией является использование двоеточий (:) для управления выравниванием столбцов. Включите столбцы в тире «подчеркивание», чтобы установить выравнивание столбцов. Эта таблица определяет выравнивание для трех из четырех столбцов:

```
|Left   |Right | Center  | Default |
|:------|-----:|:-------:|---------|
| 12345 |12345 | 12345   | 12345   |
| text  | text | text    | text    |
| 12    | 12   | 12      | 12      |
```

что дает такой результат:

Left	Right	Center	Default
12345	12345	12345	12345
text	text	text	text
12	12	12	12

Используйте двоеточия внутри «подчеркивания» заголовка столбца следующим образом:

- двоеточие в крайнем левом конце дает выравнивание по левому краю;
- двоеточие в крайнем правом конце дает выравнивание по правому краю;
- двоеточие на обоих концах дает выравнивание по центру.

См. также

На самом деле R Markdown поддерживает несколько синтаксисов таблиц – некоторые скажут: *ошеломляющее* количество синтаксисов. Этот рецепт показывает только один вариант, лишь для простоты. *См.* справочные материалы по Markdown, чтобы узнать об альтернативных вариантах.

16.10. Вставка таблицы данных

Задача

Вы хотите включить таблицу сгенерированных компьютером данных в ваш выходной документ.

Решение

Используйте функцию kable из пакета knitr. Здесь показано, как она форматирует таблицу данных с именем dfrm:

```
library(knitr)
kable(dfrm)
```

Обсуждение

В рецепте 16.9 мы показали, как поместить статическую таблицу в документ, используя простой текст. Здесь у нас есть содержимое таблицы в таблице данных, и мы хотим показать данные в выводе документа.

Можно было бы просто вывести таблицу, и в результате она оказалась бы в неотформатированном виде:

```
myTable <- tibble(
 x=c(1.111, 2.222, 3.333),
 y=c('one', 'two', 'three'),
 z=c(pi, 2*pi, 3*pi))
myTable
#> # Tibble: 3 x 3
#>       x y       z
#>   <dbl> <chr> <dbl>
#> 1 1.11  one   3.14
#> 2 2.22  two   6.28
#> 3 3.33  three 9.42
```

Но обычно нам нужно что-то более привлекательное и отформатированное. Самый простой способ реализовать это – использовать функцию kable из пакета knitr (рис. 16-9):

```
library(knitr)
kable(myTable, caption = 'My Table')
```

My Table

x	y	z
1.11	one	3.14
2.22	two	6.28
3.33	three	9.43

Рис. 16-9. Таблица, полученная с помощью функции kable

Функция kable принимает в качестве входных данных таблицу данных и ряд параметров форматирования, возвращая отформатированную таблицу, подходящую для отображения.

Она дает великолепный вывод, но многие обнаруживают, что хотят большего контроля над выводом, чем она позволяет. К счастью, kable может работать с другим пакетом, kableExtra, для – что неудивительно – обеспечения дополнительной функциональности.

Здесь мы устанавливаем округление и заголовок, используя функцию kable. Затем мы используем функцию kable_styling, чтобы сделать таблицу более узкой по сравнению с полноразмерной, добавляем заштрихованное чередование в наш вывод LaTeX и центрируем таблицу в выводе (рис. 16-10):

```
library(knitr)
library(kableExtra)
#>
#> Attaching package: 'kableExtra'
#> The following object is masked from 'package:dplyr':
#>
#>    group_rows

kable(myTable, digits = 2, caption = 'My Table') %>%
  kable_styling(full_width = FALSE,
                latex_options = c('hold_position', 'striped'),
                position = "center",
                font_size = 12)
```

My Table

x	y	z
1.11	one	3.14
2.22	two	6.28
3.33	three	9.42

Рис. 16-10. Таблица, полученная с помощью kableExtra

Функция kable_styling принимает таблицу kable в качестве входных данных (это не таблица данных) плюс параметры форматирования, а затем возвращает отформатированную таблицу.

Некоторые параметры в kable_styling по-разному влияют на вывод в зависимости от его формата. В нашем предыдущем примере, когда мы устанавливаем для full_width значение FALSE, в формате LaTeX (PDF) ничего не меняется, потому что таблицы в выводе LaTeX по умолчанию не имеют полной ширины. В HTML, однако, таблицы kable по умолчанию должны иметь такую ширину, поэтому этот параметр оказывает влияние.

Аналогично применяется опция latex_options = c('hold_position', 'striped') только для вывода LaTeX, а не HTML. Опция 'hold_position' гарантирует, что таблица окажется в том месте, куда мы поместили ее в своем исходнике, а не вверху или внизу страницы, как это обычно происходит в LaTeX, а опция 'striped' создает таблицы с чередующимися светлыми и темными рядами, чтобы ее было удобнее читать.

Для большего контроля над таблицами Microsoft Word мы рекомендуем использовать функцию flextable::regable, которая обсуждается в рецепте 16.14.

16.11. Вставка математических уравнений

Задача

Вы хотите вставить математическое уравнение в свой документ.

Решение

R Markdown поддерживает нотацию математических формул LaTeX. Существует два способа ввода этих формул в R Markdown.

Если формула короткая, поместите нотацию LaTeX между одинарными знаками доллара ($). Решение линейной регрессии можно было бы выразить как `$\beta = (X^{T}X)^{-1}X^{T} {\bf{y}}$`, что приведет к встроенной формуле $\beta = X^T X^{-1} X^T_y$.

Если вы имеете дело с блоками формул, вставьте блок между двойными знаками доллара ($$), например так:

```
$$
\frac{\partial \mathrm C}{ \partial \mathrm t } + \frac{1}{2}\sigma^{2}
     \mathrm S^{2} \frac{\partial^{2} \mathrm C}{\partial \mathrm C^2}
   + \mathrm r \mathrm S \frac{\partial \mathrm C}{\partial \mathrm S}\ =
   \mathrm r \mathrm C
   \label{eq:1}
$$
```

что сгенерирует такой вывод:

$$\frac{\partial C}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial C}{\partial C^2} + rS\frac{\partial C}{\partial S} = rC.$$

Обсуждение

Синтаксис разметки математических уравнений является стандартом LaTeX, который зародился в TeX. Опираясь на этот стандарт, R Markdown может отображать математические выражения в документах PDF, HTML, MS Word и MS PowerPoint. Форматы PDF и HTML поддерживают полный спектр математических уравнений LaTeX. Однако перевод в Microsoft Word и PowerPoint поддерживает только часть полного синтаксиса.

Описание подробностей нотации математических формул LaTeX выходит за рамки этой книги, но, поскольку TeX существует уже более 40 лет, в интернете и печати доступно множество великолепных ресурсов. Очень хороший онлайн-ресурс есть на сайте Wikibooks.org: https://en.wikibooks.org/wiki/LaTeX/Mathematics.

16.12. Генерация вывода HTML

Задача

Вы хотите создать документ на языке гипертекстовой разметки (HTML) из документа R Markdown.

Решение

В RStudio нажмите на стрелку, указывающую вниз, рядом с кнопкой с надписью **Knit** в верхней части окна редактирования кода. Когда вы это сделаете, то получите раскрывающийся список всех выходных форматов, доступных для вашего текущего документа. Выберите опцию **Knit to HTML**, как показано на рис. 16-11.

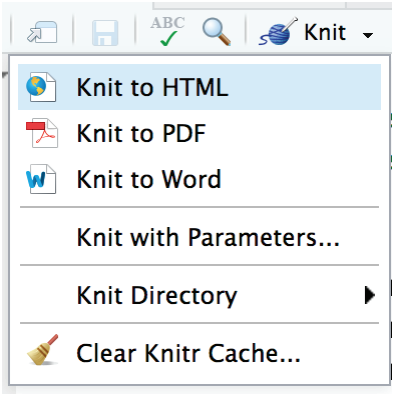


Рис. 16-11. Выбираем опцию **Knit to HTML**

Обсуждение

Когда вы выбираете опцию **Knit to HTML**, RStudio перемещает `html_document: default` в верхнюю часть выходного блока YAML вверху документа, сохраняет файл и затем запускает `rmarkdown:: render(./ВашФайл.Rmd)`. Если вы «связали» свой документ в трех разных форматах, ваш YAML может выглядеть так:

```
output:
  html_document: default
  pdf_document: default
  word_document: default
```

Если вы запустите `render(./ВашФайл.Rmd)`. в своем документе R Markdown, подставив вместо `ВашФайл.Rmd` фактическое имя файла, по умолчанию он будет создан в самом верхнем выходном формате (в данном случае HTML).

Если вы создаете документ в формате HTML, он не должен содержать какого-либо особого специфичного для LaTeX форматирования, поскольку могут возникнуть неприятные сюрпризы. Исключением, как упоминалось в предыдущих рецептах, являются математические формулы LaTeX, которые правильно отображаются в HTML благодаря библиотеке JavaScript MathJax.

См. также

См. рецепт 16.11.

16.13. Генерация вывода в формате PDF

Задача

Вы хотите создать документ в формате PDF (Adobe Portable Document Format) из документа R Markdown.

Решение

В RStudio нажмите стрелку, указывающую вниз, рядом с кнопкой с надписью **Knit** в верхней части окна редактирования кода. Когда вы это сделаете, то получите раскрывающийся список всех выходных форматов, доступных для вашего текущего документа. Выберите опцию **Knit to PDF**, как показано на рис. 16-12.

`pdf_document` переместится в верхнюю часть параметров `output` YAML:

```
---
title: "Nice Title"
output:
pdf_document: default
html_document: default
---
```

и затем будет создан документ PDF.

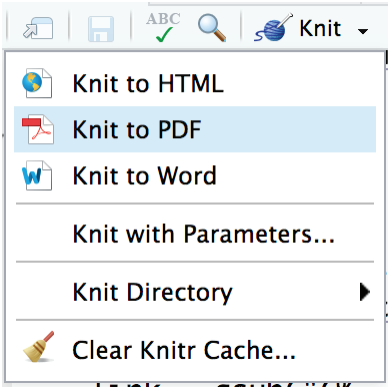


Рис. 16-12. Выбираем опцию **Knit to PDF**

Обсуждение

Вязание в PDF использует Pandoc и движок LaTeX для создания PDF-документа. Если дистрибутив LaTeX еще не установлен на вашем компьютере, проще всего получить его с помощью пакета `tinytex`. Установите `tinytex` в R и затем вызовите функцию `install_tinytex()`. `tinytex` установит небольшой и эффективный дистрибутив LaTeX на ваш компьютер:

```
install.packages("tinytex")
tinytex::install_tinytex()
```

LaTeX богат опциями, и, к счастью, большинство вещей, которые мы хотим сделать, могут быть представлены с помощью R Markdown и автоматически преобра-

зованы в LaTeX через Pandoc. Поскольку LaTeX – мощный инструмент для набора текста, с ним можно делать вещи, которых нет в R Markdown. Мы не можем перечислить здесь все его возможности, но можем рассказать о способах передачи параметров напрямую в LaTeX из R Markdown. Однако имейте в виду, что любые специфичные для LaTeX опции, которые вы используете, не будут должным образом переведены в другие форматы, такие как HTML или MS Word.

Существует два основных способа передачи информации из R Markdown в движок рендеринга LaTeX:

1) передать команды LaTeX непосредственно в компилятор LaTeX;
2) установить параметры LaTeX в заголовке YAML.

Если вы хотите передать команды LaTeX напрямую в компилятор LaTeX, можно использовать команду LaTeX, начинающуюся со знака \. Ограничение состоит в том, что если вы «свяжете» документ в любой формат, кроме PDF, команда, идущая после наклонной черты, полностью исключается из вывода.

Например, если мы поместим эту фразу в наш исходный файл R Markdown:

```
Sometimes you want to write directly in \LaTeX
```

она будет отображена, как показано на рис. 16-13.

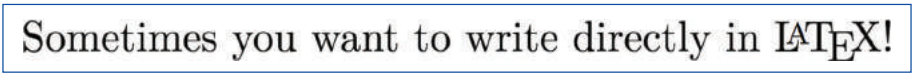


Рис. 16-13. Результат отображения фразы в LaTeX

Тем не менее если вы отобразите документ в HTML, команда `\LaTeX` будет полностью удалена, и в вашем документе останется непривлекательный пробел.

Если вы хотите установить глобальные параметры для LaTeX, это можно сделать, добавив параметры в заголовок YAML в вашем документе R Markdown. Заголовок YAML содержит метаданные верхнего уровня, а также дополнительные данные для некоторых опций. Разные параметры устанавливаются на разных уровнях отступов, поэтому мы обычно ищем их в книге *R Markdown: The Definitive Guide* (https://bookdown.org/yihui/rmarkdown/), просто чтобы быть уверенными.

Например, если у вас есть какое-либо ранее написанное содержимое LaTeX и вы хотите включить его в свой документ, можно добавить это содержимое в трех разных местах документа: в заголовке, перед содержимым тела или после содержимого тела в конец. Если вы добавите внешний контент во все три раздела, ваш заголовок YAML будет выглядеть примерно так:

```
---
title: "My Wonderful Document"
output:
   pdf_document:
     includes:
       in_header: header_stuff.tex
       before_body: body_prefix.tex
       after_body: body_suffix.tex
---
```

Еще одна распространенная опция LaTeX – это шаблон LaTeX для форматирования документа. Многие шаблоны доступны в интернете (https://www.sharelatex.

com/templates), у некоторых компаний и школ есть собственные шаблоны. Если вы хотите использовать существующий шаблон, можете сослаться на него в заголовке YAML:

```
---
title: "Poetry I Love"
output:
    pdf_document:
      template: i_love_template.tex
---
```

Вы также можете включить или выключить нумерацию страниц и разделов:

```
---
title: "Why I Love a Good ToC"
output:
    pdf_document:
      toc: true
      number_sections: true
---
```

Однако некоторые параметры LaTeX устанавливаются с метаданными YAML верхнего уровня:

```
---
title: "Custom Report"
output: pdf_document
fontsize: 12pt
geometry: margin=1.2in
---
```

Поэтому, когда вы настраиваете параметры LaTeX, обратитесь к документации R Markdown, чтобы определить, является ли заданная вами опция подопцией параметра `output:` или ее собственной опцией YAML верхнего уровня.

См. также

См. раздел «PDF Document» по адресу https://bookdown.org/yihui/rmarkdown/pdf-document.html.

См. также документацию по шаблонам Pandoc: https://pandoc.org/MANUAL.html#templates.

16.14. Генерация вывода в формате Microsoft Word

Задача

Вы бы хотели создать документ Microsoft Word из документа R Markdown.

Решение

В RStudio нажмите стрелку, указывающую вниз, рядом с кнопкой с надписью **Knit** в верхней части окна редактирования кода. Когда вы это сделаете, то получите раскрывающийся список всех выходных форматов, доступных для вашего текущего документа. Выберите опцию **Knit to Word**, как показано на рис. 16-14.

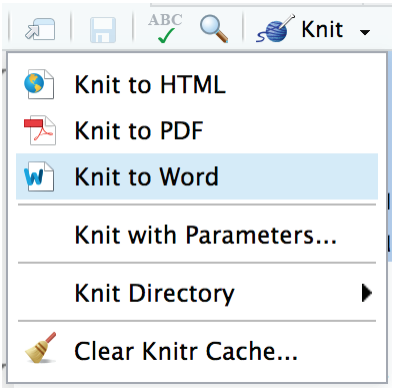


Рис. 16-14. Выбираем опцию **Knit to Word**

`word_document` переместится в верхнюю часть параметров `output` YAML, а затем будет создан ваш документ R Markdown в формате Word:

```
---
title: "Nice Title"
output:
    word_document: default
    pdf_document: default
---
```

Обсуждение

Вязание в Microsoft Word полезно для деловой и академической среды, где руководители и сотрудники ожидают увидеть документы в формате Word. Большинство функций R Markdown работают в Word очень хорошо, но есть несколько хитростей, которые мы нашли полезными при использовании вывода в формате Word.

У Microsoft есть собственный инструмент для редактирования формул. Pandoc приведет ваши уравнения LaTeX в MS Equation Editor, который хорошо работает с большинством основных уравнений, но поддерживает не все параметры уравнений LaTeX. Одна из проблем заключается в том, что MS Equation Editor не поддерживает изменение шрифтов для части уравнения. В результате матричная запись с дробями и другими формулами, для которых требуются разные шрифты, в Word может выглядеть несколько странно.

Вот пример матрицы, которая хорошо выглядит в HTML и PDF:

```
$$
M = \begin{bmatrix}
        \frac{1}{6} & \frac{1}{6} & 0 \\[0.3em]
        \frac{7}{8} & 0 & \frac{2}{3} \\[0.3em]
        0 & \frac{7}{9} & \frac{7}{7}
    \end{bmatrix}
$$
```

Вот как она отображается в следующих форматах вывода:

$$M = \begin{bmatrix} \frac{1}{6} & \frac{1}{6} & 0 \\ \frac{7}{8} & 0 & \frac{2}{3} \\ 0 & \frac{7}{9} & \frac{7}{7} \end{bmatrix}$$

Но это похоже на рис. 16-15 в MS Word.

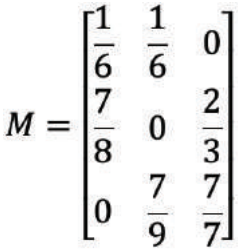
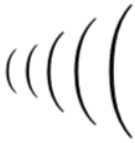


Рис. 16-15. Матрица в MS Word

Любая формула, использующая масштабирование символов, не будет работать должным образом в Word. Например, уравнение

```
$( \big( \Big( \bigg( \Bigg($
```

в HTML и LaTeX будет выглядеть так:



но будет упрощено в MS Equation Editor, как показано на рис. 16-16.

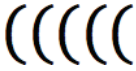


Рис. 16-16. Уравнение масштабирования шрифта в MS Word

Самое простое решение для уравнений в Word – сначала опробовать свое уравнение. Если вам не нравится вывод, перенесите свое уравнение LaTeX в бесплатный онлайн-редактор уравнений (http://www.sciweavers.org/free-online-latex-equation-editor), отредактируйте его и сохраните в виде файла изображения. Затем включите этот файл изображения в свой документ R Markdown, убедившись, что в ваших документах Word есть уравнения, которые выглядят так же хорошо, как документы HTML или LaTeX. Вы, вероятно, захотите сохранить исходный код уравнения LaTeX в текстовом файле, чтобы быть уверенным, что сможете легко изменить его позже.

Другая проблема с выводом Word заключается в том, что часто цифры здесь выглядят не так хорошо, как в HTML или PDF. Возьмем в качестве примера этот линейный график:

```
'''{r}
mtcars %>%
 group_by(cyl, gear) %>%
 summarize(mean_hp=mean(hp)) %>%
 ggplot(., aes(x = cyl, y = mean_hp, group = gear)) +
   geom_point() +
   geom_line(aes(linetype = factor(gear))) +
   theme_bw()
'''
```

В документе Word это изображение выглядит так, как показано на рис. 16-17.

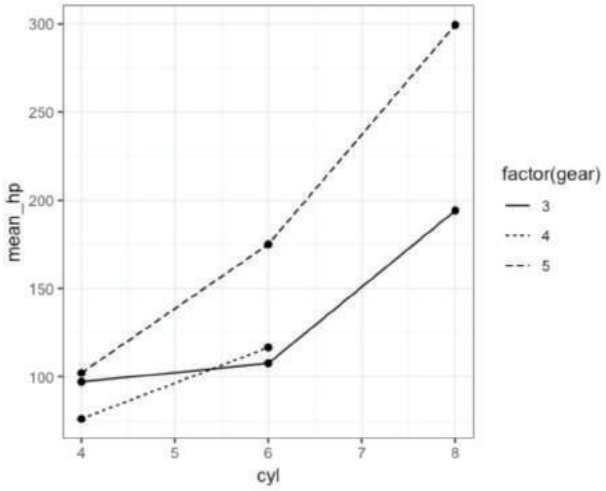


Рис. 16-17. График в Word

Выглядит довольно хорошо, но при печати изображение кажется немного массивным и нерезким.

Его можно улучшить, увеличив настройку точек на дюйм (`dpi`), используемую при «вязании» вывода. Это поможет сделать вывод более плавным и четким:

```
'''{r, dpi=300}
mtcars %>%
 group_by(cyl, gear) %>%
 summarize(mean_hp=mean(hp)) %>%
 ggplot(., aes(x = cyl, y = mean_hp, group = gear)) +
   geom_point() +
   geom_line(aes(linetype = factor(gear))) +
   theme_bw()
'''
```

Чтобы показать улучшение внешнего вида, мы соединили составное изображение, показывающее низкое значение `dpi` по умолчанию слева и более высокое `dpi` справа на рис. 16-18.

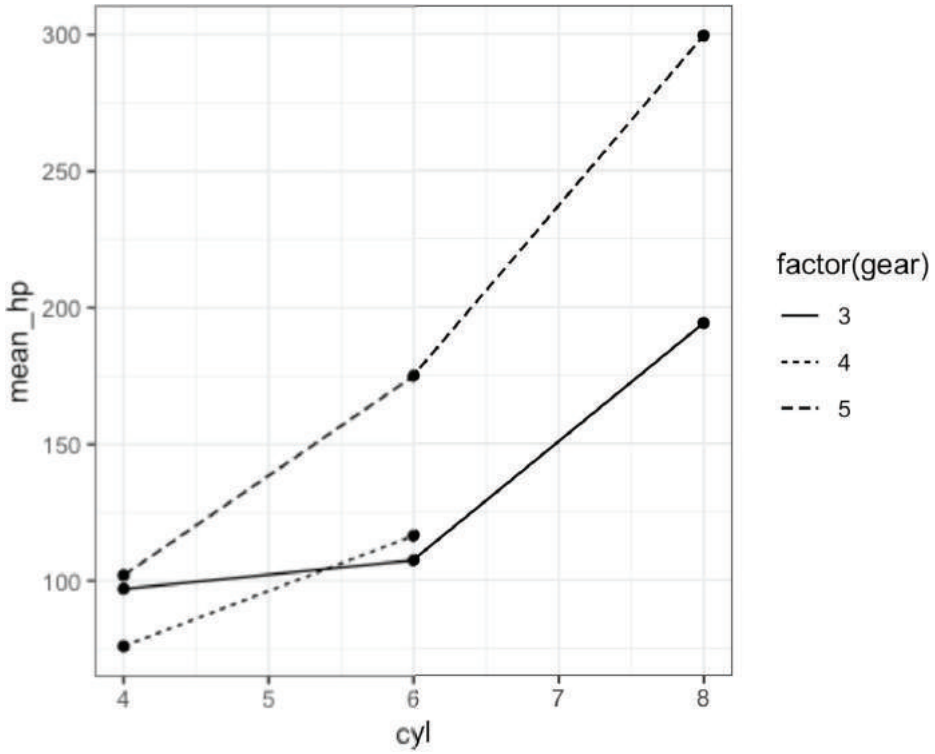


Рис. 16-18. Разрешение изображения в Word (по умолчанию низкое `dpi` в левой половине, более высокое `dpi` в правой)

В дополнение к изображениям вывод таблицы в Word иногда настроен не так, как нам бы хотелось. Использование функции `kable`, как было показано в предыдущих рецептах, приводит к созданию хорошей таблицы без излишеств в MS Word (*см.* рис. 16-19):

```
library(knitr)
myTable <- tibble(x = c(1.111, 2.222, 3.333),
                  y = c('one', 'two', 'three'),
                  z = c(5, 6, 7))
kable(myTable, caption = 'My Table in Word')
```

My Table in Word

x	y	z
1.111	one	5
2.222	two	6
3.333	three	7

Рис. 16-19. Таблица в Word

Pandoc помещает таблицу в табличную структуру Microsoft внутри документа Word. Но, как и в случае с таблицами в PDF или HTML, в Word также можно использовать пакет `flextable`:

```
library(flextable)
regulartable(myTable)
```

В результате этого мы получаем то, что показано на рис. 16-20.

x	y	z
1.111	one	5.000
2.222	two	6.000
3.333	three	7.000

Рис. 16-20. Таблица в Word, полученная с помощью функции regulartable

Мы можем использовать богатые возможности форматирования `flextable` и конвейерных цепочек, чтобы настроить ширину столбцов, добавить цвет фона для наших заголовков и сделать шрифт заголовка белым:

```
regulartable(myTable) %>%
 width(width = c(.5, 1.5, 3)) %>%
 bg(bg = "#000080", part = "header") %>%
 color(color = "white", part = "header")
```

В результате этого мы получаем то, что показано на рис. 16-21.

x	y	z
1.111	one	5.000
2.222	two	6.000
3.333	three	7.000

Рис. 16-21. Индивидуально настроенная таблица в Word, полученная с помощью функции `regulartable`

Для получения подробной информации обо всех настраиваемых параметрах `flextable` *см.* сопутствующую и онлайн-документацию по нему.

«Вязание» позволяет шаблону контролировать форматирование вывода Word. Чтобы использовать шаблон, добавьте `reference_docs: template.docx` в заголовок YAML:

```
title: "Nice Title"
output:
  word_document:
    reference_docx: template.docx
```

Когда вы вяжете файл R Markdown в Word, используя шаблон, `knitr` отображает форматирование элементов в вашем исходном документе в стили в шаблоне. По-

этому если вы хотите изменить шрифт основного текста, можно установить стиль основного текста в шаблоне Word на нужный вам шрифт. После этого `knitr` будет использовать стиль шаблона в новом документе.

Обычный рабочий процесс при первом использовании шаблона – «связать» свой документ в Word без шаблона, затем открыть получившийся документ, настроить стили каждого раздела по своему усмотрению и использовать скорректированный документ в качестве шаблона в будущем. Таким образом, вам не нужно гадать, какой стиль использует `knitr` для каждого элемента.

См. также

См. дополнительную документацию `flextable` по форматированию, `vignette('format','flextable')`, и онлайн-документацию (https://davidgohel.github.io/flextable/articles/overview.html).

16.15. Генерация выходных данных презентации

Задача

Вы хотите создать презентацию из документа R Markdown.

Решение

R Markdown и `knitr` поддерживают создание презентаций из документов R Markdown. Наиболее распространенными форматами для презентаций являются HTML (с использованием HTML-шаблонов `ioslides` или `Slidy`), PDF с Beamer или Microsoft PowerPoint. Самое большое различие между документами и презентациями R Markdown заключается в том, что презентации по умолчанию имеют альбомный макет (широкий, не длинный) и каждый раз, когда вы создаете заголовок второго уровня, начинающийся с `##`, `knitr` будет создавать новую «страницу» или слайд.

Самый простой способ начать работу с презентациями с помощью R Markdown – это использовать RStudio и выбрать **File → New File → R Markdown…** (Файл → Новый файл → R Markdown…), затем выбрать один из четырех форматов презентации, предлагаемых в диалоговом окне на рис. 16-22.

Четыре класса презентаций соответствуют трем основным классам документов, которые мы обсуждали в предыдущих рецептах документов.

Когда приходит время «связать» документ в выходной формат, в RStudio вы щелкаете по стрелке, указывающей вниз, рядом с кнопкой **Knit** и выбираете из раскрывающегося списка тип презентации, который хотите создать, как показано на рис. 16-23.

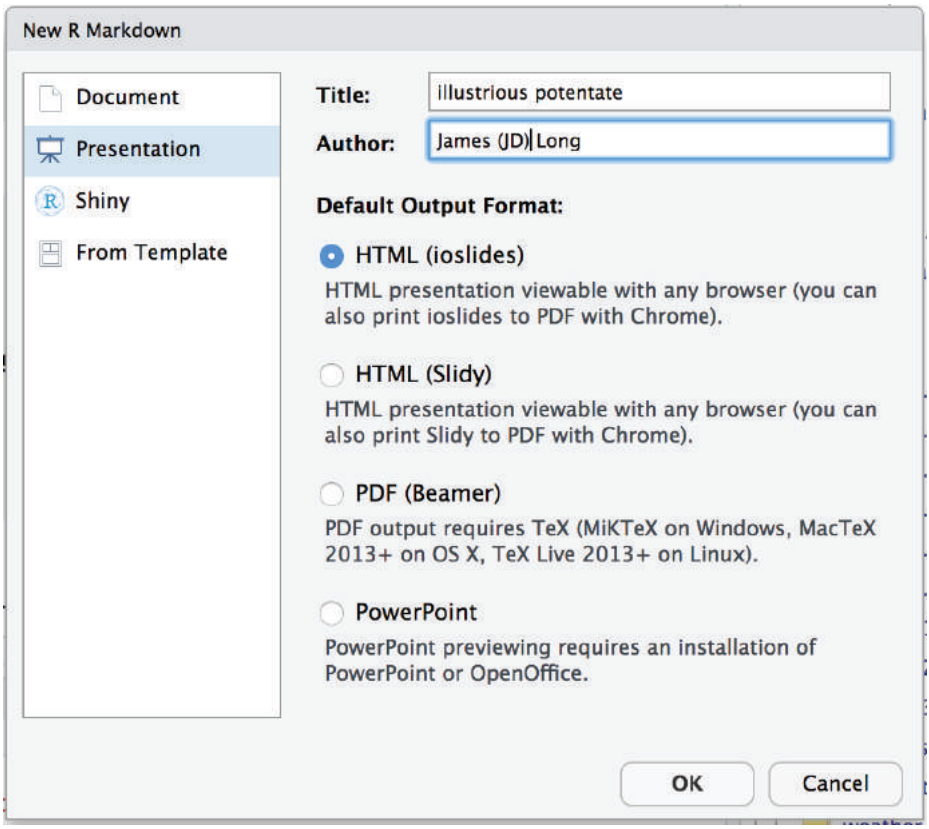


Рис. 16-22. Окно презентации в R Markdown

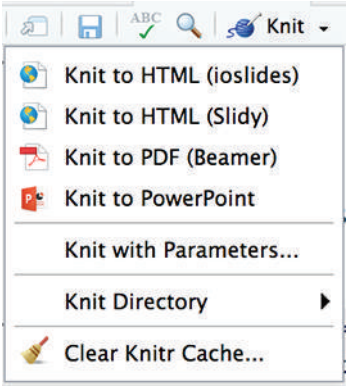


Рис. 16-23. Выбираем тип презентации

Обсуждение

Вязание в формат презентации очень похоже на вязание в обычный документ, только с разными выходными именами. Когда вы используете кнопку **Knit** в RStudio для выбора выходного формата, RStudio перемещает выбранный выходной формат в верхнюю часть параметров вывода в заголовке YAML вашего документа, а затем запускает `rmark down::render("ваш_файл.Rmd")`, который «вяжет» документ в самый верхний формат в вашем заголовке YAML.

Например, если бы мы выбрали опцию **Knit to PDF (Beamer)**, заголовок презентации мог бы выглядеть так:

```
---
title: "Best Presentation Ever"
output:
   beamer_presentation: default
   slidy_presentation: default
   ioslides_presentation: default
   powerpoint_presentation: default
---
```

Большинство параметров HTML, обсуждавшихся в предыдущих рецептах, применимы к HTML-презентациям `Slidy` и `ioslides`. Beamer – это формат на базе PDF, поэтому большинство параметров LaTeX и PDF, описанных в предыдущих рецептах, применимы и к Beamer. И последнее, но не менее важное: PowerPoint – это формат Microsoft, поэтому те предостережения и параметры, которые мы обсуждали ранее, касательно документов Word, также применимы и к PowerPoint.

См. также

Другие рецепты, связанные с выводом R Markdown, тоже могут быть полезны, *см.* рецепты 16.12, 16.13 и 16.14.

16.16. Создание параметризованного отчета

Задача

Вы хотели бы периодически запускать один и тот же отчет с разными входными данными.

Решение

Документы R Markdown можно создавать с параметрами в заголовке YAML, которые затем можно использовать в качестве переменных в теле документа. Параметры хранятся в виде именованных элементов в списке с именем `params`, к которому можно получить доступ в своем фрагменте кода:

```
---
output: html_document
params:
 var: 2
---
'''{r}
print(params$var)
'''
```

Позже, если вы у вас возникнет желание изменить параметр(ы), у вас есть три варианта:

- отредактируйте документ R Markdown и затем визуализируйте его снова;
- визуализируйте документ изнутри R, используя команду `rmarkdown::render`, передавая параметры в виде списка:

```
rmarkdown::render("test_params.Rmd", params = list(var=3))
```

- используя RStudio, выберите **Knitr → Knit with Parameters**, и RStudio запросит у вас параметры, перед тем как приступить к вязанию.

Обсуждение

Использование параметров в R Markdown очень полезно, если у вас есть документ, который необходимо регулярно запускать с различными настройками. Распространенным вариантом использования является отчет, в котором при каждом запуске меняются только настройки даты и метка.

Вот пример документа R Markdown, иллюстрирующий, как можно передать параметры в текст документа:

```
---
title: "Example of Params"
output: html_document
params:
 effective_date: '2018-07-01'
 quarter_num: 2
---
## Illustrate Params
'''{r, results='asis', echo=FALSE}
cat('### Quarter', params$quarter_num,
    'report. Valuation date:',
    params$effective_date)
'''
```

Результат показан на рис. 16-24.

Example of Params

1 Illustrate Params

1.1 Quarter 2 report. Valuation date: 2018-07-01

Рис. 16-24. Вывод параметров

В заголовке фрагмента мы устанавливаем для `results` значение `'asis'`, потому что наш фрагмент кода будет генерировать текст Markdown напрямую. Мы хотим выгрузить этот текст в наш документ без префикса `##`, что обычно происходит с выводом из фрагмента кода. Кроме того, внутри блока кода мы используем команду `cat` для конкатенации нашего текста. Мы применяем `cat` вместо `paste`, потому что `cat` выполняет меньше преобразования текста, чем при вызове `paste`. Это

гарантирует, что текст просто собирается и передается в документ Markdown без изменений.

Если мы хотим отобразить документ с другими параметрами, можно отредактировать значения по умолчанию в заголовке YAML и затем связать их, или можно использовать меню **Knitr** (рис. 16-25).

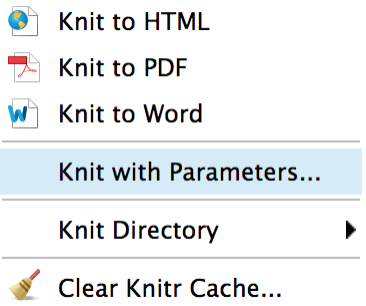


Рис. 16-25. Пункт меню **Knit with parameters...**

После этого у нас запрашивают ввод параметров, как показано на рис. 16-26.

Или мы можем отобразить документ из R, передав новые параметры в виде списка:

```
rmarkdown::render("example_of_params.Rmd",
params = list(quarter_num=2, effective_date='2018-07-01'))
```

В качестве альтернативы использованию меню **Knitr**, если мы хотим, чтобы у нас запросили параметры, можно установить для `params` значение `"ask"`, когда мы вызываем `rmarkdown::render`, и R запросит у нас входные данные:

```
rmarkdown::render("example_of_params.Rmd", params="ask")
```

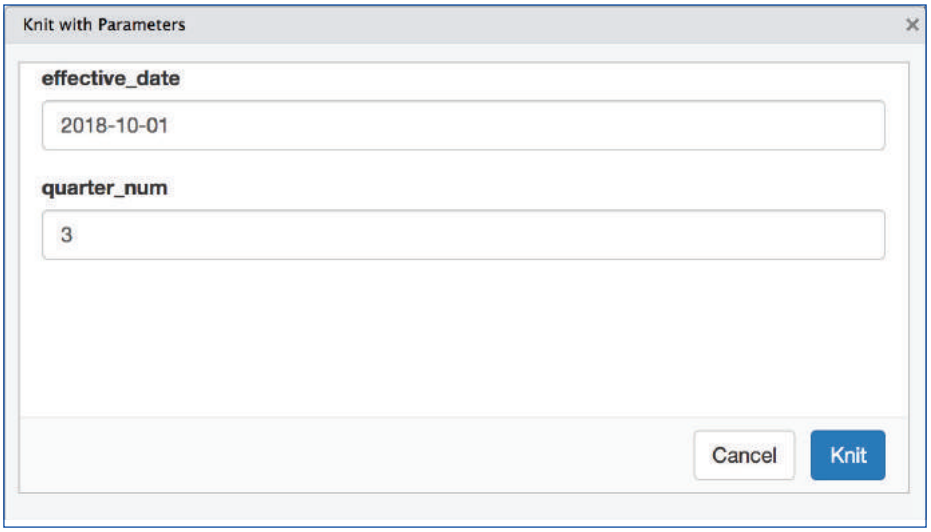


Рис. 16-26. Диалоговое окно **Knit with parameters...**

См. также

См. раздел «Parameterized reports» по адресу https://bookdown.org/yihui/rmarkdown/parameterized-reports.html.

16.17. Организация рабочего процесса в R Markdown

Задача

Вы хотите организовать свой проект R Markdown таким образом, чтобы он был эффективным, гибким и продуктивным.

Решение

Лучший способ получить контроль над своим проектом – организовать рабочий процесс. Организация требует немного усилий, поэтому наличие высокоструктурированного проекта было бы излишне, если ваш документ R Markdown представляет собой только одну страницу вывода с тремя небольшими кусками кода. Тем не менее большинство людей считают, что организация их рабочего процесса стоит дополнительных усилий.

Вот четыре совета по организации вашего рабочего процесса, чтобы в будущем вашу работу было легче читать, редактировать и обслуживать:

1) используйте проекты RStudio;
2) называйте каталоги интуитивно;
3) создайте пакет R для повторно используемой логики;
4) R Markdown должен фокусироваться на контенте и логике исходников.

Используйте проекты RStudio

RStudio включает в себя понятие проекта RStudio (RStudio Project) (обратите внимание на заглавную букву P). Это способ хранения метаданных и настроек, связанных с логическим проектом. Когда вы открываете проект в RStudio, одна из вещей, которые делает RStudio, – это установка рабочего каталога в путь, в котором находится проект. Каждый проект должен обитать в собственном уникальном каталоге. Весь код запускается из этого рабочего каталога, а это значит, что ваш код не должен содержать команды `setwd`, которые не позволят выполнять анализ на чужом компьютере.

Интуитивное именование каталогов

Хорошей идеей будет организовать файлы в каталоге вашего проекта в подкаталоги, а затем вдумчиво назвать свои файлы в этих каталогах. По мере увеличения количества файлов в проекте возрастает и важность организации и интуитивно понятных имен. Одна из распространенных структур, рекомендованная проектом Software Carpentry (https://software-carpentry.org), выглядит так:

```
my_project
  |- data
  |- doc
  |- results
  |- src
```

В этой структуре необработанные исходные данные помещаются в каталог *data*, документация – в *doc*, результаты анализа – в *results*, а исходный код R – в *src*.

Если у вас есть структура каталогов для размещения вашей работы, отдельные файлы должны быть названы так, чтобы их было удобно читать людям и компьютерам. Это помогает поддерживать ваш код в будущем и избавляет от многих проблем. Одни из лучших советов по именованию файлов дает Дженни Брайан (http://www2.stat.duke.edu/~rcs46/lectures_2015/01-markdown-git/slides/naming-slides/naming-slides.pdf):

- используйте подчеркивания вместо пробелов в именах файлов; пробелы вызывают слишком много проблем в дальнейшем;
- если вы указали даты в именах файлов, используйте даты в формате ISO 8601: *ГГГГ-ММ-ДД*;
- используйте префикс в ваших сценариях, чтобы они сортировались правильно, например *00_start_here.R, 01_data_scrub.R, 02_report_output.Rmd*.

Использование числовых префиксов в ваших сценариях и дат в формате ISO 8601 помогает обеспечить значительную сортировку ваших файлов по умолчанию. Это очень полезно, когда кто-либо еще или даже вы в будущем попытается понять ваш проект.

Создание пакета R для повторно используемой логики

Если у вас есть хорошая структура каталогов и рациональное именование, вам следует задуматься над тем, какая логика куда помещается. Вам нужно подумать о создании пакета R для логики, которую вы используете в более чем трех разных проектах. Пакеты R – это коллекции функций и другого кода, которые обеспечивают функциональность, недоступную в базовой версии R. В этой книге мы использовали несколько пакетов, и ничто не мешает вам написать пакет для ваших функций, которые вы используете неоднократно. Создание пакета выходит за рамки этой книги, но презентация Джима Хестера «Вы можете сделать пакет за 20 минут» (https://rstudio.com/resources/videos/you-can-make-a-package-in-20-minutes/) – одно из лучших введений по данной теме.

R Markdown должен фокусироваться на контенте и логике исходников

Большинство из нас начинает проект с одного большого файла *.Rmd*, который наполнен всей нашей логикой в кусках кода. По мере роста документа и расширения фрагментов кода им может стать трудно управлять. Вы можете обнаружить, что форматирование вашего кода смешано с кодом, который изменяет данные и извлекает что-либо из файлов и баз данных. Сочетание логики, форматирования и кода представлений может усложнить последующее изменение кода, и понять его будет еще сложнее. Мы рекомендуем хранить блоки кода в вашем основном отчётном файле *.Rmd*, который сосредоточен на содержимом, таблицах и графиках, а логику манипуляций хранить в файлах * *.R*, которые вы добавляете с помощью функции `source`.

Использование `source` для извлечения внешнего кода включает в себя передачу имени вашего R-файла в функцию `source`:

```
source("my_logic_file.R")
```

R запустит все содержимое `my_logic_file.R` в том месте кода, где вы вызываете функцию `source`. Хороший пример – исходные файлы, которые извлекают таблицы данных и преобразуют ваши данные в форму, необходимую для создания графиков или таблиц в документе. Затем в вашем основном файле *.Rmd* вы храните преимущественно код, который готовит графики и таблицы.

Помните, что это шаблон проектирования для управления большими, громоздкими файлами R Markdown. Если ваш проект не очень большой, вам, вероятно, следует просто хранить весь свой код в файле *.Rmd*.

См. также

Полезные ссылки включают в себя:

- https://www.tidyverse.org/articles/2017/12/workflow-vs-script/;
- https://swcarpentry.github.io/r-novice-gapminder/02-project-intro/;
- http://r-pkgs.had.co.nz;
- http://www2.stat.duke.edu/~rcs46/lectures_2015/01-markdown-git/slides/naming-slides/naming-slides.pdf;
- https://arxiv.org/pdf/1609.00037.pdf.

Об авторах

Дж. Д. Лонг – специалист по экономике сельского хозяйства юга страны. В настоящее время он работает в компании Renaissance Re в Нью-Йорке. Он заядлый пользователь Python, R, AWS и красочных метафор, а также часто выступает с докладами на конференциях, посвященных языку R, и основатель Chicago R User Group. Дж. Д. Лонг живет в Джерси-Сити, штат Нью-Джерси, со своей женой, оставившим практику адвокатом, и 11-летней дочерью.

Пол Титор – программист со степенью магистра в области статистики и информатики. Он специализируется на аналитике и разработке программного обеспечения для управления инвестициями, торговли ценными бумагами и управления рисками. Работает с хедж-фондами, маркетмейкерами и портфельными менеджерами в Большом Чикаго.

Колофон

Птица, изображенная на обложке книги, – южноамериканская гарпия (Harpia harpyja). Будучи одним из 50 видов орлов мировой фауны, она обитает в тропических дождевых лесах Центральной и Южной Америки и предпочитает гнездиться в верхнем пологе леса. Название этого рода и вида относится к гарпиям из древнегреческой мифологии – злобным существам с лицом женщины и телом орла или стервятника.

В среднем гарпии весят около 18 фунтов, длина их тела составляет от 36 до 40 дюймов, а размах крыльев – от 6 до 7 футов, хотя самки неизменно крупнее самцов. Однако оперение обоих полов идентично: темно-серое оперение покрывает верхнюю половину тела птицы, а нижняя сторона – белая или светло-серая. На голове светло-серого цвета – двойной хохол из крупных перьев, который птицы могут поднимать при проявлении враждебности.

Гарпии моногамны, и пара растит только одного птенца раз в два-три года. Самки обычно откладывают по два яйца одновременно, а после того как вылупляется первый птенец, про второе яйцо забывают. Хотя птенец и оперится в течение шести месяцев, оба родителя продолжают ухаживать за ним и кормить его не менее года. Из-за такого низкого темпа роста популяции гарпии особенно подвержены посягательствам на их среду обитания и гибели в результате охоты на них. На протяжении всего своего диапазона охранный статус этой птицы варьируется от категории животных, которым угрожает опасность, до находящихся на грани полного исчезновения.

Многие из животных, изображенных на обложках книг издательства O'Reilly, находятся под угрозой исчезновения; все они важны для мира.

Иллюстрация на обложке выполнена Карен Монтгомери на основе черно-белой гравюры из многотомника *Animate Creation* Дж. Г. Вуда. Шрифты на обложке – Gilroy Semibold и Guardian Sans. Шрифт текста – Adobe Minion Pro; шрифт заголовка – Adobe Myriad Condensed; а шрифт для кодов – Ubuntu Mono от компании Dalton Maag.

Предметный указатель

От редакции

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте **www.dmkpress.com**, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу **dmkpress@gmail.com**, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу **http://dmkpress.com/authors/publish_book/** или напишите в издательство по адресу **dmkpress@gmail.com**.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг – возможно, ошибку в тексте или в коде, – мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу **dmkpress@gmail.com**, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и Packt очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или вебсайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу электронной почты **dmkpress@gmail.com** со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Книги издательства «ДМК Пресс» можно заказать
в торгово-издательском холдинге «Планета Альянс» наложенным платежом,
выслав открытку или письмо по почтовому адресу:
115487, г. Москва, 2-й Нагатинский пр-д, д. 6А.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.a-planeta.ru**.
Оптовые закупки: тел. **(499) 782-38-89**.
Электронный адрес: **books@alians-kniga.ru**.

Дж. Д. Лонг и Пол Титор

R. Книга рецептов:
Проверенные рецепты для статистики, анализа и визуализации данных

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Редактор *Балдин Е. М.*
Перевод *Беликов Д. А.*
Корректор *Синяева Г. И.*
Верстка *Луценко С. В.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.
Гарнитура «PT Serif». Печать цифровая.
Усл. печ. л. 41,28. Тираж 200 экз.

Веб-сайт издательства: www.dmkpress.com