Сергей Мастицкий

Визуализация данных с помощью ggplot2

Москва, 2017

УДК 311:004.9R
ББК 60.6с515
М32

Мастицкий С. Э.
М32 Визуализация данных с помощью ggplot2. – М.: ДМК Пресс, 2017. – 222 с.: ил.

ISBN 978-5-97060-470-0

Визуализация данных играет важную роль на всех этапах статистического анализа – от первичного ознакомления со свойствами данных до диагностики качества построенных моделей и представления полученных результатов. Из всего разнообразия статистических программ выделяется R – интенсивно развивающаяся и свободно распространяемая система статистических вычислений, в которой реализовано множество классических и современных методов анализа данных. Программные реализации алгоритмов, входящих в базовую версию R, проверены на практике не одним поколением пользователей и ученых. Кроме того, пользователи R постоянно разрабатывают многочисленные дополнения для этой системы. Настоящая книга посвящена ggplot2 – одному из таких пакетов, который значительно расширяет и без того богатые базовые графические возможности R.

В 2015 г. ggplot2 был установлен более миллиона раз. Такая популярность этого пакета обусловлена несколькими причинами, среди которых можно отметить эстетическую привлекательность и пригодное для публикации качество получаемых с его помощью графиков, возможность создавать пользовательские типы диаграмм, а также большой набор опций для тонкой настройки внешнего вида графиков. В этой книге описаны основы работы с ggplot2 и приведены многочисленные примеры кода, которые читатели легко могут модифицировать для собственных нужд.

Книга окажется полезной для всех пользователей R, желающих освоить новый мощный инструмент анализа данных.

УДК 311:004.9R
ББК 60.6с515

ISBN 978-5-97060-470-0

Посвящаю эту книгу
своей жене Светлане

Оглавление

Предисловие

«...нет статистического метода более мощного,
чем хорошо подобранный график.»
(Chambers et al., 1983[1])

Визуализация данных играет важную роль на всех этапах статистического анализа — от первичного ознакомления со свойствами данных до диагностики качества построенных моделей и представления полученных результатов. Существует много компьютерных программ для выполнения сложных статистических расчетов и создания не менее сложных графиков. Из всего этого разнообразия выделяется `R` — интенсивно развивающаяся и свободно распространяемая система статистических вычислений, в которой реализовано множество классических и современных методов анализа данных. Язык `R` имеет почти полувековую историю. Он был создан в середине 1990–х г. в Университете Окленда (Unverity of Auckland) Робертом Джентельменом (Robet Gentleman) и Росом Ихакой (Ross Ihaka) на основе языка `S`[2], который, в свою очередь, был разработан в AT&T Bell Laboratories Джоном Чемберсом (John Chambers), Риком Бекером (Rick Becker) и их коллегами в 1976 году[3].

Программные реализации алгоритмов, входящих в ядро `R`, проверены на практике не одним поколением пользователей и ученых. Кроме того, пользователи `R` постоянно разрабатывают многочисленные дополнения (т. н. «пакеты») для этой системы[4]. Представляемая вашему вниманию книга посвящена `ggplot2` — одному из таких пакетов, который значительно расширяет и без того богатые базовые возможности `R` по визуализации данных. Основные создатели `ggplot2` — Хэдли Уикхэм[5] (Hadley Wickham)

[1] Chambers J. M., Cleveland W. S., Tukey P. A., Kleiner B. (1983) Graphical Methods for Data Analysis. Duxbury Press.

[2] `http://cran.r-project.org`.

[3] Подробнее о создании языка `S` можно узнать из очень интересного выступления Рика Бекера «Forty Years of S» (конференция UseR, Стэнфорд, 2016 г.): `http://bit.ly/29lKfNm`.

[4] В сентябре 2016 г. количество пакетов в хранилище CRAN превысило 9000. Актуальную статистику можно всегда узнать на странице `http://bit.ly/2d6ugbn`.

[5] `http://had.co.nz`.

и Уинстон Ченг[6] (Winston Chang) — проделали огромную работу, результатами которой сегодня пользуются сотни тысяч людей[7]. Популярность пакета обусловлена несколькими причинами, среди которых можно отметить эстетическую привлекательность и пригодное для публикации качество получаемых с его помощью графиков, возможность создавать пользовательские типы диаграмм, а также широкий набор инструментов для настройки внешнего вида графиков. Кроме того, логика и синтаксис команд `ggplot2` базируются на интуитивно понятных идеях «грамматики графических элементов» (Wilkinson, 1999[8]), что облегчает программирование.

К сожалению, информации о `ggplot2` на русском языке крайне мало. Цель данной книги — заполнить этот информационный пробел, представив описание основных возможностей пакета. Следует, однако, подчеркнуть, что у меня не было намерения дать сколь–либо исчерпывающее описание `ggplot2`: такими полными источниками всегда будут книги Х. Уикхэма (Wickham, 2009, 2016[9]). Кроме того, важным справочным источником является официальная онлайн–документация по `ggplot2`[10], которую нет смысла дублировать на бумаге.

Настоящая книга предназначена для широкой аудитории — для всех, кто сталкивается с необходимостью визуализации данных и интересуется соответствующими методами и средствами. Предполагается, что читатель имеет некоторое представление об основных статистических понятиях (на уровне вводного университетского курса статистики) и обладает уверенными навыками работы с `R`. Последнее обстоятельство особенно важно, поскольку команды, не имеющие непосредственного отношения к `ggplot2`, подробно здесь не обсуждаются. Читателям, не знакомым с `R`, можно порекомендовать следующие книги на русском языке:

- Зарядов И. С. (2010) Введение в статистический пакет R: типы переменных, структуры данных, чтение и запись информации, графика. Российский университет дружбы народов.

- Шипунов А. Б., Балдин Е. М., Волкова П. А., Коробейников А. И., Назаров С. А., Петров С. В., Суфиянов В. Г. (2012) Наглядная статистика. Используем R! ДМК Пресс.

- Мастицкий С. Э., Шитиков В. К. (2015) Статистический анализ и визуализация данных с помощью R. ДМК Пресс.

Я благодарен Дмитрию Мовчану и всей команде «ДМК Пресс» за помощь с подготовкой и изданием этой книги, а также Петру Валь, Алексею Кожевину, Игорю Магдееву, Александру Маслову, Виталию Скальскому и Тимуру Шеферу за внимательное прочтение рукописи и высказанные ими предложения по улучшению текста.

[6] `https://github.com/wch`.

[7] В 2015 г. пакет был установлен более миллиона раз (Wickham, 2016).

[8] Wilkinson L. (1999) The Grammar of Graphics. Springer.

[9] Wickham H. (2009) ggplot2: Elegant graphics for data analysis. Springer. Второе издание книги вышло в июне 2016 г.

[10] `http://docs.ggplot2.org/current`.

Эту книгу следует считать приложением к моему блогу «R: Анализ и визуализация данных»[11], целью которого является популяризация R среди русскоязычных пользователей. Все примеры кода и данные можно найти на GitHub–странице книги[12]. Любые замечания и пожелания вы можете направлять по электронной почте rtutorialsbook@gmail.com.

Сергей Мастицкий,
Лондон, сентябрь 2016 г.

[11] http://r-analytics.blogspot.com.
[12] https://github.com/ranalytics/ggplot2-ru.

Глава 1

Введение

1.1 Что представляет собой ggplot2?

Пакет является дополнением к системе статистических вычислений R и служит для визуализации данных. В основе ggplot2 лежат идеи «грамматики графических элементов» Лиланда Уилкинсона (Leland Wilkinson), что проявляется в концептуальной целостности и логичном синтаксисе этого пакета. В своей работе Л. Уилкинсон (Wilkinson, 1999[1]) описал набор элементарных графических компонентов, комбинируя которые, можно послойно создавать самые замысловатые графики. Все эти компоненты и операции для их комбинирования доступны в ggplot2, благодаря чему пользователь практически не ограничен в выборе заранее определенных типов графиков и имеет возможность изображать данные в точном соответствии со своими потребностями. При этом тщательно продуманные и заданные по умолчанию настройки ggplot2 позволяют создавать эстетически привлекательные графики с использованием лаконичного синтаксиса.

1.2 Инсталляция ggplot2 и начало работы

Инсталляция пакета ggplot2 не составляет никакого труда. Убедитесь, что на вашем компьютере установлена последняя версия R[2] и что он подключен к сети Интернет, после чего выполните следующую обычную в таких случаях команду:

```
install.packages("ggplot2")
```

Перед использованием установленного пакета ggplot2 необходимо загрузить его при помощи команды

```
library("ggplot2")
```

[1] Wilkinson L. (1999) The Grammar of Graphics. Springer.

[2] Приведенные в книге примеры были созданы с использованием R v3.3.1 и ggplot2 v2.0.0.

1.3 Грамматика графических элементов

Согласно Л. Уилкинсону (Wilkinson, 1999), статистический график — это результат преобразования исходных данных в *геометрические объекты* (например, точки, линии, столбцы), обладающие определенными *эстетическими атрибутами* (цвет, форма, размер). График строится с в рамках некоторой *системы координат* и дополнительно может изображать результаты *статистических преобразований* исходных данных (например, средние значения, доверительные интервалы, сглаживающие кривые и т. п.). Для одновременной визуализации сгруппированных данных можно разбить область рисунка на отдельные ячейки («панели») и разместить в них графики, соответствующие каждой группе. Комбинирование перечисленных компонентов и дает возможность создавать разнотипные статистические графики.

Ниже приведены основные термины «грамматики графических элементов», понимание которых важно для освоения пакета `ggplot2`:

- `data` — подлежащие визуализации *данные*;
- `mapping` — процедура присваивания *координат, формы, размера и цвета* изображаемым на графике объектам в соответствии со значениями анализируемых переменных;
- `geom` (сокращение от *«geometric object»*) — *«геометрические объекты»*, используемые для изображения данных (точки, линии, многоугольники, и т. п.). Между типом этих объектов и типом графика существует тесная связь. Так, на диаграммах рассеяния для изображения данных обычно используют точки, для построения гистограмм — прямоугольные столбики, а для изображения временных рядов применяют линии. В состав пакета `ggplot2` входят более 35 типов геометрических объектов[3], которые можно комбинировать в любых сочетаниях;
- `stat` (сокращение от *«statistical transformations»*) — *«статистические преобразования»*, применяемые к данным для обобщения заключенной в них информации. Объединение значений количественных переменных в дискретные классы для построения гистограмм и представление связи между двумя переменными в виде линии регрессии служат примерами таких преобразований. Статистические преобразования не являются обязательными элементами графиков, однако во многих случаях они оказываются очень полезными;
- `scale` (дословно *«шкала»*) — функция, выполняющая отображение пространства данных на пространство эстетических атрибутов. Результатом работы таких функций является преобразование данных в то, что мы можем воспринять визуально, — координаты, форма, размер, цвет, тип линии и т. д.;
- `coord` — *система координат*, в которой строится график. Обычно используется декартова система координат, однако в пакете `ggplot2`

[3] Этот список (доступен по команде `??ggplot2::geom`) растет с каждой новой версией пакета.

реализованы и другие системы (например, полярная система координат и разнообразные картографические проекции);

- `facet` — сокращение от *«faceting»*, что означает *разбиение данных на группы* и изображение графиков для каждой из этих групп на одном рисунке. В русскоязычной литературе по статистике такие графики часто называют *«категоризованными»*. В англоязычной литературе используются также термины «lattice plots» и «trellis plots».

1.4 Данные, используемые в примерах

Примеры в книгах и статьях по R часто строятся на одних и тех же наборах данных (`iris`, `mtcars` и т. п.). Я решил изменить этой старой доброй традиции и привнести некоторое разнообразие, заодно рассказав читателям о том, чего они раньше, возможно, не знали.

Помните, как в школе на уроках биологии вы изучали инфузорию–туфельку? Оказывается, в природе существует и множество других видов инфузорий. При этом многие из них, в отличие от инфузории–туфельки, живут не просто в воде луж, прудов и озер, а населяют внутренние полости тела других водных животных. Один из таких видов — это *конхофтирус остроконечный* (научное название *Conchophthirus acuminatus*). Конхофтирус обитает в полостях тела *дрейссены речной* (научное название *Dreissena polymorpha*) — моллюска, который широко распространен в пресноводных водоемах Европы и Северной Америки (рис. 1.1)[4]. Никакого вреда эти инфузории не оказывают — они просто подхватывают и с удовольствием поедают остатки пищи (водоросли, бактерии, органические частички и т. п.), отфильтрованной хозяином из толщи воды[5].

Большинство используемых в книге примеров основано на моих собственных данных по количеству инфузорий *C. acuminatus*, обнаруженных в дрейссене из трех озер — Нарочь, Мястро и Баторино (Республика Беларусь). Подробнее об этом исследовании можно узнать в статье Mastitsky (2012)[6]. Полученные данные опубликованы на сайте сервиса «figshare»[7], откуда их можно загрузить непосредственно в R при помощи команды

```
dreissena <- read.delim(
          "http://files.figshare.com/1360878/Dreissena.txt")
```

В состав таблицы `dreissena` входят следующие переменные:

- `Month` — качественная переменная с тремя уровнями, соответствующими времени отбора проб дрейссены: `May` (май), `July` (июль) и `September` (сентябрь);
- `Day` — день отбора проб (с даты начала проведения исследований);

[4] Подробнее см. статью в Википедии: `http://bit.ly/2cdwukq`.

[5] Такие взаимоотношения между двумя видами называются *комменсализмом*.

[6] Mastitsky S. E. (2012) Infection of *Dreissena polymorpha* (Bivalvia: Dreissenidae) with *Conchophthirus acuminatus* (Ciliophora: Conchophthiridae) in lakes of different trophy. BioInvasions Records 1(3): 161–169.

[7] Mastitsky S. (2012) Infection of the zebra mussel with its commensal ciliate *Conchophthirus acuminatus*. `figshare:http://dx.doi.org/10.6084/m9.figshare.95449`.

Рисунок 1.1. Дрейссена речная (*Dreissena polymorpha*) — пресноводный моллюск родом из низовьев рек, впадающих в Черное и Азовское моря. Один из наиболее активно расселяющихся чужеродных видов в Европе и Северной Америке. На фотографии видно несколько десятков моллюсков, почти полностью облепивших небольшой булыжник

- `Lake` — качественная переменная с тремя уровнями, обозначающими изученные озера: `Batorino`, `Myastro` и `Naroch`;
- `Site` — качественная переменная с девятью уровнями, обозначающими места отбора проб (`S1` – `S9`). В каждом озере моллюсков собирали на трех постоянных станциях;
- `Length` — длина раковины моллюсков (мм);
- `Infection` — количество инфузорий, обнаруженных в каждом моллюске (далее по тексту будут использоваться также термины *«интенсивность инвазии»* и *«уровень инвазии»*).

Со структурой этих данных можно ознакомиться при помощи стандартной команды `str()`:

```
str(dreissena)
```

```
'data.frame':        476 obs. of  6 variables:
 $ Month    : Factor w/ 3 levels "July","May","September": 2 ...
 $ Day      : int  1 1 1 1 1 1 1 1 1 1 ...
 $ Lake     : Factor w/ 3 levels "Batorino","Myastro",..: 1 1 ...
 $ Site     : Factor w/ 9 levels "S1","S2","S3",..: 3 3 3 3 3 ...
 $ Length   : num  14.9 14 13 14 12 14 12 19 16.5 18 ...
 $ Infection: int  36 30 331 110 4 171 31 887 525 497 ...
```

Набор данных `dreissena` очень удобен для иллюстрации возможностей `ggplot2`, поскольку обладает небольшим размером (476 наблюдений) и содержит 5 разнотипных переменных (количественные и качественные), чьи

значения изменяются во времени. Для правильного отображения хронологической последовательности месяцев на графиках, которые мы будем строить в дальнейшем, необходимо сообщить R о том, что Month является качественной переменной с упорядоченными (англ. *ordered*) уровнями. Для этого следует выполнить следующую команду:

```
dreissena$Month <- factor(dreissena$Month, ordered = TRUE,
                   levels = c("May", "July", "September"))
```

Хотя обычно R без труда «понимает» кириллические текстовые выражения, определенные настройки операционной системы компьютера могут сопровождаться некорректным распознаванием таких выражений. Поэтому почти на всех приведенных в книге графиках имена переменных из таблицы dreissena, а также значения качественных переменных из этой таблицы на русский язык не переводятся (например, Lake, а не Озеро, May, а не май, и т. д.). Выбор в пользу оригинальных имен был сделан осознанно, чтобы исключить обусловленные кодировкой проблемы при воспроизведении примеров.

Другие использованные в книге наборы данных будут описаны непосредственно в ходе рассмотрения соответствующих примеров.

Глава 2

Функция `qplot()`: быстрое решение для задач визуализации

Функция `qplot()` получила свое название от двух слов — *quick* и *plot*, что значит «быстрый» и «график» соответственно. Название этой функции полностью соответствует ее назначению — она позволяет строить самые разнообразные статистические графики с использованием одной–двух строк кода. Если вы уже знакомы с `plot()` — базовой графической функцией `R`, то освоение `qplot()` не составит никакого труда.

2.1 Аргументы функции `qplot()`

Функция `qplot()` имеет следующие основные аргументы:

- `x` и `y` — переменные X и Y соответственно;
- `data` — таблица данных (*«data frame»* в терминах `R`), содержащая переменные X и Y. Если этот аргумент не указан, то функция `qplot()` попытается автоматически извлечь векторы `x` и `y` из текущей рабочей среды и объединить их в таблицу;
- `facets` — формула, определяющая способ разбиения рисунка на отдельные подобласти при создании категоризованных графиков (см. разд. 2.4 и главу 7);
- `margins` — аргумент, используемый при создании категоризованных графиков. Позволяет включать (`TRUE`) или отключать (`FALSE`) отображаемые по краям графика названия уровней качественной переменной, в соответствии с которыми рисунок разбивается на подобласти;
- `geom` — текстовый вектор с названиям геометрических объектов, используемых для изображения данных. Если на функцию `qplot()`

поданы две переменные — X и Y, то аргумент `geom` по умолчанию примет значение `"point"` («точка»). Если же подана только количественная переменная Y, то значением по умолчанию будет `"histogram"` («гистограмма»). Возможно совмещение нескольких типов геометрических объектов на одном рисунке;

- `stat` — текстовый вектор, определяющий тип статистического преобразования данных;
- `xlim` и `ylim` — задают границы значений переменных X и Y соответственно (в виде `c(нижняя граница, верхняя граница)`);
- `log` — позволяет логарифмически «растянуть» ось X (`log = "x"`), ось Y (`log = "y"`), или обе оси одновременно (`log = "xy"`);
- `main` — текстовый вектор и (или) математическое выражение, образующие заголовок графика;
- `xlab` и `ylab` — текстовые векторы и (или) математические выражения, образующие подписи осей X и Y соответственно;
- `asp` — число, задающее отношение длины X к длине оси Y.

2.2 Построение диаграмм рассеяния с помощью qplot()

Первые два аргумента `qplot()` — `x` и `y` — задают переменные, значения которых будут отложены по соответствующим координатным осям создаваемого графика. Если `x` и `y` представляют собой самостоятельные векторы, то функция `qplot()` попытается автоматически объединить их в одну таблицу. Такой подход не защищен от возникновения непредвиденных ошибок, в связи с чем рекомендуется всегда предварительно объединять необходимые данные в одну таблицу и далее ссылаться на нее при помощи аргумента `data`. Ниже приведен пример обычной *диаграммы рассеяния*, построенной с помощью функции `qplot()` по данным из таблицы `dreissena` (рис. 2.1).

Код для рис. 2.1

```
qplot(x = Length, y = Infection, data = dreissena)
```

На рис. 2.1 видно, что количество инфузорий *C. acuminatus* положительно и нелинейно связано с длиной раковины моллюсков *D. polymorpha*. Мы можем «выровнять» эту зависимость путем логарифмирования обеих переменных (рис. 2.2).

Код для рис. 2.2

```
qplot(x = log(Length),
      y = log(Infection + 1), data = dreissena)
# поскольку некоторые значения Infection равны 0,
# логарифмирование выполнено для (Infection + 1)
```

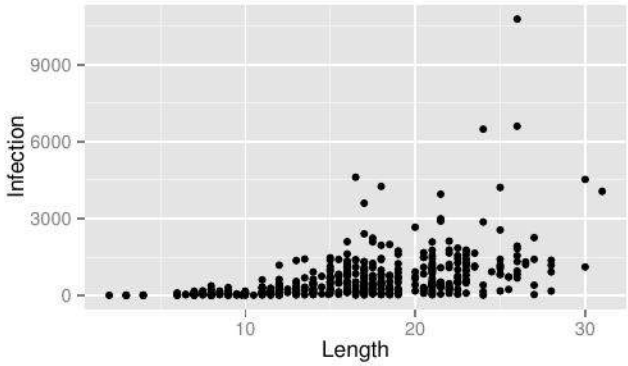


Рисунок 2.1. Пример диаграммы рассеяния, построенной с помощью функции `qplot()`. Изображена связь между длиной раковины дрейссены и интенсивностью инвазии *C. acuminatus*

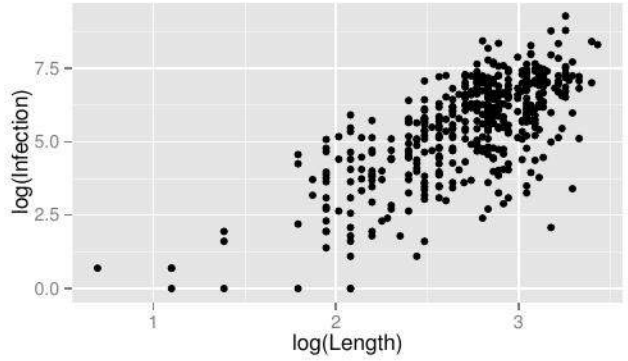


Рисунок 2.2. То же, что и на рис. 2.1, но после логарифмирования исходных данных

Одно из существенных отличий функции `qplot()` от базовой `R`–функции `plot()` состоит в том, каким образом точкам на графике присваиваются эстетические атрибуты, т. е. цвет, размер и форма. В случае с `plot()` пользователь должен самостоятельно конвертировать уровни интересующей его качественной переменной (например, «зима», «весна», «лето», «осень») в соответствующие значения эстетических атрибутов (например, цвет для разных сезонов года: «белый», «голубой», «зеленый», «оранжевый»). Функция же `qplot()` выполняет такие преобразования автоматически, одновременно создавая легенду с цветовой шкалой, которую пользователь может изменить в соответствии со своими требованиями.

На рис. 2.3 показано, как к графику зависимости между двумя количественными переменными можно добавить информацию о третьей — качественной — переменной, изменяя цвет точек (аргумент `colour`) или их форму (аргумент `shape`). Автоматическое присваивание значений эс-

тетических атрибутов можно отменить, воспользовавшись стандартной R–функцией `I()`[1] (например, «вручную» задав значения `colour = I("red")` или `shape = I(2)`).

Код для рис. 2.3

```
qplot(log(Length), log(Infection + 1), data = dreissena,
      colour = Month)
qplot(log(Length), log(Infection + 1), data = dreissena,
      shape = Lake)
```

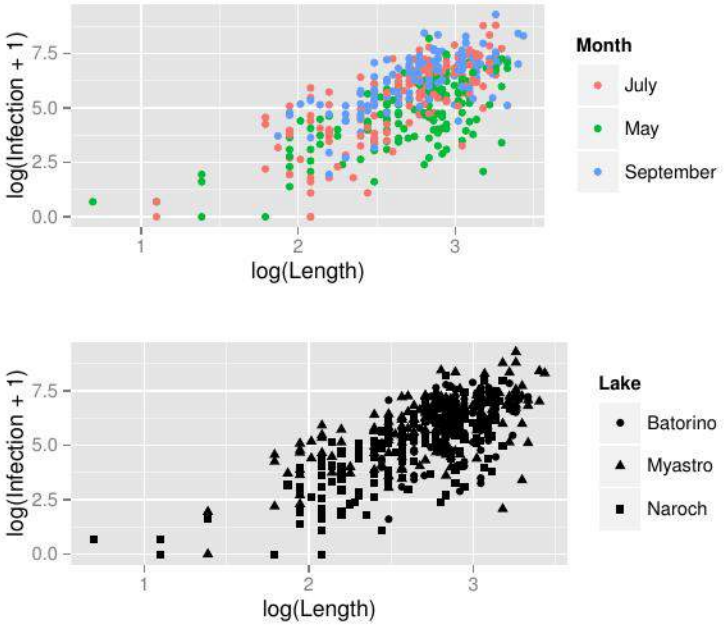


Рисунок 2.3. Примеры присвоения точкам атрибутов «цвет» (верхний график, в соответствии с датой отбора проб) и «форма» (нижний график, в соответствии с водоемом, из которого были отобраны пробы)

Обычно при работе с данными большого объема точки на диаграммах рассеяния накладываются друг на друга, что затрудняет выявление заключенных в данных закономерностей (см., например, рис. 2.1). Полезным приемом для облегчения восприятия таких графиков является использование полупрозрачного цвета (рис. 2.4). Этот прием можно реализовать при помощи аргумента `alpha`, который принимает значения от `0` (полная прозрачность) до `1` (полная непрозрачность). Одним из решений для визуализации данных большого объема является также использование *категоризованных графиков* (см. разд. 2.4 и главу 7).

[1] Функция `I()` подавляет любые преобразования объекта R, возвращая его с сохранением исходного класса.

Код для рис. 2.4

```
qplot(Length, Infection, alpha = I(1/2), data = dreissena)
qplot(Length, Infection, alpha = I(1/4), data = dreissena)
qplot(Length, Infection, alpha = I(1/8), data = dreissena)
```

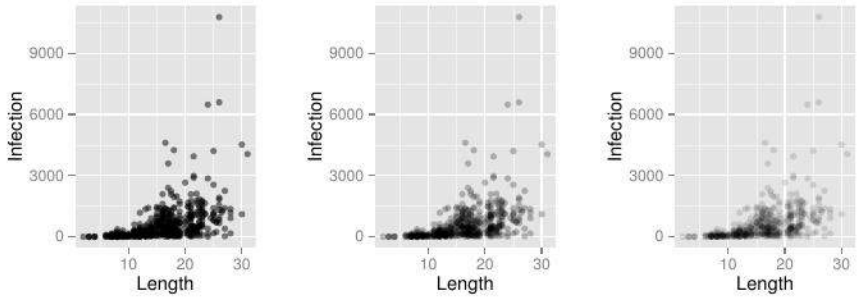


Рисунок 2.4. Примеры использования полупрозрачных точек на диаграммах рассеяния с большим количеством наблюдений. На приведенных графиках (слева направо) значения аргумента `alpha` составляют `0.5`, `0.25` и `0.125`

Следует помнить, что разные эстетические атрибуты неодинаково хорошо подходят для работы с качественными и количественными переменными. Так, цвет и форма хорошо разграничивают уровни качественных переменных, тогда как атрибут «размер» лучше работает с количественными переменными. Размер точек (и других графических объектов) в `qplot()` задается при помощи аргумента `size` (рис. 2.5).

Код для рис. 2.5

```
qplot(log(Length), log(Infection + 1), data = dreissena,
      size = Day, alpha = I(0.25), colour = I("magenta"))
```

2.3 Другие примеры использования qplot()

Безусловно, с помощью функции `qplot()` можно создавать не только диаграммы рассеяния: варьируя значения аргумента `geom`, пользователь получает возможность построить практически все распространенные типы статистических графиков. Аргумент `geom` определяет тип геометрических объектов, используемых для изображения данных. Так, к наиболее часто используемым значениям `geom` относятся следующие:

- `geom = "point"` — изображает данные в виде точек (см., например, рис. 2.1);

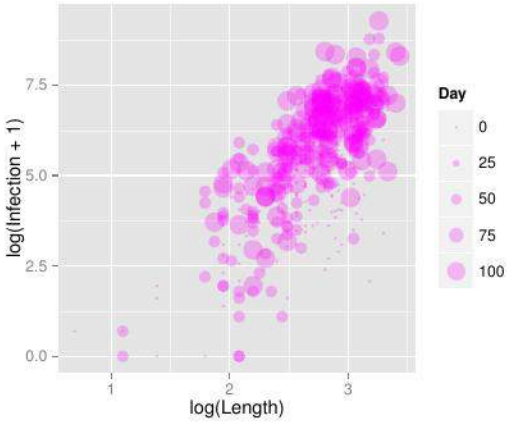


Рисунок 2.5. Диаграмма рассеяния, на которой точкам присвоен атрибут «размер» (`size`) в соответствии со значениями третьей (количественной) переменной

- `geom = "smooth"` — подгоняет сглаживающую кривую к данным и одновременно изображает ее 95%–ную доверительную область;
- `geom = "jitter"` — создает одномерные диаграммы рассеяния;
- `geom = "boxplot"` — создает диаграммы размахов;
- `geom = "path"` и `geom = "line"` — соединяют точки линиями. Традиционно используются для изображения временных изменений количественных переменных (`geom = "line"`). Однако точки могут соединяться не только в соответствии с ходом времени, т. е. слева направо, но и любым другим образом (`geom = "path"`).

При анализе свойств только одной переменной выбор возможных значений аргумента `geom` будет определяться типом этой переменной:

- количественные переменные: значение `geom = "histogram"` приведет к созданию гистограммы, `geom = "freqpoly"` — полигона распределения частот, а `geom = "density"` — кривой плотности вероятности;
- качественные переменные: значение `geom = "bar"` приведет к созданию столбиковой диаграммы.

2.3.1 Линии тренда

Как было отмечено ранее, на диаграммах рассеяния с большим количеством наблюдений бывает сложно увидеть какие–либо четкие закономерности. Помимо использования полупрозрачного цвета точек (см. рис. 2.4 и 2.5), полезным приемом в таких случаях может оказаться также добавление к графику *сглаживающей линии* (англ. *smoother*), или *линии*

тренда. В ggplot2 для этого служит геометрический объект типа smooth. Обратите внимание на то, как в приведенном ниже коде для рис. 2.6 два типа геометрических объектов — точки и сглаживающая линия — были совмещены стандартным для R образом, т. е. с помощью функции конкатенации c(). Слои (см. разд. 3.2) с соответствующими геометрическими объектами появятся на графике в порядке перечисления этих объектов в скобках команды c().

Код для рис. 2.6

```
qplot(log(Length), log(Infection + 1),
      geom = c("point", "smooth"), data = dreissena)
```

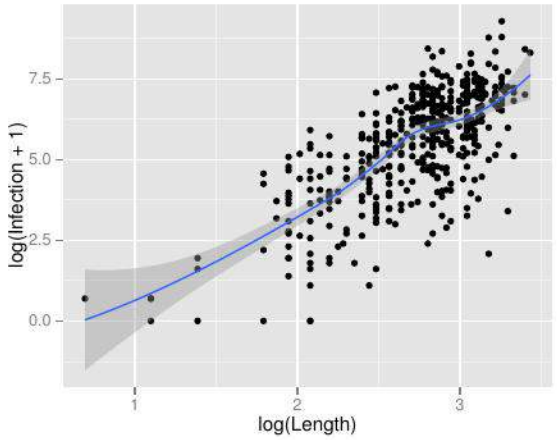


Рисунок 2.6. Пример добавления сглаживающей линии к диаграмме рассеяния

2.3.2 Одномерные диаграммы рассеяния

Одномерная диаграмма рассеяния (англ. *strip chart* или *strip plot*) является подходящим инструментом для визуализации значений какой–либо количественной переменной в соответствии с уровнями качественной переменной. Для создания таких диаграмм в ggplot2 служит геометрический объект типа "jitter". Во избежание излишнего перекрытия точек на графике к их X–координатам случайным образом добавляется небольшой «шум» (рис. 2.7). Поскольку при этом используется встроенный в R генератор псевдослучайных чисел, каждый раз при выполнении соответствующего кода внешний вид рисунка будет несколько изменяться[2].

[2] Для воспроизведения внешнего вида графика при повторном исполнении кода следует воспользоваться функцией set.seed().

Код для рис. 2.7

```
qplot(Lake, log(Infection + 1), data = dreissena,
      geom = "jitter", alpha = I(0.6))
```

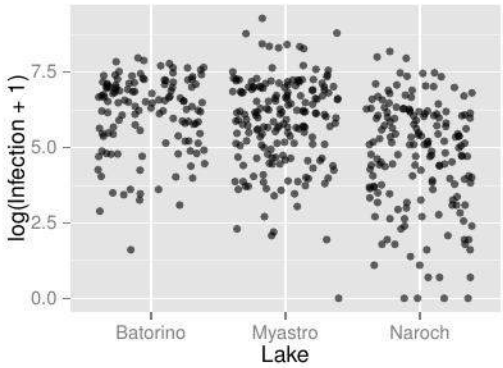


Рисунок 2.7. Пример одномерной диаграммы рассеяния: изображены логарифмированные значения количества инфузорий *C. acuminatus* в дрейссене из трех озер

Точкам на одномерных диаграммах рассеяния можно легко присвоить необходимые эстетические атрибуты, воспользовавшись уже рассмотренными ранее в разд. 2.2 аргументами `color`, `shape` и `size` (рис. 2.8).

Код для рис. 2.8

```
qplot(Lake, log(Infection + 1), data = dreissena,
      geom = "jitter", alpha = I(0.6), colour = Month)
qplot(Lake, log(Infection + 1), data = dreissena,
      geom = "jitter", alpha = I(0.6), colour = Month,
      size = Length)
```

2.3.3 Диаграммы размахов

Диаграммы размахов (англ. *boxplots* или *box–whisker plots*) служат той же цели, что и рассмотренные в предыдущем разделе одномерные диаграммы рассеяния, — они характеризуют вариабельность количественных переменных в соответствии с уровнями качественных переменных. Однако, в отличие от диаграмм рассеяния, на которых изображаются все исходные значения анализируемой количественной переменной (см., например, рис. 2.8), на диаграммах размахов представлена *обобщенная* статистическая информация о распределении значений количественной переменной в соответствующих группах. В классическом случае каждая группа данных изображается в виде прямоугольника (отсюда жаргонные названия

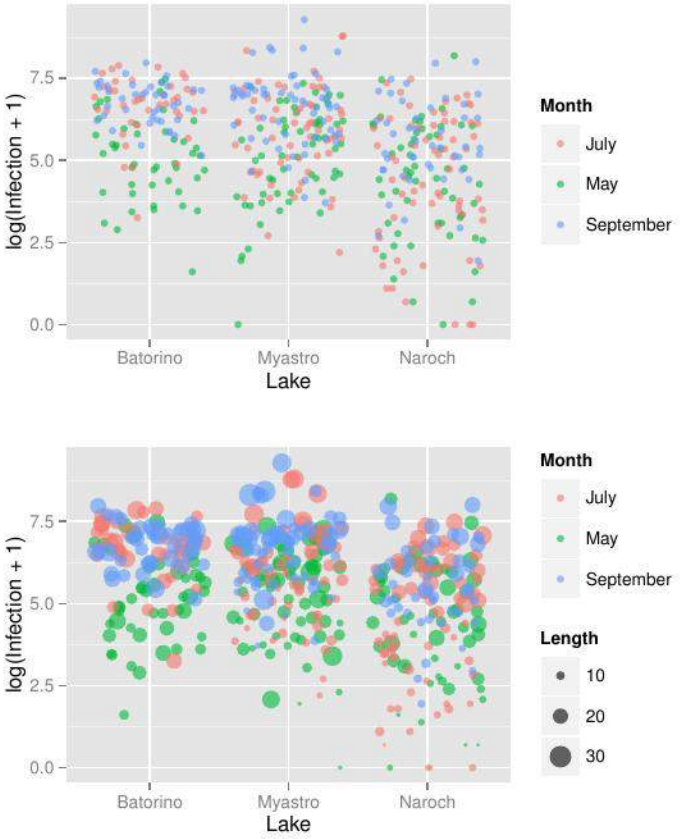


Рисунок 2.8. Примеры одномерных диаграмм рассеяния, на которых точкам присвоены эстетические атрибуты в соответствии со значениями переменных–ковариат

на русском языке — «ящики с усами», «коробочки с усами», «коробчатые графики»), длина которого равна *интерквартильному размаху*, т. е. разнице между первым и третьим *квартилями* ($IQR = Q_3 - Q_1$). Внутри этого прямоугольника находится отрезок, обозначающий *медианное* значение количественной переменной в соответствующей группе. Кроме того, от торцов прямоугольника отходят «усы» — отрезки, чьи концы имеют следующие координаты: верхний отрезок — $\min(\max x, Q_3 + 1.5 \times IQR)$, нижний отрезок — $\min(\max x, Q_1 - 1.5 \times IQR)$. Наблюдения x, лежащие вне ограниченного «усами» интервала, изображаются в виде отдельных точек и потенциально могут быть выбросами (рис. 2.9).

Для построения диаграмм размахов служит геометрический объект типа `"boxplot"`. Пользователь имеет возможность изменять такие эстетические атрибуты, как цвет линий (аргумент `colour`), цвет заливки «ящика» (`fill`) и толщина линий (`size`) (рис. 2.10). Цвет точек, обозначающих потенциальные выбросы, изменяют при помощи аргумента `outlier.colour` (см. код для рис. 2.11).

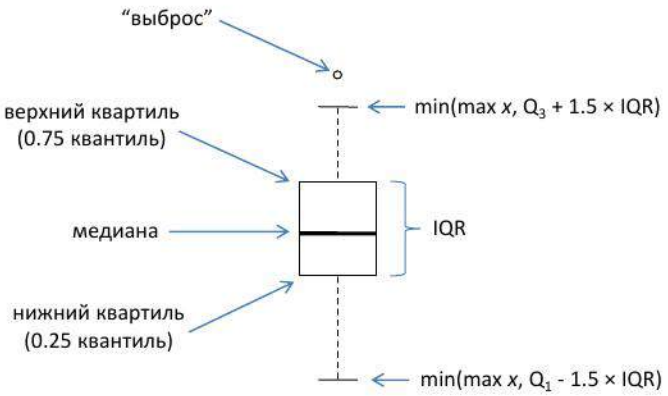


Рисунок 2.9. Анатомия диаграммы размахов

Код для рис. 2.10

```
qplot(Lake, log(Infection + 1), data = dreissena,
      geom = "boxplot")
qplot(Lake, log(Infection + 1), data = dreissena,
      geom = "boxplot", colour = "red")
qplot(Lake, log(Infection + 1), data = dreissena,
      geom = "boxplot", fill = "coral")
```

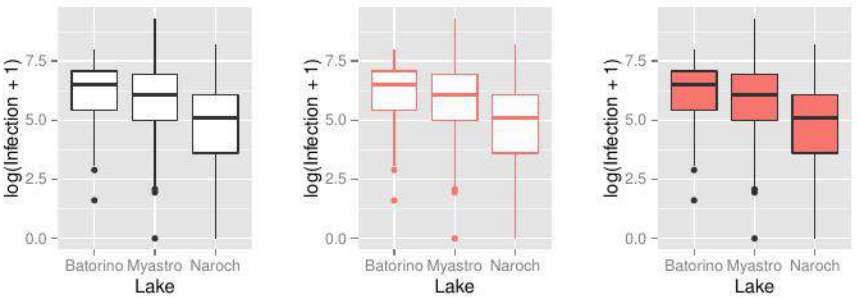


Рисунок 2.10. Примеры диаграмм размахов с разными эстетическими атрибутами. *Слева:* диаграмма, построенная в соответствии с автоматическими настройками. *В центре:* цвет линий изменен на красный. *Справа:* «ящики» залиты цветом кораллового оттенка

Более полно свойства данных можно охарактеризовать, совместив диаграмму размахов с одномерной диаграммой рассеяния (рис. 2.11). Для этого при вызове функции `qplot()` достаточно указать соответствующие типы геометрических объектов, объединив их в один вектор с помощью

функции конкатенации c() (этот прием уже был использован нами при рассмотрении сглаживающих линий в разд. 2.3.1).

Код для рис.2.11

```
qplot(Lake, log(Infection + 1), data = dreissena,
      geom = c("jitter", "boxplot"), alpha = I(1/5),
      outlier.colour = NA)
# отображение "выбросов" отключено
# во избежание дублирования точек, являющихся
# одновременно частью диаграммы рассеяния
```

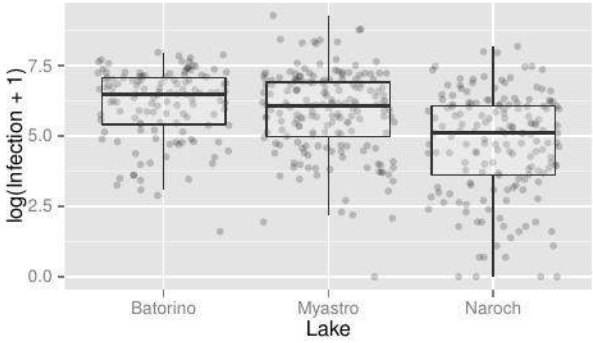


Рисунок 2.11. Пример диаграммы размахов, совмещенной с одномерной диаграммой рассеяния

2.3.4 Гистограммы, кривые плотности вероятности, полигоны частот

Гистограмма является важным инструментом статистики, позволяющим наглядно представить распределение значений анализируемой количественной переменной. Гистограммы отражают больше информации об отдельных группах данных, чем диаграммы размахов, но с их помощью сложнее сравнивать несколько групп одновременно (хотя и возможно — см. ниже). Для построения гистограмм в пакете ggplot2 имеется геометрический объект "histogram". При вызове функции qplot() с аргументом geom = "histogram" будет построена гистограмма с автоматически выбранным количеством столбцов. Такого рисунка, получаемого с использованием автоматических настроек, может оказаться вполне достаточно (например, при проведении быстрого разведочного анализа данных), однако часто требуется дополнительное экспериментирование с размером шага (*классового промежутка*), в соответствии с которым происходит разбиение данных на классы. Слишком большой шаг может замаскировать истинные свойства анализируемой переменной, тогда как слишком малый шаг приведет к отображению излишних деталей. Для подбора оптимального классового промежутка служит аргумент binwidth (рис. 2.12).

Код для рис. 2.12

```
qplot(Infection, data = dreissena,
      geom = "histogram", xlim = c(0, 11000))
qplot(Infection, data = dreissena,
      geom = "histogram", binwidth = 1000,
      xlim = c(0, 11000))
qplot(Infection, data = dreissena,
      geom = "histogram", binwidth = 50, xlim = c(0, 11000))
```

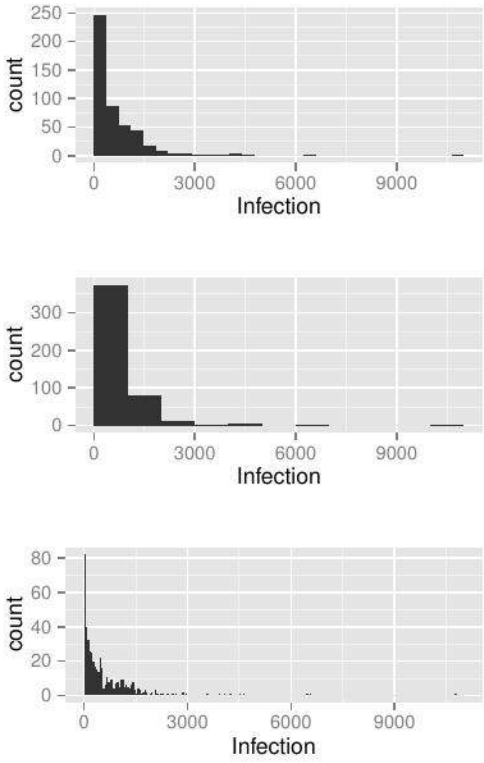


Рисунок 2.12. Примеры гистограмм с разными классовыми промежутками. *Вверху:* промежуток выбран по умолчанию ($[\max - \min]/30$). *В центре:* `binwidth = 1000`. *Внизу:* `binwidth = 50`

Для одновременного сравнения распределений в нескольких группах необходимо воспользоваться аргументом `fill`. В результате действия этого аргумента каждому наблюдению будет присвоен определенный цвет в соответствии с группой, к которой оно принадлежит. Это приведет к построению графика, изображающего несколько наложенных друг на друга гистограмм разного цвета (рис. 2.13).

Код для рис. 2.13

```
qplot(log(Infection + 1), data = dreissena,
      geom = "histogram", fill = Lake)
```

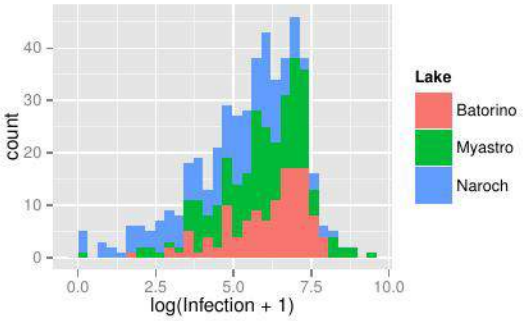


Рисунок 2.13. Одновременное сравнение распределений нескольких совокупностей с помощью гистограмм, залитых разным цветом

Часто при работе с непрерывными количественными переменными вместо гистограмм используют *кривые плотности вероятности*. Для построения графиков такого типа в пакете `ggplot2` служит геометрический объект типа `"density"`. *Ядерное оценивание плотности вероятности* (англ. *kernel probability density estimation*) выполняется при помощи базовой R–функции `density()`, которая вызывается функцией `qplot()` незаметно для пользователя. Процедура оценивания контролируется аргументом `kernel` (возможные значения: `"gaussian"` (принято по умолчанию), `"rectangular"`, `"biweight"`, `"epanechnikov"`, `"triangular"`, `"cosine"` и `"optcosine"`). Гладкость получаемой кривой задается с помощью аргумента `adjust` (более высокие его значения соответствуют большей степени сглаживания) (рис. 2.14). Как и в случае с гистограммами, мы можем изобразить одновременно несколько кривых плотности вероятности на одном графике. Для этого следует воспользоваться аргументом `colour`, указав в качестве его значения имя качественной переменной, по уровням которой исходная совокупность данных разбивается на группы. Площадь под кривой плотности вероятности при желании можно залить цветом с помощью уже известного нам аргумента `fill`. Применив при этом полупрозрачный цвет (аргумент `aplha`), можно получить привлекательный график, позволяющий очень наглядно продемонстрировать различия в распределениях нескольких групп (рис. 2.15).

Код для рис. 2.14

```
qplot(log(Infection + 1), data = dreissena, geom = "density")
qplot(log(Infection + 1), data = dreissena,
      geom = "density", adjust = 0.5)
```

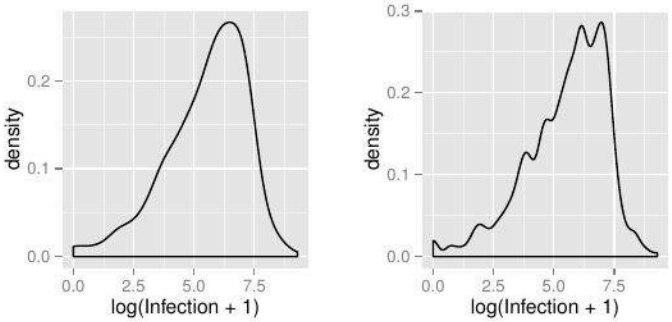


Рисунок 2.14. Примеры кривых плотности вероятности. *Слева:* степень сглаживания выбрана автоматически. *Справа:* степень сглаживания уменьшена (аргумент `adjust = 0.5`)

Код для рис. 2.15

```
qplot(log(Infection + 1), data = dreissena, geom = "density",
      colour = Lake)
qplot(log(Infection + 1), data = dreissena, geom = "density",
      fill = Lake, alpha = I(1/4))
```

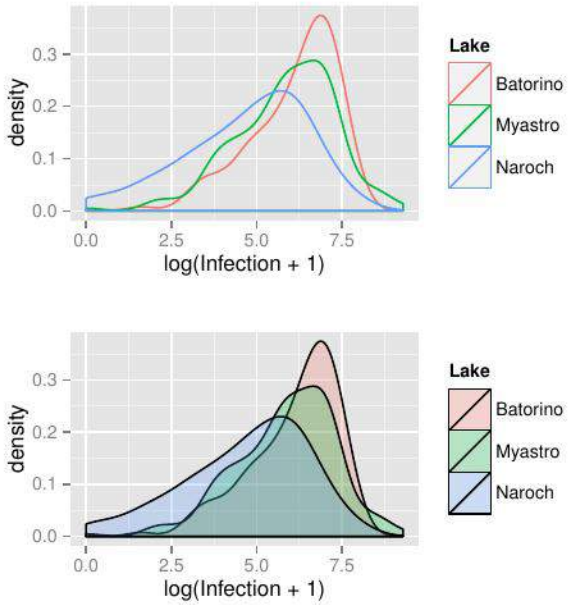


Рисунок 2.15. Примеры кривых плотности вероятности, совмещенных на одном графике

Полигон распределения (англ. *frequency polygon*) — еще один, хотя и реже используемый, способ визуализации распределений количественных переменных. По оси абсцисс на таких графиках откладывают значения анализируемой количественной переменной, а по оси ординат — частоты их встречаемости. В `ggplot2` для создания полигона распределения служит графический объект типа `"freqpoly"`. Как и в случае с гистограммами, размер классового промежутка контролируется аргументом `binwidth`. Кроме того, с помощью аргумента `colour` можно одновременно изобразить несколько кривых на одном графике (рис. 2.16).

Код для рис. 2.16

```
qplot(log(Infection+1), data = dreissena, geom = "freqpoly")
qplot(log(Infection+1), data = dreissena,
      geom = "freqpoly", binwidth = 0.8)
qplot(log(Infection+1), data = dreissena,
      geom = "freqpoly", binwidth = 0.7, colour = Month)
```

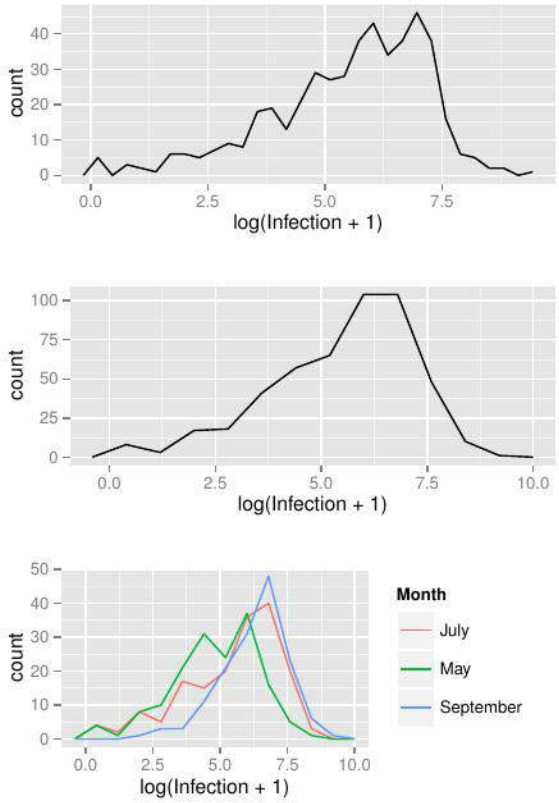


Рисунок 2.16. Примеры полигонов распределений. *Вверху*: график построен с использованием автоматических настроек. *В центре*: гладкость кривой увеличена при помощи аргумента `binwidth = 0.8`. *Внизу*: три полигона частот совмещены на одном графике (см. аргумент `colour`)

2.3.5 Столбиковые диаграммы

Столбиковые диаграммы (англ. *bar plots* или *bar charts*) служат «дискретным аналогом» гистограмм: они изображают число наблюдений в группах, выделенных в соответствии с уровнями той или иной качественной переменной. В отличие от используемой для этого базовой R–функции `barplot()`, функция `qplot()` не требует никаких предварительных вычислений — она самостоятельно сводит в таблицу объемы наблюдений по каждой группе. Для построения столбиковых диаграмм служит геометрический объект типа `"barplot"`. Если стоит задача обобщить данные каким–либо иным способом, нежели подсчет объема наблюдений в каждой группе, то можно воспользоваться аргументом `weight`. Например, мы могли бы автоматически подсчитать и изобразить на графике суммарное количество инфузорий, обнаруженных в дрейссене из озер Нарочь, Мястро и Баторино за весь период проведения исследований (рис. 2.17).

Код для рис. 2.17

```
qplot(Lake, data = dreissena, geom = "bar")
qplot(Lake, data = dreissena, geom = "bar",
      weight = Infection)
```

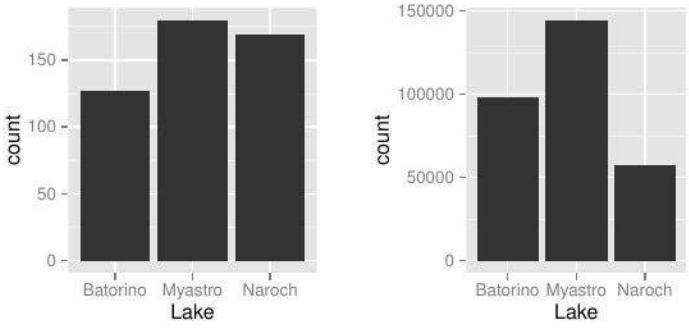


Рисунок 2.17. Примеры столбиковых диаграмм. *Слева*: число особей *D. polymorpha*, обследованных в каждом из трех озер. *Справа*: суммарное число инфузорий, обнаруженное в моллюсках из каждого озера

Мы можем добавить дополнительную информацию к столбиковой диаграмме, разбив данные на подгруппы в соответствии с уровнями второй качественной переменной. Например, к рис. 2.17 (слева) можно добавить информацию о том, сколько особей *D. polymorpha* было обследовано в каждом месяце. Для этого необходимо воспользоваться аргументом `fill` со значением `"Month"`, что приведет к разбиению каждого исходного столбика на три сегмента (май, июль, сентябрь) с соответствующими высотой и цветом (рис. 2.18). График такого типа называют *составной столбиковой диаграммой*, или *диаграммой с накоплением*.

Код для рис. 2.18

```
qplot(Lake, data = dreissena, geom = "bar", fill = Month)
```

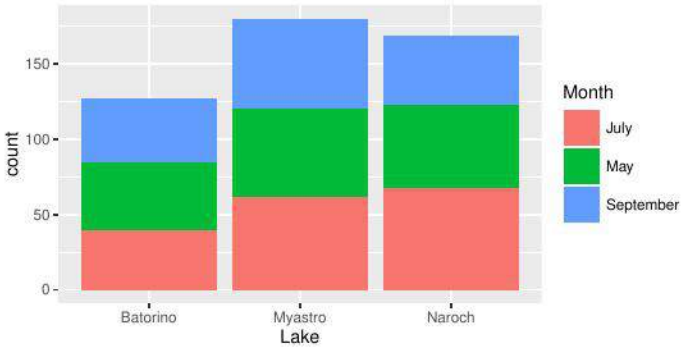


Рисунок 2.18. Пример столбиковой диаграммы, на которой данные разбиты на группы в соответствии с уровнями двух качественных переменных

2.4 Категоризованные графики

В разд. 2.2 было показано, как эстетические атрибуты «цвет» и «форма» помогают обозначить данные, принадлежащие к разным группам. Одним из альтернативных способов визуализации свойств нескольких групп данных является использование т. н. *категоризованных графиков*. Этот способ заключается в том, что для каждой группы строят отдельный график определенного типа, а затем все эти графики (их еще называют *«панелями»*) компонуют на одном рисунке. Такой подход значительно облегчает сравнительный анализ нескольких групп данных.

Для построения категоризованных графиков с помощью функции `qplot()` необходимо воспользоваться ее аргументом `facet` («грань», «ячейка»). На этот аргумент подают формулу в виде `rowvar~colvar`, где `rowvar` и `colvar` — качественные переменные, определяющие упорядочение отдельных «панелей» по строкам и столбцам соответственно. Если стоит задача разбить данные в соответствии с уровнями только одной качественной переменной, то вместо второй переменной в формуле приводят точку (например, `rowvar~.`). Примеры категоризованных графиков представлены на рис. 2.19, 2.20 и 2.21.

Код для рис. 2.19

```
qplot(log(Infection + 1), data = dreissena,
      facets = Lake ~ ., geom = "histogram", binwidth = 0.2)
```

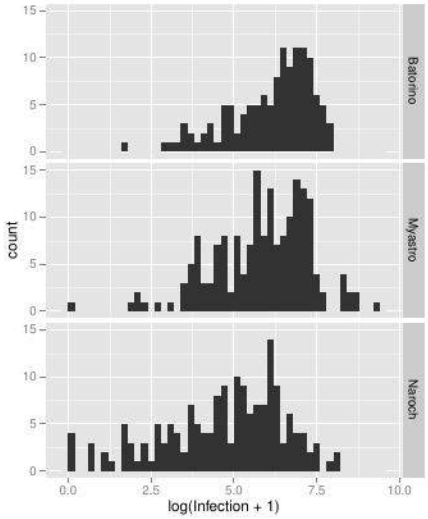


Рисунок 2.19. Распределения значений интенсивности инвазии дрейссены инфузорией *C. acuminatus* в озерах Нарочь, Мястро и Баторино

Код для рис. 2.20

```
qplot(Lake, log(Infection + 1), data = dreissena,
     facets = . ~ Month, geom = "boxplot",
     colour = I("blue"), outlier.colour = "red")
```

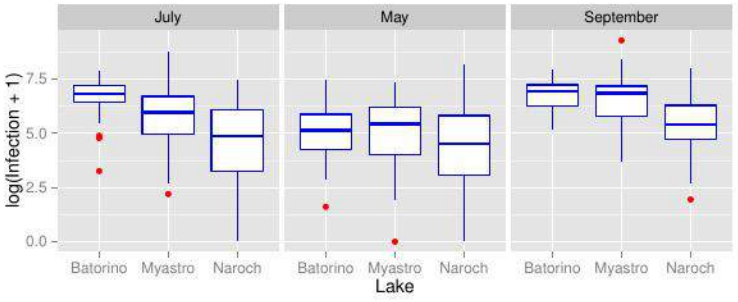


Рисунок 2.20. Количество инфузорий *C. acuminatus* в дрейссене из озер Нарочь, Мястро и Баторино в разные месяцы 2003 г.

Код для рис. 2.21

```
qplot(Length, log(Infection + 1), data = dreissena,
      facets = Lake ~ Month, geom = c("point","smooth"))
```

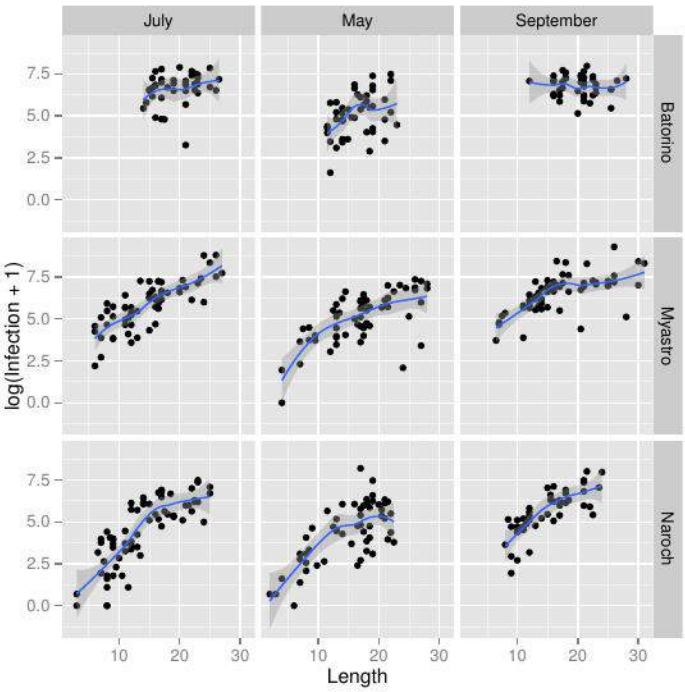


Рисунок 2.21. Зависимость между интенсивностью инвазии *C. acuminatus* и длиной раковины моллюсков *D. polymorpha* из проб, собранных в озерах Нарочь, Мястро и Баторино в разные месяцы 2003 г.

Глава 3

Построение графиков слой за слоем

Описанная в предыдущей главе функция `qplot()` позволяет строить разнообразные графики с использованием очень лаконичного синтаксиса, и для многих целей этой функции вполне может оказаться достаточно (например, при выполнении разведочного анализа данных). Однако для раскрытия полного потенциала пакета `ggplot2` от пользователя потребуется освоение другой его функции — `ggplot()`, также целого ряда функций, контролирующих свойства разнообразных геометрических объектов. О них и пойдет речь в этой главе.

3.1 Аргументы функции `ggplot()`

Рассмотренная нами ранее функция `qplot()` автоматически инициировала новый график, добавляла к нему слои с геометрическими объектами и выводила результат на экран. Однако для более детальной настройки графика следует использовать функцию `ggplot()` в связке с подходящими ситуации вспомогательными функциями. `ggplot()` имеет два основных аргумента:

- `data` — имя таблицы с данными, на основе которых строится график;
- `aes` (от англ. *aesthetics*, что значит «эстетика») — функция, которая присваивает эстетические атрибуты геометрическим объектам, используемым для изображения данных на графике. У разных типов геометрических объектов эти атрибуты будут разными.

Вот, например, как можно изобразить значения переменной `Length` по оси X, а значения переменной `Infection` — по оси Y и раскрасить точки в соответствии со значениями переменной `Month`, используя данные из таблицы `dreissena`:

```
p <- ggplot(data = dreissena, aes(x = Length,
     y = Infection, colour = Month))
```

Однако при попытке вывести этот график на графическое устройство R (например, командой `print(p)`) программа выдаст ошибку `Error: No layers in plot`. Не отчаивайтесь — возникновение этой ошибки вполне логично. Просто на графике пока еще нечего показывать, поскольку мы не добавили ни одного *слоя* (англ. *layer*) с геометрическими объектами, изображающими данные!

3.2 Слои

Создание графиков путем наложения нескольких слоев с геометрическими объектами друг на друга является особенностью пакета `ggplot2` и в какой–то мере напоминает работу с изображениями в программе Adobe Photoshop. Добавление слоев осуществляется при помощи функции `layer()`, которая в простейшем случае задает тип геометрического объекта для отображения данных (обратите внимание на использование знака `+` для добавления слоя — это характерная часть синтаксиса `ggplot2`):

```
p <- p + layer(geom = "point")
```

Функция `layer()` принимает много разных аргументов, позволяя выполнять тонкую настройку элементов соответствующего слоя. Например, в результате выполнения приведенных ниже команд будет построена гистограмма со столбиками голубого цвета, на которой данные разбиты на классы с промежутком 100 (рис. 3.1).

Код для рис. 3.1

```
p <- ggplot(data = dreissena, aes(x = Infection))
p + layer(geom = "bar", stat = "bin",
          position = "identity",
          params = list(fill = "blue", binwidth = 100))
```

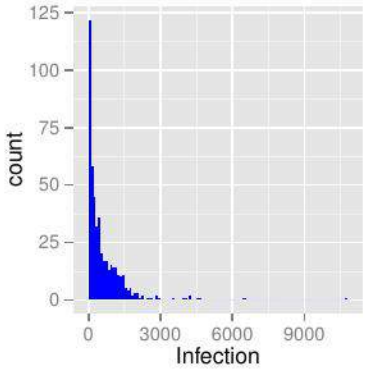


Рисунок 3.1. Пример гистограммы, построенной при помощи функций `ggplot()` и `layer()`

Как видим, с помощью функции `layer()` можно детально описать все желаемые настройки. Хотя ее использование значительно облегчает понимание структуры создаваемого графика, код очень быстро может стать длинным и плохо читаемым, особенно при построении сложных пользовательских графиков с несколькими слоями. В связи с этим в пакете `ggplot2` реализован целый ряд функций «быстрого доступа», позволяющих сделать код лаконичнее. Использование таких функций возможно благодаря тому, что каждый тип геометрических объектов, используемых для отображения данных, имеет свой набор заданных по умолчанию параметров. Например, каждому типу геометрических объектов соответствует свой тип статистического преобразования (разбиение на классы в случае с гистограммами, расчет медианы и квартилей в случае с диаграммами размахов и т. д.). С другой стороны, каждому типу статистических преобразований соответствует свой заданный по умолчанию тип геометрических объектов. Благодаря этому принципу для полного определения свойств слоя пользователю достаточно указать либо необходимый тип `geom`, либо необходимый тип `stat`. Например, следующая команда эквивалента приведенному выше коду для рис. 3.1:

```
p + geom_histogram(binwidth = 100, fill = "blue")
```

Названия всех функций быстрого доступа имеют следующий вид (подробные примеры работы с этими функциями будут приведены в главе 4):

```
geom_XXX(mapping, data, position, ...)
stat_XXX(mapping, data, position, ...)
```

где

- `XXX` — тип геометрического объекта или статистического преобразования (например, `geom_histogram`, `geom_boxplot`, `geom_smooth`, `stat_bin`, `stat_density` и т. д.);
- `mapping` — аргумент, задающий эстетические атрибуты геометрических объектов. Используется в виде `mapping = aes(x, y, ...)`, например:

  ```
  mapping = aes(x = Length, y = Infection, colour = Month)
  ```

  Подобно аргументам большинства функций в `R`, имя аргумента `mapping` можно в явном виде не указывать — достаточно привести только команду `aes(...)`[1]:

  ```
  geom_histogram(aes(x = Infection, colour = Month),
                 binwidth = 100, fill = "blue")
  ```

- `data` — имя таблицы с данными, которые необходимы для создания добавляемого слоя. Обычно этот аргумент опускают, поскольку в большинстве случаев для создания слоя используются данные из основной таблицы, уже заданной при помощи соответствующего аргумента функции `ggplot()`;

[1] Этот прием будет использоваться нами на протяжении всей книги.

- `position` — определяет способ взаимного расположения перекрывающихся геометрических объектов;
- `...` — другие параметры, характерные для соответствующего типа `geom` или `stat` (например, `binwidth` для `geom_histogram()` или `bandwidth` для `geom_smooth()`).

Следует отметить, что слои можно добавлять к графикам, созданным как при помощи функции `ggplot()`, так и при помощи функции `qplot()`. Это и неудивительно — ведь `qplot()` делает все то же самое, что и функция `ggplot()` (т. е. инициирует новый графический объект и добавляет к нему слои с соответствующими геометрическими объектами), но в автоматическом режиме. Ниже приведены примеры, когда использование `qplot()` и `ggplot()` приведет к получению идентичных результатов:

```
# Пример 1. Диаграмма рассеяния:
qplot(Length, Infection, data = dreissena)
# эквивалентная ggplot-команда:
ggplot(data = dreissena, aes(Length, Infection)) + geom_point()

# Пример 2. Диаграмма рассеяния со сглаживающей кривой:
qplot(Length, Infection, data = dreissena,
      geom = c("point", "smooth"))
# эквивалентная ggplot-команда:
ggplot(data = dreissena, aes(Length, Infection)) +
       geom_point() + geom_smooth()

# Пример 3: гистограмма
qplot(data = dreissena, Infection,
      geom = "histogram", fill = I("red"))
# эквивалентная ggplot-команда:
ggplot(data = dreissena, aes(Infection)) +
       geom_histogram(fill = I("red"))
```

Объекты, создаваемые функциями `ggplot()` и `qplot()`, являются обычными для R *списками*. Следовательно, сводную информацию о таких объектах можно получить с помощью обычной функции `summary()`:

```
p <- ggplot(data = dreissena, aes(Length, Infection))

summary(p)


data: Month, Day, Lake, Station, Length, Infection [476x6]
mapping:  x = Length, y = Infection
faceting: facet_null()
```

Добавив к графику слой, мы увидим информацию и о нем:

```
p <- ggplot(data=dreissena, aes(Length, Infection)) + geom_point()

summary(p)
```

```
data: Month, Day, Lake, Station, Length, Infection [476x6]
mapping:  x = Length, y = Infection
faceting: facet_null()
-----------------------------------
geom_point: na.rm = FALSE
stat_identity:
position_identity: (width = NULL, height = NULL)
```

Слои тоже представляют собой самостоятельные объекты, которые можно сохранить в виде отдельных переменных и затем использовать по мере необходимости. Это значительно упрощает написание кода и облегчает его чтение. В приведенном ниже примере сначала создается слой `smoother` с командами для построения сглаживающей кривой, а затем этот слой добавляется к диаграмме рассеяния:

```
smoother <- geom_smooth(se = T, colour = I("red"), size = 2)
p <- ggplot(data = dreissena, aes(log(Length),
            log(Infection+1))) + geom_point()
p + smoother
# эквивалентная команда:
qplot(log(Length), log(Infection+1), data = dreissena) + smoother
```

3.3 Требования к данным

В отличие от стандартных графических функций R, которые могут работать с векторами, списками и таблицами данных, функции `qplot()` и `ggplot()` ожидают на входе таблицы данных (*«data frames»* в терминах R). Наряду с некоторыми другими преимуществами такого ограничения, пользователь получает возможность без труда воспроизводить один и тот же тип графика для разных наборов данных — достаточно лишь подать новую таблицу на уже существующий график. Для выполнения такой замены применяется оператор `%+%` (рис. 3.2).

Код для рис. 3.2

```
p <- ggplot(data = dreissena, aes(Lake, Length)) +
            geom_boxplot()
# Исключим данные по озеру Баторино:
newdata <- dreissena[dreissena$Lake != "Batorino", ]
p %+% newdata
```

При создании графического объекта с помощью `ggplot()` или `qplot()` копия таблицы с данными *включается в состав этого объекта*. Такой подход имеет два важных следствия: 1) при изменении исходных данных график автоматически не изменится; 2) графические объекты, созданные при помощи функций пакета `ggplot2`, совершенно независимы от исходной таблицы данных — их можно сохранить на жесткий диск (при помощи команды `save()`; например, `save(p, "myplot.RData")`) и при необходимости загрузить с него в другой сессии R (при помощи команды `load()`; например, `load("myplot.RData")`).

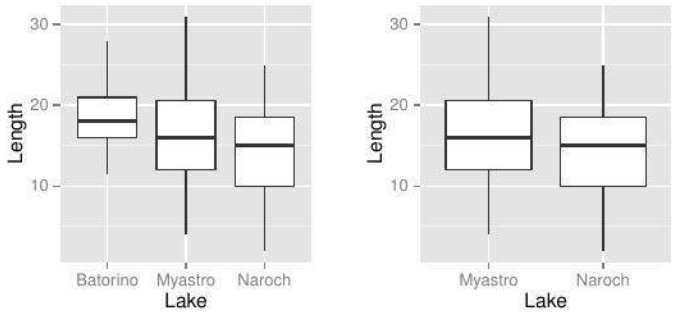


Рисунок 3.2. Результат вывода на печать одного и того же `ggplot`–объекта, на который были поданы разные наборы данных. См. объяснения в тексте

3.4 Присваивание эстетических атрибутов

Как было отмечено в разд. 3.1, присваивание эстетических атрибутов геометрическим объектам выполняется при помощи функции `aes()`[2]. В качестве аргументов эта функция принимает перечень пар `атрибут = переменная`, как, например, в следующей команде:

```
aes(x = Length, y = Infection, colour = Lake)
```

В приведенном примере мы сообщаем программе о том, что по оси X необходимо отложить значения переменной `Length`, по оси Y — значения переменной `Infection`, и что цвет геометрических объектов на графике (например, точек) должен обозначать уровни качественной переменной `Lake`. Имена первых двух аргументов (`x` и `y`) можно в явном виде не указывать — достаточно лишь указать их значения (однако порядок важен — первым должно идти значение `x`, а за ним — значение `y`):

```
aes(Length, Infection, colour = Lake)
```

Все переменные, подаваемые на функцию `aes()`, должны храниться в таблице с данными, на основе которой строится весь график или соответствующий добавляемый к графику слой (см. разд. 3.3). Ссылки на переменные из внешних таблиц (например, `table$variable`) не допускаются.

Присваивание эстетических атрибутов может быть выполнено как в ходе инициализации графического объекта функцией `ggplot()`, так и позднее при добавлении соответствующего слоя к этому объекту. Все три приведенных ниже набора команд дадут идентичный результат — построение графика зависимости интенсивности инвазии *C. acuminatus* от длины раковины дрейссены, на котором цвет точек будет обозначать озеро, из которого были отобраны пробы.

[2] В английском языке для обозначения этой процедуры применяется термин «*mapping*».

```
# Вариант 1: сначала происходит инициализация объекта p, а
# затем присваиваются эстетические атрибуты (функция aes())
# и добавляется слой с точками (geom_point())

p <- ggplot(data = dreissena)
p + aes(Length, Infection, colour = Lake) + geom_point()

# Вариант 2: инициализация объекта p с одновременным
# присваиванием эстетических атрибутов и последующим
# добавлением слоя с точками (geom_point())

p <- ggplot(data = dreissena,
            aes(Length, Infection, colour = Lake))
p + geom_point()

# Вариант 3: сначала происходит инициализация объекта p;
# затем к нему добавляется слой с точками таким образом,
# что присваивание эстетических атрибутов происходит
# внутри самой команды geom_point()

p <- ggplot(data = dreissena)
p + geom_point(aes(Length, Infection, colour = Lake))
```

Часто возникает необходимость присвоить тому или иному эстетическому атрибуту некоторое фиксированное значение. Предположим, что вместо трех цветов, обозначающих наблюдения из разных озер, мы решили использовать только один цвет (например, голубой вместо заданного по умолчанию черного цвета). Это можно сделать, присвоив соответствующее значение (`"blue"`) параметру `colour` слоя с точками (рис. 3.3).

Код для рис. 3.3

```
ggplot(data = dreissena, aes(Length, Infection)) +
       geom_point(colour = "blue")
ggplot(data = dreissena, aes(Length, Infection)) +
       geom_point(aes(colour = "blue"))
```

Обратите внимание на то, что в коде для рис. 3.3 значение `"blue"` было присвоено параметру `colour` без использования функции `aes()`. Дело в том, что вызов этой функции в виде `geom_point(aes(colour = "blue"))` привел бы к автоматическому созданию в таблице с данными новой качественной переменной с одним–единственным значением — `"blue"`, и программа восприняла бы это значение просто как уровень вновь созданной качественной переменной, а не как название цвета (см. результат на рис. 3.3 справа — точки имеют не голубой, а автоматически выбранный программой коралловый цвет)[3].

[3] См. также разд. 2.2 и примеры использования функции `I()` для присваивания значений эстетических атрибутов.

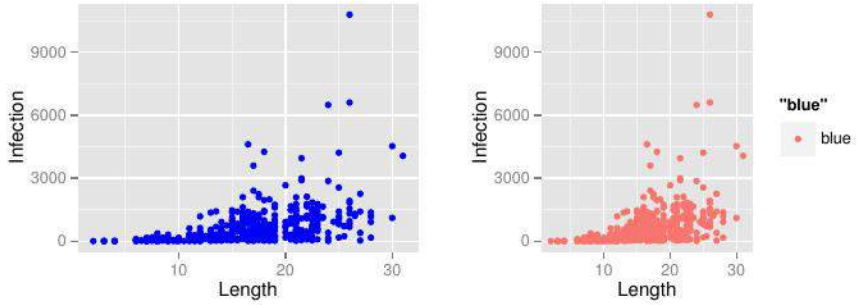


Рисунок 3.3. Результаты построения одного и того же графика с использованием команд `geom_point(colour = "blue")` и `geom_point(aes(colour = "blue"))`. См. объяснения в тексте

3.5 Группирование данных

Предположим, что перед нами стоит задача изобразить сезонную динамику среднего количества инфузорий *C. acuminatus* в дрейссене в каждом из трех исследованных озер. Для начала рассчитаем эти средние значения (в приведенном ниже примере для этого использован пакет `doBy`, однако тот же результат можно было бы достичь многими другими способами).

```
library(doBy)
# Функция summaryBy() из пакета doBy использована для расчета
# средних значений интенсивности инвазии в каждом озере
# для каждого дня отбора проб:
InfectionMeans <- summaryBy(Infection ~ Lake + Day,
                            data = dreissena)
names(InfectionMeans)[3] <- "Infection"

# Содержимое таблицы InfectionMeans:
#       Lake Day Infection
# 1 Batorino   1     316.4
# 2 Batorino  62    1022.5
# 3 Batorino 109    1024.0
# 4  Myastro   1     356.6
# 5  Myastro  59     786.5
# 6  Myastro 108    1251.8
# 7   Naroch   0     262.7
# 8   Naroch  58     289.4
# 9   Naroch 108     510.8
```

Полученные средние значения изображены на рис. 3.4.

Код для рис. 3.4

```
ggplot(data = InfectionMeans, aes(Day, Infection)) +
       geom_point()
```

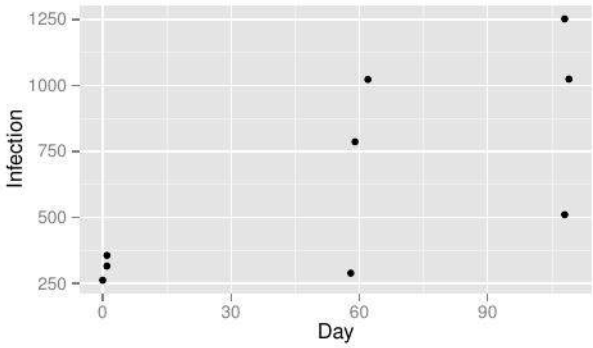


Рисунок 3.4. Средние значения количества инфузорий *C. acuminatus* в дрейссены из трех исследованных озер. Все точки имеют одинаковый цвет, в связи с чем невозможно сказать, к какому из озер относится то или иное наблюдение

Рисунок 3.4 вышел не очень информативным. Во–первых, все точки на нем оказались одного цвета, из–за чего мы не можем соотнести их с конкретными водоемами. Во–вторых, временные изменения количественных переменных принято изображать с использованием линий, а не в виде отдельных точек. Для изображения временных рядов в пакете `ggplot2` имеется функция `geom_line()`. Воспользовавшись ею, мы получим рис. 3.5.

Код для рис. 3.5

```
ggplot(data = InfectionMeans, aes(Day, Infection)) +
       geom_line()
```

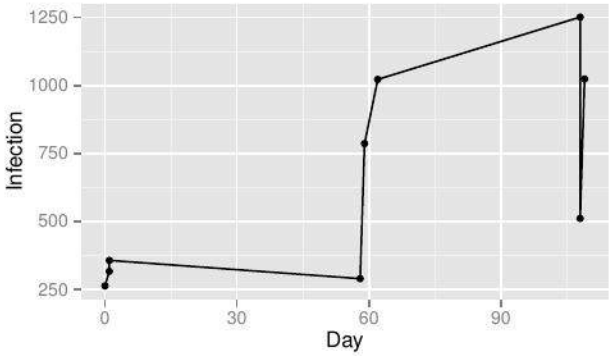


Рисунок 3.5. Пример применения функции `geom_line()` к сгруппированным данным. Эта функция не способна автоматически распознать группы данных, из–за чего происходит просто последовательное соединение точек

Как видно на рис. 3.5, функция `geom_line()` не была «осведомлена» о том, что наши данные сгруппированы по озерам. В результате точки оказались последовательно соединены одной линией. Чтобы сообщить программе о наличии сгруппированных данных, необходимо воспользоваться аргументом `group` функции `aes()`. В качестве значения этого аргумента указывают имя группирующей переменной (`Lake` в рассматриваемом примере; рис. 3.6).

Код для рис. 3.6

```
ggplot(data = InfectionMeans, aes(Day, Infection,
       group = Lake)) + geom_line()
```

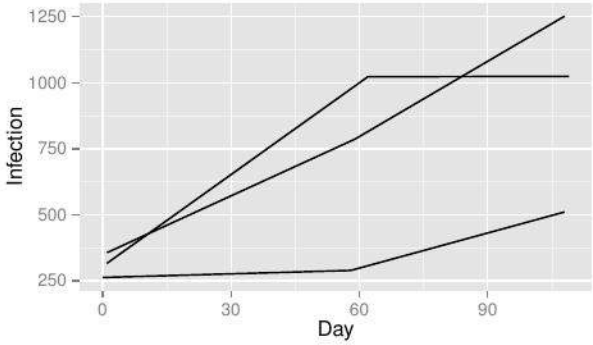


Рисунок 3.6. Пример визуализации динамики количественной переменной, чьи значения сгруппированы в соответствии со значениями другой, качественной переменной

Хотя на рис. 3.6 теперь четко видны три группы данных, мы все еще не можем идентифицировать эти группы, поскольку линии не обладают никакими отличительными эстетическими атрибутами. Это легко исправить: достаточно, например, присвоить линиям разные цвета (рис. 3.7).

Код для рис. 3.7

```
ggplot(data = InfectionMeans,
       aes(Day, Infection, group = Lake, colour = Lake)) +
       geom_line()
```

3.6 Геометрические объекты, реализованные в ggplot2

Как мы уже знаем, на статистических графиках данные изображаются в виде определенных геометрических объектов. На момент написания этой

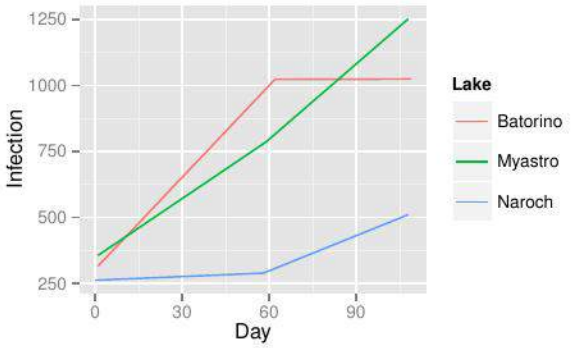


Рисунок 3.7. Временные изменения количественной переменной, чьи значения сгруппированы в соответствии со значениями другой, качественной переменной. Для идентификации групп использован эстетический атрибут «цвет»

книги в пакете `ggplot2` было реализовано более 35 типов таких объектов (табл. 3.1)[4].

Каждый геометрический объект в `ggplot2` обладает свойственным ему набором эстетических атрибутов. Например, объект типа `"point"` (точка) характеризуется прежде всего координатами X и Y. Кроме того, у точки может быть определенная форма, размер и цвет. Объект `"bar"` («столбик») можно описать по таким параметрам, как высота, ширина и цвет, а также тип и цвет окаймляющей его линии. Эстетические атрибуты всех основных типов геометрических объектов `ggplot2` будут подробно рассмотрены в следующей главе.

Каждому типу геометрических объектов соответствует свой заданный по умолчанию тип статистического преобразования исходных данных (и наоборот — каждому преобразованию соответствует свой объект). Например, преобразование типа `"bin"` автоматически применяется для разбиения данных на классы в процессе построения гистограмм. Допускается использование нестандартных статистический преобразований (т. е. несвойственных для конкретного типа геометрических объектов), однако не стоит удивляться, если в результате применения таких преобразований полученный график будет выглядеть «странно».

Таблица 3.1. Основные типы геометрических объектов, реализованные в `ggplot2`

Объект	Результат применения объекта
`abline`	Линия, заданная уравнением $y = a + bx$
`area`	Площадь под кривой
`bar`	Столбиковая диаграмма
`bind2d`	Тепловая карта для двух переменных, значения которых разбиты на классовые промежутки

[4] Актуальный перечень всегда доступен на сайте с официальной документацией по `ggplot2`: `http://docs.ggplot2.org/current`.

Таблица 3.1: *продолжение*

Объект	Результат применения объекта
`blank`	Пустой слой
`boxplot`	Диаграмма размахов
`contour`	Контурная диаграмма
`crossbar`	Прямоугольник, внутри которого изображена линия, параллельная его торцам; линия соответствует медиане или среднему значению
`density`	Диаграмма плотности вероятности
`density2d`	2D–диаграмма плотности вероятности
`dotplot`	Точечная диаграмма Уилкинсона
`errorbar`	Диаграмма диапазонов (с использованием вертикальных отрезков)
`errorbarh`	Диаграмма диапазонов (с использованием горизонтальных отрезков)
`freqpoly`	Полигон частот
`hex`	«Сотовая диаграмма»: координатная плоскость разбита на гексагоны, цвет заливки которых соответствует плотности расположения точек
`histogram`	Гистограмма
`hline`	Горизонтальная линия
`jitter`	Точечная диаграмма, на которой к координатам точек добавлен небольшой «шум»
`line`	Линия, соединяющая упорядоченные по оси X наблюдения
`linerange`	Один из вариантов диаграммы диапазонов (с использованием вертикальных отрезков)
`map`	Географическая карта (и другие похожие многоугольники)
`path`	Линия, соединяющая наблюдения в порядке, который задан одной из переменных в таблице с данными
`point`	Диаграмма рассеяния
`pointrange`	Точка с исходящими из нее отрезками
`polygon`	Многоугольник
`quantile`	Линии квантильной регрессии
`raster`	Растровое изображение
`rect`	Прямоугольник
`ribbon`	Ленточная диаграмма
`rug`	Небольшие перпендикулярные координатной оси отрезки, обозначающие отдельные наблюдения
`segment`	Линии, координаты начала и конца которых заданы пользователем
`smooth`	Сглаживающая линия (линия тренда)
`step`	Эмпирическая кумулятивная функция плотности вероятности
`text`	Текстовые аннотации
`tile`	Плоскость, разбитая на прямоугольники
`violin`	«Скрипичная диаграмма»: смесь диаграммы размахов с диаграммой плотности вероятности
`vline`	Вертикальная линия

3.7 Статистические преобразования

Как следует из их названия, *статистические преобразования* представляют собой функции, которые выполняют определенные математические операции над исходными данными. Статистические преобразования, реализованные в пакете `ggplot2` на момент написания этой книги, перечислены в табл. 3.2.

Таблица 3.2. Основные типы статистических преобразований, реализованные в `ggplot2`

Объект	Описание
`bin`	Разбиение данных на классы
`bin2d`	Разбиение данных на классы для построения двухмерных диаграмм плотности вероятности
`bindot`	Сортировка данных для построения точечных диаграмм
`binhex`	Разбиение данных на классы для построения «сотовых диаграмм»
`boxplot`	Вычисления, необходимые для построения диаграмм размахов
`contour`	Вычисления, необходимые для построения контурных диаграмм
`density`	Одномерное ядерное оценивание плотности вероятности
`density2d`	Двухмерное ядерное оценивание плотности вероятности
`ecdf`	Сортировка данных для построения эмпирической кумулятивной функции плотности вероятности
`function`	Любая пользовательская функция для преобразования данных
`identity`	Возвращает данные в неизменном виде
`qq`	Вычисления, необходимые для построения квантильных графиков
`quantile`	Расчет линий квантильной регрессии
`smooth`	Подгонка сглаживающей линии
`summary`	Позволяет создавать пользовательские функции для расчета сводных статистических показателей по значениям переменной Y для каждого уникального значения переменной X
`summary_hex`	Позволяет создавать пользовательские функции для расчета показателей, отражаемых на «сотовых диаграммах»
`summary2d`	Позволяет создавать пользовательские функции для расчета показателей, отражаемых на двухмерных диаграммах плотности вероятности
`unique`	Удаление повторяющихся значений из выборки
`ydensity`	Одномерное ядерное оценивание плотности вероятности, необходимое для построения диаграмм типа «виолончель»

Как было отмечено в разд. 3.2, функции, выполняющие статистические преобразования, имеют в своем названии приставку `"stat"` (например, `stat_bin()`, `stat_point()`, `stat_boxplot()` и т. д.). Эти функции принимают таблицу с исходными данными и возвращают таблицу с новыми переменными, содержащими результаты вычислений. Часто новые перемен-

ные добавляются непосредственно в исходную таблицу[5], и им можно присвоить эстетические атрибуты. Так, `stat_qq()` — функция, применяемая для построении графиков нормальной вероятности, — возвращает таблицу со следующими дополнительными переменными: `sample` (выборочные квантили) и `theoretical` (теоретически ожидаемые квантили нормального распределения). Пример использования `stat_qq()` приведен на рис. 3.8.

Код для рис. 3.8

```
ggplot(dreissena, aes(sample = Infection, colour = Lake)) +
       stat_qq()
```

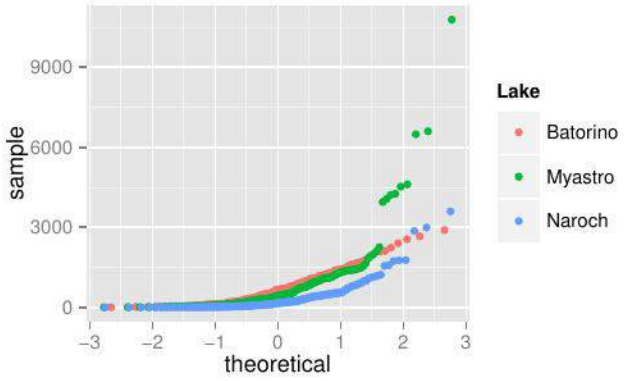


Рисунок 3.8. Пример графика нормальной вероятности

К дополнительным переменным, которые создаются функциями статистических преобразований, можно обращаться, окружая их имена двойными точками (например, `..sample..` и `..theoretical..` в случае с `stat_qq()`; рис. 3.9). Наличие двойных точек позволяет избежать возникновения ошибок при совпадении имен новых переменных с именами переменных, которые могли присутствовать в данных изначально. Кроме того, наличие двойных точек помогает идентифицировать новые переменные при написании кода и его чтении. С перечнем имен переменных, создаваемых функциями статистических преобразований, можно ознакомиться в соответствующих справочных файлах.

Код для рис. 3.9

```
# По умолчанию по оси X графика типа "квантиль-квантиль"
# откладываются теоретически ожидаемые квантили нормального
# распределения, а по оси Y - выборочные наблюдения.
# Мы можем изменить такое поведение программы и
# "перевернуть" оси следующим образом:
```

[5] Имеется в виду копия таблицы с данными, входящая в состав графического объекта, — см. разд. 3.3.

```
ggplot(data = dreissena, aes(sample = Infection,
       colour = Lake)) +
       stat_qq(aes(x = ..sample.., y = ..theoretical..))
```

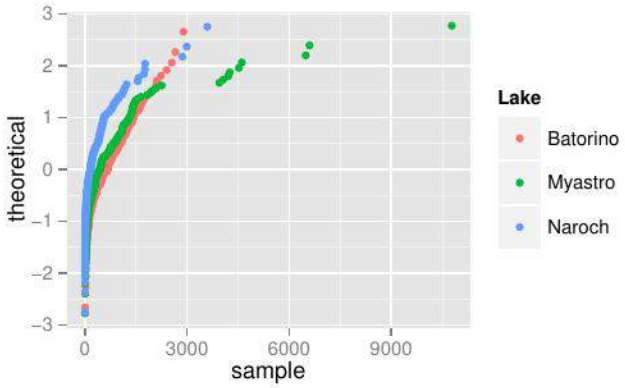


Рисунок 3.9. График нормальной вероятности с повернутыми осями

Глава 4

Основные типы статистических графиков

В этой главе собраны многочисленные примеры использования функции `ggplot()` и функций, реализующих основные типы геометрических объектов. Материал разбит на темы в соответствии с типичными задачами, которые возникают при визуализации данных (например, изображение распределений анализируемых переменных, сводной информации о свойствах данных в нескольких группах, связи между двумя и более переменными и т. п.). Все подразделы построены по одной схеме, напоминающей структуру справочных файлов `ggplot2`: сначала обсуждается перечень аргументов функции, реализующей геометрический объект того или иного типа, а затем приводятся примеры использования этой функции для построения соответствующих графиков.

4.1 Общие аргументы `geom`– и `stat`–функций

Все рассматриваемые в этой главе функции спецификации геометрических объектов, т. е. функции, в названии которых есть приставка `geom` или `stat`, имеют пять одинаковых аргументов. С целью экономии места имеет смысл привести список этих аргументов только один раз и не повторять его в дальнейшем при описании каждой конкретной функции:

- `data` — имя таблицы данных, специфичной для конкретного слоя. Применяется, когда необходимо изобразить на слое данные, отличные от тех, которые были заданы при инициализации графика функцией `ggplot()`;
- `mapping` — служит для присваивания эстетических атрибутов (обычно при помощи функции `aes()`). Используется только при необходимости отменить глобальные настройки графика и задать определенные эстетические атрибуты на уровне слоя;
- `stat` — задает статистическое преобразование, которое применяется к изображаемым на слое данным.

- `position` — определяет взаимное расположение перекрывающихся геометрических объектов. Значения этого аргумента будут разными у разных функций;

- `na.rm` — логический аргумент, определяющий действия в отношении пропущенных данных. При `na.rm = FALSE` (значение, принятое по умолчанию) пропущенные наблюдения будут удалены и пользователь увидит на экране соответствующее предупреждение. При `na.rm = TRUE` пропущенные значения также будут удалены, но без вывода предупреждающего сообщения.

4.2 Визуализация одномерных распределений

Изучение характера распределения значений той или иной переменной может многое сказать о ее свойствах и потому широко используется в ходе разведочного анализа данных и для диагностики статистических моделей. Существует несколько способов визуализации одномерных распределений, что определяется как стоящей перед исследователем задачей, так и тем, является ли переменная непрерывной количественной, дискретной количественной или качественной. В этом разделе рассмотрены основные способы создания соответствующих графиков средствами `ggplot2`.

4.2.1 Точечные диаграммы Уилкинсона: `geom_dotplot()`

Точечные диаграммы Уилкинсона (англ. *Wilkinson dot plots*) служат для визуализации распределений непрерывных количественных переменных. Это один из простейших статистических графиков, хорошо подходящий для работы с малыми выборками (20–30 наблюдений). Данные на диаграмме Уилкинсона изображаются в виде точек, определенным образом упорядоченных друг над другом вдоль координатной оси. Хотя внешне диаграммы Уилкинсона напоминают гистограммы, по лежащему в их основе алгоритму они больше близки к методам ядерного оценивания плотности вероятности[1]. Графики этого типа хорошо подходят для выявления выбросов и кластеров наблюдений.

Аргументы

- `binaxis` — переменная, значения которой подлежат разбиению на классы (по умолчанию `binaxis = "x"`).

- `method` — название алгоритма, используемого для разбиения наблюдений на классы: `method = "dotdensity"` для разбиения с учетом плотности вероятности и `method = "histodot"` для разбиения с применением фиксированного классового промежутка (как при построении гистограмм).

[1] Алгоритмы, используемые в пакете `ggplot2` для построения таких диаграмм, описаны в работе: Wilkinson L. (1999) Dot plots. The American Statistician 53(3): 276–281.

- `binwidth` — при `method = "dotdensity"` задает степень сглаживания для алгоритма оценивания плотности вероятности; при `method = "histodot"` задает фиксированный размер классового промежутка. По умолчанию этот параметр равен $(max - min)/30$.

- `binpositions` — при `method = "dotdensity"` значение `binpositions = "bygroup"` указывает на необходимость расчета плотности вероятности в пределах каждой из групп данных. При `binpositions = "all"` оценивание плотности вероятности выполняется для всех имеющихся данных.

- `stackdir` — задает направление, вдоль которого точки на графике должны укладываться в «стопки». Возможные значения: `"up"` (по умолчанию), `"down"`, `"center"` и `"centerwhole"` (см. примеры ниже).

- `stackratio` — определяет степень перекрытия точек. По умолчанию `stackratio = 1` (точки почти касаются друг друга). При меньших значениях точки частично перекрывают друг друга.

- `dotsize` — относительный размер точек (по умолчанию `dotsize = 1`).

- `stackgroups` — логический аргумент, включающий тот же эффект для точечных диаграмм, что и `position = "stack"` в случае со столбиковыми диаграммами (см. следующий подраздел).

- другие аргументы — см. разд. 4.1.

Эстетические атрибуты

- `x`* и `y` — переменные X и Y соответственно[2].

- `alpha` — степень прозрачности цвета.

- `colour` — цвет линии, окаймляющей точки.

- `fill` — цвет точек.

Примеры

Код для рис. 4.1

```
ggplot(data = subset(dreissena,
       Lake == "Naroch" & Month == "May"),
       aes(x = Infection)) + geom_dotplot()

# В соответствии с автоматическими настройками
# приведенная выше команда создаст график с осью Y.
# Однако в случае с одномерной точечной диаграммой
# эта ось не имеет смысла. Ее можно отключить при
# помощи функции scale_y_continuous() следующим образом:
```

[2] Здесь и далее обязательные аргументы отмечены знаком *.

```
ggplot(data = subset(dreissena,
       Lake == "Naroch" & Month == "May"),
       aes(x = Infection)) + geom_dotplot() +
       scale_y_continuous(name = "", breaks = NULL)
```

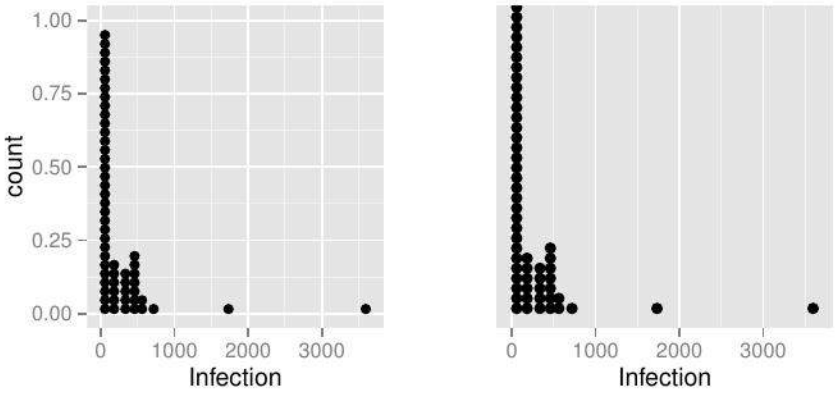


Рисунок 4.1. Пример точечной диаграммы Уилкинсона, на которой изображено распределение значений интенсивности инвазии дрейссены инфузорией *C. acuminatus* в озере Нарочь в мае 2005 г. *Слева:* график построен с использованием автоматических настроек. *Справа:* ось Y удалена за ненадобностью

Код для рис. 4.2

```
p <- ggplot(data = subset(dreissena,
            Lake == "Naroch" & Month == "May"),
            aes(x = Infection)) +
     scale_y_continuous(name = "", breaks = NULL)
p + geom_dotplot()
p + geom_dotplot(binwidth = 100)
p + geom_dotplot(binwidth = 250)
```

Код для рис. 4.3

```
p <- ggplot(data = subset(dreissena,
            Lake == "Naroch" & Month == "May"),
            aes(x = Infection)) +
            scale_y_continuous(name = "", breaks = NULL)
p + geom_dotplot(method = "histodot", binwidth = 100)
p + geom_dotplot(method = "histodot", binwidth = 250)
```

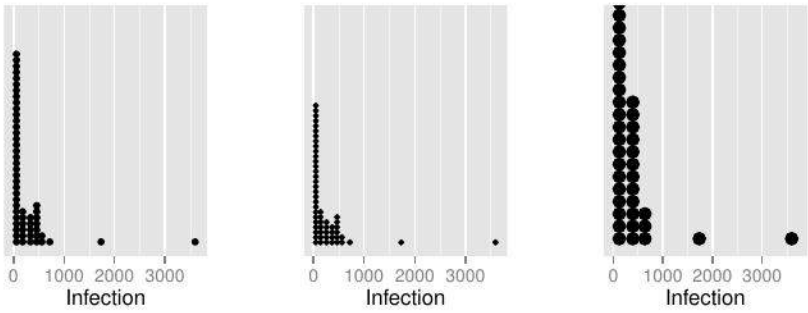


Рисунок 4.2. Точечные диаграммы Уилкинсона, построенные с использованием разных значений аргумента `binwidth`. *Слева:* значение, заданное по умолчанию (`binwidth` $= (max - min)/30$). *В центре:* `binwidth = 100`. *Справа:* `binwidth = 250`

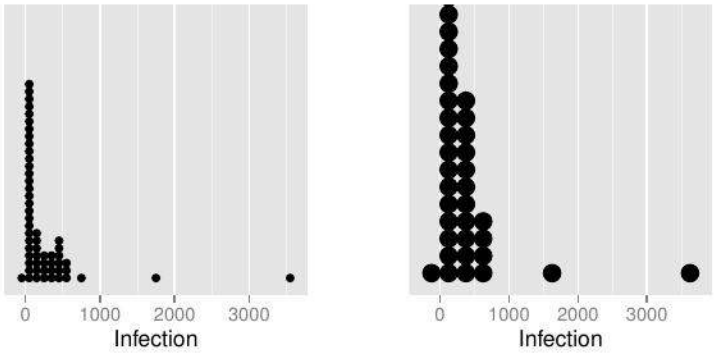


Рисунок 4.3. Точечные диаграммы Уилкинсона, построенные с использованием метода `histodot`: точки разбиваются на классы с использованием фиксированного классового промежутка, заданного параметром `binwidth`. *Слева:* `binwidth = 100`. *Справа:* `binwidth = 250`

Код для рис. 4.4

```
p <- ggplot(data = subset(dreissena,
            Lake == "Batorino" & Month == "May"),
            aes(x = Site, y = Infection))
p + geom_dotplot(binaxis = "y", stackdir = "up",
                 binwidth = 50)
p + geom_dotplot(binaxis = "y", stackdir = "down",
                 binwidth = 50)
p + geom_dotplot(binaxis = "y", stackdir = "center",
                 binwidth = 50)
```

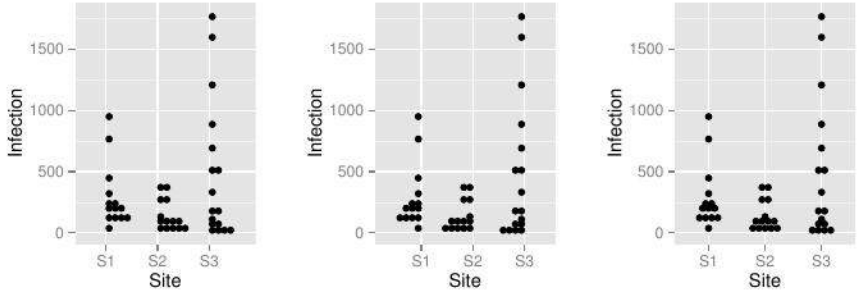


Рисунок 4.4. Иллюстрация действия аргумента `stackdir`. *Слева:* `stackdir = "up"` (значение, принятое по умолчанию). *В центре:* `stackdir = "down"`. *Справа:* `stackdir = "center"`. Обратите внимание также на то, что наблюдения объединяются в классы вдоль оси Y. В связи с этим аргументу `binaxis` присвоено значение `"y"`

Код для рис. 4.5

```
p <- ggplot(data = subset(dreissena, Lake == "Batorino"),
            aes(x = Month, fill = Site, y = Infection))
p + geom_dotplot(binaxis = "y")
p + geom_dotplot(binaxis = "y", position = "dodge")
```

4.2.2 Столбиковые диаграммы: `geom_bar()`

Столбиковые диаграммы используются для визуализации распределений как количественных, так и качественных переменных. В случае с качественной переменной на столбиковой диаграмме изображаются частоты встречаемости каждого из ее уровней (абсолютные либо относительные). В случае же с количественной переменной столбиковая диаграмма превращается в гистограмму. Слой с гистограммой в `ggplot2` можно создать как с помощью функции `geom_bar()`, так и специально предназначенной для этого функции `geom_histogram()` (см. следующий подраздел).

Довольно часто столбиковые диаграммы используются для изображения показателей центральной тенденции (среднее значение, мода, медиана). Такой подход обладает существенным недостатком: при визуализации того или иного показателя центральной тенденции важна *высота* столбика, тогда как ограниченная этим столбиком площадь не несет никакой дополнительной информации. Согласно Cleveland & McGill (1984)[3], более подходящим типом графиков для таких случаев является точечная диаграмма[4].

[3] Cleveland W. S., McGill R. (1984) Graphical perception: theory, experimentation, and application to the development of graphical methods. Journal of the American Statistical Association 79(387): 531–554.

[4] См. статью «Базовые графические возможности R: точечные диаграммы Кливленда»: `http://bit.ly/2cxNqbd`.

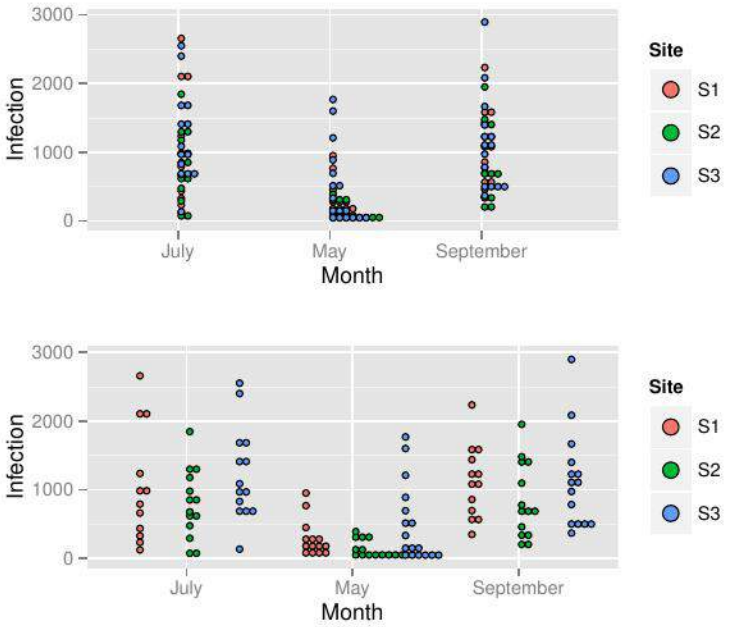


Рисунок 4.5. С помощью аргумента `fill` точкам можно присвоить атрибут «цвет» (на рисунке вверху цвет задан в соответствии со станциями отбора проб). Рисунок внизу иллюстрирует действие аргумента `position`: значение `position = "dodge"` позволяет избежать «наползания» точек из разных групп друг на друга

Аргументы

- `weight` — переменная, по значениям которой выполняется «взвешивание» значений переменной X (см. объяснения ниже).
- другие аргменты — см. разд. 4.1.

Эстетические атрибуты

- `x`* — переменная X.
- `alpha` — степень прозрачности цвета.
- `colour` — цвет линии, окаймляющей столбики.
- `linetype` — тип линии, окаймляющей столбики.
- `size` — толщина линии, окаймляющей столбики.
- `fill` — цвет заливки столбиков.

Примеры

Код для рис. 4.6

```
ggplot(data = subset(dreissena, Lake == "Batorino"),
       aes(x = Site)) + geom_bar()
ggplot(data = subset(dreissena, Lake == "Batorino"),
       aes(x = Infection)) + geom_bar()
```

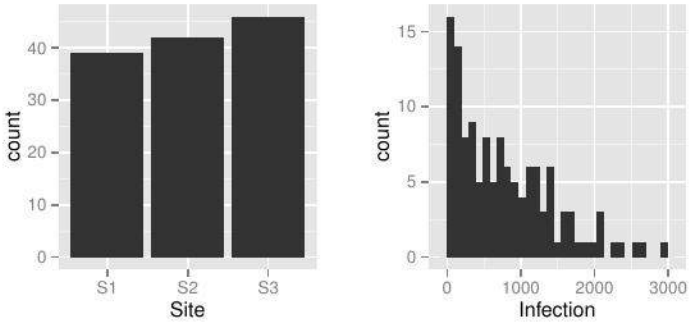


Рисунок 4.6. Примеры столбиковой диаграммы (*слева*) и гистограммы (*справа*), построенных при помощи функции `geom_bar()`

Код для рис. 4.7

```
p <- ggplot(data = subset(dreissena,
     Lake == "Batorino"), aes(x = Site))
p + geom_bar(width = 0.5)
p + geom_bar(width = 0.5) + coord_flip()
```

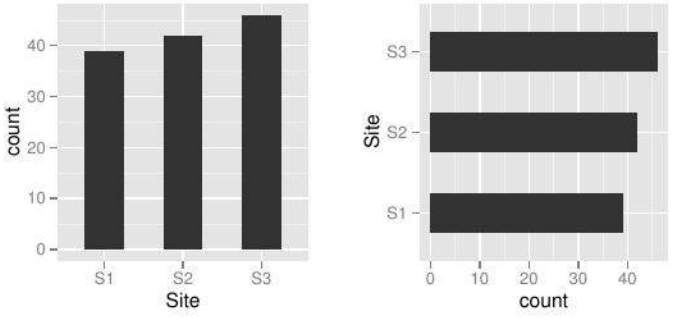


Рисунок 4.7. Настройка внешнего вида столбиковой диаграммы. *Слева:* ширина столбиков задана с помощью аргумента `width = 0.5`. *Справа:* координатные оси повернуты с помощью функции `coord_flip()`

Код для рис. 4.8

```
p <- ggplot(data = subset(dreissena,
            Lake == "Batorino"), aes(x = Site))
p + geom_bar(fill = "coral", colour = "blue",
             size = 1.2)
p + geom_bar(fill = "coral", colour = "blue",
             size = 1.2, linetype = 2)
```

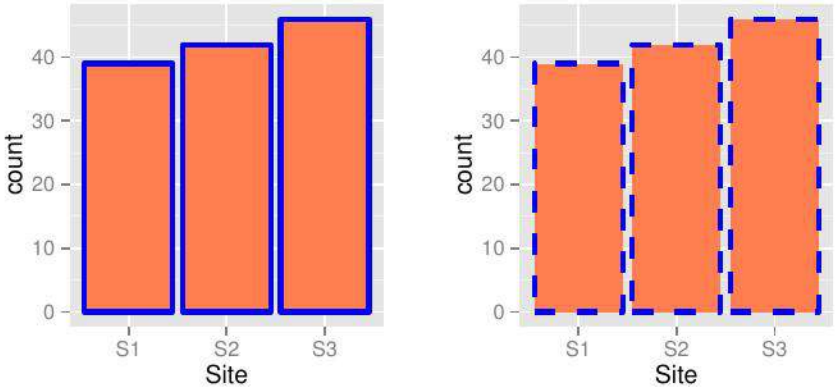


Рисунок 4.8. Настройка внешнего вида столбиковой диаграммы. *Слева:* цвет заливки столбиков задан аргументом `fill = "coral"`, цвет окаймляющей линии — `colour = "blue"`, толщина линии — `size = 1.2`. *Справа:* тип окаймляющей линии изменен при помощи аргумента `linetype = 2`

Код для рис. 4.9

```
p <- ggplot(data = subset(dreissena,
            Lake == "Batorino"), aes(x = Site))
p + geom_bar(aes(fill = Month), position = "stack")
p + geom_bar(aes(fill = Month), position = "fill")
p + geom_bar(aes(fill = Month), position = "dodge")
```

Код для рис. 4.10

```
p <- ggplot(data = subset(dreissena,
            Lake == "Naroch" & Month == "May"),
            aes(x = Site))
p + geom_bar()
p + geom_bar(aes(weight = Infection))
```

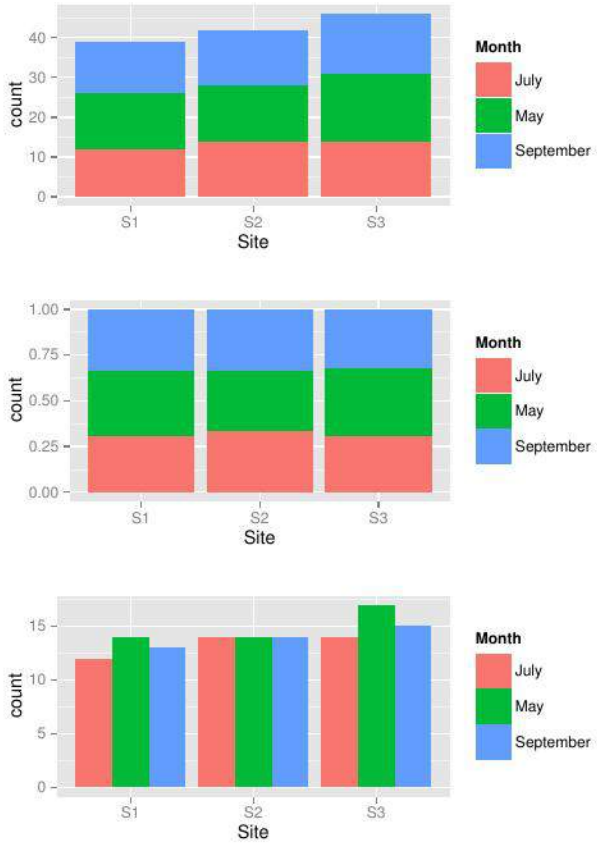


Рисунок 4.9. Три типа взаимного расположения столбиков при визуализации информации по двум качественным переменным. *Вверху:* цвет заливки столбиков задан в соответствии с датой отбора проб (`aes(fill = Month)`); столбики выстраиваются друг над другом, образуя «стопки» (`position = "stack"`). *В центре:* столбики также выстраиваются друг над другом, однако их длина пропорциональна доле от общего числа наблюдений (`position = "fill"`). *Внизу:* столбики выстраиваются бок о бок (`position = "dodge"`)

Код для рис. 4.11

```
library(doBy)
# Расчет средних значений интенсивности инвазии
# с сохранением результатов в виде таблицы данных:
meanInfection <- summaryBy(Infection ~ Lake, data = dreissena)

# Точечная диаграмма со средними значениями:
ggplot(data = meanInfection,
       aes(x = Lake, y = Infection.mean)) + geom_point()
```

```
# Столбиковая диаграмма с теми же средними значениями
# (обратите внимание на аргумент stat = "identity"
# функции geom_bar(): он заставляет эту функцию принять
# средние значения "как есть", без выполнения каких-либо
# дополнительных преобразований):
ggplot(data = meanInfection,
       aes(x = Lake, y = Infection.mean)) +
       geom_bar(stat = "identity")
```

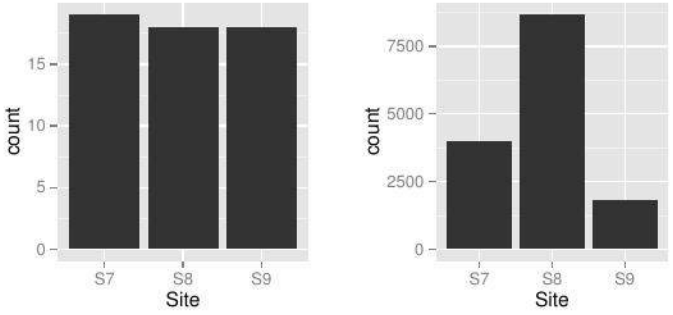


Рисунок 4.10. Иллюстрация действия аргумента `weight`. *Слева:* столбиковая диаграмма, на которой ось Y соответствует числу особей дрейссены из озера Нарочь, обследованных в мае (атрибут `weight` не задействован). *Справа:* значения переменной `Station` «взвешены» по значениям переменной `Infection` (`aes(weight = Infection)`), что привело к подсчету суммарного количества инфузорий *C. acuminatus*, обнаруженных во всех исследованных особях дрейссены на каждой станции

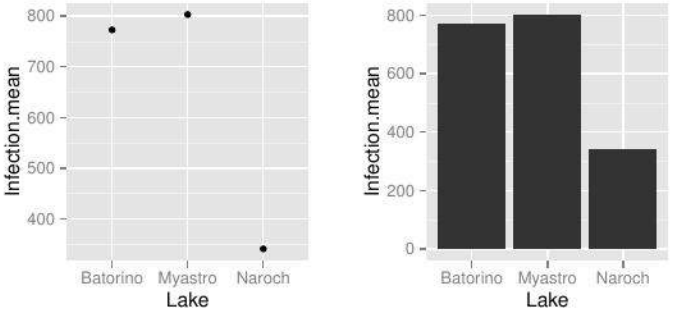


Рисунок 4.11. *Слева:* точечная диаграмма со средними значениями количественной переменной. *Справа:* столбиковая диаграмма с теми же средними значениями

4.2.3 Гистограммы: `geom_histogram()`

Гистограмма представляет собой вариант столбиковой диаграммы, применяемый для визуализации распределений количественных переменных с относительно большим размахом значений. При построении гистограммы значения анализируемой переменной упорядочиваются по возрастанию, а затем разбиваются на *классы* в соответствии с некоторым *классовым промежутком*. Получаемый в итоге график выглядит как совокупность из нескольких столбиков, ширина которых соответствует величине классового промежутка, а высота — частоте встречаемости соответствующего класса. Таким образом, гистограмма является приближением распределения плотности вероятности анализируемой переменной.

Аргументы

- `binwidth` — размер классового промежутка.
- `weight` — переменная, по значениям которой происходит «взвешивание» значений переменной X (см. рис. 4.10).
- другие аргументы — см. разд. 4.1.

Эстетические атрибуты

- `x`* — переменная X.
- `alpha` — степень прозрачности цвета.
- `colour` — цвет линии, окаймляющей столбики (см. рис. 4.8).
- `fill` — цвет заливки столбиков (см. рис. 4.8).
- `linetype` — тип линии, окаймляющей столбики (см. рис. 4.8).
- `size` — толщина линии, окаймляющей столбики (см. рис. 4.8).

Примеры

Код для рис. 4.12

```
p <- ggplot(data = dreissena, aes(x = Infection))
p + geom_histogram()
p + geom_histogram(binwidth = 50)
```

Код для рис. 4.13

```
p <- ggplot(data = subset(dreissena, Lake == "Naroch"),
            aes(x = Infection))
p + geom_histogram(aes(y = ..density..))
p + geom_histogram(aes(y = ..density..)) +
    geom_density(colour = "magenta")
```

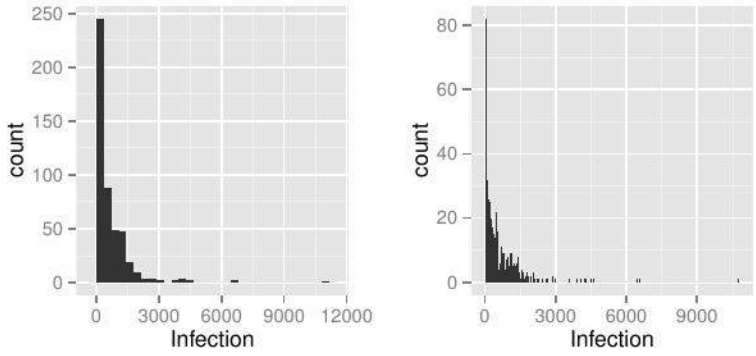


Рисунок 4.12. Примеры гистограмм с разными классовыми промежутками. *Слева:* размер промежутка задан в соответствии с автоматическими настройками $((max - min)/30)$. *Справа:* размер промежутка задан пользователем при помощи аргумента `binwidth = 50`

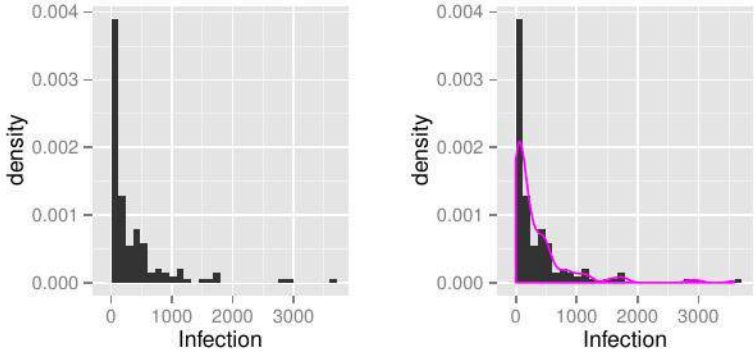


Рисунок 4.13. *Слева:* пример гистограммы, на которой по оси Y приведены значения плотности вероятности (команда `aes(y = ..density..)`; подробнее о двойных точках, окружающих имя переменной, см. разд. 3.7). *Справа:* при помощи функции `geom_density()` к той же гистограмме добавлен слой с кривой плотности вероятности

Код для рис. 4.14

```
p <- ggplot(data = subset(dreissena, Lake == "Naroch"),
            aes(x = Infection))
p + geom_histogram(aes(fill = Month))
p + geom_histogram(aes(fill = Month), position = "dodge")
```

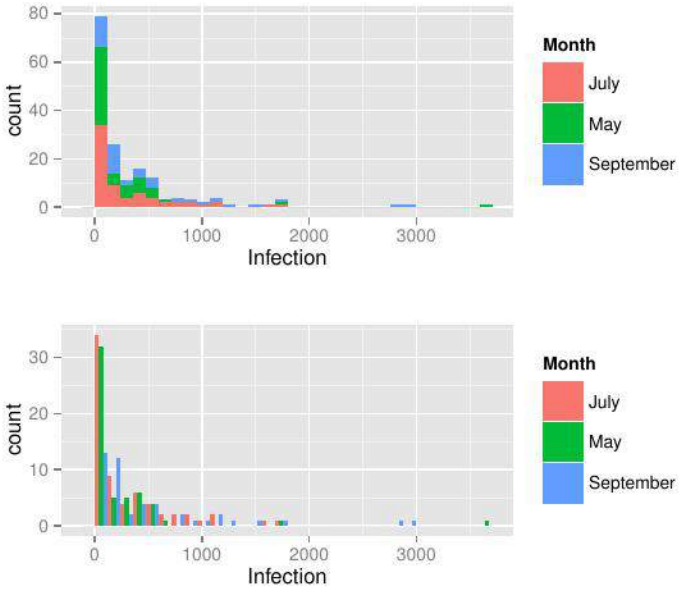


Рисунок 4.14. Управление взаимным расположением столбиков на гистограмме при помощи аргумента `position` функции `geom_histogram()`. *Вверху:* цвет столбиков задан в соответствии с датой отбора проб (`aes(fill = Month)`). По умолчанию столбики располагаются друг над другом, образуя «стопки» (`position = "stack"`; см. также рис. 4.9). *Внизу:* столбики выстраиваются бок о бок (`position = "dodge"`)

4.2.4 Полигоны частот: `geom_freqpoly()`

Подобно гистограмме, полигон частот представляет собой приближенный вариант распределения плотности вероятности количественной переменной. По сути, единственное различие между этими двумя типами графиков состоит в том, что полигон частот изображают в виде сплошной ломаной линии. Как видно на рис. 4.15, координаты узловых точек этой линии соответствуют вершинам столбиков гистограммы. Кроме того, ломаная касается оси X по обеим сторонам распределения в точках, соответствующих ближайшим классам с нулевыми значениями частот. В результате этого образуется замкнутая фигура («полигон»).

Аргументы

- `binwidth` — размер классового промежутка.
- другие аргументы — см. разд. 4.1.

Эстетические атрибуты

- `x`* — переменная X.
- `alpha` — степень прозрачности цвета.

- `colour` — цвет линии.
- `linetype` — тип линии.
- `size` — толщина линии.

Примеры

Код для рис. 4.15

```
p <- ggplot(data = subset(dreissena, Lake == "Batorino"),
            aes(x = Infection))
p + geom_histogram(binwidth = 500)
p + geom_histogram(binwidth = 500) +
    geom_freqpoly(binwidth = 500, colour = "magenta")
```

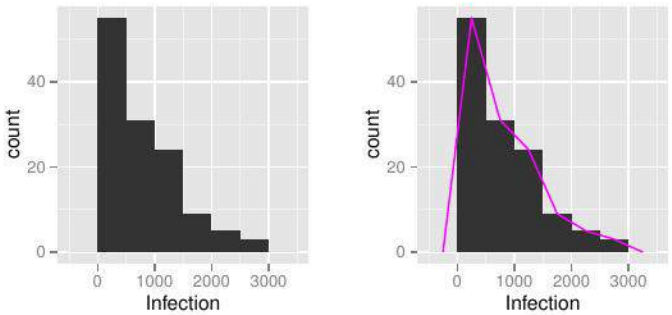


Рисунок 4.15. *Слева:* гистограмма распределения количественной переменной. *Справа:* к графику с той же гистограммой добавлен слой с полигоном распределения

Код для рис. 4.16

```
p <- ggplot(data = subset(dreissena, Lake == "Batorino"),
            aes(x = Infection))
p + geom_freqpoly(binwidth = 50, colour = "blue")
p + geom_freqpoly(binwidth = 100, colour = "darkgreen",
                  size = 1.2)
```

Код для рис. 4.17

```
# Объект p определен как на рис. 4.16
p + geom_freqpoly(aes(colour = Month), binwidth = 500)
p + geom_freqpoly(aes(y = ..density.., colour = Month),
                  binwidth = 500)
```

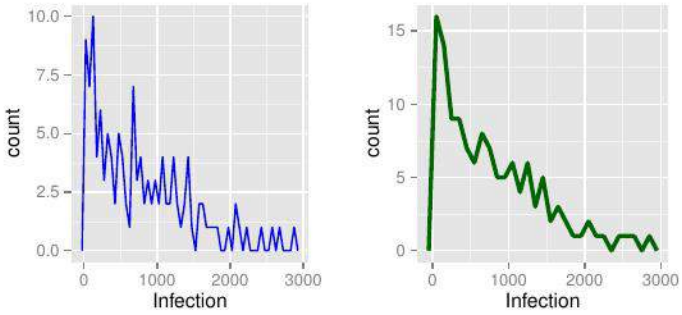


Рисунок 4.16. Подобно гистограммам, у полигонов частот можно изменять размер классового промежутка при помощи аргумента `binwidth` (слева: `binwidth = 50`; справа: `binwidth = 100`). Кроме того, при помощи аргументов `colour` и `size` можно легко изменить цвет и толщину линии, формирующей полигон

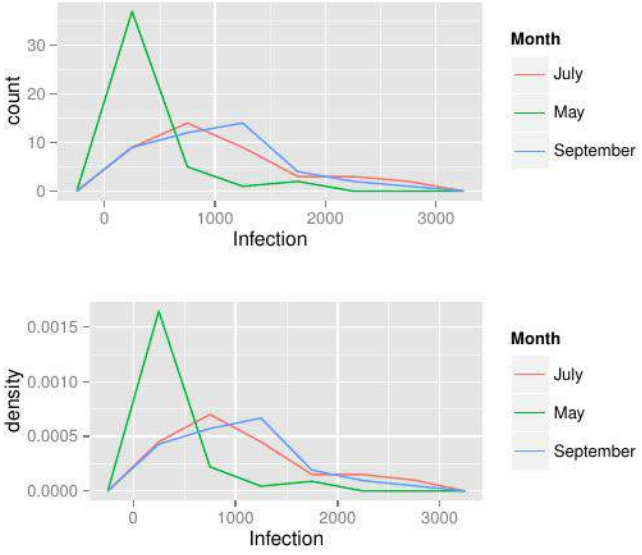


Рисунок 4.17. Полигоны частот для нескольких групп данных. *Вверху:* цвет линии задан в соответствии с уровнями качественной переменной (команда `aes(colour = Month)`). *Внизу:* то же, но по оси Y отложены значения плотности вероятности (команда `aes(y = ..density..)`)

4.2.5 Кривые плотности вероятности: `geom_density()`

Функцией плотности вероятности непрерывной количественной переменной X называют функцию, которая описывает относительную вероятность того, что значение x этой переменной лежит в пределах некоторого

интервала (англ. *probability density function*). Для нахождения вероятности данную функцию необходимо проинтегрировать в пределах соответствующего сегмента кривой плотности. В такой интерпретации понятие плотности вероятности применимо только к непрерывным количественным переменным. В случае с дискретными переменными говорят просто о *функции вероятности* (англ. «*probability mass function*»). Последняя непосредственно задает вероятность того, что дискретная переменная примет определенное значение.

Для добавления к `ggplot`–графику слоя с кривой плотности вероятности служит описанная ниже функция `geom_density()`. Ядерное оценивание значений плотности вероятности по выборочным данным в действительности выполняется другой функцией из пакета `ggplot2` — `stat_density()`, которая незаметно для пользователя вызывает базовую `R`–функцию `density()` (подробнее см. файл помощи, доступный по команде `?density`, а также стр. 27).

Аргументы

- `kernel` — задает алгоритм ядерного оценивания плотности вероятности; возможные значения: `"gaussian"` (принято по умолчанию), `"rectangular"`, `"biweight"`, `"epanechnikov"`, `"triangular"`, `"cosine"` и `"optcosine"`.

- `adjust` — определяет степень сглаживания кривой плотности вероятности (более высокие значения соответствуют большей степени сглаживания — см. также стр. 27).

- `trim` — логический аргумент. При `trim = TRUE` (значение, принятое по умолчанию) рассчитанные вероятности ограничиваются размахом выборочных значений анализируемой переменной. При `trim = FALSE` диапазон оцениваемых вероятностей расширяется на некоторую небольшую величину (подробнее см. справочный файл по функции `density()`).

- `weight` — переменная, по значениям которой происходит «взвешивание» значений переменной X.

- другие аргументы — см. разд. 4.1.

Эстетические атрибуты

- `x`* и `y` — переменные X и Y соответственно.

- `alpha` — степень прозрачности цвета.

- `colour` — цвет линии.

- `fill` — цвет заливки площади под кривой.

- `linetype` — тип линии.

- `size` — толщина линии.

Примеры

Код для рис. 4.18

```
p <- ggplot(data = subset(dreissena, Lake == "Batorino"),
            aes(x = log(Infection + 1)))
p + geom_density(kernel = "rectangular")
p + geom_density(kernel = "epanechnikov")
p + geom_density(kernel = "rectangular", adjust = 1/3)
p + geom_density(kernel = "epanechnikov", adjust = 1/3)
```

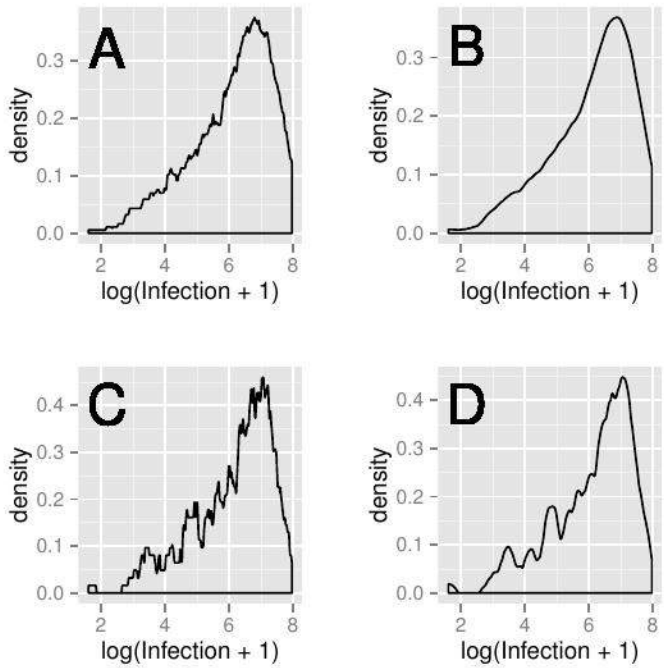


Рисунок 4.18. Примеры кривых плотности вероятности, рассчитанных с использованием разных параметров функции `geom_density()`. *A:* `kernel = "rectangular"`. *B:* `kernel = "epanechnikov"`. *C:* `kernel = "rectangular"` и `adjust = 1/3`. *D:* `kernel = "epanechnikov"` и `adjust = 1/3`

Код для рис. 4.19

```
p <- ggplot(data = subset(dreissena, Lake == "Batorino"),
            aes(x = log(Infection + 1)))
p + geom_density(aes(fill = Month), size = 1.1)
p + geom_density(aes(fill = Month), colour = "blue",
                 linetype = 2, alpha = 0.6)
```

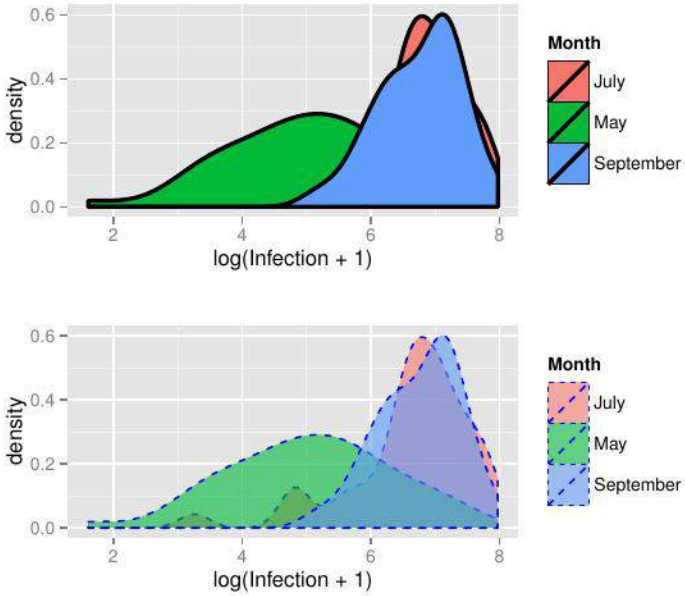


Рисунок 4.19. Настройка внешнего вида графиков плотности вероятности. *Вверху:* кривые плотности вероятности построены отдельно для каждого месяца отбора проб дрейссены. Площадь под каждой кривой залита разным цветом (команда `aes(fill = Month)`), что позволяет идентифицировать отдельные группы данных. Толщина линии задана при помощи аргумента `size = 1.1`. *Внизу:* прозрачность цвета заливки изменена при помощи аргумента `alpha = 0.6`. Цвет и тип линии изменены при помощи аргументов `colour = "blue"` и `linetype = 2` соответственно

Код для рис. 4.20

```
p <- ggplot(data = dreissena, aes(x = log(Infection + 1)))
p + geom_density(aes(group = factor(Day)))
p + geom_density(aes(group = factor(Day), colour = Day))
```

4.2.6 Кумулятивные функции распределения: `geom_step()`

Кумулятивная функция распределения (или просто *«функция распределения»*) описывает вероятность того, что вещественнозначная случайная переменная X примет некоторое значение, не превышающее либо равное x. В пакете `ggplot2` слой с кумулятивной функцией распределения выборочных данных (т. н. *«эмпирической функцией распределения»*) можно добавить к графику при помощи описанной ниже функции `geom_step()`. В простейшем виде функция `geom_step()` создает слой, на котором упорядоченные значения анализируемой переменной образуют ступенчатую

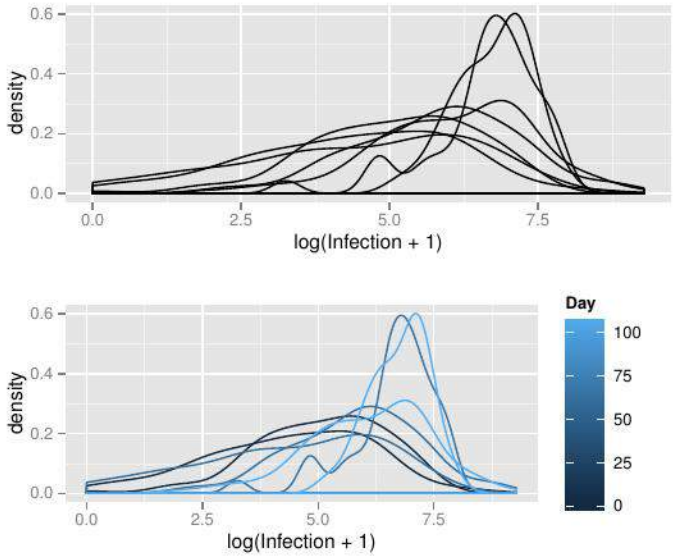


Рисунок 4.20. *Вверху:* пример кривых плотности вероятности для нескольких групп данных (см. команду `aes(group = factor(Day))` в коде для этого рисунка; функция `factor()` использована для преобразования количественной переменной `Day` в фактор). *Внизу:* то же, но цвет линий задан в соответствии со значениями непрерывной количественной переменной `Day` (команда `aes(group = factor(Day), colour = Day)`)

кривую (см. примеры ниже). Однако в сочетании со статистическим преобразованием типа `"ecdf"` (от англ. *empirical cumulative distribution function*) она позволяет также создать кривую собственно кумулятивной функции распределения.

Аргументы

- `direction` — определяет характер чередования линий, соединяющих соседние точки на графике и образующих «ступеньки». При значении `direction = "vh"` вертикальные линии чередуются с горизонтальными. При значении `direction = "hv"` порядок изменяется на обратный (см. рис. 4.22).

- другие аргументы — см. разд. 4.1.

Эстетические атрибуты

- `x*` и `y*` — переменные X и Y соответственно.

- `alpha` — степень прозрачности цвета.

- `colour`, `linetype` и `size` — цвет, тип и толщина линии.

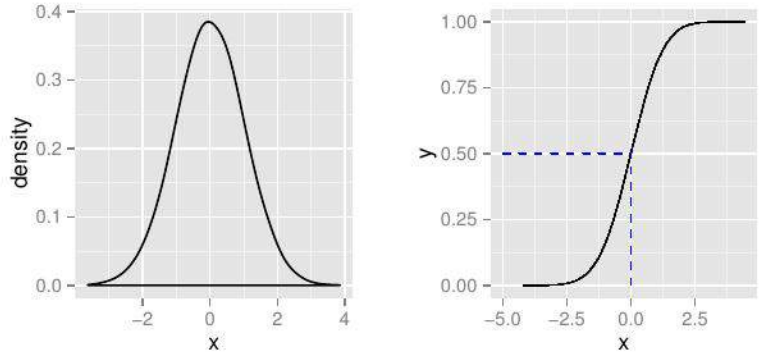


Рисунок 4.21. Иллюстрация понятия «кумулятивная функция распределения». *Слева:* стандартное нормальное распределение со средним значением, равным 0, и стандартным отклонением, равным 1. *Справа:* кумулятивная функция этого распределения. Например, вероятность того, что переменная X примет значение, меньшее либо равное 0, составляет 0.5 (отмечено синими пунктирными линями)

Примеры

Код для рис. 4.22

```
# Для наглядности отсортируем исходную таблицу с данными,
# выбрав только те значение переменной Infection, которые
# лежат в диапазоне от 100 до 200:
dreissena2 <- subset(dreissena,
                     Infection > 100 & Infection < 200)

# Упорядочим значения переменной Infection по возрастанию:
dreissena2 <- dreissena2[order(dreissena2$Infection), ]
p <- ggplot(data = dreissena2,
            aes(x = 1:nrow(dreissena2),
                y = Infection))
p + geom_step(direction = "hv") +
    geom_step(direction = "vh", colour = "blue")
p + geom_step(colour = "red", size = 1.1)
```

Код для рис. 4.23

```
p <- ggplot(data = dreissena2,
            aes(x = Infection))
p + geom_step(stat = "ecdf")
p + geom_step(aes(colour = Lake), stat = "ecdf")
```

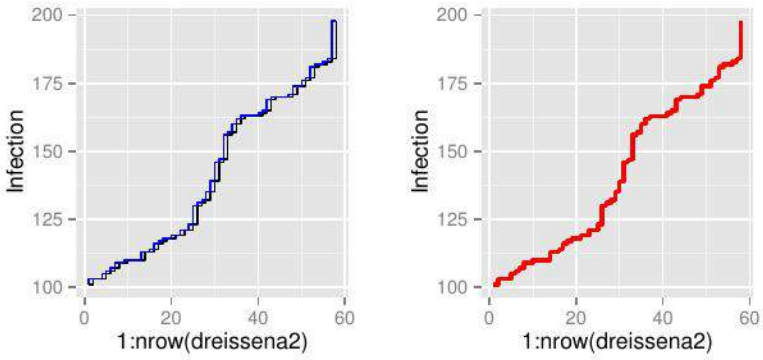


Рисунок 4.22. *Слева:* кривые, соединяющие упорядоченные по возрастанию значения интенсивности инвазии дрейссены инфузорией *C. acuminatus*. Черная кривая построена с использованием аргумента `direction = "hv"` (значение, заданное по умолчанию), а синяя кривая — с использованием аргумента `direction = "vh"` . *Справа:* цвет и толщина линии изменены при помощи аргументов `colour = "red"` и `size = 1.1` соответственно

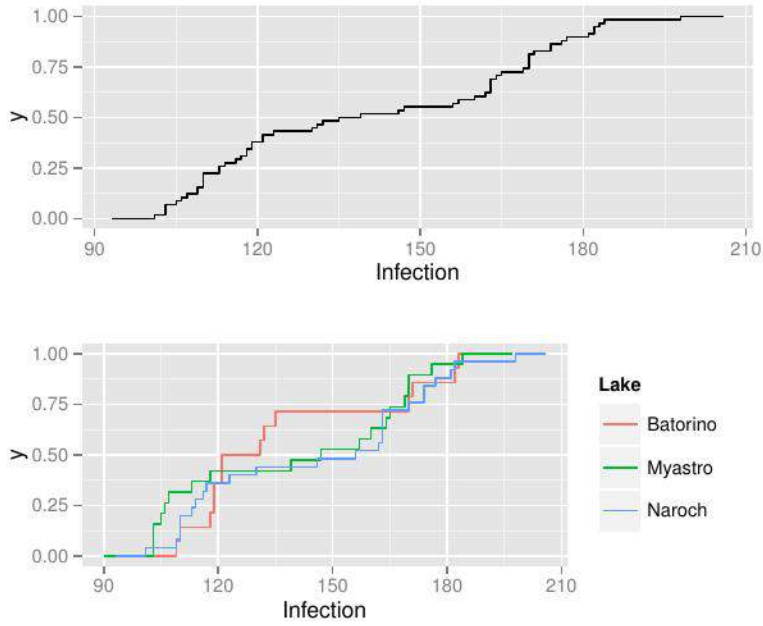


Рисунок 4.23. *Вверху:* эмпирическая кумулятивная функция распределения. *Внизу:* эмпирические функции распределения для трех групп данных

4.2.7 Квантильные графики: stat_qq()

Квантильные графики (также известны как *«графики квантиль–квантиль»*; англ. *«quantile–quantile plots»*, или просто *«q–q plots»*) служат для сравнения двух распределений. Как следует из названия, по осям таких графиков откладывают квантили сравниваемых распределений (чаще всего это квантили распределения выборочных значений и теоретически ожидаемые квантили нормального распределения). Очевидно, что если сравниваемые распределения похожи, то точки на таком графике будут примерно лежать на линии $y = x$. В пакете ggplot2 для создания слоев с квантильными графиками служит описанная ниже функция stat_qq().

Аргументы

- distribution — определяет закон распределения вероятностей, с которым сравнивается эмпирическое распределение. Это может быть любой закон распределения, квантили для которого можно рассчитать с использованием соответствующей базовой функции R. В качестве значения аргумента distribution указывают имя такой функций (оно всегда начинается с буквы "q" — qnorm(), qt(), qexp(), qpois() и т. д.). По умолчанию аргумент distribution принимает значение "qnorm" (т. е. выполняется сравнение с квантилями нормального распределения). Помимо целого ряда законов распределения вероятностей, доступных в базовой версии R[5], многие дополнительные распределения можно найти в специализированных пакетах (например, actuar и VGAM).
- dparams — вектор с параметрами распределения, заданного при помощи аргумента distribution (см. пример ниже).
- geom — геометрический объект, используемый для изображения рассчитываемых функцией stat_qq() значений. По умолчанию используются точки (geom = "point").
- x — задаваемый пользователем вектор значений для отображения по оси X. Необходимость в использовании этого аргумента может возникнуть в случаях, когда пользователь желает отложить по оси X значения, отличные от тех теоретически ожидаемых квантилей, которые по умолчанию рассчитываются функцией stat_qq().
- y — задаваемый пользователем вектор значений для отображения по оси Y.
- другие аргументы — см. разд. 4.1.

Эстетические атрибуты

- sample* — вектор со значениями анализируемой переменной.
- другие атрибуты — определяются значением аргумента geom. Например, если geom = "point", то это будут эстетические атрибуты, свойственные точкам (размер, форма, цвет).

[5] См. статью «Законы распределения вероятностей, реализованные в R»: http://bit.ly/2bLUdy4.

Примеры

Код для рис. 4.24

```
p <- ggplot(data = subset(dreissena,
            Lake == "Batorino" & Month == "May"),
            aes(sample = Infection))
p + stat_qq()
p + stat_qq(aes(colour = Site))
```

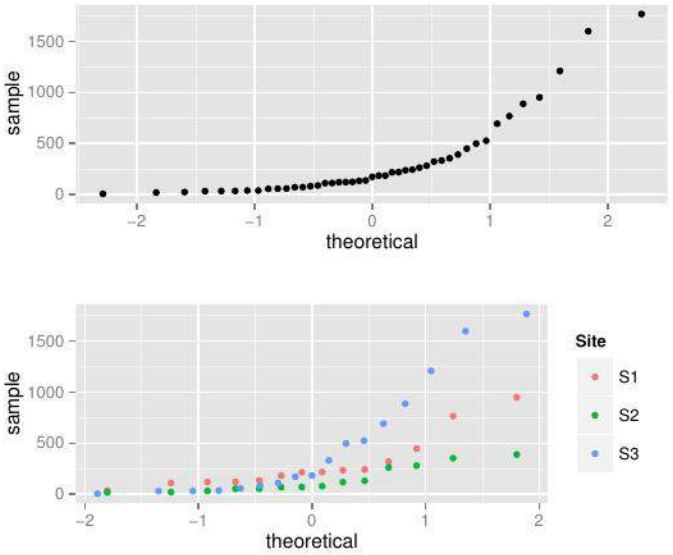


Рисунок 4.24. *Вверху:* квантильный график для данных по интенсивности инвазии дрейссены инфузорией *C. acuminatus* в озере Баторино в мае 2005 г. Хорошо видно значительное отклонение этого эмпирического распределения от ожидаемого нормального распределения. *Внизу:* те же данные, но сгруппированные в соответствии с местом отбора проб

Код для рис. 4.25

```
# Интенсивность инвазии дрейссены инфузорией C. acuminatus -
# дискретная количественная переменная. Часто для описания
# распределения таких переменных подходят распределение
# Пуассона или негативное биномиальное распределение.
# Проверим, какой из этих двух законов распределения
# вероятностей подходит лучше для данных по озеру Баторино,
# полученных в мае 2005 г.

# Для удобства сохраним данные в виде отдельной таблицы:
dreissena2 <- subset(dreissena,
                     Lake == "Batorino" & Month == "May")
```

```
# Оценим параметры теоретических распределений по
# выборочным данным (использована функция fitdistr() из
# базового R-пакета MASS):
library(MASS)
params1 <- as.list(fitdistr(dreissena2$Infection,
                  "Negative Binomial")$estimate)
params2 <- as.list(fitdistr(dreissena2$Infection,
                            "Poisson")$estimate)

# Построим квантильные графики:
ggplot(dreissena2, aes(sample = Infection)) +
       stat_qq(distribution = qpois, dparams = params2)
ggplot(dreissena2, aes(sample = Infection)) +
       stat_qq(distribution = qnbinom, dparams = params1)
```

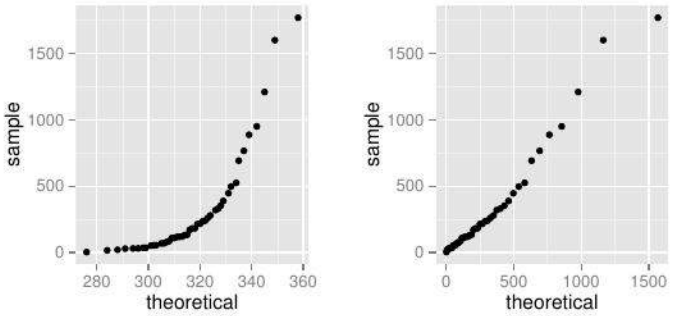


Рисунок 4.25. Квантильные графики для данных по интенсивности инвазии дрейссены инфузорией *C. acuminatus* в озере Баторино в мае 2005 г. *Слева:* по оси X отложены квантили распределения Пуассона. *Справа:* по оси X отложены квантили негативного биномиального распределения. Видно, что второе распределение гораздо лучше подходит для описания этой выборки

4.3 Визуализация 2D– и 3D–распределений

В этом разделе мы продолжим рассмотрение способов визуализации распределений количественных переменных и перейдем к случаям двух– и трехмерных распределений. Следует отметить, что в пакете `ggplot2` 3D–графики (т. е. графики с тремя координатными осями) как таковые не реализованы. Тем не менее имеется возможность строить графики с изолиниями, отражающими изменение той или иной количественной переменной в зависимости от двух других количественных переменных (например, среднегодовая температура в зависимости от географических координат на картах).

4.3.1 Контуры плотности вероятности: geom_density2d()

В подразд. 2.3.4 и 4.2.5 были описаны способы построения графиков одномерных функций плотности вероятности. Однако идею оценки плотности вероятности можно применить и к ситуации, когда каждое выборочное наблюдение описывается по двум количественным переменным. Плотность двухмерного распределения можно отобразить с помощью контурных линий, ограничивающих области с одинаковой плотностью вероятности. В пакете `ggplot2` для построения таких графиков служит рассмотренная ниже функция `geom_density2d()`. Вычисления, необходимые для нахождения границ контуров плотности, выполняются функцией `stat_density2d()`, которая, в свою очередь, основана на функции `kde2d()` из базового `R`–пакета `MASS`. Запуск работы функции `stat_density2d()` происходит незаметно для пользователя при вызове `geom_density2d()`.

Аргументы

- `contour` — логический аргумент. Значение `contour = TRUE` (задано по умолчанию) включает отображение контуров плотности вероятности на графике.
- `n` — число, определяющее гладкость контурных линий.
- другие аргументы — см. разд. 4.1.

Эстетические атрибуты

- `x`* и `y`* — переменные X и Y соответственно.
- `alpha` — степень прозрачности цвета.
- `colour` — цвет линии.
- `linetype` — тип линии.
- `size` — толщина линии.

Примеры

Код для рис. 4.26

```
p <- ggplot(data = subset(dreissena, Lake == "Naroch"),
            aes(x = Length, y = log(Infection + 1))) +
    geom_point()
p + stat_density2d(); p + stat_density2d(n = 10)
```

Код для рис. 4.27

```
p <- ggplot(data = subset(dreissena,
                          Lake == "Naroch" & Month == "May"),
            aes(x = Length, y = log(Infection + 1))) +
            geom_point(aes(colour = Site))
```

```
p + stat_density2d(aes(colour = Site))
p + stat_density2d(aes(colour = Site), alpha = 0.4)
```

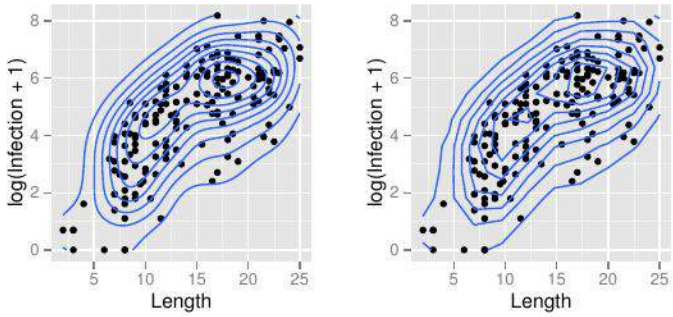


Рисунок 4.26. Контуры плотности вероятности двухмерного распределения. *Слева:* график, построенный с использованием автоматических настроек. *Справа:* степень сглаживания контурных линий изменена при помощи аргумента `n = 10`

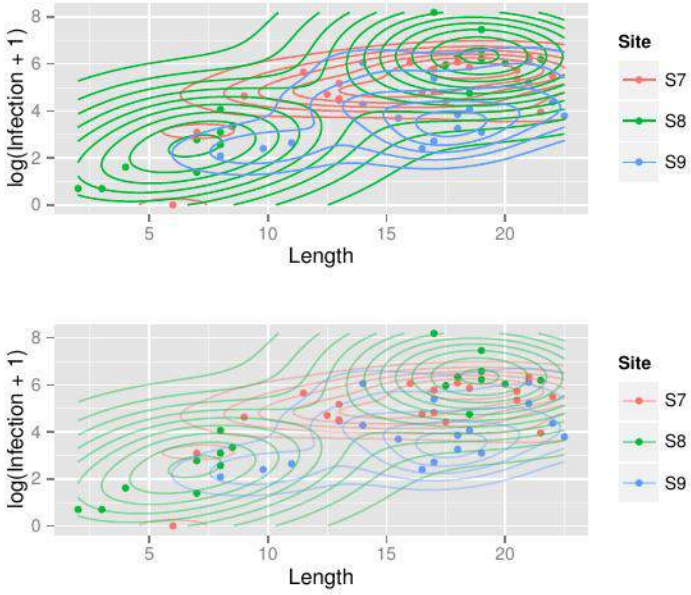


Рисунок 4.27. Контуры плотности вероятности распределения для трех групп данных. *Вверху:* график, построенный с использованием автоматических настроек. *Внизу:* прозрачность цвета контурных линий изменена при помощи аргумента `alpha = 0.4`

4.3.2 Изолинии: geom_contour()

Изолиния, или *контурная линия*, функции двух переменных представляет собой линию, вдоль которой значение этой функции постоянно. Графики с использованием изолиний можно встретить во многих научных областях — в картографии, метеорологии, океанографии, экологии и т. д. В зависимости от контекста изолинии могут иметь разные названия (изобаты, изобары, изотермы, изоклины и т. п.). В пакете `ggplot2` для построения графиков с изолиниями служит описанная ниже функция `geom_contour()`. Вычисление координат изолиний выполняется функцией `stat_contour()`, которая активируется незаметно для пользователя при вызове `geom_contour()`.

Аргументы

- `bins` — задает размер классового промежутка, используемого для объединения нескольких соседних изолиний в один класс.
- другие аргументы — см. разд. 4.1.

Эстетические атрибуты

- `x`* и `y`* — переменные X и Y соответственно.
- `z`* — переменная, являющаяся функцией от `x` и `y`.
- `alpha` — степень прозрачности цвета.
- `colour` — цвет линии.
- `linetype` — тип линии.
- `size` — толщина линии.

Примеры

В приведенных ниже примерах использованы данные по топологии вулкана Maunga Whau (Новая Зеландия), входящие в состав базовой версии `R` в виде матрицы под названием `volcano`. Строки этой матрицы соответствуют географическим координатам в направлении с востока на запад, а столбцы — координатам в направлении с юга на север[6].

Код для рис. 4.28

```
# матрица volcano преобразована в таблицу данных:
volcano3d <- reshape2::melt(volcano)
names(volcano3d) <- c("x", "y", "z")

# Построение графиков:
p <- ggplot(data = volcano3d, aes(x = x, y = y, z = z))
p + geom_contour(); p + geom_contour(bins = 5)
```

[6] Подробнее см. файл помощи, доступный по команде `?volcano`.

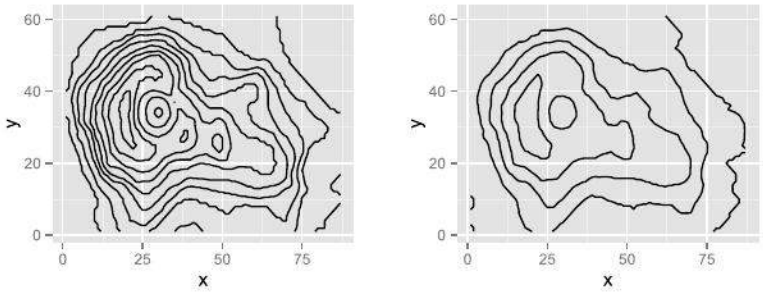


Рисунок 4.28. Контурная карта вулкана Maunga Whau. *Слева:* график построен с использованием автоматических настроек. *Справа:* несколько соседних изолиний объединены в более крупные классы при помощи аргумента `bins = 5`

Код для рис. 4.29

```
p + geom_contour(bins = 50, colour = "rosybrown")
p + geom_contour(bins = 50, size = 0.5,
                 colour = "rosybrown", alpha = 0.5) +
    geom_contour(bins = 5, size = 1)
```

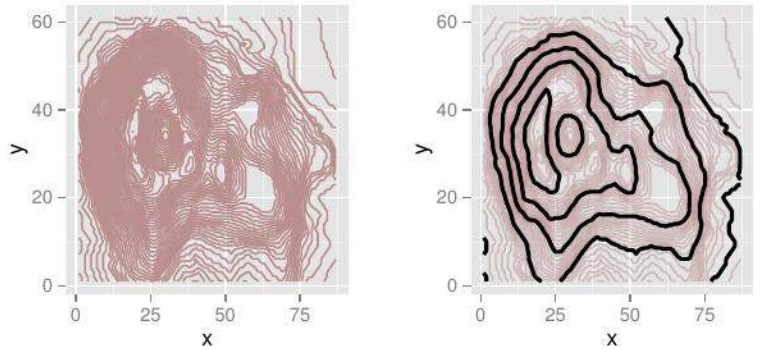


Рисунок 4.29. *Слева:* частота и цвет контурных линий изменены при помощи аргументов `bins = 50` и `colour = "rosybrown"` соответственно. *Справа:* две контурные карты с разными настройками наложены друг на друга

Код для рис. 4.30

```
p + geom_contour(aes(colour = ..level..))
p + geom_tile(aes(fill = z)) + geom_contour()
```

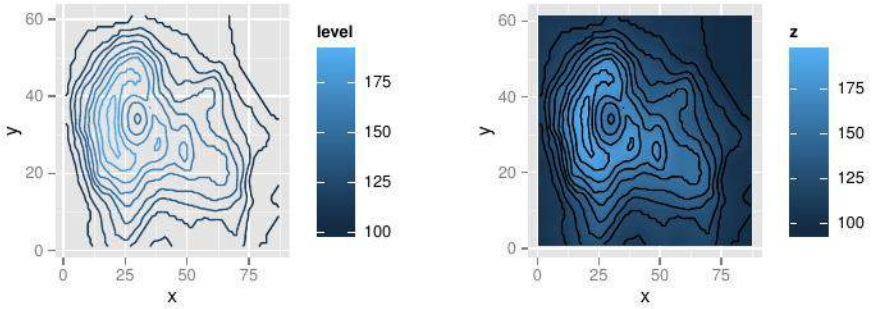


Рисунок 4.30. *Слева:* цвет контурных линий задан в соответствии с градиентом значений переменной `z` (см. команду `aes(colour = ..level..)`, где `level` — переменная, создаваемая в ходе вычислений, которые выполняет функция `stat_contour()`). *Справа:* контурная карта совмещена с графиком, созданным при помощи функции `geom_tile()` (см. подразд. 4.7)

4.3.3 Сотовые диаграммы: `geom_hex()`

При работе с данными большого объема визуализация зависимости между двумя переменными на диаграмме рассеяния неизбежно сопровождается «наползанием» точек друг на друга, что затрудняет выявления характера этой зависимости. Одним из возможных способов решения этой проблемы является использование «*сотовых диаграмм*» (англ. *hexagon plots*). На такой диаграмме координатная плоскость разбивается на гексагоны, которые закрашиваются цветом в соответствии с градиентом плотности попавших в них точек (в англоязычной литературе этот процесс имеет название «*hexagon binning*»). При этом гексагоны с нулевым количеством попавших в них точек на графике не изображаются. Таким образом, сотовые диаграммы представляют собой вариант двухмерной гистограммы. В пакете `ggplot2` для построения сотовых диаграмм служит рассмотренная ниже функция `geom_hex()`. Вычисления, связанные с разбиением координатной плоскости на гексагоны и нахождением плотности точек в них, выполняются функцией `stat_binhex()`, которая активируется незаметно для пользователя при вызове `geom_hex()`[7].

Аргументы

- `bins` — определяет размер гексагонов. Это может быть скалярный вектор или вектор из двух чисел. В последнем случае контролируется протяженность гексагонов вдоль координатных осей (см. примеры ниже).

- другие аргументы — см. разд. 4.1.

[7] При выполнении вычислений функция `stat_binhex()` обращается к функциям из пакета `hexbin`. Соответственно, этот пакет должен быть предварительно установлен на вашем компьютере.

Эстетические атрибуты

- x* и y* — переменные X и Y соответственно.
- `alpha` — степень прозрачности цвета заливки гексагонов.
- `colour` — цвет линии, окаймляющей гексагон.
- `size` — толщина линии, окаймляющей гексагон.

Примеры

Код для рис. 4.31

```
library(hexbin)
p <- ggplot(data = dreissena,
            aes(x = Length, y = log(Infection + 1)))
p + geom_point()
p + geom_hex()
p + geom_hex(bins = 10)
```

Код для рис. 4.32

```
# Объект p определен как на рис. 4.31
p + geom_hex(bins = c(2, 30), alpha = 0.6)
p + geom_hex(bins = 10, colour = "red")
```

4.4 Визуализация сводной статистической информации о количественных переменных

В этом разделе обсуждаются способы визуализации сводной информации о количественных переменных, значения которых разбиты на группы в соответствии со значениями качественных переменных (факторов). Под «сводной информацией» имеются в виду меры центральной тенденции (арифметическое среднее, мода, медиана), а также показатели вариабельности данных (минимальное и максимальное значения, стандартное отклонение, квартили). Особым типом рассмотренных здесь способов визуализации являются «скрипичные диаграммы» — смесь диаграмм размахов с кривыми плотности вероятности.

4.4.1 Диаграммы диапазонов: `geom_linerange()`, `geom_pointrange()`, `geom_errorbar()`, `geom_crossbar()`

Диаграммы диапазонов служат для визуализации интервалов значений анализируемых переменных (минимум и максимум, нижний и верхний

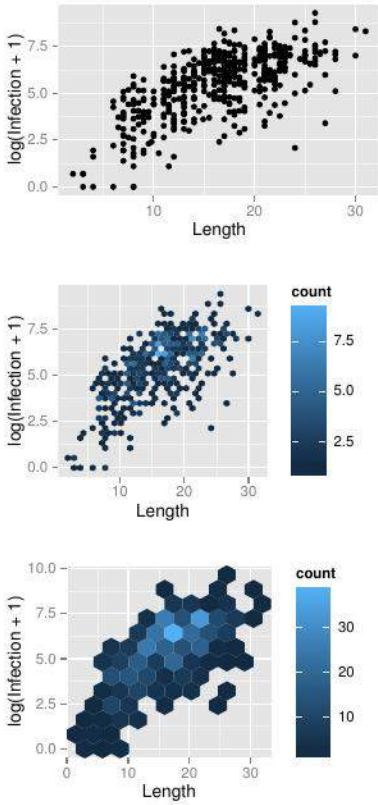


Рисунок 4.31. Примеры сотовых диаграмм. *Вверху:* исходные данные. *В центре:* график построен в соответствии с автоматическими настройками. *Внизу:* размер гексагонов изменен с помощью аргумента `bins = 10`

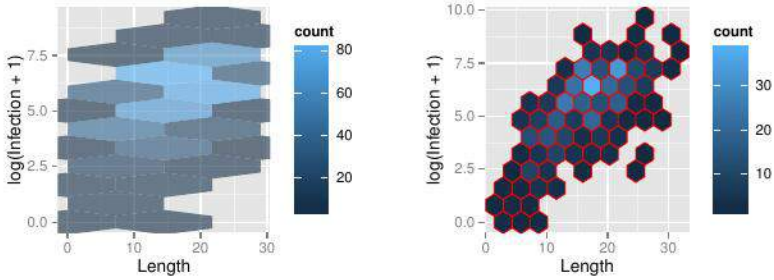


Рисунок 4.32. Настройка внешнего вида сотовых диаграмм. *Слева:* размер гексагонов задан при помощи аргумента `bins = c(2, 30)`. Прозрачность цвета заливки гексагонов изменена при помощи аргумента `alpha = 0.6`. *Справа:* цвет линий, окаймляющих гексагоны, изменен на красный с помощью аргумента `colour = "red"`

квартили и т. п.) и определенных статистических показателей (стандартные отклонения, стандартные ошибки, доверительные интервалы и т. п.). Способы изображения этих интервалов варьируют, в связи с чем в пакете `ggplot2` имеется несколько соответствующих функций:

- `geom_linerange()` — создает слой, на котором интервалы значений количественных переменных представлены в виде вертикальных или горизонтальных линий;

- `geom_pointrange()` — создает слой, на котором интервалы значений количественных переменных представлены в виде вертикальных или горизонтальных линий с точкой посередине (например, *среднее значение* ± *стандартное отклонение*);

- `geom_errorbar()` — создает слой с вертикальными отрезками, обозначающими стандартные ошибки, доверительные интервалы и другие подобные им статистические показатели. Эти отрезки по обоим концам ограничены более короткими перпендикулярными отрезками (см. пример на рис. 4.35 и 4.36). Обычно слой, созданный при помощи `geom_errorbar()`, добавляют к уже существующему `ggplot`–графику с точками, символизирующими, например, средние значения какой–либо переменной. Для создания слоя с аналогичными горизонтальными отрезками служит функция `geom_errorbarh()`;

- `geom_crossbar()` — создает слой, на котором небольшие горизонтальные линии, соответствующие какой–либо мере центральной тенденции, окаймлены прямоугольниками, длина которых соответствуют какой–либо мере разброса данных (см. пример на рис. 4.37).

В силу того, что перечисленные функции приводят к построению концептуально сходных графиков, все они обладают рядом одинаковых аргументов, которые были описаны ранее в разд. 4.1. Кроме того, эти функции имеют несколько одинаковых аргументов, задающих эстетические атрибуты:

- `x`* — переменная X (функция `geom_crossbar()` требует также обязательного указания Y–координат для линий, изображающих выбранную пользователем меру центральной тенденции);

- `ymax`* — вектор с Y–координатами, соответствующими верхним границам интервалов;

- `ymin`* — вектор с Y–координатами, соответствующими нижним границам интервалов;

- `alpha` — степень прозрачности цвета;

- `colour` — цвет линии;

- `linetype` — тип линии;

- `size` — толщина линии.

В то же время некоторые из перечисленных выше функций имеют и свои специфичные аргументы:

- функция `geom_pointrange()`: аргумент `shape` позволяет задать форму символа, изображаемого посередине отрезка, а аргумент `fill` — его цвет;
- функция `geom_errorbar()`: аргумент `width` служит для указания длины перпендикулярных отрезков, ограничивающих интервалы ошибок (см. примеры на рис. 4.35). В случае с функцией `geom_errorbarh()` вместо `width` используется аргумент `height`;
- функция `geom_errorbarh()`: аргументы `xmin` и `xmax` служат для указания векторов, содержащих значения нижних и верхних границ интервалов ошибок. Аргумент `height` используется для настройки длины перпендикулярных отрезков, ограничивающих интервалы (см. примеры на рис. 4.36);
- функция `geom_crossbar()`: аргумент `fatten` задает коэффициент пропорциональности для толщины линии, которая изображает меру центральной тенденции (например, медиану). Аргумент `fill` определяет цвет заливки прямоугольника, изображающего интервал анализируемых значений (см. примеры на рис. 4.37).

Примеры

Код для рис. 4.33

```
library(doBy)
# Таблица с минимальными значениями:
MIN <- summaryBy(Length ~ Month, FUN = min,
                 data = subset(dreissena, Lake == "Myastro"))
# Таблица с максимальными значениями:
MAX <- summaryBy(Length ~ Month, FUN = max,
                 data = subset(dreissena, Lake == "Myastro"))
# Таблица со средними значениями:
MEAN <- summaryBy(Length ~ Month, FUN = mean,
                  data = subset(dreissena, Lake == "Myastro"))
# Таблица со стандартными ошибками:
SE <- summaryBy(Length ~ Month,
                FUN = function(x) sd(x)/sqrt(length(x)),
                data = subset(dreissena, Lake == "Myastro"))
# Сводная таблица с параметрами описательной статистики:
dat <- data.frame(MIN[1], MIN[2], MAX[2], MEAN[2], SE[2])
names(dat) <- c("Month", "Min", "Max", "Mean", "SE")
# Построение графиков:
p <- ggplot(data=dat, aes(x = Month, ymin = Min, ymax = Max))
p + geom_linerange() + ylab("Length")
p + geom_pointrange(aes(y = Mean))
```

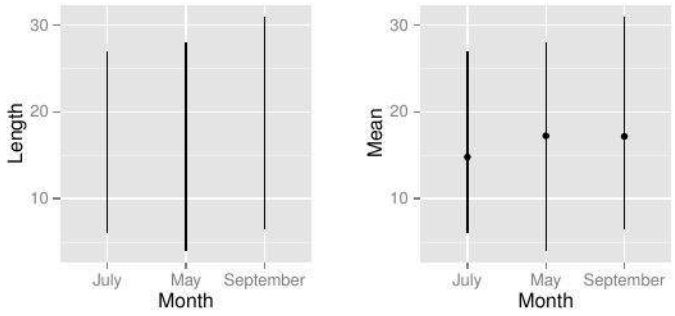


Рисунок 4.33. *Слева:* сезонная динамика размаха значений длины раковины дрейссены в озере Мястро (нижняя граница отрезка — минимум, верхняя граница — максимум). *Справа:* то же, но с точками, обозначающими средние значения длины раковины

Код для рис. 4.34

```
# объект p определен как на рис. 4.33
p + geom_pointrange(aes(y = Mean), shape = 2, linetype = 2)
p + geom_pointrange(aes(y = Mean, colour = Month))
```

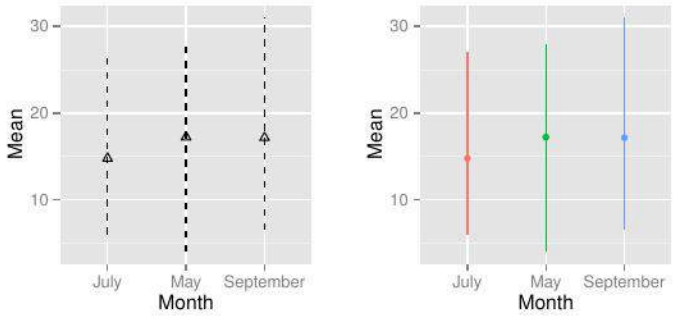


Рисунок 4.34. Те же данные, что и на рис. 4.33. *Слева:* тип линий изменен при помощи аргумента `linetype = 2`, а форма точек — при помощи аргумента `shape = 2`. *Справа:* цвет линий и точек соответствует месяцу отбора проб (аргумент `colour = Month`)

Код для рис. 4.35

```
p <- ggplot(data = dat, aes(x = Month,
           ymin = Mean - SE, ymax = Mean +  SE))
p + geom_point(aes(y = Mean)) + geom_errorbar()
p + geom_point(aes(y = Mean)) + geom_errorbar(width = 0.25)
```

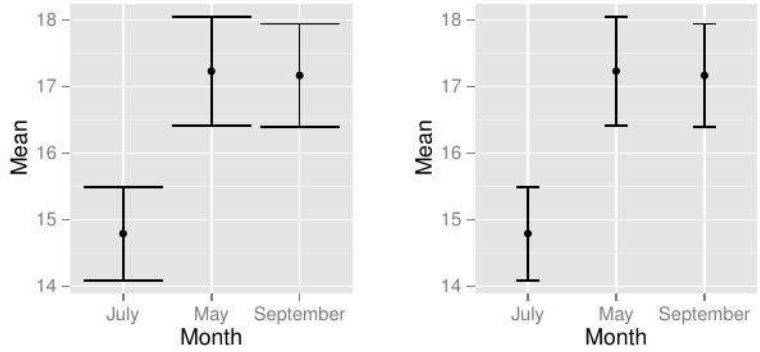


Рисунок 4.35. Средняя длина раковины дрейссены в озере Мястро (± стандартная ошибка). *Слева:* интервалы ошибок изображены в соответствии с автоматическими настройками. *Справа:* длина перпендикулярных отрезков, ограничивающих интервалы ошибок, уменьшена при помощи аргумента `width = 0.25`)

Код для рис. 4.36

```
# объект p определен как на рис. 4.35
p + geom_bar(aes(y = Mean, fill = Month),
    stat = "identity") + geom_errorbar(width = 0.25)
q <- ggplot(data = dat, aes(x = Mean, y = Month))
q + geom_point() + geom_errorbarh(aes(xmax = Mean + SE,
    xmin = Mean - SE), height = 0.25)
```

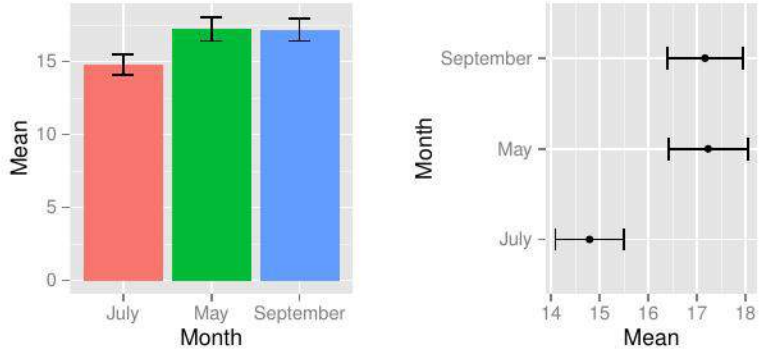


Рисунок 4.36. Те же данные, что и на рис. 4.35. *Слева:* средние значения длины раковины дрейссены представлены столбиками. *Справа:* интервалы ошибок изображены горизонтально (функция `geom_errorbarh()`)

Код для рис. 4.37

```
# объект p определен как на рис. 4.35
p + geom_crossbar(aes(y = Mean))
p + geom_crossbar(aes(y = Mean, fill = Month), fatten = 5)
```

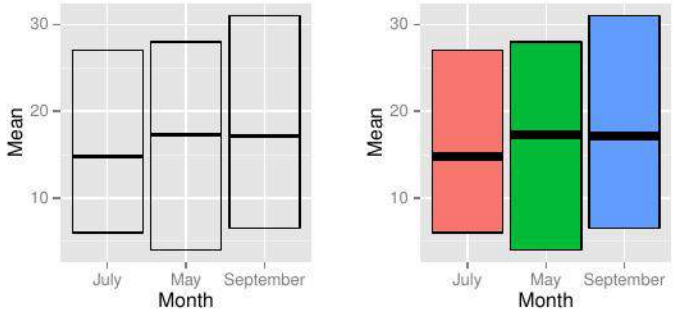


Рисунок 4.37. График, построенный при помощи функции `geom_crossbar()`. Представлены те же данные, что и на рис. 4.33. *Слева:* использованы автоматические настройки функции. *Справа:* цвет заливки прямоугольников соответствует месяцу отбора проб (см. аргумент `fill = Month`). Толщина линий, изображающих средние значения, увеличена при помощи аргумента `fatten = 5`

4.4.2 Диаграммы размахов: `geom_boxplot()`

Как было отмечено в подразд. 2.3.3, диаграммы размахов позволяют очень полно представить на одном графике сводную статистическую информацию одновременно для нескольких групп данных, выделенных в соответствии с уровнями той или иной качественной переменной (или переменных). Вспомним, в частности, что для каждой группы данных изображаются медиана, нижний и верхний квартили, интервал («усы»), в который попадает подавляющее большинство наблюдений, а также наблюдения–выбросы, оказавшиеся за пределами этого интервала. По бокам «ящиков» могут также присутствовать *«насечки»* (англ. *notches*) шириной $1.5 \times \mathrm{IQR}/\sqrt{n}$. Ширина насечек примерно соответствует 95%–ным доверительным интервалам медиан, что позволят выполнять быструю визуальную оценку различий между группами данных (если эти интервалы перекрываются, то делается заключение об отсутствии различий между генеральными медианами соответствующих групп)[8]. Добавление слоев с диаграммами размахов выполняется при помощи описанной ниже функции `geom_boxplot()`. При этом вычисления над исходными данными, необходимые для создания таких слоев, выполняются незаметно для пользователя функцией `stat_boxplot()`.

[8] Подробнее см.: McGill R., Tukey J. W., Larsen W. A. (1978) Variations of box plots. The American Statistician 32: 12–16.

Аргументы

- `notch` — логический аргумент, включающий отображение «насечек» (по умолчанию `notch = FALSE`).
- `notchwidth` — задает глубину насечки относительно ширины «ящика» (по умолчанию `notch = 0.5`).
- другие аргументы — см. разд. 4.1.

Эстетические атрибуты

- `lower` и `upper` — нижняя и верхняя Y–координаты прямоугольников. По умолчанию это квартили, однако при помощи аргументов `lower` и `upper` можно задать и другие процентили (см. пример на рис. 4.41).
- `middle` — Y–координаты линий, изображающих меру центральной тенденции в каждой группе данных (по умолчанию это медиана).
- `x*` — переменная X.
- `ymin` и `ymax` — нижняя и верхняя Y–координаты отрезков, отходящих от прямоугольников.
- `outlier.color` — цвет точек, обозначающих выбросы.
- `outlier.shape` — форма точек, обозначающих выбросы.
- `outlier.size` — размер точек, обозначающих выбросы.
- `alpha` — степень прозрачности цвета.
- `colour` — цвет линий.
- `linetype` — тип линий.

Примеры

Код для рис. 4.38

```
p <- ggplot(data = dreissena,
            aes(x = Lake, y = log(Infection+1)))
p + geom_boxplot()
p + geom_boxplot(fill = "coral", color = "blue")
```

Код для рис. 4.39

```
# здесь и до рис. 4.42 объект p определен как на рис. 4.38
p + geom_boxplot(size = 1.2)
p + geom_boxplot(outlier.shape = 2,
                 outlier.colour = "blue")
```

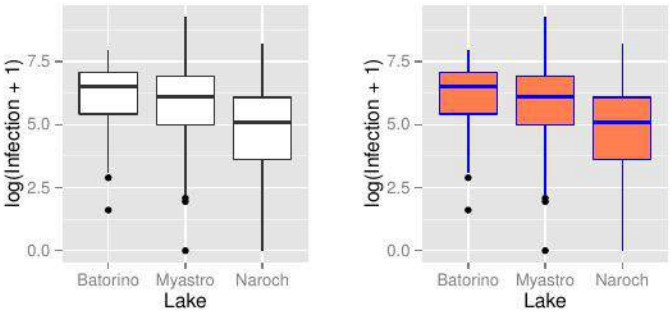


Рисунок 4.38. *Слева:* диаграмма размахов, созданная с использованием автоматических настроек функции `geom_boxplot()`. *Справа:* цвет заливки «ящиков» изменен при помощи аргумента `fill = "coral"`, а цвет линий — при помощи аргумента `colour = "blue"`

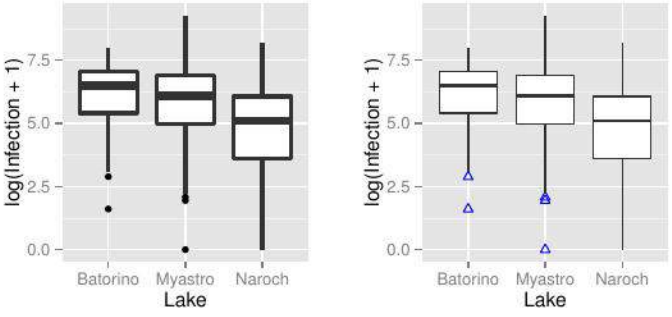


Рисунок 4.39. *Слева:* толщина линий изменена при помощи аргумента `size = 1.2`. *Справа:* форма и цвет точек, изображающих потенциальные выбросы, изменены при помощи аргументов `outlier.shape = 2` и `outlier.color = "blue"` соответственно

Код для рис. 4.40

```
p + geom_boxplot(notch = TRUE)
p + geom_boxplot(notch = TRUE, notchwidth = 0.2)
```

Код для рис. 4.41

```
# График слева:
p + geom_boxplot()
# График справа:
require(doBy)
quant10 <- function(x) quantile(x, p = 0.10)
quant90 <- function(x) quantile(x, p = 0.90)
stats <- summaryBy(data = dreissena,
```

```
                    log(Infection + 1) ~ Lake,
                    FUN = c(min, max, mean, quant10, quant90))
names(stats) <- c("Lake", "Min", "Max", "Mean", "Q10", "Q90")
q <- ggplot(stats)
q + geom_boxplot(aes(x = Lake, ymin = Min, ymax = Max,
                     lower = Q10, middle = Mean,
                     upper = Q90), stat = "identity") +
                     ylab("log(Infection+1)")
```

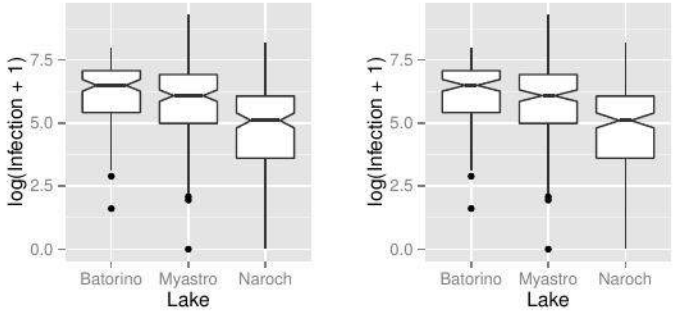


Рисунок 4.40. Диаграммы размахов с «насечками». *Слева:* график создан с использованием автоматических настроек. *Справа:* глубина насечек изменена при помощи аргумента `notchwidth = 0.2`

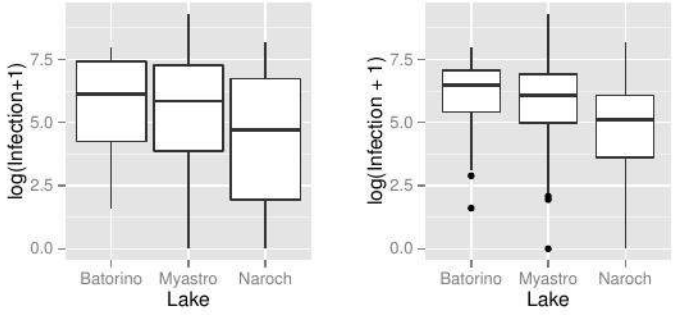


Рисунок 4.41. *Слева*: Нижняя и верхняя границы «ящиков» соответствуют 0.1– и 0.9–процентилям (вместо принятых по умолчанию 0.25– и 0.75–процентилей). Горизонтальные линии внутри «ящиков» отражают средние значения (вместо медиан). «Усы» ограничивают интервалы между минимальным и максимальным значениями в каждой группе. *Справа*: для сравнения приведена диаграмма из рис. 4.38, которая была создана с использованием автоматических настроек

Код для рис. 4.42

```
p + geom_boxplot(aes(fill = Month))
```

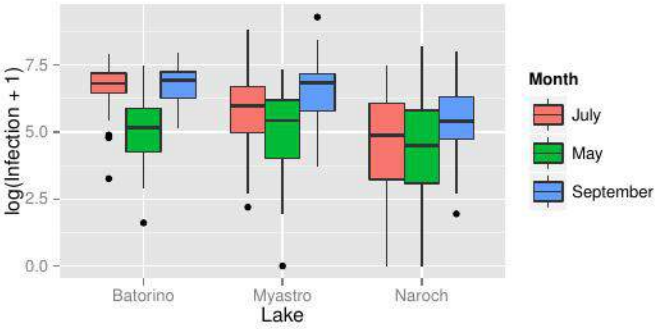


Рисунок 4.42. Диаграмма размахов, на которой значения количественной переменной (уровень инвазии) разбиты на группы в соответствии с уровнями двух факторов (озеро и месяц отбора проб)

4.4.3 Скрипичные диаграммы: `geom_violin()`

Рассмотренные в предыдущем разделе диаграммы размахов позволяют представить информацию о нескольких статистических параметрах анализируемых групп данных. Однако мы можем сделать график еще более информативным при помощи «*скрипичных диаграмм*» (англ. «*violin plots*»)[9], которые объединяют в себе идеи диаграмм размахов и кривых плотности вероятности (см. стр. 27 и разд. 4.2.5). Суть достаточно проста: продольные края «ящиков» на диаграмме размахов замещаются кривыми плотности вероятности. В итоге получаются симметричные фигуры, чьи очертания напоминают очертания скрипки — отсюда название этого типа диаграмм. В пакете `ggplot2` для создания скрипичных диаграмм служит рассмотренная ниже функция `geom_violin()`.

Аргументы

Аргументы функции `geom_violin()` аналогичны аргументам `geom_density()` (см. подразд. 4.2.5). Кроме того, имеется специфичный аргумент `scale`, который определяет форму «скрипок». При `scale = "area"` (значение по умолчанию) наблюдения в каждой группе нормализуются таким образом, чтобы все «скрипки» имели одинаковую площадь. При `scale = "count"` площади «скрипок» пропорциональны объемам наблюдений в соответствующих группах. Наконец, при `scale = "width"` максимальная ширина у всех «скрипок» будет одинакова (рис. 4.43).

[9] Скрипичные диаграммы были впервые предложены в работе: Hintze J. L., Nelson R. D. (1998) Violin plots: a box plot–density trace synergism. The American Statistician 52(2): 181–184. PDF–версия этой статьи доступна по адресу `http://bit.ly/2cgBdlz`.

Эстетические атрибуты

- `alpha` — степень прозрачности цвета.
- `fill` — цвет заливки «скрипок».
- `colour` — цвет линий.
- `linetype` — тип линий.
- `size` — толщина линий.

Примеры

Код для рис. 4.43

```
p <- ggplot(data = dreissena,
            aes(x = Lake, y = log(Infection+1)))
p + geom_violin(scale = "area")
p + geom_violin(scale = "count")
p + geom_violin(scale = "width")
```

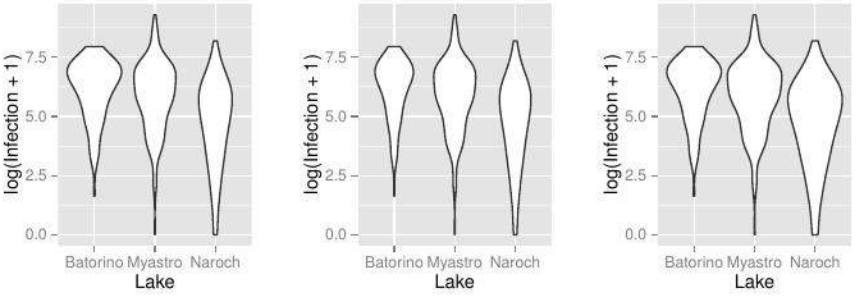


Рисунок 4.43. Примеры скрипичных диаграмм. *Слева:* скрипичная диаграмма, созданная с использованием автоматических настроек функции `geom_violin()` (площадь всех фигур на диаграмме одинакова). *В центре:* площадь «скрипок» пропорциональна объемам наблюдений в соответствующих группах (`scale = "count"`). *Справа:* максимальная ширина у всех «скрипок» одинакова (`scale = "width"`)

Код для рис. 4.44

```
# здесь и до рис. 4.48 объект p определен как на рис. 4.43
p + geom_violin(trim = TRUE)
p + geom_violin(adjust = 0.4, trim = FALSE)
p + geom_violin(adjust = 0.4, trim = FALSE) +
    coord_flip()
```

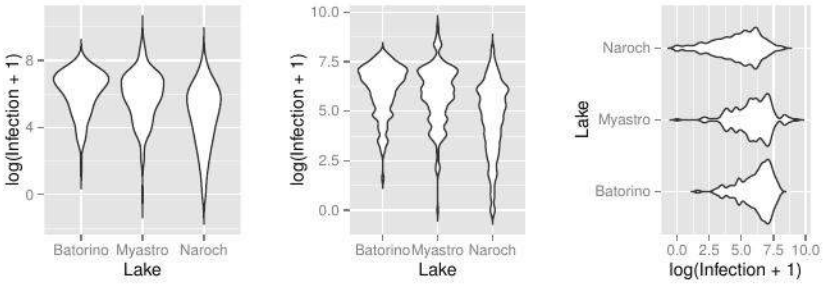


Рисунок 4.44. *Слева:* концы «скрипок» не ограничены минимальным и максимальным значениями в соответствующих группах (аргумент `trim = FALSE`). *В центре:* то же, что и слева, но гладкость кривых плотности вероятности уменьшена при помощи аргумента `adjust = 0.4`. *Справа:* то же, что и в центре, но оси повернуты при помощи функции `coord_flip()`

Код для рис. 4.45

```
p + geom_violin(trim = FALSE, fill = "red")
p + geom_violin(trim = FALSE, fill = "red", colour = "blue")
p + geom_violin(trim = FALSE, fill = "red", alpha = 0.5,
                colour = "blue", size = 1.5)
```

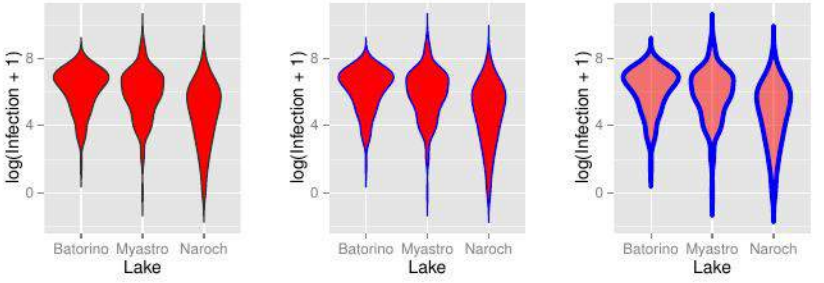


Рисунок 4.45. *Слева:* «скрипки» залиты красным цветом (аргумент `colour = "red"`). *В центре:* то же, что и слева, но цвет окаймляющих линий изменен на синий (`colour = "blue"`). *Справа:* то же, что и в центре, но цвет заливки «скрипок» сделан полупрозрачным (`alpha = 0.5`) и увеличена толщина линий (`size = 1.5`)

Код для рис. 4.46

```
p + geom_violin(trim = FALSE) +
    geom_point(shape = 19, alpha = 0.2)
p + geom_violin(trim = FALSE) +
    geom_jitter(shape = 19, alpha = 0.2)
```

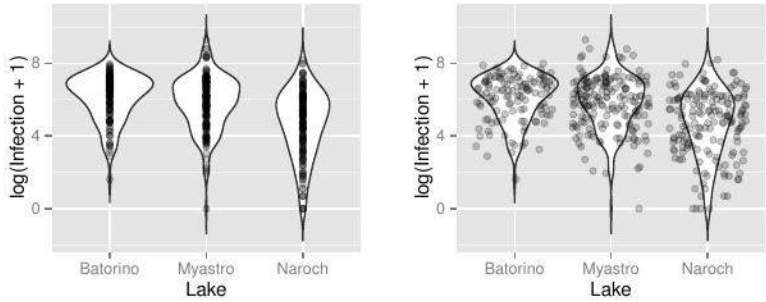


Рисунок 4.46. Скрипичные диаграммы в сочетании с точками, изображающими исходные наблюдения. *Слева:* слой с точками добавлен при помощи функции `geom_point()`). *Справа:* слой с точками добавлен при помощи функции `geom_jitter()`

Код для рис. 4.47

```
p + geom_violin(aes(fill = Month), trim = FALSE)
```

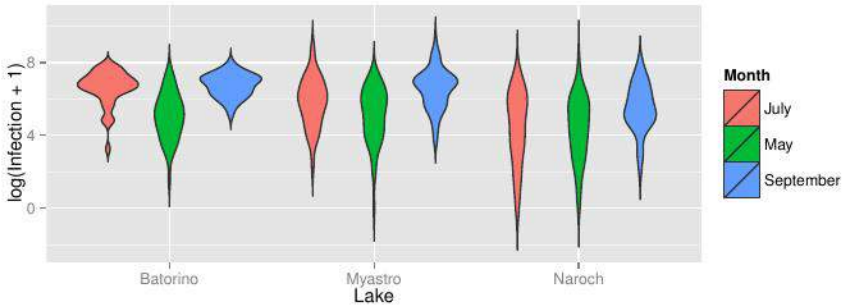


Рисунок 4.47. Скрипичные диаграммы для данных, разбитых на группы в соответствии с уровнями двух факторов (место и месяц отбора проб)

Код для рис. 4.48

```
p + geom_violin(aes(fill = Lake),
                trim = FALSE, alpha = 0.3) +
    geom_boxplot(aes(fill = Lake),
                 width = 0.2, outlier.colour = NA)
```

4.5 Визуализация зависимостей

В этом разделе описаны приемы визуализации зависимостей между двумя количественными переменными при помощи диаграмм рассеяния. Этот

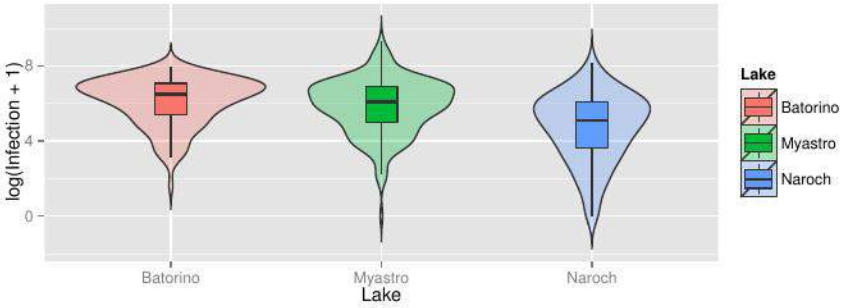


Рисунок 4.48. Скрипичная диаграмма в сочетании с диаграммой размахов

тип диаграмм уже встречался нам в разд. 2.2, где были приведены примеры использования функции `qplot()`. Здесь вы узнаете, как похожие диаграммы рассеяния можно создать при помощи функций `ggplot()` и `geom_point()`. Кроме того, будут рассмотрены способы визуализации регрессионных моделей, описывающих зависимость между двумя количественными переменными.

4.5.1 Диаграммы рассеяния: `geom_point()`

На диаграммах рассеяния значения двух переменных изображают в виде точек (или каких–либо других символов), размещенных на декартовой плоскости. Координаты точек представляют собой пары значений соответствующих переменных. Если есть основания предполагать существование некоего реального механизма, благодаря которому изменение одной переменной влечет за собой определенные изменения другой переменной, то значения *независимой переменной* откладывают по оси абсцисс (X), а значения *зависимой переменной* — по оси ординат (Y). В отсутствие такого механизма выбор оси для отображения той или иной переменной не принципиален. Часто размер точек на диаграмме рассеяния (в частности, их площадь) делают пропорциональным значениям какой–либо третьей количественной переменной, что приводит к получению «*пузырьковой диаграммы*» (англ. *bubble plot*). Хотя пузырьковые диаграммы изображают информацию о трех измерениях, они все же являются двухмерными графиками. В отличие от настоящих трехмерных диаграмм, т. е. графиков с тремя координатными осями, пузырьковые диаграммы легче воспринимать и интерпретировать. В пакете `ggplot2` диаграммы рассеяния и пузырьковые диаграммы создаются при помощи рассмотренной ниже функции `geom_point()`.

Аргументы

См. разд. 4.1.

Эстетические атрибуты

- `x*` и `y*` — переменные X и Y соответственно.

- `alpha` — степень прозрачности цвета точек.
- `colour` — цвет линий, окаймляющих точки.
- `fill` — цвет точек.
- `size` — размер точек.
- `shape` — форма точек.
- `stroke` — толщина линий, окаймляющих точки.

Примеры

Код для рис. 4.49

```
p <- ggplot(data = dreissena,
            aes(x = Length, y = log(Infection + 1)))
p + geom_point()
p + geom_point(shape = 22, size = 4, fill = "pink")
p + geom_point(shape = 15, size = 3, colour = "blue",
               alpha = 0.2)
```

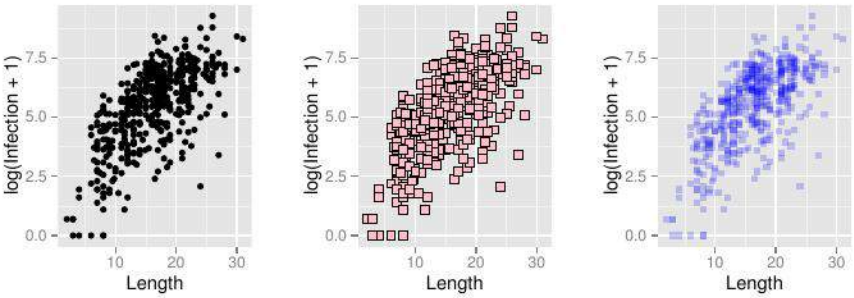


Рисунок 4.49. Примеры диаграмм рассеяния. *Слева*: график создан с использованием автоматических настроек. *В центре*: форма и размер точек изменены при помощи аргументов `shape` и `size` соответственно. *Справа*: форма, размер, а также цвет и степень прозрачности цвета точек изменены при помощи аргументов `shape`, `size`, `colour` и `alpha` соответственно

Код для рис. 4.50

```
# здесь и до рис. 4.54 объект p определен как на рис. 4.49
p + geom_point(aes(colour = Lake))
p + geom_point(aes(size = Day),
               colour = "tomato", alpha = 0.3)
```

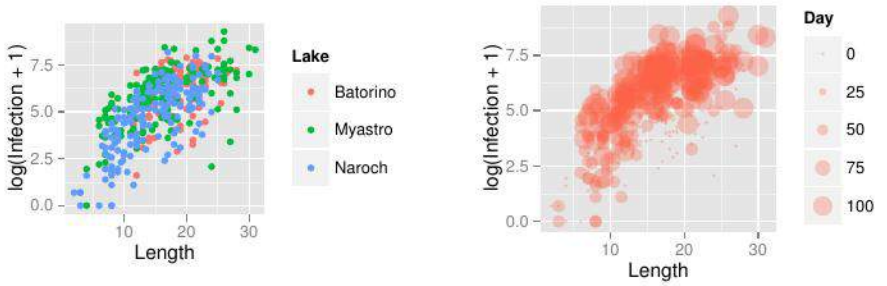


Рисунок 4.50. *Слева*: пример диаграммы рассеяния, на которой цвет точек задан в соответствии с уровнями качественной переменной. *Справа*: пузырьковая диаграмма, на которой размер точек задан в соответствии со значениями третьей количественной переменной

Код для рис. 4.51

```
p + geom_point(aes(colour = Day))
p + geom_point(colour = "gray30", size = 3.5) +
    geom_point(aes(colour = Day))
```

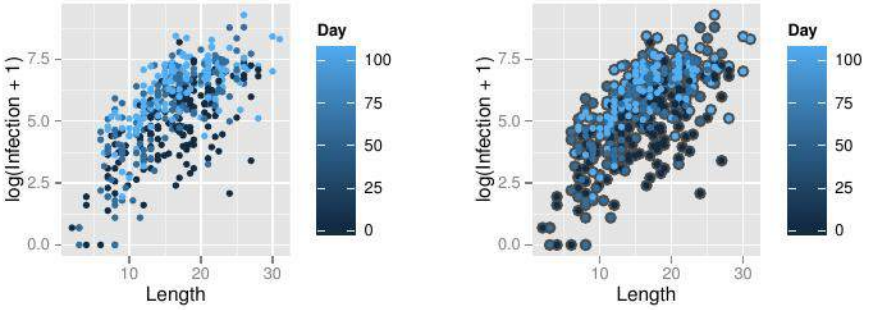


Рисунок 4.51. *Слева*: диаграмма рассеяния, на которой цвет точек задан в соответствии со значениями третьей количественной переменной. *Справа*: то же, но к исходному графическому объекту, инициализированному при помощи функции `ggplot()`, добавлены два слоя с точками (см. код)

4.5.2 Линии тренда: `geom_smooth()`

Как было отмечено в подразд. 2.3.1, добавление регрессионных линий к диаграммам рассеяния помогает лучше выявить зависимость между анализируемыми переменными, особенно при работе с данными большого объема. Одной из функций, позволяющих добавлять слои с такими линиями, является описанная ниже функция `geom_smooth()`.

Аргументы

- `method` — статистический метод, используемый для расчета сглаживающей линии. Реализованы следующие методы:
  - `method = "loess"`: метод *локально взвешенной полиномиальной регрессии*[10]. Степень сглаживания определяется параметром `span`, который варьирует в пределах от `0` (наиболее низкая степень сглаживания) до 1 (максимально гладкая кривая). При объеме наблюдений $n > 1000$ `loess` начинает работать медленно, и поэтому вместо него автоматически включается другой метод сглаживания (см. ниже);
  - `method = "gam"`: сглаживание на основе *обобщенной аддитивной модели* (англ. *generalized additive model*). Выполняется при помощи функции `gam()` из базового `R`–пакета `mgcv` (этот пакет следует предварительно загрузить в рабочую среду программы командой `library(mgcv)`). Подгонка обобщенной аддитивной модели сходна с подгонкой сплайн–моделей, за исключением того, что оптимальное число степеней свободы рассчитывается автоматически, исходя из свойств данных[11]. Спецификация подгоняемой модели задается формулой в виде `formula = y ~ s(x)` (при больших объемах данных рекомендуется применять модели, основанные на кубических сплайнах: `formula = y ~ s(x, bs = "cs")`);
  - `method = "lm"`: сглаживание на основе *линейной модели* (англ. *linear model*). По умолчанию подгоняется прямая линия регрессии. Кроме того, имеется возможность подогнать полиномиальную регрессионную модель любой степени (например, `formula = y ~ poly(x, 2)`) или загрузить стандартный пакет `splines` (командой `library(splines)`) и подогнать сплайн–модель (например, натуральный сплайн: `formula = y ~ ns(x, 3)`);
  - `method = "rlm"`: сглаживание с использованием *робастной линейной модели* (англ. *robust linear model*). Работает сходным с `"lm"` образом, однако использует алгоритм, благодаря которому наблюдения–выбросы не оказывают значительного влияния на параметры получаемой линейной модели. Функция `rlm()` является частью базового пакета `MASS`, который перед использованием необходимо загрузить в рабочую среду `R` командой `library(MASS)`.
- `formula` — формула, по которой выполняется подгонка соответствующей модели (см. примеры выше).
- `se` — логический аргумент. Включает (`se = TRUE`; значение, принятое по умолчанию) отображение доверительной области линии регрессии.
- `fullrange` — логический аргумент. При `fullrange = TRUE` регрессионная линия будет проходить через всю область диаграммы, тогда

[10] Подробнее см. справочный файл, доступный по команде `?loess`.
[11] Подробнее см. справочный файл функции `gam()`, доступный по команде `?gam`.

как при `fullrange = FALSE` — только через облако точек, по которым эта линия была рассчитана.

- `level` — уровень доверительной области регрессионной линии (0.95 по умолчанию).
- `n` — количество точек, по которым следует рассчитывать регрессионную модель.
- другие аргументы — см. разд. 4.1.

Эстетические аргументы

- `x*` и `y*` — переменные X и Y соответственно.
- `alpha` — степень прозрачности цвета.
- `colour` — цвет линии.
- `fill` — цвет заливки доверительной области регрессионной линии.
- `size` — толщина линии.

Примеры

Код для рис. 4.52

```
p + geom_point(alpha = 1/2) + geom_smooth()
p + geom_point(alpha = 1/2) + geom_smooth(method = "lm")
p + geom_point(alpha = 1/2) +
    geom_smooth(method = "lm", se = FALSE)
```

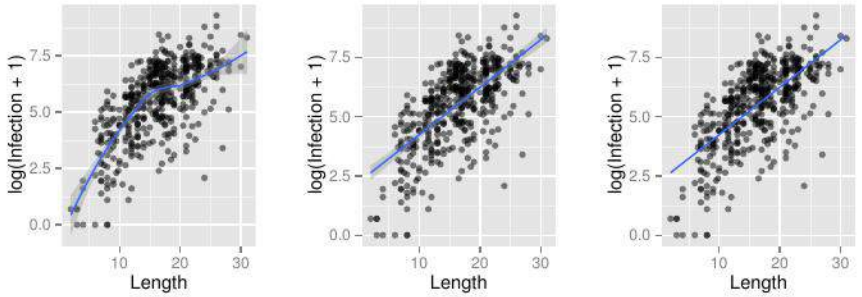


Рисунок 4.52. Примеры линий тренда, добавленных к диаграммам рассеяния. *Слева*: автоматически выбранная линия локально взвешенной полиномиальной регрессии. *В центре*: заданная пользователем линейная регрессия. *Справа*: то же, что и в центре, но с отключенной 95%–ной доверительной областью регрессионной линии

Код для рис. 4.53

```
library(mgcv); library(splines); library(MASS)
p + geom_smooth(method = "gam",
                formula = y ~ s(x, bs = "cs"))
p + geom_smooth(method = "lm", formula = y ~ poly(x, 2))
p + geom_smooth(method = "rlm", formula = y ~ ns(x, 3))
```

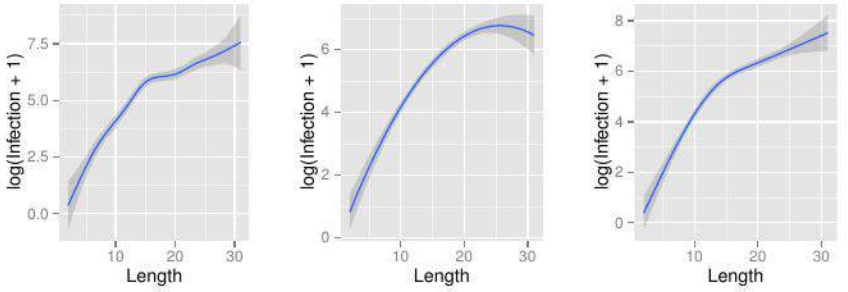


Рисунок 4.53. Примеры линий тренда, рассчитанных при помощи разных методов (для облегчения восприятия различий между этими графиками отображение точек отключено). *Слева*: обобщенная аддитивная модель на основе кубических сплайнов. *В центре*: полином второй степени. *Справа*: робастная регрессия на основе натурального кубического сплайна

Код для рис. 4.54

```
p + geom_smooth(fill = "yellow")
p + geom_smooth(fill = "blue", colour = "red", alpha = 0.2)
p + geom_smooth(fill = "skyblue", colour = "darkblue",
                linetype = 2, size = 2)
```

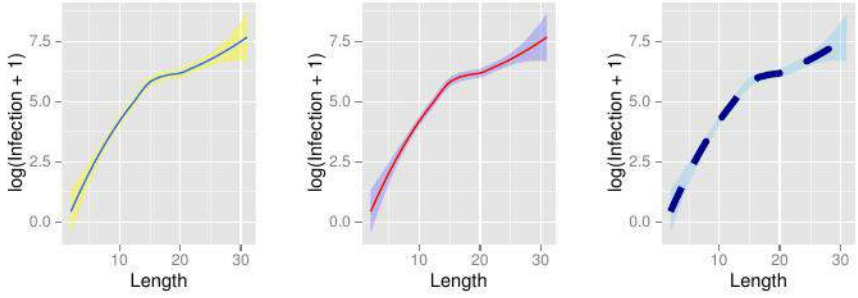


Рисунок 4.54. Настройка внешнего вида линий тренда при помощи аргументов `fill`, `colour`, `alpha`, `linetype` и `size` (см. код)

Код для рис. 4.55

```
p <- ggplot(data = dreissena, aes(x = Length,
            y = log(Infection + 1), colour = Lake))
p + geom_smooth(aes(fill = Lake), alpha = 0.2, size = 2)
```

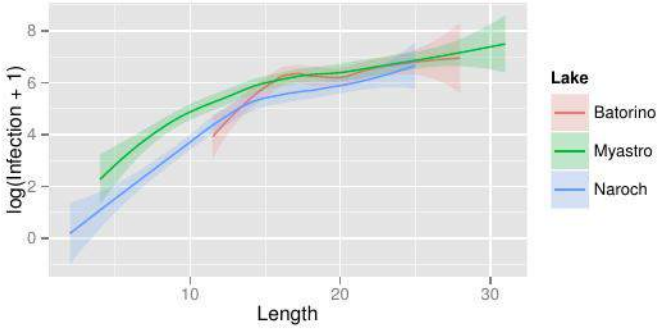


Рисунок 4.55. Линии тренда для трех групп, заданных в соответствии с уровнями качественной переменной

4.5.3 Линии квантильной регрессии: `geom_quantile()`

Квантильная регрессия (англ. *quantile regression*) представляет собой разновидность регрессионного анализа, в котором ставится задача оценить параметры модели, предсказывающей *медиану* или какие–либо другие квантили зависимой переменной на основе одной или нескольких независимых переменных[12]. Этим квантильная регрессия отличается от классической линейной регрессии, задача которой состоит в предсказании *среднего* значения зависимой переменной по заданным значениям независимых переменных. Кроме того, в отличие от классической регрессии, параметры которой оцениваются по *методу наименьших квадратов*, параметры квантильной регрессии оцениваются с использованием *симплекс–метода*[13]. В пакете `ggplot2` для добавления слоев с линиями квантильной регрессии служит описанная ниже функция `geom_quantile()`.

Аргументы

- `quantiles` — вектор с квантилями, подлежащими предсказанию с помощью квантильной регрессии.
- `formula` — формула, определяющая характер связи между X и Y.

[12] Хорошее введение в этот метод можно найти в статье: Cade B. F., Noon B. R. (2003) A gentle introduction to quntile regression for ecologists. Frontiers in Ecology and Environment 1(8): 412–420.

[13] `http://en.wikipedia.org/wiki/Simplex_method`.

- `method` — название метода, применяемого для оценки параметров квантильной регрессии. В настоящее время поддерживается только функция `rq()` из пакета `quantreg`, который предварительно должен быть установлен на вашем компьютере[14].
- другие аргументы — см. разд. 4.1.

Эстетические атрибуты

- `x*` и `y*` — переменные X и Y соответственно.
- `alpha` — степень прозрачности цвета.
- `colour` — цвет линии.
- `linetype` — тип линии.
- `size` — толщина линии.

Примеры

Код для рис. 4.56

```
library(quantreg) # для функции rq()
library(splines) # для функции ns()
p <- ggplot(data = dreissena,
            aes(x = Length, y = log(Infection + 1))) +
            geom_point(alpha = 0.2)
p + geom_quantile()
p + geom_quantile(formula = y ~ poly(x, 2))
p + geom_quantile(formula = y ~ ns(x, 3))
```

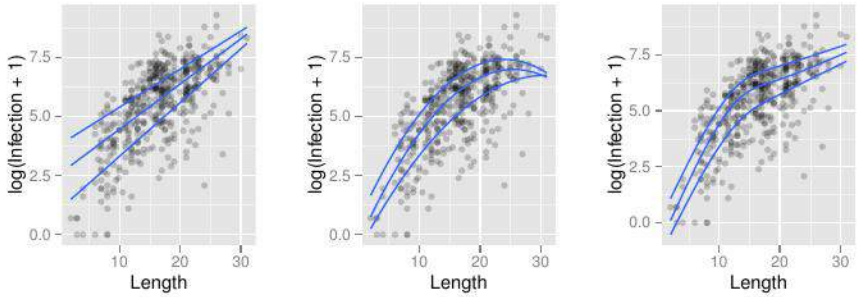


Рисунок 4.56. *Слева*: линии квантильной регресии, рассчитанные с использованием автоматических настроек функции `geom_quantile()`. Показаны медиана (средняя линия), а также 0.25 (нижняя линия) и 0.75 квантили (верхняя линия). *В центре*: квантильные линии для полинома второй степени. *Справа*: квантильные линии для кубического сплайна

[14] Подробнее см. руководство к пакету `quantreg`, доступное по адресу `http://bit.ly/2bZRf3Z`.

Код для рис. 4.57

```
# здесь и до рис. 4.58 объект p определен как на рис. 4.56
p + geom_quantile(formula = y ~ x,
                  quantiles = c(0.5), size = 1) +
    geom_smooth(method = "lm", colour = "red",
                size = 1, se = FALSE)
p + geom_quantile(formula = y ~ ns(x, 3), colour = "red",
                  quantiles = c(0.025, 0.5, 0.975))
p + geom_quantile(formula = y ~ ns(x, 3), colour = "magenta",
                  linetype = 2, size = 1,
                  quantiles = c(0.025, 0.5, 0.975))
```

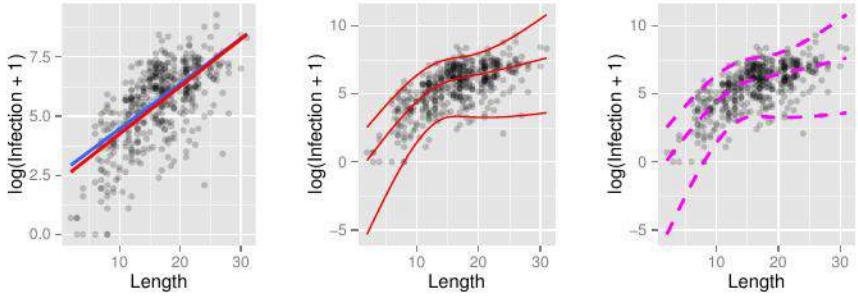


Рисунок 4.57. *Слева*: сравнение линий тренда, рассчитанных при помощи обычной линейной регрессии (обозначена красным цветом) и квантильной линейной регрессии для медианы (обозначена синим цветом). *В центре*: квантильная регрессия на основе кубического сплайна. Показаны линии для медианы, а также для 0.025 (нижняя линия) и 0.975 (верхняя линия) квантилей. Цвет линий задан при помощи аргумента `colour = "red"`. *Справа*: то же, что и в центре, однако цвет, тип и толщина линий изменены при помощи соответствующих аргументов (см. код)

Код для рис. 4.58

```
p + aes(colour = Lake) +
    geom_quantile(formula = y ~ ns(x, 3),
                  quantiles = c(0.5), size = 1)
p + geom_quantile(aes(colour = ..quantile..),
                  formula = y ~ ns(x, 3),
                  quantiles = seq(0.05, 0.95, by = 0.05))
```

4.6 Визуализация временных рядов

Динамику того или иного процесса принято изображать в виде линии, которая последовательно (слева направо) соединяет наблюдения, получен-

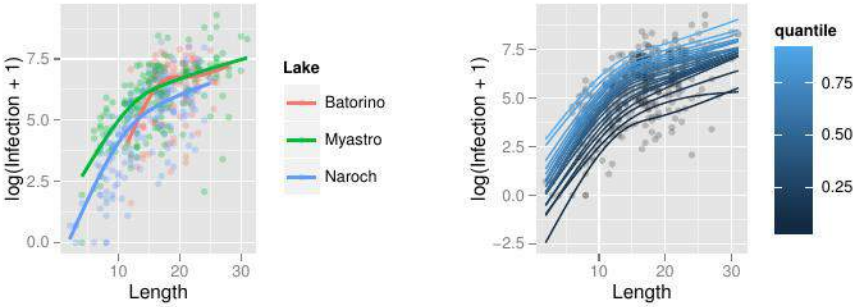


Рисунок 4.58. *Слева*: линии квантильной регрессии (медиана) на основе кубического сплайна, рассчитанные отдельно для каждого уровня качественной переменной. *Справа*: линии квантильной регрессии на основе кубического сплайна, рассчитанные для квантилей $0.05-0.95$ с шагом 0.05

ные в разные моменты времени. В этом разделе мы воспользуемся данными из входящей в состав `ggplot2` таблицы `economics`, которая содержит многолетние наблюдения по нескольким показателям экономики США. Мы сосредоточимся только на одном из эти показателей — численности безработных (`unemploy`, тысяч человек). Для построения временных рядов служит функция `geom_line()`. Кроме того, мы рассмотрим функцию `geom_ribbon()` — с ее помощью к линии временного ряда можно добавить полосу, границы которой обозначают показатели разброса или точности (например, минимальное и максимальное значения анализируемой переменной для каждой даты, нижняя и верхняя доверительные границы и т. п.).

4.6.1 Функция `geom_line()`

Аргументы

См. разд. 4.1.

Эстетические атрибуты

- `x*` и `y*` — переменные X и Y соответственно.
- `alpha` — степень прозрачности цвета.
- `colour` — цвет линии.
- `linetype` — тип линии.
- `size` — толщина линии.

Примеры

Код для рис. 4.59

```
p <- ggplot(data = economics, aes(x = date, y = unemploy)) +
     xlab("Дата") + ylab("Численность, тыс.")
```

```
p + geom_line()
p + geom_line(colour = "blue", size = 1)
```

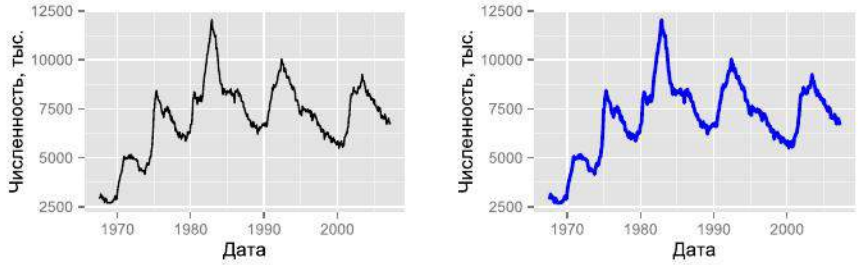


Рисунок 4.59. Динамика численности безработных в США в период с 1967 по 2007 г. *Слева*: график построен с использованием автоматических настроек функции `geom_line()`. Обратите внимание: переменная `date` в таблице `economics` принадлежит к классу `Date` и содержит даты в стандартном для R формате. Функция `geom_line()` автоматически распознала класс этой переменной и правильно упорядочила наблюдения вдоль оси X. *Справа*: цвет и толщина линии изменены при помощи аргументов `colour = "blue"` и `size = 1` соответственно

Код для рис. 4.60

```
econ <- economics # копия исходной таблицы
# добавим новую переменную - год проведения исследования:
econ$year <- 1900 + as.POSIXlt(economics$date)$year
# добавим новую переменную - месяц (внимание: в R
# по умолчанию январь имеет порядковый номер 0):
econ$month <- as.POSIXlt(economics$date)$mon

# Построение графиков:
p <- ggplot(data = econ, aes(x = month, y = unemploy,
                             group = factor(year))) +
     xlab("Месяц") + ylab("Численность, тыс.")

# Использование переменной year в качестве фактора
# (функция theme() использована для отключения легенды;
# подробнее эта функция будет рассмотрена в главе 8):
p + geom_line(aes(colour = factor(year))) +
    theme(legend.position = "NA")

# year как количественная переменная:
p + geom_line(aes(colour = year))
```

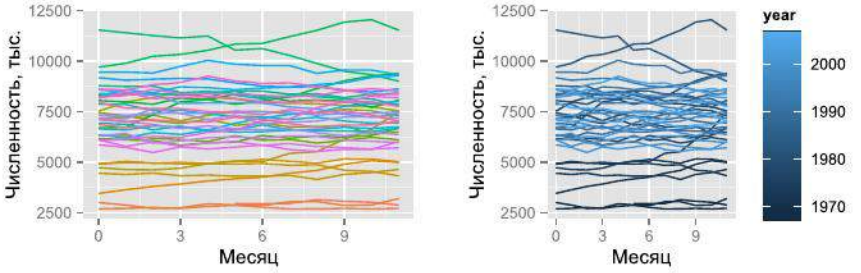


Рисунок 4.60. Динамика безработицы в США в пределах каждого года наблюдений (с 1967–го по 2007–й). *Слева*: добавленная в таблицу с данными количественная переменная `year` была преобразована в фактор, после чего линиям на графике были присвоены дискретные цвета в соответствии с уровнями этого фактора. Легенда графика была отключена из–за большого количества уровней переменной `factor(year)`. *Справа*: количественная переменная `year` не была преобразована фактор, в связи с чем линиям присвоены плавно переходящие друг в друга цвета

4.6.2 Функция `geom_ribbon()`

Аргументы

См. разд. 4.1.

Эстетические атрибуты

- `x`* — переменная X (время или какая–либо другая количественная переменная).
- `ymax`* — вектор с Y–координатами, задающими верхнюю границу полосы.
- `ymin`* — вектор с X–координатами, задающими нижнюю границу полосы.
- `fill` — фоновый цвет полосы.
- `alpha` — степень прозрачности фонового цвета.
- `colour`, `linetype`, `size` — цвет, тип и толщина окаймляющих полосу линий.

Примеры

Код для рис. 4.61

```
p <- ggplot(data = economics,
            aes(x = date, ymax = unemploy + 1000,
            ymin = unemploy - 1000)) +
     xlab("Дата") + ylab("Численность, тыс.")
p + geom_ribbon()
```

```
p + geom_ribbon(fill = "lightblue",
                colour = "blue", linetype = 3)
p + geom_ribbon(fill = "orange", colour = "red",
    size = 1) + geom_line(aes(y = unemploy))
```

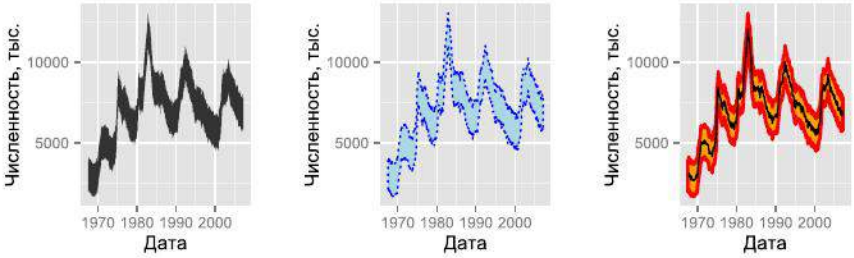


Рисунок 4.61. Полоса, верхняя и нижняя границы которой были рассчитаны по данным из таблицы `economics` как `(unemploy + 1000)` и `(unemploy - 1000)` соответственно. *Слева*: диаграмма, построенная с использованием автоматических настроек. *В центре*: цвет полосы (аргумент `fill`), а также цвет (`colour`) и тип (`linetype`) окаймляющих полосу линий изменены. *Справа*: изменен цвет полосы, а также цвет и ширина (`size`) окаймляющих ее линий; кроме того, поверх полосы добавлена линия черного цвета, отражающая динамику безработицы

4.7 Тепловые карты: `geom_tile()`

Тепловая карта (англ. *«heat map»*) представляет собой один из способов визуализации трехмерных наборов данных (см. также подразд. 4.3.2). Первые две переменные при этом являются количественными дискретными или могут быть преобразованы в таковые. Значения этих двух переменных задают координатную плоскость, которая разбивается на прямоугольные сегменты одинакового размера. Каждый сегмент далее закрашивается цветом, градиент которого определяется значениями третьей количественной переменной (непрерывной или дискретной). Тепловые карты широко используются для анализа биржевых данных, в молекулярной биологии, веб–аналитике и других областях. В пакете `ggplot2` для создания такого рода диаграмм служит описанная ниже функция `geom_tile()`[15].

Аргументы

См. разд. 4.1.

[15] При работе с данными большого объема стоит воспользоваться `geom_raster()` — специально оптимизированным вариантом функции `geom_tile()`. Обе функции имеют почти одинаковые аргументы, и поэтому `geom_raster()` здесь отдельно не рассматривается (см. справочный файл, доступный по команде `?geom_raster`).

Эстетические атрибуты

- `x*` и `y*` — векторы с X– и Y–координатами[16].
- `alpha` — степень прозрачности цвета.
- `fill` — цвет заливки прямоугольных сегментов. Как правило, на этот аргумент подается имя третьей количественной переменной (например, `fill = z`).
- `colour`, `linetype`, `size` — цвет, тип и толщина линий, окаймляющих прямоугольные сегменты.

Примеры

Код для рис. 4.62

```
# Этот пример основан на искусственных данных,
# сгенерированных следующим образом:
x = seq(-10, 10, 1)
df <- expand.grid(X = x, Y = x)
df$z <- sin(df$X)*cos(df$Y/3)
# Построение графиков:
p <- ggplot(df, aes(x = X, y = Y, fill = z))
p + geom_tile()
# В приведенной ниже команде функция scale_fill_gradient()
# использована для смены цветовой гаммы. Эта функция будет
# подробно рассмотрена позднее:
p + geom_tile() +
    scale_fill_gradient(low = "red", high = "yellow")
```

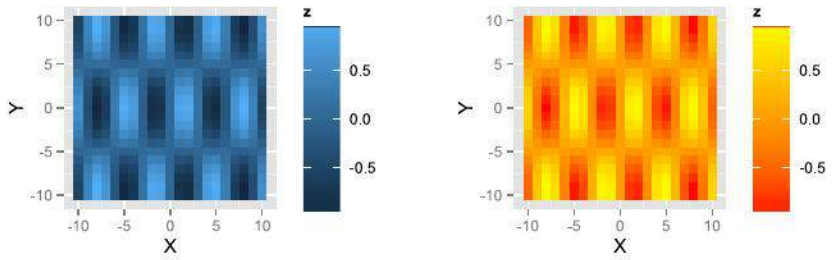


Рисунок 4.62. *Слева*: пример тепловой карты, построенной с использованием автоматических настроек. *Справа*: то же, но цветовая гамма изменена при помощи функции `scale_fill_gradient()`

[16] См. также пример на рис. 4.30.

4.8 Другие геометрические объекты

Данный раздел посвящен диаграммам, которые сложно отнести к какому–либо из рассмотренных нами ранее основных типов. Среди прочего вы узнаете, например, о том, как добавить к графику слои с вертикальными и горизонтальными линиями, ломаными кривыми, прямоугольниками и многоугольниками произвольной формы, текстовыми подписями и др.

4.8.1 «График–щетка»: `geom_rug()`

«График–щетка»[17] представляет собой один из способов визуализации распределения количественных переменных. Суть этого способа очень проста: к числовой оси (или осям) на графике добавляются небольшие перпендикулярные отрезки, символизирующие значения соответствующей переменной. О характере распределения анализируемой переменной судят по плотности расположения таких отрезков. Как правило, графики–щетки в самостоятельном виде не используются — вместо этого их комбинируют с кривыми плотности вероятности (см. подразд. 4.2.5) или диаграммами рассеяния (подразд. 4.5.1). В `ggplot2` для построения графика–щетки служит описанная ниже функция `geom_rug()`.

Аргументы

- `sides` — текстовое значение, определяющее стороны графика, по которым будут располагаться «щетки». Может включать комбинации из следующих четырех букв: `"t"` — вверху (от *top*), `"r"` — справа (*right*), `"b"` — внизу (*bottom*) и `"l"` — слева (*left*). По умолчанию этот аргумент имеет значение `sides = "bl"`.

- другие аргументы — см. разд. 4.1.

Эстетические атрибуты

- `alpha` — степень прозрачности цвета линий.

- `colour`, `linetype`, `size` — цвет, тип и толщина линий.

Примеры

Код для рис. 4.63

```
p <- ggplot(data = subset(dreissena, Lake == "Naroch"),
            aes(x = log(Infection + 1)))
p + geom_density(fill = "orchid") + geom_rug()
p + geom_density(fill = "orchid") + geom_rug(colour = "red")
p + geom_density(fill = "orchid") +
    geom_rug(colour = "red", alpha = 0.3)
```

[17] В оригинале используется название *«rug plot»*, что можно перевести также как «график–коврик».

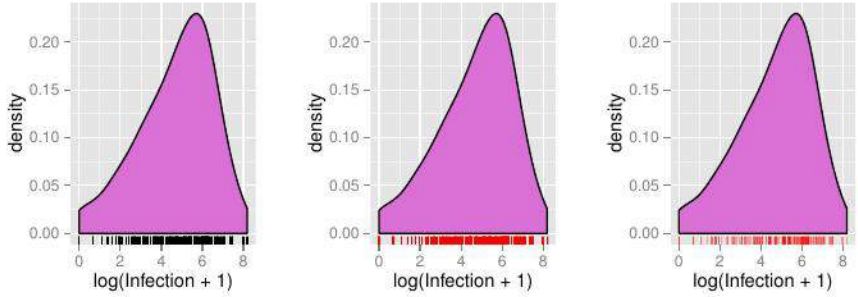


Рисунок 4.63. Примеры «графиков–щеток». *Слева*: «щетка» добавлена к диаграмме плотности вероятности; использованы автоматические настройки. *В центре*: цвет линий изменен на красный (аргумент `colour = "red"`). *Справа*: то же, что и в центре, но прозрачность цвета линий увеличена (аргумент `alpha = 0.3`)

Код для рис. 4.64

```
p <- ggplot(data = subset(dreissena, Lake == "Naroch"),
            aes(x = Length, y = log(Infection + 1)))
p + geom_rug()
p + geom_point() + geom_rug()
p + geom_point() + geom_rug(sides = "tr")
```

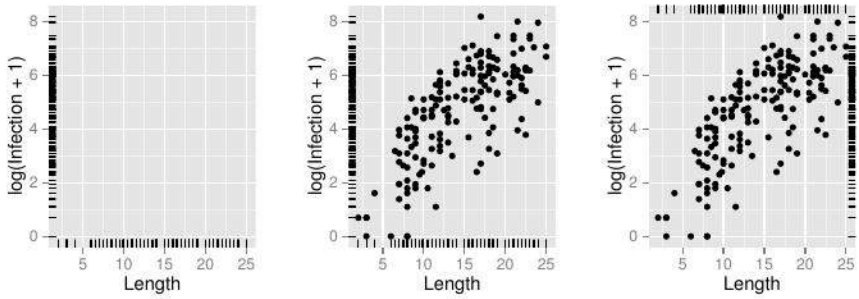


Рисунок 4.64. «Щетки», добавленные к диаграммам рассеяния. *Слева*: собственно диаграмма рассеяния не отображена, что делает использование «щеток» малоинформативным. *В центре*: по умолчанию «щетки» располагаются в нижней и левой частях графика. *Справа*: расположение «щеток» изменено при помощи аргумента `sides = "tr"`

4.8.2 Горизонтальные и вертикальные линии: geom_hline(), geom_vline()

Горизонтальные и вертикальные линии могут оказаться полезными при добавлении аннотаций к графикам. В частности, такие линии можно использовать для обозначения пороговых значений анализируемых переменных. Примеры подобных аннотаций будут рассмотрены ниже. Для добавления к графикам горизонтальных и вертикальных линий служат функции `geom_hline()` и `geom_vline()` соответственно.

Аргументы

См. разд. 4.1.

Эстетические атрибуты

Основными аргументами функций `geom_hline()` и `geom_vline()` и являются `yintercept` и `xintercept`, при помощи которых задаются X– и `Y`–координаты линий соответственно.

- `yintercept` (только для функции `geom_hline()`) — вектор с Y–координатами горизонтальных линий.
- `xintercept` (только для функции `geom_vline()`) — вектор с X–координатами вертикальных линий.
- `alpha` — степень прозрачности цвета линий.
- `colour`, `linetype`, `size` — цвет, тип и толщина линий.

Примеры

Код для рис. 4.65

```
# Отдельная таблица данных по оз. Нарочь:
Naroch <- subset(dreissena, Lake == "Naroch")

# Добавим столбец с log-значениями интенсивности инвазии:
Naroch$logInf <- log(Naroch$Infection + 1)

# Рассчитаем медиану интенсивности инвазии (на log-шкале):
med.inf <- median(Naroch$logInf)

# Добавим столбец с метками уровеня инвазии -
# ниже ("Lower") или выше ("Higher") медианы:
Naroch$Level <- with(Naroch,
                ifelse(logInf < med.inf, "Lower", "Higher"))

# Инициализация диаграммы рассеяния, на которой цвет точек
# задан в соответствии с уровнем инвазии (Level):
p <- ggplot(data = Naroch,
            aes(x = Length, y = logInf, colour = Level)) +
            theme(legend.position = "none")
```

```
# Диаграмма рассеяния с горизонтальной линией,
# символизирующей медианный уровень инвазии:
p + geom_point() + geom_hline(yintercept = med.inf)

# То же, но внешний вид горизонтальной линии изменен:
p + geom_point() + geom_hline(yintercept = med.inf,
    colour = "blue", linetype = 5, size = 1)

# Добавляем три горизонтальные линии, соответствующие
# 0.25, 0.5 и 0.75 процентилям интенсивности инвазии:
p + geom_point(colour = "gray40") +
    geom_hline(yintercept = quantile(Naroch$logInf,
                p = c(0.25, 0.5, 0.75)), colour = "blue")
```

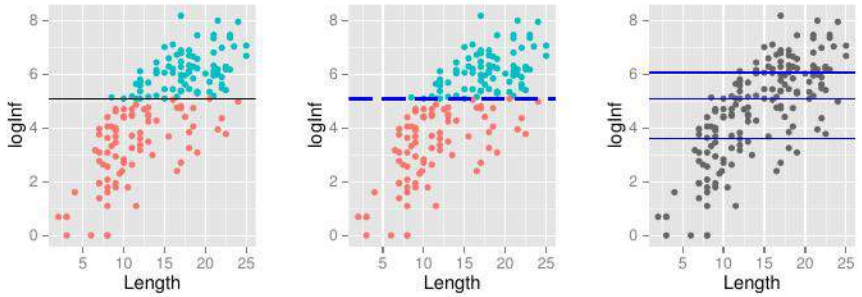


Рисунок 4.65. *Слева*: к диаграмме рассеяния добавлена горизонтальная линия, внешний вид которой задан в соответствии с автоматическими настройками. *В центре*: то же, что и слева, но эстетические атрибуты линии изменены. *Справа*: добавлены горизонтальные линии, соответствующие 0.25, 0.5 и 0.75 процентилям переменной Y

Код для рис. 4.66

```
# Рассчет среднего значения длины раковины дрейссены
# (по данным из таблицы Naroch - см. предыдущий пример):
mean.L <- mean(Naroch$Length)

# Добавляем столбец с метками класса mean.L -
# "Ниже" или "Выше" среднего значения:
Naroch$L.level <- with(Naroch,
          ifelse(Length < mean.L, "Lower", "Higher"))

# Инициализация диаграммы рассеяния, на которой цвет точек
# задан в соответствии с классом длины раковины (L.level):
p <- ggplot(data = Naroch,
     aes(x = Length, y = logInf, colour = L.level)) +
     theme(legend.position = "none")
```

```
# Диаграмма рассеяния с вертикальной линией,
# символизирующей среднюю длину раковины:
p + geom_point() + geom_vline(xintercept = mean.L)

# То же, но внешний вид вертикальной линии изменен:
p + geom_point() + geom_vline(xintercept = mean.L,
    colour = "blue", linetype = 5, size = 1)

# Добавляем три вертикальные линии, соответствующие
# 0.25, 0.5 и 0.75 процентилям длины раковины:
p + geom_point(colour = "gray40") +
    geom_vline(xintercept =
    quantile(Naroch$Length, p = c(0.25, 0.5, 0.75)),
    colour = "blue")
```

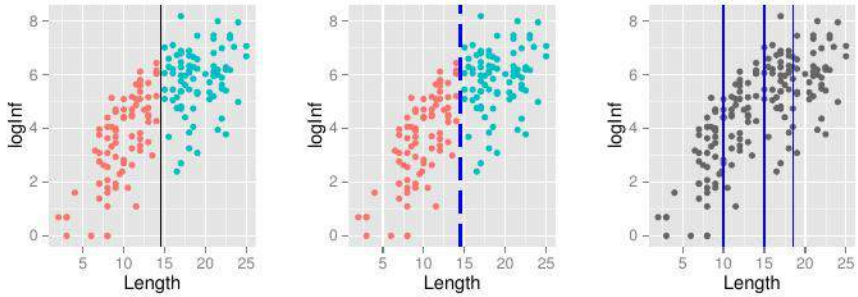


Рисунок 4.66. *Слева*: к диаграмме рассеяния добавлена вертикальная линия, внешний вид которой задан в соответствии с автоматическими настройками. *В центре*: то же, что и слева, но эстетические атрибуты линии изменены. *Справа*: к диаграмме рассеяния добавлены горизонтальные линии, соответствующие 0.25, 0.5 и 0.75 процентилям переменной X

4.8.3 Прямоугольные области: `geom_rect()`

Функция `geom_rect()` служит для добавления к `ggplot`–графикам слоев с прямоугольными областями.

Аргументы

См. разд. 4.1.

Эстетические атрибуты

- `xmin`*, `xmax`*, `ymin`* и `ymax`* — векторы с координатами прямоугольников.
- `alpha` — степень прозрачности цвета прямоугольников.

- `fill` — цвет прямоугольников.
- `colour`, `linetype`, `size` — цвет, тип и толщина линий, окаймляющих прямоугольники.

Примеры

Код для рис. 4.67

```
set.seed(1022) # для воспроизводимости примера
# Таблица, содержащая случайно сгенерированные значения,
# которые будут использованы при определении координат
# прямоугольных областей:
df <- data.frame(x = sample(10, 20, replace = TRUE),
                 y = sample(10, 20, replace = TRUE))

p <- ggplot(df, aes(xmin = x, xmax = x + 1,
                    ymin = y, ymax = y + 2))
p + geom_rect()
p + geom_rect(fill = "orange")
p + geom_rect(fill = "orange", col = "black", linetype = 2)
```

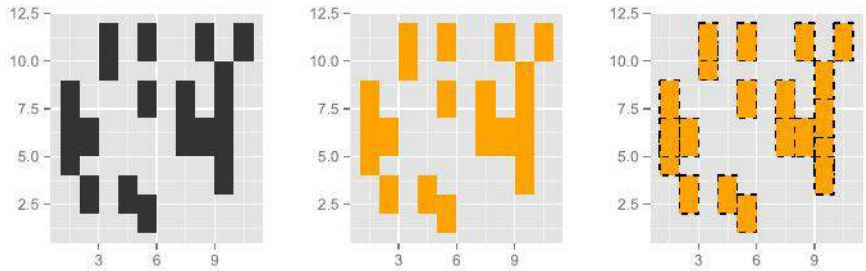


Рисунок 4.67. Примеры диаграмм с прямоугольными областями. *Слева*: диаграмма, построенная с использованием автоматических настроек. *В центре*: то же, что и слева, но цвет прямоугольников изменен (аргумент `fill = "orange"`). *Справа*: то же, что и в центре, но эстетические атрибуты линий, окаймляющих прямоугольники, изменены (см. код)

4.8.4 Отрезки: `geom_segment()`

При необходимости мы можем добавить к графику слой, содержащий один или несколько линейных отрезков. Более того, один или оба конца этих отрезков могут выглядеть как стрелки, что удобно для аннотации определенных элементов графиков. Для добавления слоя с отрезками служит описанная ниже функция `geom_segment()`.

Аргументы

- `arrow` — служит для спецификации внешнего вида головок стрелок, которые можно добавить к одному или обоим концам отрезка. По умолчанию этот аргумент принимает значение `NULL`, т. е. головки стрелок не отображаются. Остальные значения аргумента задаются при помощи функции `arrow()` из пакета `grid` (входит в базовую версию `R`). Подробное описание аргументов этой функции доступно в ее справочном файле (доступен по команде `?arrow`).

- другие аргументы — см. разд. 4.1.

Эстетические атрибуты

- `x*` и `y*` — векторы с X– и Y–координатами начала отрезка.

- `xend*` и `yend*` — векторы с X– и Y–координатами конца отрезка.

- `alpha` — степень прозрачности цвета.

- `colour`, `linetype`, `size` — цвет, тип и толщина линии.

Примеры

Код для рис. 4.68

```
# Этот пример основан на искусственных данных,
# созданных следующим образом:
df <- data.frame(X = c(1, 4, 5), XEND = c(2, 3, 6),
                 Y = c(1, 3, 5), YEND = c(2, 4, 6))
p <- ggplot(df, aes(x = X, xend = XEND, y = Y, yend = YEND))
p + geom_segment()
p + geom_segment(arrow = arrow(length = unit(0.5,"cm")),
                 colour = "blue")
p + geom_segment(arrow = arrow(length = unit(0.5,"cm"),
                 ends = "both"), colour = "red", size = 1)
```

4.8.5 Ломаные линии: `geom_path()`

Представьте, что мы наблюдаем за передвижением некоторого объекта на плоскости в течение 20 минут. Ежеминутно мы отмечаем X– и Y–координаты этого объекта. Полученные данные можно визуализировать при помощи ломаной линии, воспользовавшись описанной ниже функцией `geom_path()` (от англ. *path*, что значит *«путь»*).

Аргументы

- `lineend` — определяет внешний вид концов линии. Возможные значения: `"round"` (круглые), `"butt"` (усеченные квадратные) и `"square"` (квадратные).

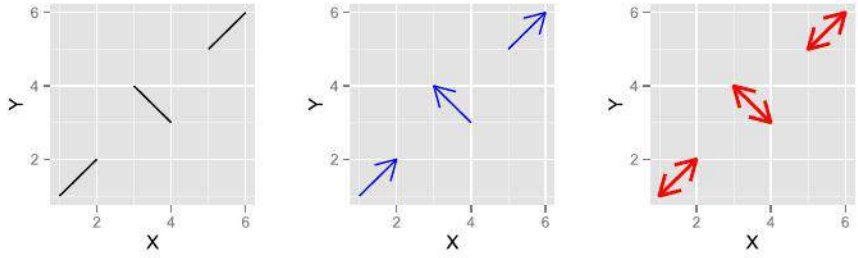


Рисунок 4.68. *Слева*: диаграмма с отрезками, построенная с использованием автоматических настроек. *В центре*: то же, что и слева, но отрезки выполнены в виде однонаправленных стрелок; кроме того, изменен цвет линий. *Справа*: то же, что и в центре, но отрезки выполнены в виде стрелок, направленных в обе стороны; изменены также цвет и ширина линий

- `linejoin` — определяет тип соединения между отдельными сегментами линии. Возможные значения: `"round"` — округлое, `"mitre"` и `"bevel"` — варианты скошенных стыков.

- `linemitre` — степень скошенности стыка между отдельными сегментами линии при `linejoin = "mitre"`. Принимает любое число, превышающее либо равное 1.

- `arrow` — определяет внешний вид головок стрелок, которые можно добавить к одному или обоим концам отрезка. По умолчанию этот аргумент принимает значение `NULL`, т. е. головки стрелок не отображаются. Остальные значения аргумента задаются при помощи функции `arrow()` из пакета `grid` (входит в базовую версию `R`). Подробное описание аргументов этой функции доступно в ее справочном файле (доступен по команде `?arrow`).

- другие аргументы — см. разд. 4.1.

Эстетические атрибуты

- `x*` и `y*` — векторы с X– и Y–координатами узловых точек, которые соединяются линией.

- `alpha` — степень прозрачности цвета.

- `colour`, `linetype`, `size` — цвет, тип и толщина линии.

Примеры

Код для рис. 4.69

```
set.seed(1100) # для воспроизводимости примера
df <- data.frame(X = sample(10, 20, replace = TRUE),
                 Y = sample(10, 20, replace = TRUE),
                 Time = 1:20)
```

```
p <- ggplot(df, aes(x = X, y = Y)); p + geom_path()
p + geom_path(aes(colour = Time), size = 1)
```

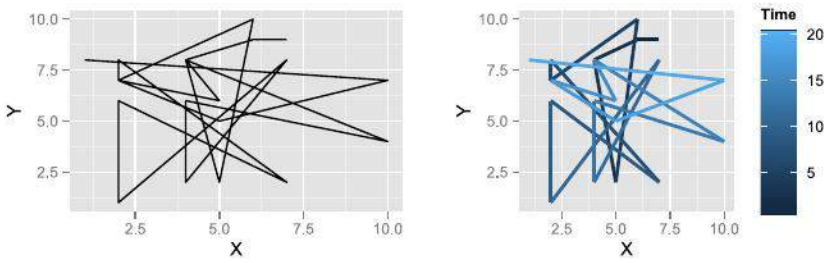


Рисунок 4.69. Пример линии, описывающей перемещение объекта на плоскости. *Слева*: диаграмма, построенная с использованием автоматических настроек. *Справа*: то же, что и слева, но цвет линии задан в соответствии с ходом времени

Код для рис. 4.70

```
df <- data.frame(x = 1:3, y = c(4, 1, 9))
p <- ggplot(df, aes(x, y)) + xlim(c(0, 4)) + ylim(c(0, 10))
p + geom_path(size = 10, lineend = "round") # "A"
p + geom_path(size = 10, lineend = "butt") # "B"
p + geom_path(size = 10, lineend = "square") # "C"
p + geom_path(size = 10, lineend = "round",
              linejoin = "round") # "D"
p + geom_path(size = 10, lineend = "round",
              linejoin = "mitre") # "E"
p + geom_path(size = 10, lineend = "round",
              linejoin = "bevel") # "F"
```

Код для рис. 4.71

```
set.seed(1100)
df <- data.frame(X = sample(10, 20, replace = TRUE),
                 Y = sample(10, 20, replace = TRUE),
                 Time = 1:20)
p <- ggplot(df, aes(x = X, y = Y))
p + geom_path(aes(size = Time),
              alpha = 0.4, colour = "tomato")
p + geom_path(aes(size = Time), alpha = 0.4,
              colour = "tomato", lineend = "round")
```

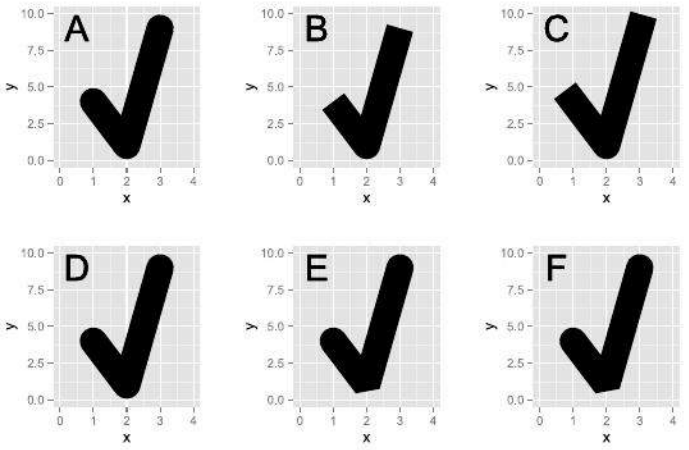


Рисунок 4.70. Пример разных вариантов сочленения сегментов ломаной линии (аргументы `lineend` и `linejoin` функции `geom_path()`)

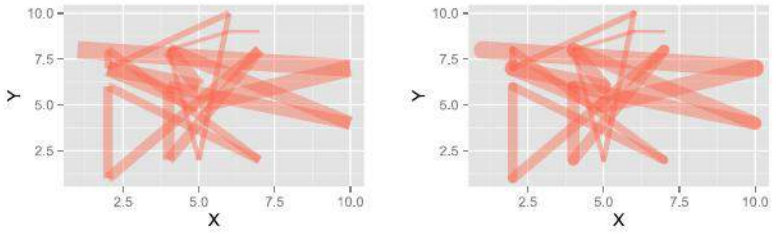


Рисунок 4.71. *Слева*: толщина линии соответствует значениям третьей количественной переменной (`aes(size = Time)`); кроме того, изменены цвет линии (`colour = "tomato"`) и степень его прозрачности (`alpha = 0.4`). *Справа*: то же, но с округлыми концами линий (`lineend = "round"`)

4.8.6 Многоугольники: `geom_polygon()`

В предыдущем подразделе были рассмотрены диаграммы, на которых данные изображаются в виде ломаных линий. Если первая и последняя точки такой линии совпадают, то мы получаем *замкнутую ломаную*, которую называют также *многоугольником*. Как правило, говоря о многоугольниках, подразумевают замкнутые ломаные без самопересечений (т. е. отдельные отрезки у таких линий нигде не пересекаются — например, как у треугольников). Однако существует и бесконечное число более сложных многоугольных фигур, образованных самопересекающимися ломаными. В пакете `ggplot2` для создания многоугольников (англ. *polygons*) служит описанная ниже функция `geom_polygon`. Помимо создания собственно многоугольных фигур, эта функция позволяет также закрасить их цветом, что можно использовать для отображения уровней какой–либо третьей (количественной) переменной.

Аргументы

См. разд. 4.1.

Эстетические атрибуты

- `x*` и `y*` — векторы с X– и Y–координатами узловых точек ломаной.
- `fill` — цвет многоугольника.
- `alpha` — степень прозрачности цвета многоугольника.
- `colour`, `linetype`, `size` — цвет, тип и толщина линии, окаймляющей многоугольник.

Примеры

Код для рис. 4.72

```
# Этот пример основан на искусственных данных.
# ids - идентификаторы отдельных частей многоугольника:
ids <- factor(c("1.1", "2.1", "1.2", "2.2", "1.3", "2.3"))
positions <- data.frame( id = rep(ids, each = 4),
  x = c(2, 1, 1.1, 2.2, 1, 0, 0.3, 1.1, 2.2, 1.1, 1.2, 2.5,
        1.1, 0.3, 0.5, 1.2, 2.5, 1.2, 1.3, 2.7, 1.2, 0.5,
        0.6, 1.3),
  y = c(-0.5, 0, 1, 0.5, 0, 0.5, 1.5, 1, 0.5, 1, 2.1,
        1.7, 1, 1.5, 2.2, 2.1, 1.7, 2.1, 3.2, 2.8, 2.1,
        2.2, 3.3, 3.2))
p <- ggplot(data = positions, aes(x = x, y = y, group = id))
p + geom_polygon()
p + geom_polygon(colour = "white", fill = "blue")
p + geom_polygon(colour = "white", fill = "blue", size = 1.6)
```

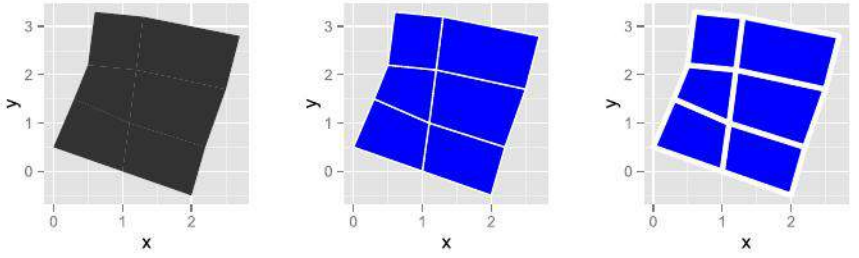


Рисунок 4.72. Примеры многоугольников, созданных при помощи функции `geom_polygon`. *Слева*: диаграмма, построенная с использованием автоматических настроек. *В центре*: цвет многоугольников и окаймляющих их линий изменены (аргументы `fill = "blue"` и `colour = "white"` соответственно). *Справа*: то же, что и в центре, но толщина окаймляющих линий увеличена (аргумент `size = 1.6`)

Код для рис. 4.73

```
# В этом примере отдельные части многоугольника, координаты
# которого хранятся в таблице positions (см. рис. 4.72),
# будут залиты цветом в соответствии со значениями третьей
# количественной переменной. Значения такой переменной обычно
# хранятся в отдельной таблице. Перед созданием графика
# обе таблицы должны быть объединены (команда merge() ниже).
values <- data.frame(id = ids,
                     value = c(4.5, 3.8, 3.2, 3, 2.5, 1.4))
polygon <- merge(values, positions, by = c("id"))
ggplot(data = polygon, aes(x = x, y = y, group = id)) +
       geom_polygon(aes(fill = value))
```

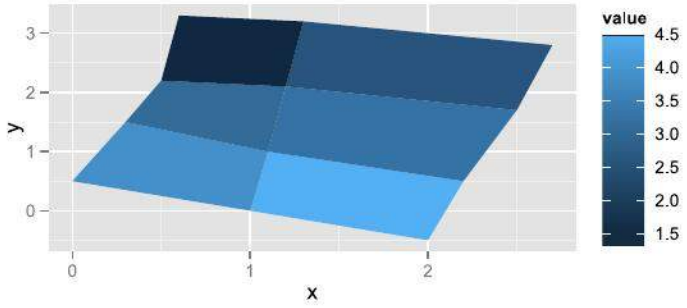


Рисунок 4.73. Многоугольник, отдельные сегменты которого залиты цветом в соответствии со значениями третьей (количественной) переменной

4.8.7 Площадь под кривой: `geom_area()`

Функция `geom_area()` позволяет создавать графики, на которых площадь, ограниченная замкнутой ломаной линией, закрашена определенным цветом. Такого рода графики можно рассматривать как аналог столбиковых диаграмм для непрерывных количественных переменных.

Аргументы

См. разд. 4.1.

Эстетические атрибуты

- `x*` и `y*` — векторы с X– и Y–координатами точек изображаемой на графике фигуры (или фигур).
- `alpha` — степень прозрачности цвета фигуры.
- `colour`, `linetype`, `size` — цвет, тип и толщина окаймляющей линии.

Примеры

Код для рис. 4.74

```
# Расчет медианных значений интенсивности инвазии
# (с сохранением результатов в отдельной таблице):
library(doBy)
medInfection <- summaryBy(
          Infection ~ Site + Day, FUN = median,
          data = subset(dreissena, Lake == "Naroch"))
p <- ggplot(data = subset(medInfection, Site == "S9"),
            aes(x = Day, y = Infection.median))
p + geom_area()
p + geom_area(fill = "skyblue")
p + geom_area(colour = 4, fill = "white",
              alpha = 0.6, linetype = 2, size = 2)
```

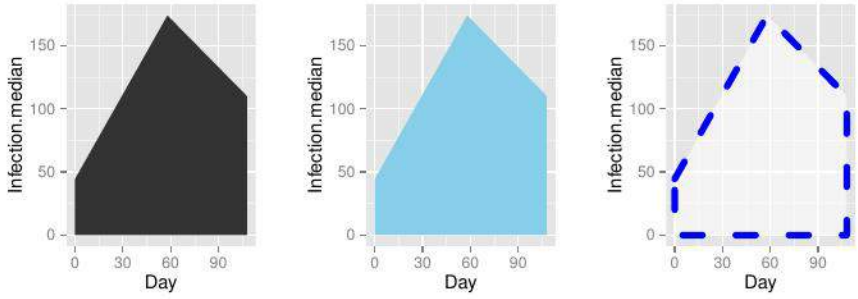


Рисунок 4.74. *Слева*: пример диаграммы типа «площадь под кривой», созданной с использованием автоматических настроек функции `geom_area()`. *В центре*: то же, но цвет фигуры изменен. *Слева*: то же, но изменены цвет фигуры, его прозрачность, а также ширина, цвет и тип окаймляющей линии

Код для рис. 4.75

```
p <- ggplot(data = medInfection,
            aes(x = Day, y = Infection.median))
p + geom_area(aes(fill = Site))
```

4.8.8 Текстовые аннотации: `geom_text()`

Часто возникает необходимость добавить к графику текстовые подписи, позволяющие, например, идентифицировать отдельные наблюдения или дать пояснения, облегчающие понимание графика. Для добавления такого рода аннотаций служит функция `geom_text()`.

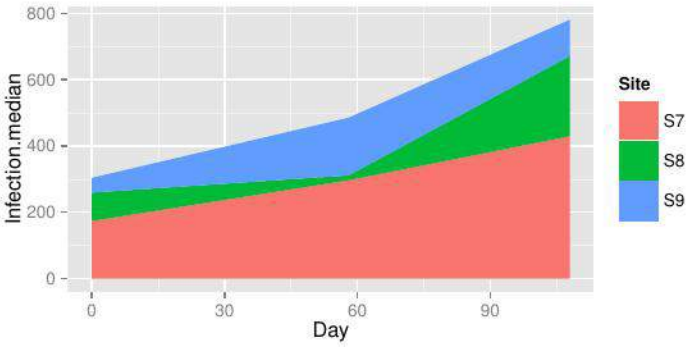


Рисунок 4.75. Пример применения диаграммы типа «площадь под кривой» для сгруппированных данных

Аргументы

- `parse` — логический аргумент, включающий (`parse = TRUE`) или отключающий (`parse = FALSE`, значение по умолчанию) парсинг текстовых значений в математические выражения (подобно тому, как это происходит при вызове базовой R–функции `plotmath()`; подробнее см. `?plotmath` и примеры ниже).

- другие аргументы — см. разд. 4.1.

Эстетические атрибуты

- `label`* — переменная, содержащая текстовые аннотации. Это может быть отдельный вектор, не входящий в состав таблицы с данными.

- `x`* и `y`* — координаты для размещения подписей.

- `colour` и `alpha` — цвет текста и степень его прозрачности соответственно.

- `angle` — угол поворота текста.

- `family` и `fontface` — семейство[18] и внешний вид шрифта (`fontface = "plain"` — обычный, `fontface = "bold"` — полужирный, `fontface = "italic"` — обычный наклонный, `fontface = "bold.italic"` — полужирный наклонный).

- `hjust` и `vjust` — параметры, позволяющие скорректировать горизонтальное и вертикальное положение текста соответственно (см. примеры ниже).

[18] Подробные примеры см. по ссылке `http://bit.ly/2ceE16C`. Можно также использовать дополнительные шрифты из пакета `extrafont`: `http://bit.ly/2bZ2XN8`.

Примеры

Код для рис. 4.76

```
library(dplyr) # для group_by(), summarise() и %>%
by.lake <- dreissena %>% group_by(Lake) %>%
    summarise(Length = round(median(Length)),
              Infection = round(median(Infection)))
ggplot(data = by.lake, aes(Length, Infection)) +
    geom_point(colour = "blue", size = 4) +
    geom_text(aes(label = Lake)) +
    xlim(11, 20) + ylim(150, 750)
ggplot(data = by.lake, aes(Length, Infection)) +
    geom_point(colour = "blue", size = 4) +
    geom_text(aes(label = Lake), hjust = 1.2) +
    xlim(11, 20) + ylim(150, 750)
ggplot(data = by.lake, aes(Length, Infection)) +
    geom_point(colour = "blue", size = 4) +
    geom_text(aes(label = Lake), hjust = 1.2,
              vjust = -1.2) + xlim(11, 20) + ylim(150, 750)
```

Код для рис. 4.77

```
# Объект by.lake определен как на рис. 4.76
ggplot(data = by.lake, aes(Length, Infection)) +
        geom_point(colour = "blue", size = 4) +
        geom_text(aes(label = Lake),
                  hjust = 1.2, colour = "red") +
        xlim(11, 20) + ylim(120, 750)
ggplot(data = by.lake, aes(Length, Infection)) +
        geom_point(colour = "blue", size = 4) +
        geom_text(aes(label = Lake), hjust = 1.2,
                  colour = "red", angle = 15) +
        xlim(11, 20) + ylim(120, 750)
```

Код для рис. 4.78

```
# Объект by.lake определен как на рис. 4.76
ggplot(data = by.lake, aes(Length, Infection)) +
    geom_point(colour = "blue", size = 4) +
    geom_text(aes(label = Lake), hjust = 1.2,
          family = "mono") + xlim(11, 20) + ylim(120, 750)
ggplot(data = by.lake, aes(Length, Infection)) +
    geom_point(colour = "blue", size = 4) +
    geom_text(aes(label = Lake, hjust = 1.2,
    fontface = c("plain", "bold",
                 "italic")[as.numeric(Lake)])) +
    xlim(11, 20) + ylim(120, 750)
```

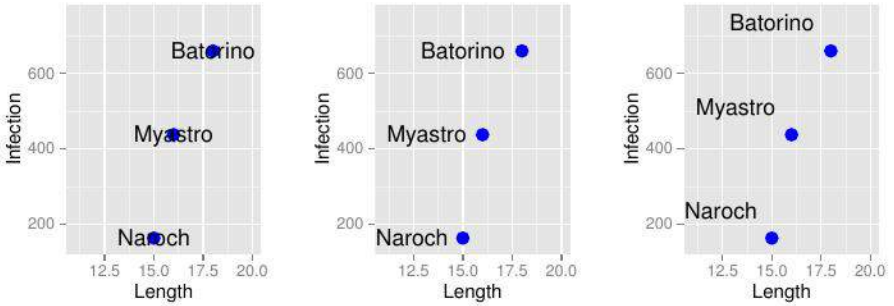


Рисунок 4.76. Пример добавления к графику текстовых подписей для идентификации отдельных наблюдений. *Слева*: подписи добавлены с использованием настроек, заданных по умолчанию. *В центре*: подписи сдвинуты влево (аргумент `hdjust = 1.2`). *Справа*: подписи сдвинуты влево и вверх (аргументы `hdjust = 1.2` и `vdjust = -1.2`)

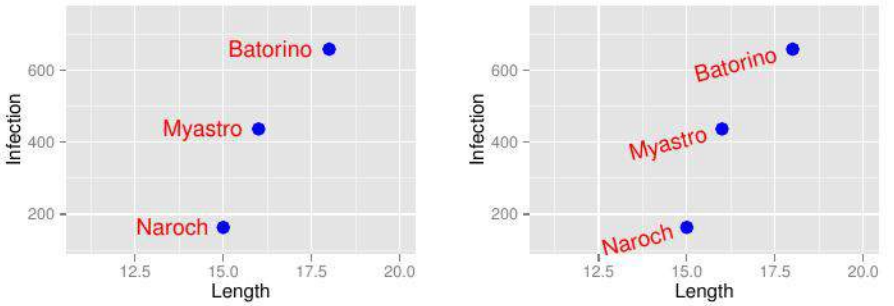


Рисунок 4.77. *Слева*: цвет шрифта изменен на красный (аргумент `colour = "red"`). *Справа*: изменен угол расположения подписей (аргумент `angle = 15`)

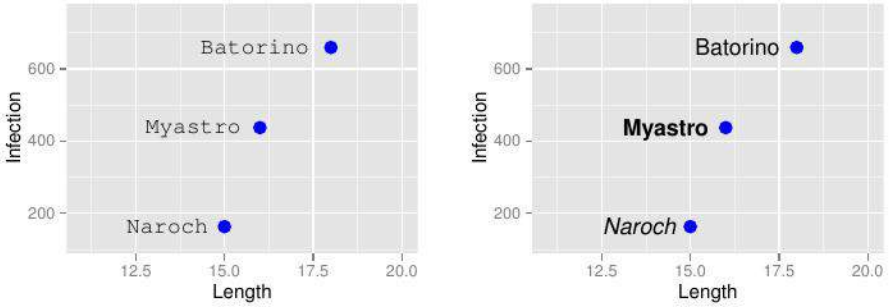


Рисунок 4.78. *Слева*: шрифт изменен на моноширинный (аргумент `fontfamily = "mono"`). *Справа*: оформление шрифта задано в соответствии с уровнями фактора `Lake` (см. код)

Код для рис. 4.79

```
# Объект by.lake определен как на рис. 4.76
ggplot(data = by.lake, aes(Length, Infection)) +
        geom_point(colour = "blue", size = 4) +
        geom_text(aes(label = Lake, size = Infection),
                  hjust = 1.2) +
        xlim(11, 20) + ylim(120, 750)
ggplot(data = by.lake, aes(Length, Infection)) +
        geom_point(colour = "blue", size = 4) +
        geom_text(data = NULL, label = "Some text",
        x = 16, y = 700) + xlim(11, 20) + ylim(120, 750)
```

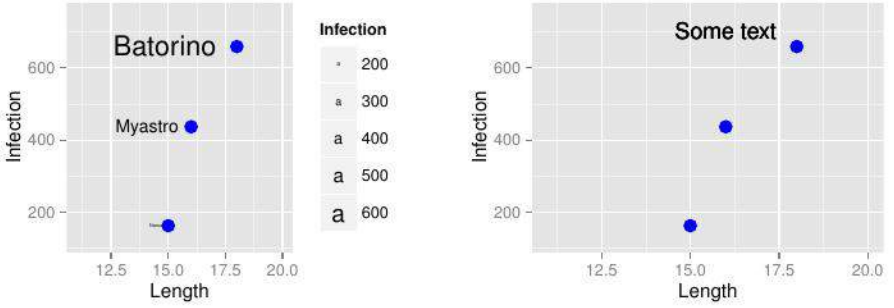


Рисунок 4.79. *Слева*: размер шрифта задан в соответствии со значениями переменной `Infection` (аргумент `size = Infection`). *Справа*: к графику добавлена произвольная подпись (`label = "Some text"`), центр которой размещен в точке координатами (16, 700) (`x = 16` и `y = 700`)

Код для рис. 4.80

```
# Объект by.lake определен как на рис. 4.76
ggplot(data = by.lake, aes(Length, Infection)) +
        geom_point(colour = "blue", size = 4) +
        geom_text(aes(label =
        paste("over(", Infection, ", host)")),
                  parse = TRUE, hjust = 1.3) +
        xlim(11, 20) + ylim(120, 750)
ggplot(data = by.lake, aes(Length, Infection)) +
        geom_point(colour = "blue", size = 4) +
        geom_text(aes(label = paste("rho[est.]==",
             round(cor(Length, Infection), 2), sep = "")),
             size = 10, parse = TRUE, x = 16.5, y = 600)
```

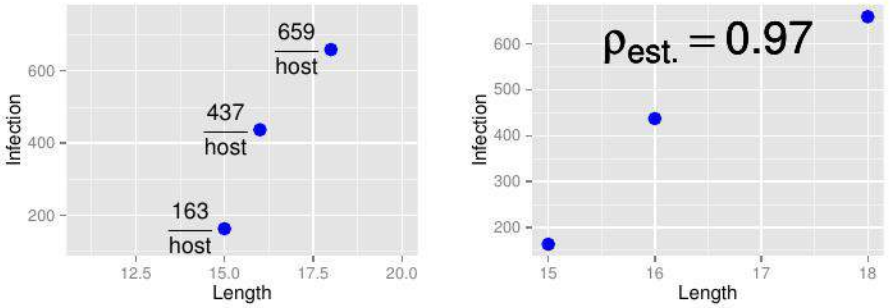


Рисунок 4.80. Примеры добавления к графику аннотаций, представляющих собой математические выражения. С особенностями парсинга таких выражений в R можно ознакомиться в справочных файлах, доступных по командам `?plotmath` и `?expression`

4.9 Географические карты: `geom_map()`

Функция `geom_map()` представляет собой модификацию рассмотренной ранее функции `geom_polygon()` (см. подразд. 4.8.6) и предназначена для создания географических карт. В частности, в этом разделе мы рассмотрим, как при помощи `geom_map()` можно строить *картограммы* — картографические изображения, на которых отдельные территориальные единицы залиты цветом разной интенсивности в соответствии со значениями той или иной количественной переменной (например, плотность населения, уровень дохода, процент проголосовавших за определенного кандидата на выборах и т. п.).

Аргументы

- `map` — таблица с географическими координатами, содержащая столбцы с именами `x` (или `long`), `y` (или `lat`) и `region` (или `id`).
- другие аргументы — см. разд. 4.1.

Эстетические атрибуты

- `map_id`* — вектор с идентификаторами территориальных единиц.
- `fill` — цвет заливки территориальных единиц.
- `alpha` — степень прозрачности цвета заливки.
- `colour`, `linetype`, `size` — цвет, тип и толщина контурных линий.

Рассмотрим создание картограмм на примере Республики Беларусь. Прежде всего нам потребуются географические координаты контурных линий, обозначающих границы страны и ее отдельных территориально–административных единиц (например, областей или районов). Подобные данные можно найти в разных источниках и в разных форматах, однако в нашем случае удобнее всего будет работать с т. н. «шейп–файлами».

Шейп–файл (англ. *shapefiile*) представляет собой распространенный векторный формат хранения информации о геометрическом положении и некоторых дополнительных атрибутах географических объектов. Несмотря на свое название, в действительности шейп–файл — это набор из нескольких файлов с одинаковым именем, но разными расширениями. Обязательными при этом являются файлы с расширениями `.shp`, `.dbf` и `.shx`. Подробную информацию о содержимом этих файлов можно найти в Википедии, а также в официальном техническом описании формата[19]. Для наших целей достаточным будет представление о шейп–файле как об источнике данных, по которым можно воспроизвести контурную карту определенной территории.

Шейп–файлы практически всех стран мира можно бесплатно[20] загрузить с сайта *Global Administrative Areas* (GADM)[21] — Глобальной базы данных административных областей. В этой базе данных доступны шейп–файлы для трех разных уровней территориального деления (страна, область, район). Имеется возможность загружать соответствующие файлы непосредственно в среду R. Так, в случае с Республикой Беларусь для загрузки данных на уровне областей и районов достаточно выполнить следующую команду[22]:

```
library(sp)
gadm <- readRDS(gzcon(url(
     "http://biogeo.ucdavis.edu/data/gadm2.8/rds/BLR_adm2.rds")))
```

В результате выполнения этой команды в рабочей среде R появится объект с именем `gadm`. С точки зрения своей внутренней организации и представления в R, `gadm` принадлежит к т. н. объектам класса `S4`[23]. Вкратце можно сказать, что такие объекты имеют сложную структуру, напоминающую структуру обычных списков R. Отдельные компоненты S4–объектов называют *слотами* (англ. *slots*). Слоты могут содержать сложные объекты любого другого класса. В отличие от обычных списков, обращение к слотам S4–объектов происходит при помощи оператора `@`, а не `$`. Кроме того, к слотам можно обращаться только по их именам, тогда как отдельные элементы списков доступны для индексирования также по их порядковым номерам. Имена слотов, входящих в состав `gadm`, легко выяснить при помощи базовой R–функции `slotNames()`:

```
slotNames(gadm)
[1] "data"        "polygons"    "plotOrder"
[4] "bbox"        "proj4string"
```

Наибольший интерес для нас представляют слоты `"data"` и `"polygons"`. Первый из них содержит обычную таблицу данных, структуру которой можно выяснить при помощи команды `str(gadm@data)`.

[19] http://arcg.is/1o0k7Lr.

[20] Разрешается только некоммерческое использование соответствующих данных.

[21] http://gadm.org.

[22] Для корректного выполнения этой команды необходимо предварительно установить пакет `sp`, который содержит классы и методы для работы с пространственными данными.

[23] Подробнее см. http://adv-r.had.co.nz/S4.html.

В свою очередь, слот "polygons" содержит список с координатами точек, определяющих контуры соответствующих территориальных единиц (в формате системы координат WGS84[24]), а также некоторые другие атрибуты (выполните команду str(gadm@polygons)).

Для корректного выполнения описанных ниже команд необходимо установить несколько дополнительных пакетов, а именно: broom, maps, maptools, rgeos, mapproj и scales.

Поскольку gadm является объектом класса S4, а функции пакета ggplot2 принимают только таблицы данных, потребуется преобразовать gadm в такую таблицу. Для этого следует воспользоваться функцией tidy() из пакета broom. На эту функцию подается объект gadm и указывается переменная, содержащая идентификаторы подлежащих отображению на карте территориально–административных единиц. Так, в случае с районами получаем:

```
library(broom) # для функции tidy()
library(maps); library(rgeos); library(maptools)
# При помощи аргумента region со значением "NAME_2"
# мы указываем переменную, в которой хранятся
# идентификаторы административных единиц - районов:
counties <- tidy(gadm, region = "NAME_2")

# Просмотр структуры объекта counties:
str(counties)
'data.frame':        42659 obs. of  7 variables:
 $ long : num  29.1 29.1 29.1 29.1 29.1 ...
 $ lat  : num  52.9 52.9 52.9 52.9 52.8 ...
 $ order: int  1 2 3 4 5 6 7 8 9 10 ...
 $ hole : logi  FALSE FALSE FALSE FALSE ...
 $ piece: Factor w/ 1 level "1": 1 1 1  ...
 $ group: Factor w/ 118 levels "Aktsyabar.1",...
 $ id   : chr  "Aktsyabar" "Aktsyabar" ...
```

Теперь мы можем построить прототип нашей будущей картограммы при помощи следующей команды (рис. 4.81):

Код для рис. 4.81

```
library(mapproj)
ggplot() + geom_map(data = counties, aes(map_id = id),
                    map = counties, color = "gray70") +
           expand_limits(x = counties$long,
                         y = counties$lat) +
           coord_map("polyconic")
```

Некоторые части последней команды требуют пояснения:

- geom_map(): эта функция создает слой с картой, внешний вид которой задается при помощи описанных выше аргументов. Так, на аргумент data была подана таблица с подлежащими изображению данными (на этом этапе построения картограммы таких данных у нас

[24] Подробнее см. http://bit.ly/2c2ORJv.

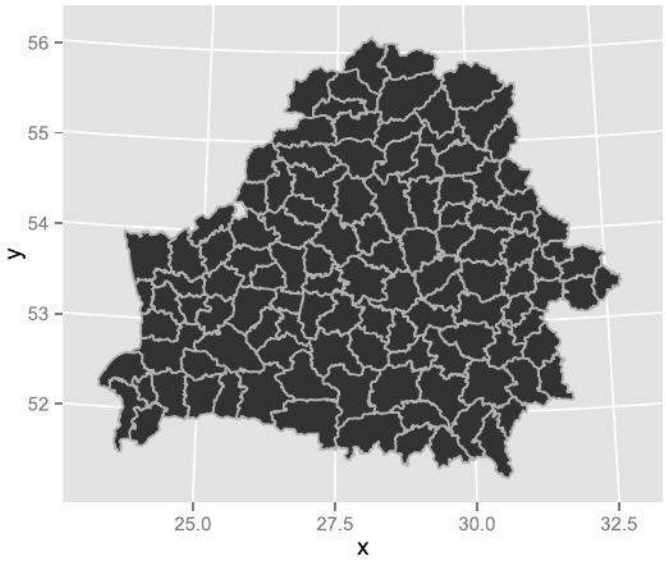


Рисунок 4.81. Контурная карта Беларуси с указанием границ районов

пока нет, в связи с чем в качестве временной «заглушки» использована таблица `counties`). Аргумент `aes()` служит для указания переменной, которая содержит идентификаторы административно–территориальных единиц (районов). Наконец, на аргумент `map` была подана таблица, содержащая координаты соответствующих полигонов;

- `expand_limits()`: в отличие от других функций `ggplot2`, которые предназначены для создания новых слоев с графическими элементами, функция `geom_map()` «не умеет» самостоятельно определять оптимальные пределы значений X– и Y–координат графика. Сделать это ей помогает функция `expand_limits()`;

- `coord_map()`: определяет тип картографической проекции. В приведенном примере использован тип `"polyconic"` — поликоническая проекция. Доступны также многие другие проекции, реализованные в пакете `mapproj` (подробнее см. справочный файл, доступный по команде `?mapproject`).

Представим теперь, что для каждого района Беларуси у нас имеются данные по какой–либо количественной характеристике. Вместо отображения на карте настоящих данных мы воспользуемся встроенным в `R` генератором псевдослучайных чисел и создадим переменную `Value`, содержащую нормально распределенные значения. Хотя случайно сгенерированные данные используются здесь лишь для иллюстрации принципа, при работе с настоящими данными последовательность выполняемых шагов будет такой же. Переменную `Value` мы добавим в новую таблицу данных, за основу которой возьмем содержимое слота `data` объекта `gadm`:

```
set.seed(1234) # для воспроизводимости результата
fake_data <- gadm@data
fake_data$Value <- rnorm(nrow(fake_data))
```

Наконец, изобразим данные из столбца `Value` таблицы `fake_data` на картограмме (рис. 4.82):

Код для рис. 4.82

```
ggplot() + geom_map(data = fake_data,
                    aes(map_id = NAME_2, fill = Value),
                    map = counties) +
        expand_limits(x = counties$long, y = counties$lat) +
        coord_map("polyconic")
```

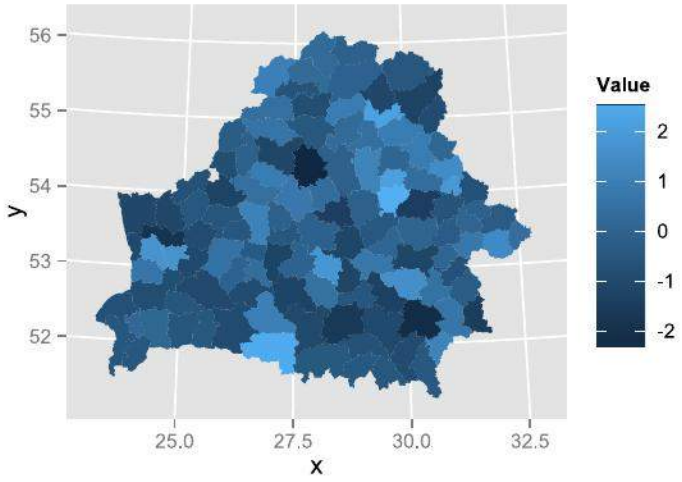


Рисунок 4.82. Картограмма Беларуси, изображающая случайно сгенерированные значения количественной переменной для каждого района. Цветовая шкала выбрана автоматически

Цветовая шкала на рис. 4.82 была выбрана программой автоматически, но ничто не мешает нам ее изменить. В рассматриваемом примере среднее значения переменной `Value` равно 0. Имеет смысл подчеркнуть это, воспользовавшись, например, белым цветом для отображения нулевых значений. Для значений же, отклоняющихся от 0 в отрицательную или в положительную сторону, можно использовать оттенки каких–либо других цветов (например, синего и красного соответственно). Для создания подобных расходящихся (англ. *diverging*) цветовых палитр в пакете `ggplot2` служит функция `scale_fill_gradient2()`, которая работает примерно следующим образом (рис. 4.83)[25]:

[25] Подробно способы настройки цветовых шкал рассмотрены в разд. 5.4.

Код для рис. 4.83

```
library(scales) # для функции muted() (см. ниже)
ggplot() + geom_map(data = fake_data,
                    aes(map_id = NAME_2, fill = Value),
                    colour = "gray",
                    map = counties) +
    expand_limits(x = counties$long,
                  y = counties$lat) +
    scale_fill_gradient2(low = muted("blue"),
                         midpoint = 0,
                         mid = "white",
                         high = muted("red"),
                         limits = c(min(fake_data$Value),
                         max(fake_data$Value))) +
    coord_map("polyconic")
```

При необходимости мы можем изменить и другие детали картограммы. Так, с помощью функции `theme()` («стиль», или «шаблон»[26]) можно отключить отображение серого фона, координатной сетки, а также осей и их названий (рис. 4.84).

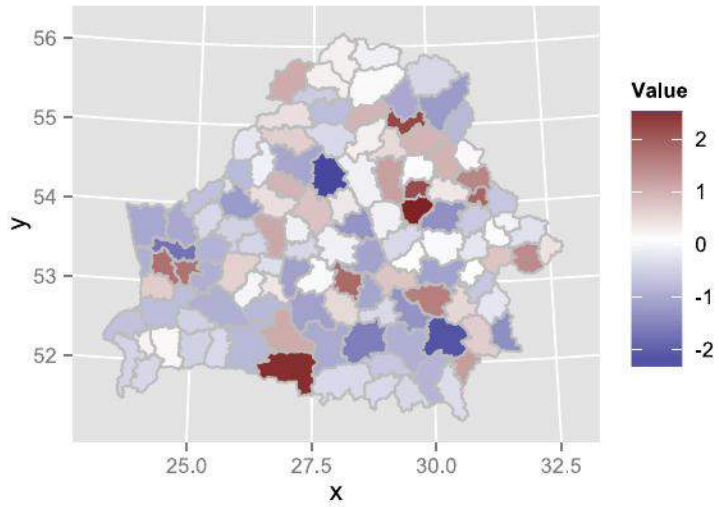


Рисунок 4.83. Пример добавления к картограмме пользовательской цветовой шкалы

[26] Подробно стили рассматриваются в разд. 8.1.

Код для рис. 4.84

```
ggplot() + geom_map(data = fake_data,
                    aes(map_id = NAME_2, fill = Value),
                    colour = "gray", map = counties) +
  expand_limits(x = counties$long, y = counties$lat) +
  scale_fill_gradient2(low = muted("blue"),
                       midpoint = 0, mid = "white",
                       high = muted("red"),
                       limits = c(min(fake_data$Value),
                                  max(fake_data$Value))) +
  coord_map("polyconic") +
  theme(axis.line = element_blank(),
    axis.text.x = element_blank(),
    axis.text.y = element_blank(),
    axis.ticks = element_blank(),
    axis.title.x = element_blank(),
    axis.title.y = element_blank(),
    plot.background = element_blank(),
    panel.grid.major = element_blank(),
    panel.grid.minor = element_blank(),
    panel.border = element_blank(),
    panel.background = element_blank() )
```

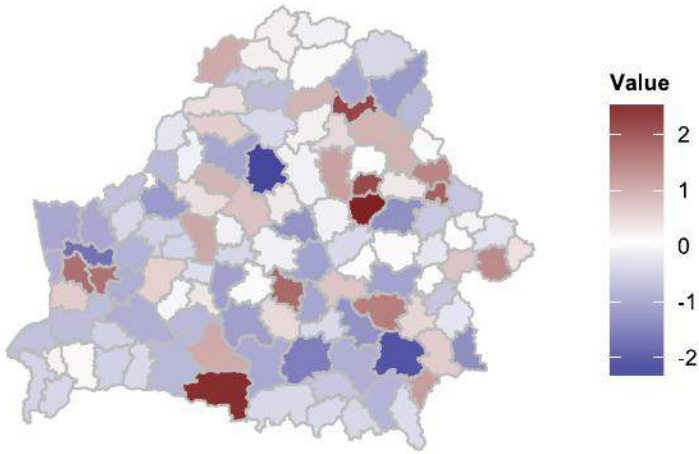


Рисунок 4.84. То же, что и на рис. 4.83, но без заднего фона, координатной сетки и осей

4.10 Добавление слоев при помощи функций семейства stat

Как было отмечено в разд. 3.7, в состав пакета ggplot2 входит целый ряд функций, осуществляющих определенные преобразования исходных данных. Все эти функции имеют в своем названии приставку "stat" (например, stat_bin(), stat_boxplot(), stat_ecdf() и т. д.). Каждому типу геометрических объектов соответствует свой заданный по умолчанию тип преобразования данных, и наоборот: каждому преобразованию соответствует свой объект. Как следствие тот же конечный результат на графике можно получить, используя как geom–, так и stat–функции. Например, идентичные диаграммы размахов можно построить следующими двумя способами:

```
p <- ggplot(data = dreissena, aes(Lake, Length))
p + geom_boxplot() # способ 1
p + stat_boxplot() # способ 2
```

Все stat–функции имеют ряд общих аргументов, рассмотренных ранее в разд. 4.1.

Кроме того, каждая stat–функция имеет свои специфичные аргументы, которые определяют ее поведение. В подавляющем большинстве случаев это те же аргументы, которые используются для настройки работы соответствующих geom–функций (например, аргумент kernel имеется как у stat_density(), так и у geom_density()). В связи с этим обстоятельством мы не будем рассматривать здесь аргументы и примеры использования всех функций статистических преобразований (см. предыдущие разделы этой главы и справочную документацию). Вместо этого мы сосредоточимся на одной очень полезной функции, которая не имеет своих заданных по умолчанию геометрических объектов, — stat_summary(). При помощи этой функции можно выполнить любые математические операции над исходными данными и изобразить результат с применением любого выбранного пользователем геометрического объекта[27]. Функция stat_summary() имеет следующие аргументы:

- fun.data — служит для указания имени функции, выполняющей желаемое преобразование исходных данных. Предполагается, что эта функция принимает данные в виде таблицы и возвращает результаты в таком же формате;
- fun.y — служит для указания имени функции, выполняющей преобразование исходных значений Y–переменной;
- fun.ymax и fun.ymin — используются для вычисления нижней и верхней границ заданного пользователем диапазона (например, минимум и максимум) по исходным значениям Y–переменной.

Предположим, что мы хотим построить одномерную диаграмму размахов (см. подразд. 2.3.2) и добавить к ней средние значения по каждой

[27] Имеется также аналог для работы с двухмерными распределениями — функция stat_summary2d().

группе, изображенные в виде точек определенного цвета и размера. В другом варианте этой диаграммы мы добавим к каждой группе также вертикальные отрезки, символизирующие 0.25 и 0.75 квантили распределения. Соответствующий код и результаты его выполнения приведены ниже (рис. 4.85).

Код для рис. 4.85

```
# Одномерная диаграмма рассеяния, на которой
# к X-координатам точек добавлен небольшой шум (geom_jitter):
p <- ggplot(data = subset(dreissena, Month == "May" &
            Lake == "Batorino"), aes(Site, Length)) +
     geom_jitter(position = position_jitter(width = 0.2),
                 alpha = 0.4)

# Добавим средние значения в виде крупных красных точек
(a <- p + stat_summary(fun.y = mean, geom = "point",
                       color = "red", size = 5))

# Добавим отрезки, символизирующие 0.25 и 0.75 квантили:
a + stat_summary(fun.y = mean,
      fun.ymin = function(x){quantile(x, p = 0.25)},
      fun.ymax = function(x){quantile(x, p = 0.75)},
      geom = "errorbar", color = "blue", width = 0.25)
```

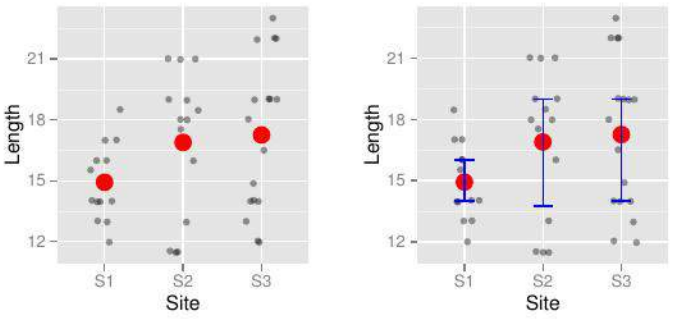


Рисунок 4.85. Примеры добавления к графику сводных статистических показателей при помощи функции `stat_summary()`

Некоторые функции, выполняющие расчет сводных показателей, требуют, чтобы данные подавались на них в виде таблиц. Для таких случаев у функции `stat_summary()` имеется аргумент `fun.data`. Продолжая пример с одномерной диаграммой рассеяния, добавим к каждой группе средние значения в виде точек и доверительные интервалы в виде вертикальных отрезков. Доверительные интервалы будут вычислены бутстреп–методом (англ. *bootstrap*) при помощи входящей в состав пакета `ggplot2` вспомогательной функции `mean_cl_boot()` (рис. 4.86):

Код для рис. 4.86

```
# Объект p определен как в коде для рис. 4.85
p + stat_summary(fun.data = mean_cl_boot,
                 color = "magenta", size = 1)
```

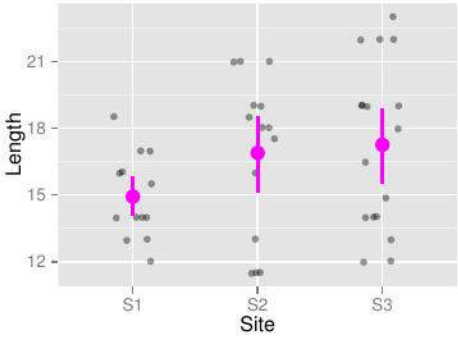


Рисунок 4.86. Пример добавления к графику сводных статистических показателей при помощи аргумента `fun.data` функции `stat_summary()`

Глава 5

Шкалы

Как вы уже могли убедиться, хорошо подобранные автоматические настройки `ggplot2` позволяют без особых усилий создавать эстетически привлекательные и легко интерпретируемые графики. Тем не менее в ряде случаев без доработки отдельных деталей графика не обойтись. В `ggplot2` для этого есть несколько способов. В частности, в этой главе мы познакомимся с таким важным понятием, как шкала, и рассмотрим способы изменения свойств разных шкал.

5.1 Шкалы и их основные типы

С технической точки зрения, *шкала* (англ. *scale*) *представляет собой функцию, которая выполняет отображение* (англ. *mapping*) *пространства данных на пространство эстетических атрибутов*. Говоря более простым языком, шкалы преобразовывают данные в то, что мы можем воспринять визуально, — координаты, размер, цвет, форму, тип линии и т. д. В `ggplot2` реализовано четыре типа шкал:

- *шкалы положения* (англ. *position scales*) — служат для создания осей графика и отображения непрерывных (включая время) и дискретных переменных на соответствующую систему координат;
- *цветовые шкалы* (англ. *colour scales*) — выполняют отображение данных на цветовое пространство;
- *пользовательские шкалы для качественных переменных* (англ. *manual scales*) — позволяют пользователю самостоятельно устанавливать соотношение между эстетическими атрибутами и значениями качественных переменных;
- *тождественные шкалы* (англ. *identity scales*) — позволяют обойти процедуру отображения в случаях, когда значения некоторой переменной в таблице с данными уже соответствуют значениям соответствующего эстетического атрибута. Например, если в таблице с данными имеется переменная `Colour` со значениями `"red"`, `"blue"`, `"green"`, то мы можем использовать эти значения непосредственно.

Как правило, процесс отображения данных на пространство эстетических атрибутов включает три стадии (Wickham, 1999[1]):

- *преобразование данных* (англ. *transformation*) — необязательная стадия, которая имеет место при работе с количественными переменными. Под преобразованием понимают применение к исходным значениям количественной переменной некоторой алгебраической функций (логарифма, квадратного корня, тангенса и т. п.);

- *обучение шкалы* (англ. *training*) — ключевая стадия, в ходе которой происходит распознавание диапазонов значений переменных, подлежащих изображению на графике. В большинстве случаев эта стадия сводится к простому определению минимального и максимального значений количественной переменной (возможно, преобразованной определенным образом на предыдущей стадии) или составлению списка уровней качественной переменной. В некоторых случаях, когда график содержит несколько слоев с разными наборами данных, стадия обучения шкалы усложняется за счет необходимости отобразить все эти наборы в пределах одной системы координат;

- *отображение* (англ. *mapping*) — стадия, в результате которой происходит проецирование данных на пространство эстетических атрибутов.

Отображение данных на пространство эстетических атрибутов одновременно приводит к созданию *ориентиров* (англ. *guides*) — элементов, помогающих нам правильно интерпретировать график. В случае с атрибутами положения, т. е. координатами, такими ориентирами являются *оси графика*. Во всех остальных случаях (форма, цвет, размер и т. п.) в качестве ориентира выступает *легенда графика*. Координатные оси и легенду следует рассматривать как функции, обратные шкале, поскольку с их помощью можно выполнить отображение пространства эстетических атрибутов на пространство исходных данных. В отличие от других графических систем `R`, `ggplot2` предоставляет пользователю относительно ограниченные возможности по непосредственному изменению осей и легенд, поскольку их свойства определяются автоматически в ходе описанных выше стадий отображения данных на пространство эстетических атрибутов. Следовательно, для изменения свойств осей и легенд пользователь должен самостоятельно изменить свойства соответствующих шкал. В следующих разделах будет показано, как именно это можно сделать.

В `ggplot2` имена функций, которые управляют свойствами шкал, имеют общий формат вида `"scale_xxx_yyy"`, где `xxx` — это название эстетического атрибута, а `yyy` — тип шкалы. Элементы названий некоторых часто используемых `scale`–функций представлены в табл. 5.1 и 5.2, а примеры таких функций приведены в табл. 5.3.

[1] Wickham H. (2009) ggplot2: Elegant graphics for data analysis. Springer.

Таблица 5.1. Названия эстетических атрибутов, входящие в имена наиболее часто используемых `scale`–функций

Эстетический атрибут (`xxx`)	Описание
`colour`	цвет символов и линий
`fill`	цвет заливки фигур
`linetype`	тип линий
`size` и `shape`	размер и форма символов
`x` и `y`	положение вдоль осей X и Y

Таблица 5.2. Названия шкал, входящие в имена некоторых часто используемых `scale`–функций

Шкала (`yyy`)	Описание
`hue`	цвета, равноудаленные друг от друга на цветовом колесе[2]
`gradient`	градиент плавно переходящих друг в друга цветов
`continuous`	шкалы для количественных переменных
`discrete`	шкалы для качественных переменных
`date` и `datetime`	шкалы для дат и времени
`manual`	пользовательские шкалы
`identity`	тождественные шкалы

Таблица 5.3. Основные функции, управляющие свойствами шкал в `ggplot2`. Звездочкой (*) отмечены функции, которые используются для переменных соответствующего типа по умолчанию

Эстетический атрибут	Качественные переменные	Количественные переменные

Цвет символов и линий	`scale_colour_brewer()`
	`scale_colour_grey()`
	`scale_colour_hue()*`
	`scale_colour_discrete()`
	`scale_colour_identity()`
	`scale_colour_manual()`
		`scale_colour_gradient()*`
		`scale_colour_gradient2()`
		`scale_colour_gradientn()`

Цвет заливки фигур	`scale_fill_brewer()`
	`scale_fill_grey()`
	`scale_fill_hue()*`
	`scale_fill_identity()`
	`scale_fill_manual()`
		`scale_fill_gradient()*`
		`scale_fill_gradient2()`
		`scale_fill_gradientn()`

Положение (X, Y)	`scale_x_discrete()*`
	`scale_y_discrete()*`
		`scale_x_continuous()*`
		`scale_y_continuous()*`
		`scale_x_date()`
		`scale_y_date()`

[2] Подробнее см., например, `http://bit.ly/2cJQSuN`.

Таблица 5.3: *продолжение*

Эстетический атрибут	Качественные переменные	Количественные переменные
		`scale_x_datetime()` `scale_y_datetime()`

Форма	`scale_shape_discrete()*`
	`scale_shape_identity()`
	`scale_shape_manual()`	

Тип линии	`scale_linetype_discrete()*`
	`scale_linetype_identity()`
	`scale_linetype_manual()`	

Размер	`scale_size_discrete()*`
	`scale_size_identity()`
	`scale_size_manual()`
		`scale_size_continuous()*`
		`scale_size_area()`

5.2 Аргументы, общие для всех scale–функций

У всех функций из табл. 5.3 есть несколько одинаковых аргументов:

- `name` — задает подпись оси или легенды. На этот аргумент можно подавать либо текстовый вектор с одним элементом (при необходимости используя `\n`, чтобы разбить текст на несколько строк), либо математическое выражение (см. справочный файл, доступный по команде `?plotmath`, а также примеры в подразд. 4.8.8). Поскольку изменение подписей осей и заголовка легенды является распространенной задачей, в `ggplot2` имеются специальные вспомогательные функции — `xlab()`, `ylab()` и `labs()`. Вот примеры использования этих функций (см. результат на рис. 5.1):

Код для рис. 5.1

```
( p <- ggplot(dreissena, aes(Lake, Length)) +
              geom_boxplot(aes(fill = Month)) )
p + xlab("Озеро") + ylab("Длина, мм") +
    scale_fill_discrete("Месяц")
# Команда, аналогичная предыдущей:
p + labs(x = "Озеро", y = "Длина, мм", fill = "Месяц")
```

- `limits` — задает диапазон значений соответствующей переменной. В случае с количественными переменными на этот аргумент подается числовой вектор из двух элементов, определяющих минимальное и максимальное значения. В случае же с качественными переменными на `limits` подается текстовый вектор с наименованиями всех уровней, *подлежащих отображению* на графике. Аргумент `limits`

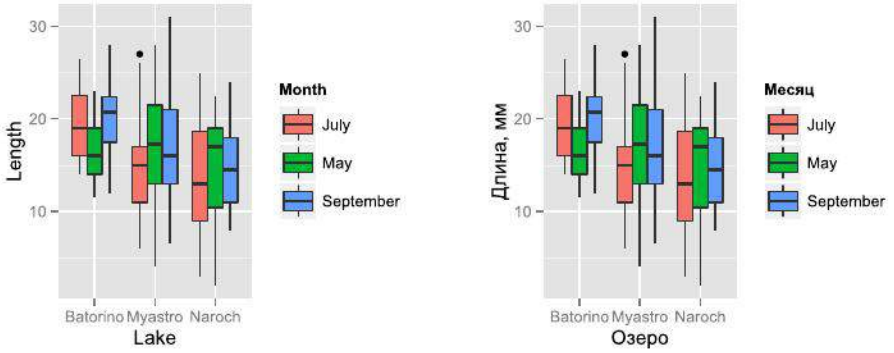


Рисунок 5.1. *Слева*: подписи осей и заголовок легенды заданы в соответствии с автоматическими настройками. *Справа*: пример изменения подписей осей (функции `xlab()`, `ylab()` или `labs()`) и заголовка легенды (функция `scale_fill_discrete()` или `labs()`)

бывает особенно полезным, когда необходимо скрыть некоторые наблюдения (это можно сделать, установив, например, более узкий, чем в исходных данных, диапазон отображаемых значений соответствующей переменной). Кроме того, использование одинаковых диапазонов значений координатных осей помогает легче сравнивать графики, на которых изображены разные группы данных;

- `breaks` — определяет интервалы, на которые будет разбита соответствующая ось графика, и какие интервалы значений количественной переменной будут представлены в легенде;
- `labels` — задает метки интервалов, на которые ось разбивается при помощи аргумента `breaks`. Очевидно, что использование `labels` имеет смысл только в связке с `breaks` (см. примеры использования этих аргументов ниже);
- `guide` — включает (`guide = "legend"`, значение по умолчанию) и отключает (`guide = "none"`) легенду графика.

Код для рис. 5.2

```
ggplot(subset(dreissena, Lake == "Naroch"),
       aes(Length, sqrt(Infection))) +
       geom_point(alpha = 0.5)
ggplot(subset(dreissena, Lake == "Naroch"),
       aes(Length, sqrt(Infection))) +
       geom_point(alpha = 0.5) +
       scale_x_continuous(breaks = c(3, 12, 22)) +
       scale_y_continuous(breaks = c(10, 45))
ggplot(subset(dreissena, Lake == "Naroch"),
       aes(Length, sqrt(Infection))) +
       geom_point(alpha = 0.5) +
```

```
        scale_x_continuous(breaks = c(3, 12, 22),
                           labels = c("L1", "L2", "L3")) +
        scale_y_continuous(breaks = c(10, 45),
                           labels = c("L4", "L5"))
```

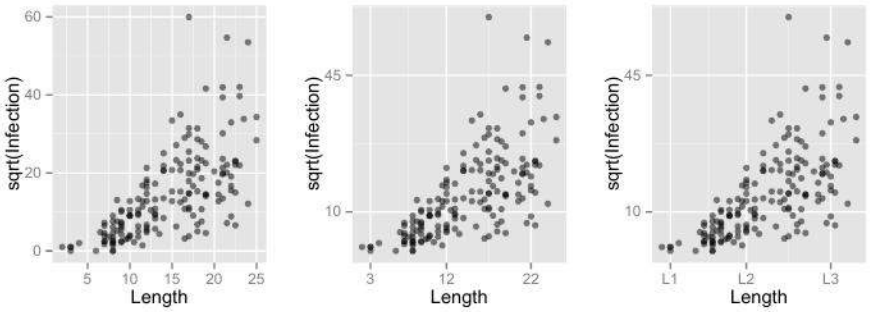


Рисунок 5.2. Пример использования аргументов `breaks` и `labels` при вызове функций `scale_x_continuous()` и `scale_y_continuous()`. *Слева*: график построен в соответствии с автоматическими настройками. *В центре*: интервалы, на которые разбиваются оси X и Y, изменены при помощи аргумента `breaks` (см. код). *Справа*: границы интервалов на оси X обозначены пользовательскими метками `"L1"` – `"L3"` (см. аргумент `labels`)

Помимо векторов с пользовательскими метками, аргумент `labels` может принимать также имена функций, которые выполняют форматирование текстовых выражений и чисел. В частности, речь идет о `abbreviate()` (базовая `R`–функция, формирующая аббревиатуры текстовых выражений) и некоторых функциях из пакета `scales`: `scientific()` (представляет числа в экспоненциальной записи), `comma()` (в числах ≥ 1000 отделяет каждые три знака запятыми, например 1,000), `percent()` (умножает число на 100 и добавляет знак процента), `dollar()` (округляет число до цента и добавляет знак доллара). На рис. 5.3 приведены примеры использования функций `abbreviate()` и `scientific()`.

Код для рис. 5.3

```
ggplot(dreissena, aes(Lake, Length)) +
       geom_boxplot() +
       scale_x_discrete(labels = abbreviate)
ggplot(dreissena, aes(Lake, Length)) +
       geom_boxplot() +
       scale_y_discrete(breaks = c(5, 15, 25),
                        labels = scientific)
```

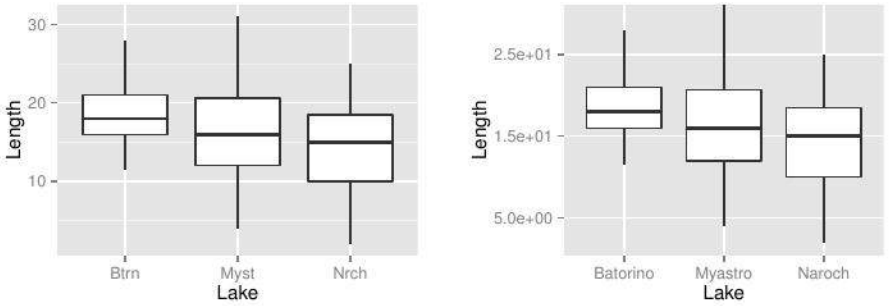


Рисунок 5.3. Примеры форматирования меток делений координатных осей. *Слева*: метки оси X сокращены при помощи функции `abbreviate()`. *Справа*: числа на оси Y представлены в экспоненциальной записи с помощью функции `scientific()`

5.3 Шкалы положения

Любой график должен иметь две шкалы положения, определяющие горизонтальное (ось X) и вертикальное (ось Y) положения геометрических объектов. В `ggplot2` такие шкалы реализованы как для количественных (включая время), так и для качественных переменных[3].

5.3.1 Шкалы положения для количественных переменных

Наиболее обычными являются шкалы, которые выполняют отображение значений количественных переменных на пространство координатных осей X и Y. Свойствами этих шкал управляют функции `scale_x_continuous()` и `scale_y_continuous()` соответственно. Поскольку в ходе анализа данных часто возникает необходимость преобразовать исходные переменные определенным образом, обе функции имеют аргумент `trans` для указания типа необходимого преобразования. В `ggplot2` реализован целый ряд таких преобразований (табл. 5.4). Некоторые из перечисленных в табл. 5.4 преобразований удобно реализованы в виде отдельных функций (табл. 5.5).

Безусловно, пользователь может самостоятельно преобразовать исходные переменные перед построением графика — применение `scale`–функций для этого не обязательно. Например, вместо использования `scale_x_sqrt()` мы можем изобразить значения `sqrt(x)` — с точки зрения расположения точек на графике оба способа приведут к идентичным результатам. Разной, однако, окажется разметка оси `x` — в результате применения `scale_x_sqrt()` разметка будет выполнена в пространстве исходных значений `x`, тогда как предварительное извлечение квадратного корня из `x` приведет к разметке в пространстве этих преобразованных значений (рис. 5.4).

[3] Следует отметить, что помимо собственно качественных переменных («факторов» в терминах `R`) здесь имеются в виду также текстовые и логические переменные, которые можно преобразовать в факторы при помощи базовой функции `as.factor()`.

Таблица 5.4. Значения аргумента `trans` и соответствующие им функции математического преобразованиям количественных переменных

Значение `trans`	Функция, $z = f(x)$	Обратная функция, $x = f^{-1}(z)$
`asn`	$\tanh^{-1}(x)$	$\tanh(z)$
`exp`	e^x	$\log(z)$
`identity`	x	y
`log`	$\log(x)$	$\exp(z)$
`log10`	$\log_{10}(x)$	10^z
`log2`	$\log_2(x)$	2^z
`logit`	$\log(\frac{x}{1-x})$	$\frac{1}{1+e^z}$
`pow10`	10^x	$\log_{10}(z)$
`probit`	$\Phi(x)$	$\Phi^{-1}(z)$
`recip`	x^{-1}	z^{-1}
`reverse`	$-x$	$-z$
`sqrt`	$x^{1/2}$	z^2

Таблица 5.5. Функции `ggplot2`, реализующие некоторые распространенные типы преобразования количественных переменных. В последнем столбце таблицы приведены примеры команд с использованием функции `scale_x_continuous()`, которые позволяют получить идентичный результат

Преобразование	Функция	Аналог
$\log_{10}(x)$	`scale_x_log10()`	`scale_x_continuous(trans = "log10")`
$\log_{10}(y)$	`scale_y_log10()`	`scale_y_continuous(trans = "log10")`
$x^{1/2}$	`scale_x_sqrt()`	`scale_x_continuous(trans = "sqrt")`
$y^{1/2}$	`scale_y_sqrt()`	`scale_y_continuous(trans = "sqrt")`
$-x$	`scale_x_reverse()`	`scale_x_continuous(trans = "reverse")`
$-y$	`scale_y_reverse()`	`scale_y_continuous(trans = "reverse")`

Код для рис. 5.4

```
ggplot(subset(dreissena, Lake == "Naroch"),
       aes(sqrt(Length), Infection)) +
       geom_point(alpha = 0.5)
ggplot(subset(dreissena, Lake == "Naroch"),
       aes(Length, Infection)) + geom_point(alpha = 0.5) +
       scale_x_continuous(trans = "sqrt")
```

В результате математических преобразований исходных переменных можно столкнуться с ситуациями, когда разметка осей выглядит не совсем «привлекательно». В качестве примера предположим, что перед нами стоит задача изобразить гистограмму log–значений переменной `Infection` из таблицы `dreissena`. Как мы уже знаем, такое преобразование можно выполнить при помощи команды `scale_x_continuous(trans = "log")`. Слева на рис. 5.5 приведен результат выполнения соответствующего кода. Проблема здесь заключается в том, что автоматически выбранные программой деления оси X не соответствуют кратным значениям переменной `Infection`.

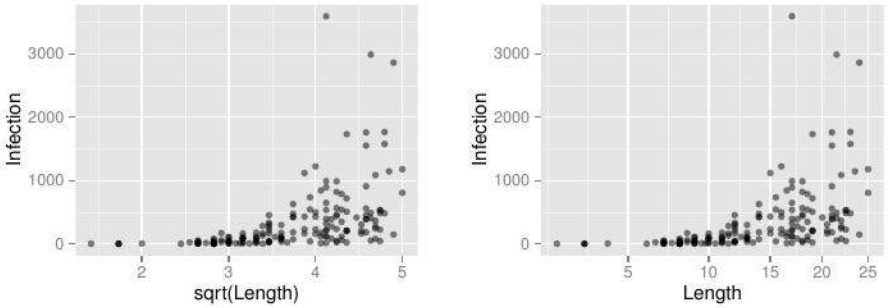


Рисунок 5.4. *Слева*: преобразование переменной `Length` выполнено с использованием команды `sqrt(Length)`. *Справа*: преобразование `Length` выполнено с использованием команды `scale_x_continuous(trans = "sqrt")`

Код для рис. 5.5

```
library(scales)
ggplot(subset(dreissena, Lake == "Naroch"), aes(Infection)) +
       geom_histogram() +
       scale_x_continuous(trans = "log")
ggplot(subset(dreissena, Lake == "Naroch"), aes(Infection)) +
       geom_histogram() +
       scale_x_continuous(trans = "log",
       breaks = trans_breaks("log", "exp"),
       labels = trans_format("log", math_format(e^.x)))
```

Для исправления этой ситуации можно разбить ось X на равномерные интервалы при помощи функции `trans_breaks()` из упомянутого выше пакета `scales`. У этой функции есть два основных аргумента: `trans` — математическая функция преобразования данных и `inv` — функция обратного преобразования (см. примеры в табл. 5.4). Кроме того, с помощью аргумента `n` можно задать желаемое (целое) число делений оси. Следует отметить, что `trans_breaks()`, как и все другие функции из пакета `scales`, является функцией *второго порядка* (англ. *second order function*).

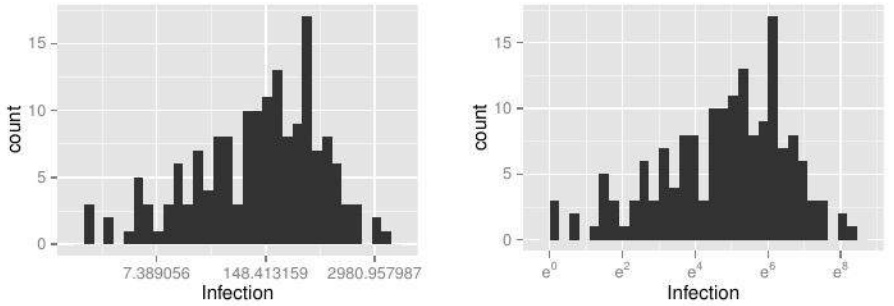


Рисунок 5.5. *Слева*: после `log`–преобразования разметка оси X выполнена в соответствии с автоматическими настройками. *Справа*: разметка выполнена с использованием функций `trans_breaks()`, `trans_format()` и `math_format()` из пакета `scales` (объяснения в тексте)

Это значит, что она возвращает другую функцию, которая далее, собственно, и выполняет разбиение некоторого вектора числовых значений на равномерные интервалы. Приведенный ниже код демонстрирует эту особенность:

```
temp_function <- trans_breaks("log", "exp")

# Объект temp_function является функцией:
temp_function

function (x)
{
    inv(pretty(trans(x), n, ...))
}
<environment: 0x0000000010018008>

# Подадим на temp_function числовой вектор:
temp_function(subset(dreissena, Lake == "Naroch")$Infection)

[1]     1.000000     7.389056    54.598150   403.428793
[5]  2980.957987 22026.465795
```

Метки делений оси, полученные при помощи функции `trans_breaks()`, являются результатом возведения основания натурального логарифма e в степени 0, 2, 4, 6, 8 и 10. В этом легко убедиться:

```
exp(c(0, 2, 4, 6, 8, 10))
[1]     1.000000     7.389056    54.598150   403.428793
[5]  2980.957987 22026.465795
```

Другими словами, деления оси соответствуют $e^0, e^2, \ldots, e^{10}$. Использование меток делений в такой сокращенной форме устранило бы указанную выше проблему с осью X гистограммы, приведенной на рис. 5.5 слева.

Теоретически мы могли бы воспользоваться встроенными в R возможностями парсинга математических выражений (см. `?plotmath`) и самостоятельно подать вектор с необходимыми метками на аргумент `labels` функции `scale_x_countinuous()` (см. разд. 5.2). Однако более удобный способ для выполнения этой задачи состоит в использовании еще одной функции из пакета `scales` — `trans_format()`, которая выполняет заданный пользователем тип форматирования меток *после* преобразования значений некоторой переменной. Эта функция принимает два аргумента: уже известный нам `trans` (некоторая математическая функция) и `format` (тип форматирования). На аргумент `format` подается одна из функций форматирования, входящих в состав пакета `scales`: `scientific_format()`, `math_format()`, `date_format()`, `percent_format()` или `dollar_format()`. Для рассматриваемого примера подойдет `math_format()` — функция, выполняющая парсинг любого математического выражения в символьное выражение. Для получения меток в виде «e^x» синтаксис этой функции будет выглядеть следующим образом: `math_format("log", e^.x)` (обратите внимание на наличие обязательной точки перед `x`). Таким образом, для нашей гистограммы вызов функции `scale_x_countinuous()` должен выглядеть так:

```
scale_x_continuous(trans = "log",
            breaks = trans_breaks("log", "exp"),
            labels = trans_format("log", math_format(e^.x)))
```

Результат выполнения данной команды приведен справа на рис. 5.5[4].

Завершая рассмотрение шкал для количественных переменных, отметим, что по умолчанию границы осей X и Y простираются несколько дальше, чем диапазон значений соответствующих переменных (исходных или преобразованных), — благодаря этому предотвращается «наползание» отдельных наблюдений на оси. Размер этих дополнительных полей контролируется аргументом `expand`, который принимает вектор из двух чисел, задающих величину мультипликативного и аддитивного расширений исходного диапазона шкалы соответственно. Дополнительные поля можно отменить при помощи `expand = c(0, 0)` (рис. 5.6).

Код для рис. 5.6

```
ggplot(subset(dreissena, Lake == "Naroch"),
   aes(Length, sqrt(Infection))) + geom_point(alpha = 0.5) +
   scale_x_continuous(expand = c(0, 0))
ggplot(subset(dreissena, Lake == "Naroch"),
   aes(Length, sqrt(Infection))) + geom_point(alpha = 0.5) +
   scale_x_continuous(expand = c(0.5, 0))
ggplot(subset(dreissena, Lake == "Naroch"),
   aes(Length, sqrt(Infection))) + geom_point(alpha = 0.5) +
   scale_x_continuous(expand = c(0, 8))
```

[4] Дополнительные примеры по работе с метками делений осей можно найти в разд. 7 документа, доступного по адресу `http://bit.ly/1v7B9as`.

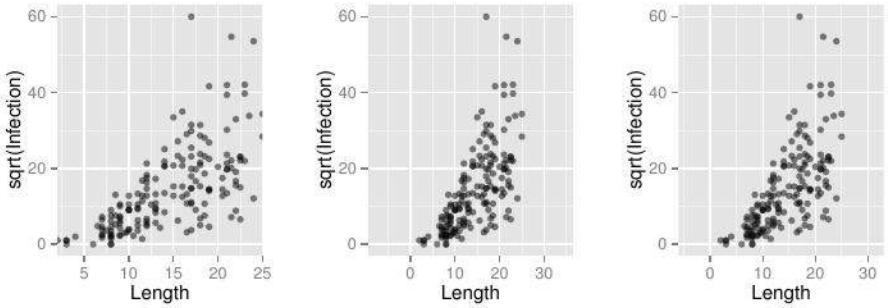


Рисунок 5.6. Примеры использования аргумента `expand` функции `scale_x_continuous()`. *Слева*: `expand = c(0, 0)`. *В центре*: `expand = c(0.5, 0)`. *Справа*: `expand = c(0, 8)`

5.3.2 Шкалы положения для дат и времени

Даты и время представляют собой количественные величины, однако большое разнообразие форматов, в которых они могут быть представлены, требует особого подхода. В связи с этим в пакете `ggplot2` имеются специальные функции для работы со шкалами дат и времени — `scale_x_date()`, `scale_y_date()`, `scale_x_datetime()` и `scale_y_datetime()`. Все эти функции имеют одинаковые наборы аргументов. Разница заключается лишь в том, что первые две из них используются преимущественно для работы с датами (класс `Date`), а последние две — для работы со временем (класс `POSIXct`). Для преобразования имеющихся значений дат или времени в необходимые класс и формат следует воспользоваться базовыми R–функциями `as.Date()` или `as.POSIXct()`.

Помимо аргументов, описанных в разд. 5.2, функции для работы со шкалами дат и времени имеют также свои специфичные аргументы — рассмотренный в предыдущем подразделе `expand`, а также аргумент `minor_breaks`, позволяющий настраивать промежуточные деления координатных осей. В большинстве случаев автоматические настройки этих функций дают хорошие результаты. При необходимости более тонкой настройки делений шкал и их меток следует воспользоваться аргументами `breaks` и `labels`. Первый из этих двух аргументов принимает текстовое выражение, которое задает интервал для разметки оси. Например, значение `breaks = "2 weeks"` приведет к тому, что основные деления на оси будут проставлены с интервалом в 2 недели. Кроме того, аргумент `breaks` можно применять в связке с функцией `date_breaks()` из пакета `scales`, например: `breaks = date_breaks(width = "2 weeks")`. Для указания временных интервалов используются следующие единицы: `"sec"` (секунда), `"min"` (минута), `"hour"` (час), `"day"` (день), `"week"` (неделя), `"month"` (месяц) и `"year"` (год), а также кратные им варианты в формате `"целое_число + пробел + единица_времени + s"` (например, `"2 weeks"`, `"5 years"`, `"10 hours"` и т. п.). Второй из указанных аргументов — `breaks` — рекомендуется использовать в паре с функцией `date_format()` из пакета `scales`, которая выполняет форматирование дат в соответствии со стандартной для R `POSIX`–спецификацией. Например,

дата 15/12/2014 в POSIX–формате выглядит как `"%d%m%Y"`. Соответственно, для отображения дат на графике в таком виде на аргумент `labels` необходимо было бы подать команду `date_format("%d%m%Y")`. Составные элементы POSIX–формата представлены в табл. 5.6. Примеры работы со шкалами положения для дат и времени, приведены на рис. 5.7 и 5.8.

Таблица 5.6. Элементы POSIX–формата

Элемент	Значение
`%S`	секунда (00–59)
`%M`	минута (00–59)
`%l`	час (1–12)
`%I`	час (01–12)
`%H`	час (00–23)
`%a`	день недели (`Mon`–`Sun`)
`%A`	день недели (`Monday`–`Sunday`)
`%e`	день месяца (1–31)
`%d`	день месяца (01–31)
`%m`	месяц (01–12)
`%b`	месяц (`Jan`–`Dec`)
`%B`	месяц (`January`–`December`)
`%y`	год (00–99)
`%Y`	год (0000–9999)

Код для рис. 5.7

```
( p <- ggplot(economics, aes(date, psavert/100)) +
       geom_line() + scale_y_continuous(labels = percent) +
       labs(x = "Year", y = "Personal savings rate") )
( p2 <- p + scale_x_date(breaks = date_breaks("5 years"),
                         labels = date_format("%Y")) )
```

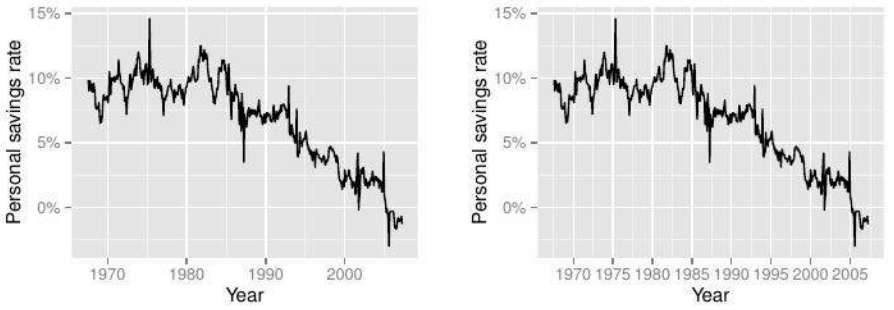


Рисунок 5.7. *Слева*: ось времени разбита на интервалы в соответствии с автоматическими настройками. *Справа*: ось времени разбита на 5–летние интервалы (см. код)

Код для рис. 5.8

```
# Пример основан на искусственных данных:
dat <- data.frame(times =
                  seq(as.POSIXct("2011-10-20 00:00:00"),
                      as.POSIXct("2011-10-20 23:59:59"),
                      by = "30 min"),
                  y = rnorm(48))
# Метки делений отражают только время:
(p3 <- ggplot(dat, aes(x = times, y = y)) + geom_line() +
       scale_x_datetime(breaks = date_breaks("4 hours"),
                        labels = date_format("%H:%M:%S")))
# Метки делений отражают дату (день) и время:
p3 + scale_x_datetime(breaks = date_breaks("6 hours"),
                  labels = date_format("%b%d\n%H:%M:%S"))
```

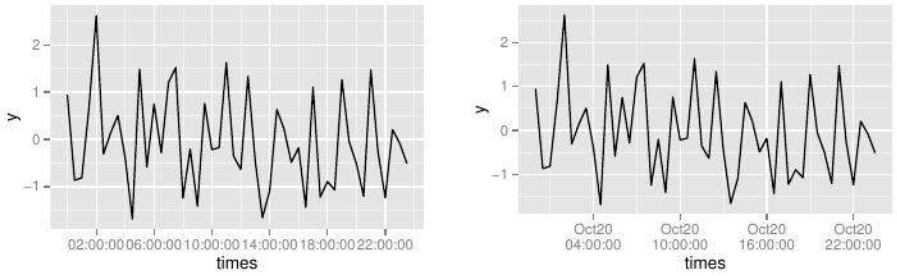


Рисунок 5.8. Пример работы функции `scale_x_datetime()`. *Слева*: метки делений на оси времени отражают только время. *Справа*: метки отражают дату и время (см. код)

5.3.3 Шкалы положения для качественных переменных

Настройка шкал положения для качественных переменных выполняется при помощи функций `scale_x_discrete()` и `scale_y_discrete()`. Их основные аргументы перечислены в разд. 5.2. Кроме того, обе эти функции имеют дополнительный аргумент `guide`, который используется для изменения заголовка легенды графика (при наличии таковой).

Отображение значений качественных переменных на координатные оси происходит путем преобразования уровней таких переменных в *целые числа*. Эти числа затем используются как порядковые номера уровней при их расстановке вдоль соответствующей оси. Например, у качественной переменной `Lake` из таблицы `dreissena` имеются три уровня:

```
levels(dreissena$Lake)
[1] "Batorino" "Myastro"  "Naroch"
```

Преобразование этих уровней в целые числа дает следующий результат:

```
as.numeric(as.factor(c("Batorino", "Myastro", "Naroch")))
[1] 1 2 3
```

Другими словами, при построении графиков с участием `Lake` первым вдоль соответствующей координатной оси будет располагаться уровень `"Batorino"`, вторым — `"Maystro"`, а третьим — `"Naroch"` (рис. 5.9, слева). Такой порядок неудивителен, поскольку R автоматически располагает уровни факторов в алфавитном порядке. Изменить этот порядок можно путем преобразования исходной качественной переменной в переменную с упорядоченными уровнями (англ. *ordered factor variable*) (см. справочный файл, доступный по команде `?as.factor`). Кроме того, очень часто возникает необходимость упорядочить уровни качественной переменной в соответствии со значениями некоторой другой, количественной переменной. В таких ситуациях следует воспользоваться базовой R–функцией `reorder()` (см. пример на рис. 5.9 справа). На рис. 5.10 представлены некоторые дополнительные особенности работы со шкалами положения для качественных переменных.

Код для рис. 5.9

```
library(dplyr) # для group_by(), summarise() и %>%
by.lake <- dreissena %>% group_by(Lake) %>%
           summarise(Length = round(median(Length)),
                     Infection = round(median(Infection)))
ggplot(by.lake, aes(Lake, Infection)) + geom_point() +
       geom_segment(aes(x = Lake, y = 0,
                        xend = Lake,
                        yend = Infection))
# Изменение порядка расположения уровней переменной Lake
# в соответствии со значениями переменной Length:
ggplot(by.lake, aes(reorder(Lake, Length), Infection)) +
       geom_point() + geom_segment(aes(x = Lake, y = 0,
                                       xend = Lake,
                                       yend = Infection))
```

Код для рис. 5.10

```
(p <- ggplot(dreissena, aes(Lake, Length)) +
           geom_boxplot() +
           scale_x_discrete(limits = c("Batorino",
                                       "Naroch")))
p + scale_x_discrete(name = "Two Lakes",
                     limits = c("Batorino", "Naroch"),
                     labels = c("Batorino" = "B", "Naroch" = "N"))
```

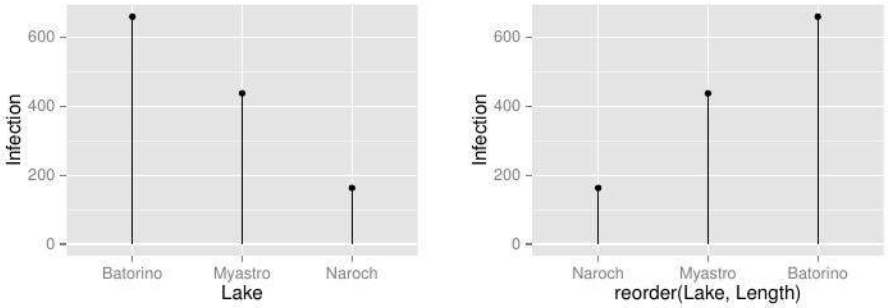


Рисунок 5.9. *Слева*: график построен с использованием автоматических настроек (уровни переменной `Lake` расположены вдоль оси X в алфавитном порядке). *Справа*: уровни переменной `Lake` упорядочены в соответствии со значениями переменной `Length` (при помощи команды `reorder(Lake, Length)` — см. код)

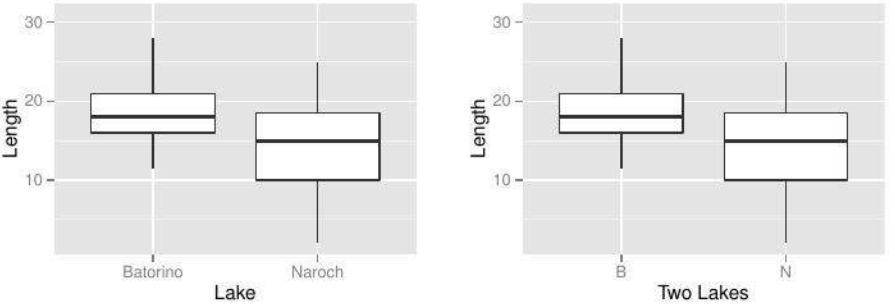


Рисунок 5.10. *Слева*: для отображения на оси X выбраны только два уровня переменной `Lake` (аргумент см. `limits`). *Справа*: изменены метки делений (аргумент `labels`) и подпись оси X (аргумент `name`)

5.4 Цветовые шкалы

Как известно, свет представляет собой электромагнитное излучение, спектральный состав которого определяет его цвет, воспринимаемый глазом человека. Существует множество моделей цветопередачи, используемых в разных устройствах для воспроизведения цветов. Одной из наиболее распространенных является RGB–модель, в которой цвет кодируется сочетанием интенсивностей красного (*red*), зеленого (*green*) и синего (*blue*) света. Несмотря на свое широкое распространение (используется в мониторах компьютеров, экранах телевизоров и т. п.), эта и другие, подобные ей модели имеют ряд недостатков, которые в целом сводятся к тому, что они некорректно отражают физический процесс восприятия цвета человеком. В глазу человека есть два типа клеток–рецепторов света — палочки и колбочки. Палочки обладают высокой светочувствительностью, но способны различать только интенсивность света, тогда как колбочки могут различать цвета, но лучше всего функционируют при ярком свете. Одной из современных моделей цветопередачи, которая хорошо отражает эти осо-

бенности человеческого глаза, является HCL–модель. Аббревиатура HCL происходит от следующих трех компонентов:

- H (*hue*) — *цветовой тон*. Представлен числом из диапазона от 0 до 360 (градусов);
- C (*chroma*) — *насыщенность цвета*. Нижнее значение этого параметра равно 0 (серый цвет), тогда как максимальное значение определяется величиной яркости;
- L (*luminance*) — *светлота* (яркость). Изменяется от 0 (черный) до 1 (белый).

В пакете `ggplot2` модель HCL принята по умолчанию. Однако, как мы увидим позднее, имеются и другие способы спецификации цветовых схем.

5.4.1 Цветовые шкалы для количественных переменных

В `ggplot2` есть три типа цветовых шкал для количественных переменных. Эти типы различаются числом основных цветов в градиенте:

- *двухцветные шкалы* (функция `scale_colour_gradient()` для управления цветом символов и линий и `scale_fill_gradient()` для управления цветом заливки различных фигур) — представлены градиентом оттенков между двумя опорными цветами, соответствующими низким и высоким значениям количественной переменной. Для указания эти двух цветов используются аргументы `low` и `high`;
- *трехцветные шкалы* (`scale_colour_gradient2()` и `scale_fill_gradient2()`) — помимо двух опорных цветов, расположенных на границах градиента (аргументы `low` и `high`), эти шкалы имеют еще один опорный цвет посередине градиента (аргумент `mid`). Такие шкалы подходят, в частности, для работы с переменными, которые принимают как положительные, так и отрицательные значения. По умолчанию опорный цвет, расположенный посередине градиента, соответствует значению 0. Это значение, однако, можно изменить при помощи аргумента `midpoint` (см. пример на рис. 4.83 в разд. 4.9);
- *многоцветные шкалы* (`scale_colour_gradientn()` и `scale_fill_gradientn()`) — для спецификации этих шкал пользователь должен подать вектор со значениями необходимых цветов на аргумент `colours` указанных функций. По умолчанию опорные цвета будут равномерно распределены вдоль диапазона значений соответствующей переменной. При необходимости неравномерного расположения цветов следует воспользоваться аргументом `values` и подать на него вектор значений, соответствующих опорным цветам градиента. Аргумент `values` может принимать либо абсолютные значения количественной переменной, либо ее нормализованные значения (т. е. из диапазона от 0 до 1). В последнем случае нормализацию необходимо активировать при помощи аргумента `rescale = TRUE`.

Помимо упомянутых выше аргументов (см. также разд. 5.2), функции, управляющие цветовыми шкалами для количественных переменных, имеют два специфичных аргумента: `na.value`, который определяет цвет для отображения отсутствующих значений, и `space`, определяющий цветовое пространство[5] (по умолчанию принимает значение `"Lab"`[6], которое рекомендуется не изменять).

Стандартным способом обозначения цвета в R является использование шестнадцатеричной схемы, в которой код каждого цвета состоит из знака # и некоторого уникального сочетания из 6 чисел и букв латинского алфавита. Например, красному цвету в этой системе соответствует код `#FF0000`, а черному — `#000000`. Вместо шестнадцатеричных кодов можно использовать также названия цветов. Полный список встроенных в R цветов включает 657 наименований (рис. 5.11). Вектор с названиями всех 657 цветов (в алфавитном порядке), перечисленных на рис. 5.11, можно получить при помощи команды `colours()`. Например, цвет под номером 404 носит название `"lightcoral"`:

```
colours()[404]
```

```
[1] "lightcoral"
```

Для выяснения шестнадцатеричного кода интересующего нас цвета можно воспользоваться следующей удобной функцией:

```
GetColorHex <- function(color){

    c <- col2rgb(color)
                sprintf("#%02X%02X%02X", c[1],c[2],c[3])

}

# Пример использования функции GetColorHex():

GetColorHex("lightcoral")
```

```
[1] "#F08080"
```

Для получения дополнительной информации по работе со стандартными цветами R стоит посетить страницу Эрла Глинна (Earl F. Glynn, Stowers Institute for Medical Research; `http://bit.ly/1wisAsW`).

На рис. 5.12 и 5.13 приведены примеры работы с цветовыми шкалами для количественных переменных.

[5] Цветовое пространство определяется всеми возможными цветами, которые задаются той или иной цветовой моделью. Подробнее см.: `http://bit.ly/1zeDL7X` и `http://bit.ly/1zsSZeK`.

[6] Подробнее о цветовом пространстве LAB см.: `http://bit.ly/1xdguIv`.

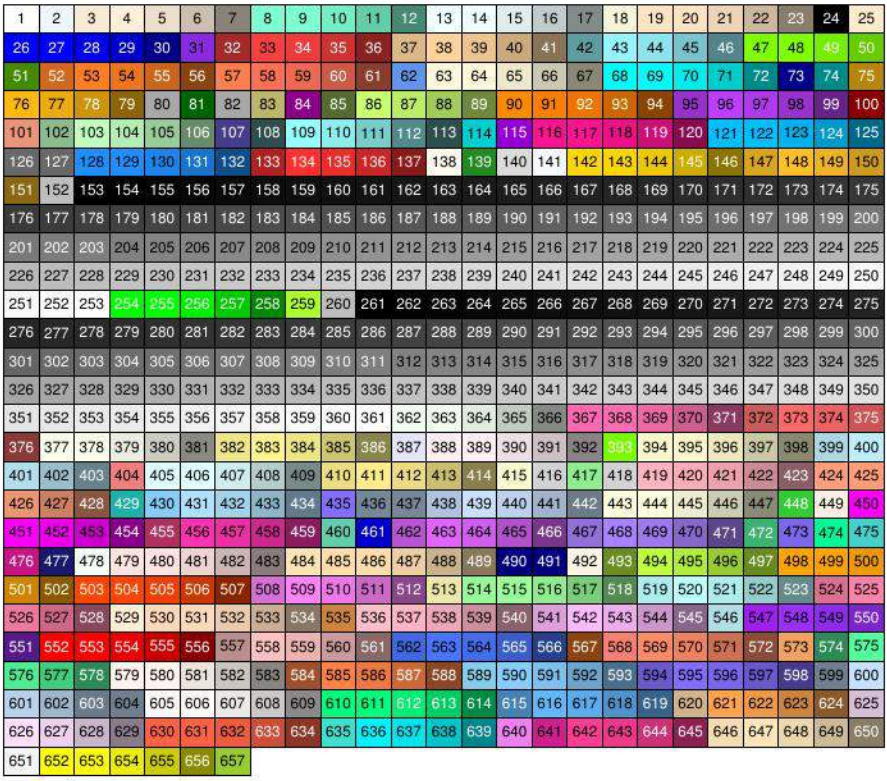


Рисунок 5.11. Стандартная цветовая палитра R. Цвета приведены в соответствии с алфавитным порядком их названий (см. объяснения в тексте). Код для создания этого рисунка был заимствован с сайта `http://bit.ly/1wisAsW`

Код для рис. 5.12

```
library(dplyr) # для filter() и %>%
nar <- dreissena %>% filter(Lake == "Naroch")

# цвета для низких и высоких значений Day заданы при помощи
# аргументов low и high функции scale_color_gradient():
ggplot(nar, aes(Length, Infection/1000, colour = Day)) +
       geom_point() +
       scale_x_continuous(trans = "sqrt") +
       scale_y_continuous(trans = "sqrt") +
       scale_color_gradient(low = "blue", high = "red")

# Вместо названий цветов можно использовать их
# шестнадцатеричные коды. Выясним, например, эти коды для
# цветов "black" и "pink":
GetColorHex("black")
[1] "#000000"
```

```
GetColorHex("pink")
[1] "#FFC0CB"

# Цвета заливки столбцов гистограммы указаны с использованием
# шестнадцатеричной схемы:
ggplot(nar, aes(Infection/1000, fill = ..count..)) +
      geom_histogram() +
      scale_x_continuous(trans = "sqrt") +
      scale_fill_gradient(low = "#000000",
                          high = "#FFC0CB")
```

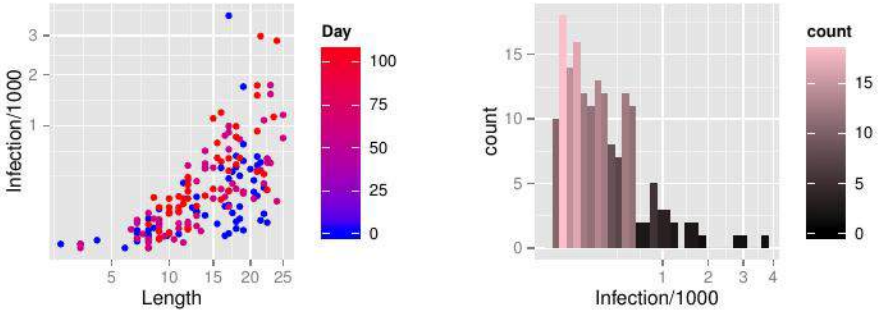


Рисунок 5.12. Примеры работы с цветовыми шкалами для количественных переменных (см. комментарии к коду). *Слева*: график зависимости между двумя количественными переменными, на котором цвет точек задан в соответствии со значениями третьей количественной переменной. *Справа*: гистограмма, цвет столбцов которой задан согласно частоте встречаемости соответствующего класса

Код для рис. 5.13

```
# Объект nar определен как на предыдущем рисунке
ggplot(nar, aes(Infection/1000, fill = ..count..)) +
       geom_histogram() +
       scale_x_continuous(trans = "sqrt") +
       scale_fill_gradient2(low = "red",
                            mid = "white",
                            high = "blue",
                            midpoint = 10)
ggplot(nar, aes(Infection/1000, fill = ..count..)) +
       geom_histogram() +
       scale_x_continuous(trans = "sqrt") +
       scale_fill_gradientn(colours = terrain.colors(4))
```

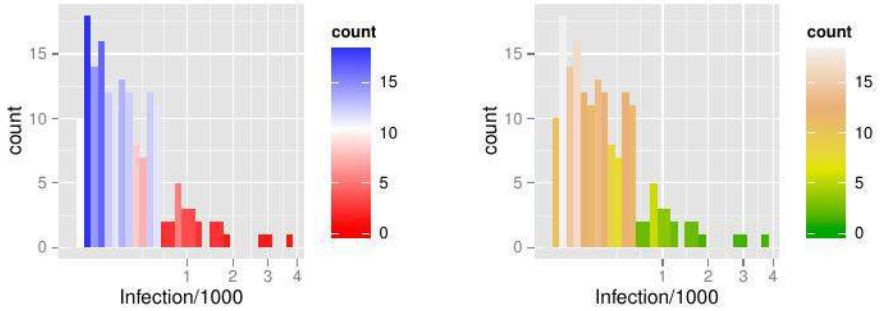


Рисунок 5.13. *Слева*: градиент задан тремя опорными цветами при помощи функции `scale_fill_gradient2()` (белый цвет соответствует значению 10). *Справа*: пример многоцветной шкалы, заданной при помощи `scale_fill_gradientn()`. Градиент представлен четырьмя опорными цветами, автоматически подобранными при помощи базовой `R`–функции `terrain.colors()` (существуют и другие стандартные функции для создания цветовых палитр — см. `?rainbow`, `?heat.colors`, `?topo.colours` и `?cm.colors`. Кроме того, можно воспользоваться функциями из других пакетов, таких как `RColorBrewer` или `vcd`)

5.4.2 Цветовые шкалы для качественных переменных

В `ggplot2` есть два типа цветовых шкал для качественных переменных[7]:

- *HCL–шкалы* (`scale_colour_hue()` для управления цветом символов и линий и `scale_fill_hue()` для управления цветом заливки фигур) — выполняют автоматический подбор цветов, равноудаленных друг от друга на «колесе» HCL–тонов (стр. 150). Многочисленные примеры использования автоматически подобранных цветов для отображения уровней качественных переменных были приведены в главе 4.

- *шкалы ColorBrewer* (`scale_colour_brewer()` и `scale_fill_brewer()`) — позволяют использовать специально подобранные палитры хорошо сочетаемых цветов (включая палитры, воспринимаемые людьми с нарушениями цветового зрения). Название этих шкал происходит от названия научного проекта ColorBrewer, в рамках которого они были разработаны (`http://colorbrewer2.org`)[8].

HCL–шкалы хорошо работают в случаях, когда число уровней качественной переменной не превышает восьми (рис. 5.14). При большем числе

[7] Как уже отмечалось ранее, помимо собственно качественных переменных, эти шкалы подходят также для работы с текстовыми и логическими переменными, которые можно преобразовать в качественные переменные при помощи функции `as.factor()`.

[8] См. инструкции по выбору подходящей палитры на странице `http://bit.ly/1JQrE9D`.

уровней разница между соседними цветами становится трудно различимой. Кроме того, все цвета в автоматически подобранных HCL–палитрах имеют одинаковую насыщенность и яркость, в связи с чем при печати на черно–белых принтерах эти цвета выглядят как неразличимые оттенки серого.

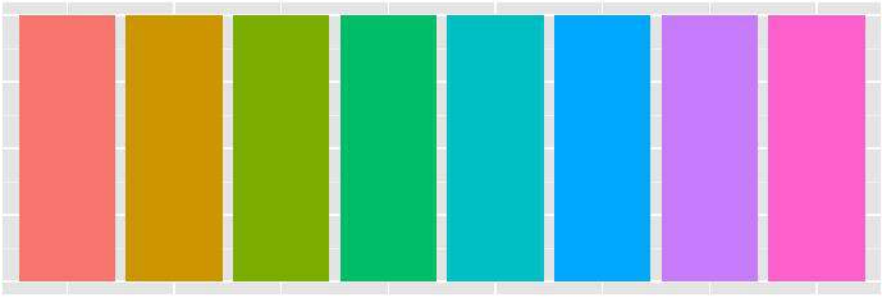


Рисунок 5.14. HCL–цвета, автоматически задаваемые в `ggplot2` для качественных переменных

Шкалы ColorBrewer лучше подходят для ситуаций, когда результат визуализации данных предполагается использовать на разных носителях или устройствах (LCD–мониторы, печатные издания и т. д.). Соответствующие цветовые палитры представлены тремя группами (рис. 5.15):

- *последовательные палитры* (англ. *sequential palettes*) — предназначены для работы с качественными переменными, чьи уровни упорядочены (например, [«холодный», «теплый», «горячий»] или [«май», «июнь», «июль»]). Различия между отдельными уровнями таких переменных подчеркиваются при помощи светлоты — светлые цвета соответствуют низким значениям, а темные — высоким. Имеется 18 таких палитр, включающих до 9 цветов;

- *качественные палитры* (англ. *qualitative palettes*) — используются при работе с качественными переменными, уровни которых невозможно упорядочить каким–либо естественным образом (например, [«Лондон», «Париж», «Нью–Йорк»]). Имеется 8 таких палитр. Число входящих в них цветов составляет от 3 до 12. Для кодирования цвета точек и линий особенно хорошо подходят палитры `"Set1"` и `"Dark2"`, а для закрашивания фигур стоит обратиться к палитрам `"Set2"`, `"Pastel1"`, `"Pastel2"` или `"Accent"`;

- *расходящиеся палитры* (англ. *divergent palettes*) — используются для придания одинакового веса значениям, находящимся как в центре, так и в «хвостах» распределения значений качественной переменной. Для уровня, находящегося в центре распределения, обычно используется какой–либо светлый цвет, тогда как уровни, удаляющиеся от центра, представлены более темными цветами контрастирующих тонов. Имеется 9 таких палитр, каждая из которых включает от 3 до 11 цветов.

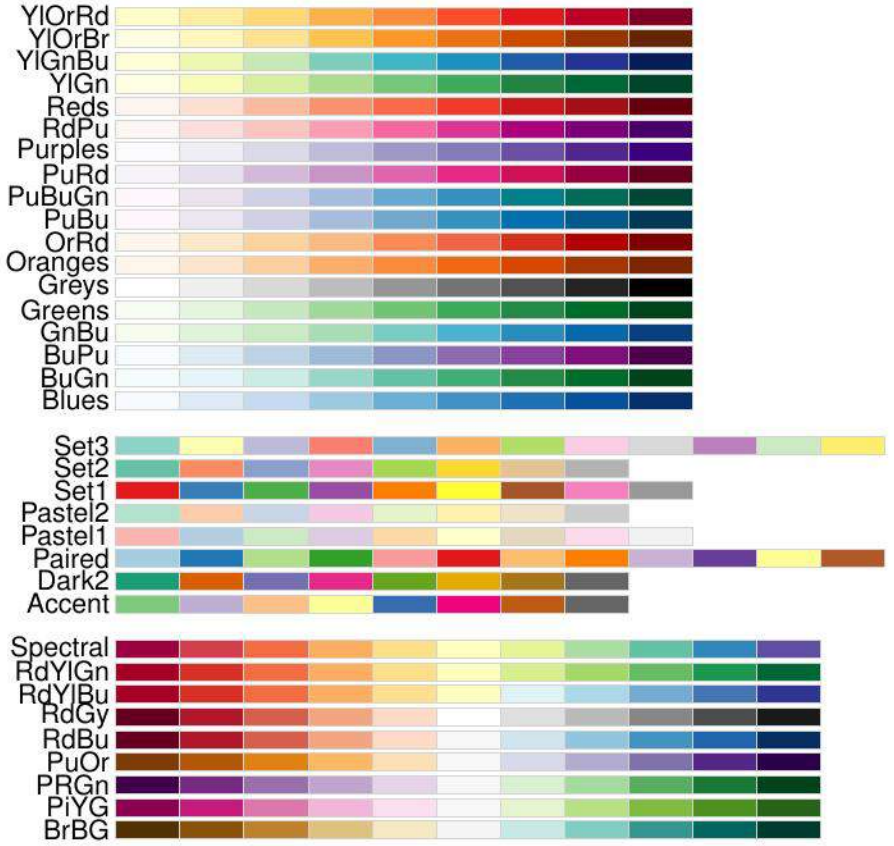


Рисунок 5.15. Цветовые палитры ColorBrewer, объединенные в три группы: последовательные (вверху), качественные (в центре) и расходящиеся (внизу). Каждая палитра представлена максимальным числом входящих в нее цветов. Названия палитр указаны слева

Функции, управляющие свойствами шкал HCL и ColorBrewer, имеют ряд как общих (см. разд. 5.2), так и специфичных для них аргументов. В частности, функции `scale_colour_hue()` и `scale_fill_hue()` имеют следующие специфичные аргументы:

- `h` — вектор из двух целых чисел, определяющих границы диапазона цветовых тонов (нижняя допустимая граница — `0`, верхняя — `360`);
- `c` — целое число (от `0` до `100`), задающее насыщенность цвета;
- `l` — целое число (от `0` до `100`), задающее светлоту цвета;
- `h.start` — целое число, определяющее начальную точку отсчета на цветовом колесе;
- `direction` — направление движения по цветовому колесу (`1` — по часовой стрелке, `-1` — против часовой стрелки);

- `na.value` — цвет для отображения отсутствующих значений.

На рис. 5.16 представлены варианты одной и той же диаграммы рассеяния с использованием разных сочетаний параметров `h`, `c` и `l`.

Функции `scale_colour_brewer()` и `scale_fill_brewer()` имеют следующие специфичные для них аргументы (рис. 5.17):

- `type` — тип палитры: `"seq"` (последовательная), `"qual"` (качественная) или `"div"` (расходящаяся);
- `palette` — название палитры (см. рис. 5.15).

Код для рис. 5.16

```
# Объект nar определен как на рис. 5.12
( p <- ggplot(nar, aes(Length, Infection/1000,
            colour = Month)) + geom_point() +
            scale_x_continuous(trans = "sqrt") +
            scale_y_continuous(trans = "sqrt") )
p + scale_color_hue(h = c(200, 360))
p + scale_color_hue(h = c(200, 360), c = 30)
p + scale_color_hue(h = c(200, 360), c = 30, l = 80)
```

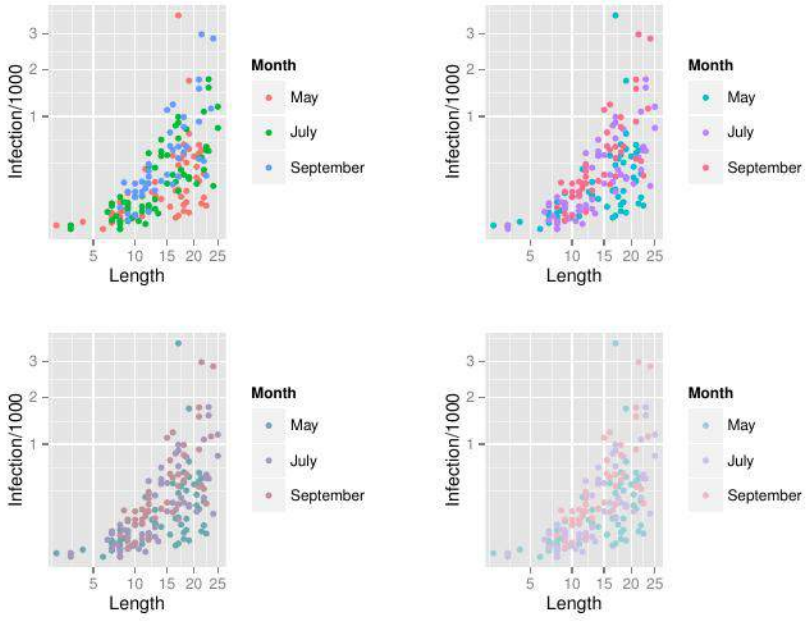


Рисунок 5.16. *Слева вверху*: цвета HCL–шкалы заданы в соответствии с автоматическими настройками. *Справа вверху*: диапазон цветовых тонов изменен (`h = c(200, 360)`). *Слева внизу*: то же, но изменена также насыщенность цвета (`c = 30`). *Справа внизу*: то же, но изменена также светлота цвета (`l = 80`)

Код для рис. 5.17

```
# Объект nar определен как на рис. 5.12
ggplot(nar, aes(Length, Infection/1000, colour = Month)) +
       geom_point() + scale_x_continuous(trans = "sqrt") +
       scale_y_continuous(trans = "sqrt") +
       scale_colour_brewer(type = "qual", palette = "Set1")
ggplot(nar, aes(Infection/1000, fill = Month)) +
       geom_histogram() +
       scale_x_continuous(trans = "sqrt") +
       scale_fill_brewer(type = "qual", palette = "Accent")
```

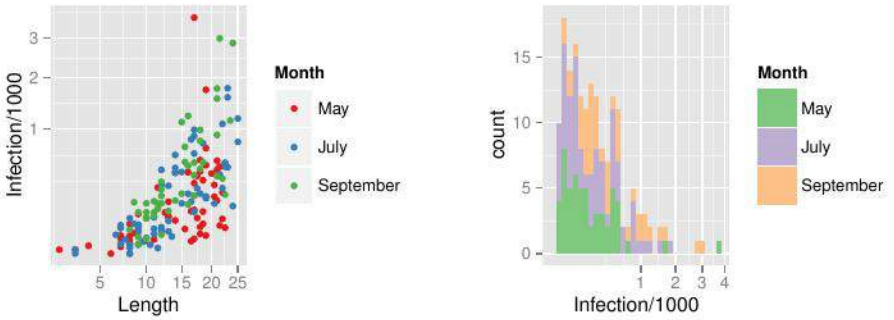


Рисунок 5.17. Примеры настройки шкал ColorBrewer. *Слева*: использована палитра `"Set1"`. *Справа*: использована палитра `"Accent"`

5.5 Пользовательские шкалы для качественных переменных

При желании пользователь может самостоятельно задать значения эстетических атрибутов, предназначенных для обозначения уровней качественных переменных (форма, размер и цвет символов, тип, толщина и цвет линий, а также цвет заливки фигур). Все `scale`–функции, позволяющие выполнять такое «ручное» управление шкалами, имеют в своем названии слово `manual` (например, `scale_size_manual()`, `scale_fill_manual()` и т. п. — см. табл. 5.3). Помимо аргументов, перечисленных в разд. 5.2, эти функции имеют также специфичный для них аргумент `value`, который служит для указания пользовательских значений соответствующих эстетических атрибутов. Примеры применения таких функций приведены ниже.

Код для рис. 5.18

```
library(dplyr) # для filter() и %>%
nar <- dreissena %>% filter(Lake == "Naroch",
                            Month %in% c("May", "July"))
p <- ggplot(nar, aes(Length, Infection/1000)) +
```

```
        geom_point() + scale_x_continuous(trans = "sqrt") +
        scale_y_continuous(trans = "sqrt")
p + aes(shape = Month) + scale_shape_manual(values = c(1, 2))
p + aes(shape = Month, size = Month) +
        scale_shape_manual(values = c(1, 2)) +
        scale_size_manual(values = c(5, 2))
```

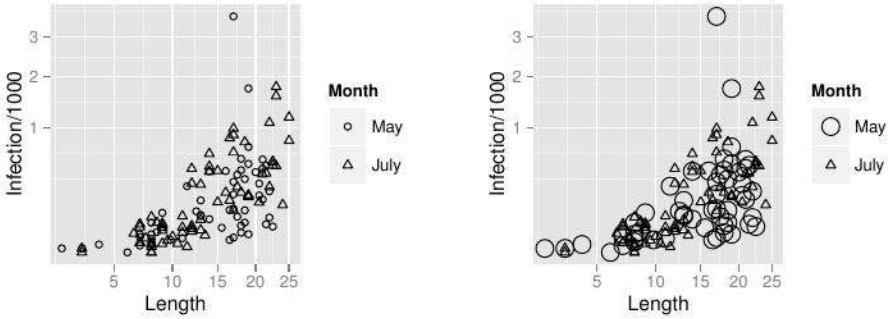


Рисунок 5.18. *Слева*: заданы пользовательские значения формы символов, используемых для изображения данных. *Справа*: заданы пользовательские значения размера этих символов

Код для рис. 5.19

```
# Объекты nar и p определены как на предыдущем рисунке
library(RColorBrewer)
p + aes(colour = Month) +
        scale_colour_manual(values = c("blue", "pink"))
# Цвета из палитры Dark2 пакета RColorBrewer:
cols <- brewer.pal(3, "Dark2")
( p3 <- p + aes(colour = Month) +
        scale_colour_manual(values = cols) )
```

Код для рис. 5.20

```
# Объект nar определен как на рис. 5.18
library(dplyr) # для group_by(), summarise() и %>%
by.month <- dreissena  %>% group_by(Lake, Month) %>%
            summarise(Infection = median(Infection))
# Цвета из палитры Set1 пакета RColorBrewer:
cols <- brewer.pal(3, "Set1")
ggplot(by.month, aes(Month, Infection,
       group = Lake, colour = Lake)) + geom_point() +
       geom_line() + scale_colour_manual(values = cols)
ggplot(nar, aes(Infection, fill = Month)) +
       geom_histogram() + scale_fill_manual(values = cols)
```

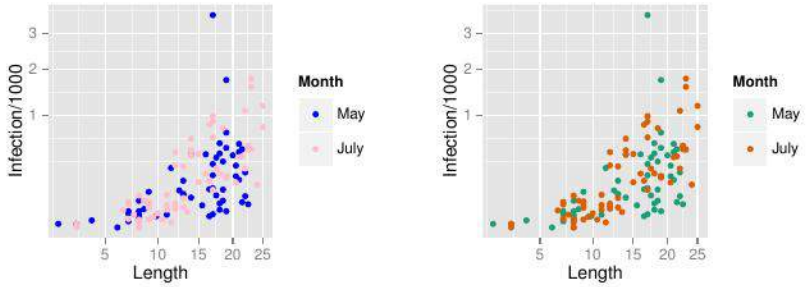


Рисунок 5.19. *Слева*: заданы пользовательские значения цвета символов, используемых для изображения данных. *Справа*: то же, но использованы цвета из палитры `"Dark2"` (пакет `RColorBrewer`)

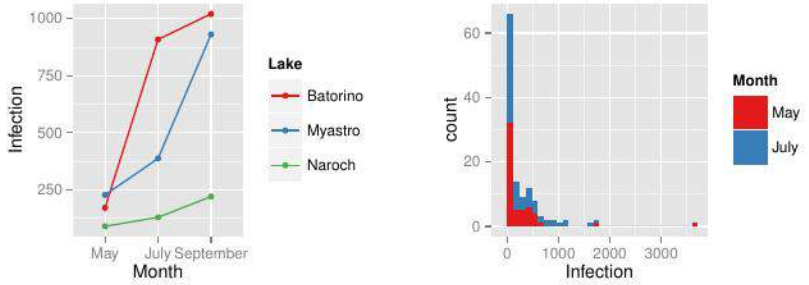


Рисунок 5.20. *Слева*: заданы пользовательские значения цвета символов и линий, используемых для изображения данных. *Справа*: заданы пользовательские значения цвета столбцов гистограммы

5.6 Тождественные шкалы

Тождественные шкалы используются в случаях, когда в таблице с данными уже имеются переменные со значениями соответствующих эстетических атрибутов. Рассмотрим следующий простой пример:

```
library(dplyr) # для group_by(), %>% и summarise()
by.month <- dreissena %>% group_by(Lake, Month) %>%
            summarise(Infection = median(Infection))
by.month
      Lake     Month Infection
1 Batorino       May       171
2 Batorino      July       908
3 Batorino September      1020
4  Myastro       May       227
5  Myastro      July       387
6  Myastro September       930
7   Naroch       May        90
8   Naroch      July       129
9   Naroch September       220
```

Таблица `by.month` содержит медианные значения переменной `Infection`, рассчитанные для каждого сочетания уровней переменных `Lake` и `Month` (см. также рис. 5.20). Чтобы задать цвет линий и точек на этом графике, мы применили функцию `scale_colour_manual()`, на аргумент `values` которой был подан вектор со значениями цветов палитры `"Set1"` из пакета `RColorBrewer`. Однако представим ситуацию, когда значения цвета уже присутствуют в таблице с данными:

```
by.month$Color <- c(rep("black",3), rep("red",3), rep("blue",3))
by.month
      Lake     Month Infection Color
1 Batorino       May       171 black
2 Batorino      July       908 black
3 Batorino September      1020 black
4  Myastro       May       227   red
5  Myastro      July       387   red
6  Myastro September       930   red
7   Naroch       May        90  blue
8   Naroch      July       129  blue
9   Naroch September       220  blue
```

Мы можем непосредственно использовать значения из столбца `Color` для создания цветовой шкалы. Для этого необходимо воспользоваться функцией `scale_colour_identity()` (рис. 5.21). Обратите внимание на то, что при работе с тождественными шкалами пользователю придется самостоятельно сконструировать легенду графика (с помощью таких аргументов, как `name`, `labels`, `breaks` и др., — см. разд. 5.2).

Код для рис. 5.21

```
ggplot(by.month, aes(Month, Infection,
                     group = Lake, colour = Color)) +
       geom_point() + geom_line() +
       scale_colour_identity(guide = "legend", name = "Lake",
       labels = c("Batorino", "Naroch", "Myastro"))
# Аргумент guide = "legend" использован для включения легенды
```

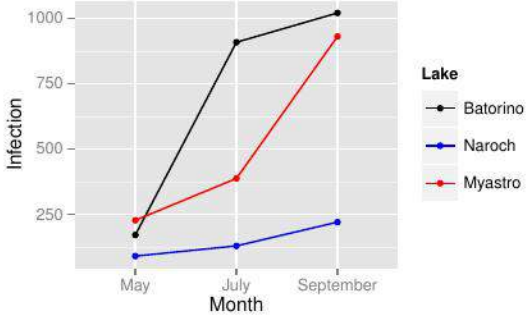


Рисунок 5.21. Пример создания тождественной цветовой шкалы при помощи функции `scale_colour_identity()`

Глава 6

Системы координат

Как было отмечено в предыдущей главе, в ходе отображения данных на пространство эстетических атрибутов, выполняемого той или иной `scale`–функцией, происходит создание координатных осей. Оси являются важной составной частью графика, без которой в большинстве случаев его правильная интерпретация невозможна. В этой главе мы кратко рассмотрим системы координат, реализованные в пакете `ggplot2`. В общем виде под системой координат понимают некую *опорную систему, позволяющую определить положение точки на плоскости или в пространстве.* В настоящее время в `ggplot2` есть 6 таких систем (табл. 6.1).

Таблица 6.1. Системы координат, реализованные в `ggplot2`

Название	Описание
`cartesian`	декартова (картезианская) система координат
`fixed`	декартова система с фиксированным масштабом осей
`flip`	декартова система с перевернутыми осями
`trans`	преобразованная декартова система
`map`	картографические проекции
`polar`	полярная система координат

Имена функций, отвечающих за формирование той или иной системы координат, образуются путем добавления приставки `coord_` к названию соответствующей системы (например, `coord_cartesian()`, `coord_polar()`, `coord_map()` и т. д.). Все реализованные в `ggplot2` системы координат являются двухмерными, т. е. задают положение точки на плоскости. Очевидно, однако, что в зависимости от рассматриваемой системы координаты точки будут представлять собой разные сущности. Так, в декартовой системе положение точки на плоскости задается обычными X– и Y–координатами (т. е. расстояниями от этой точки до двух взаимно перпендикулярных координатных осей), в полярной системе мы имеем дело с углом и радиусом, а в картографической проекции — с широтой и долготой.

6.1 Декартова система и ее разновидности

Построение всех графиков в ggplot2 по умолчанию выполняется в *прямоугольной декартовой (картезианской) системе координат*, свойства которой контролируются функцией `coord_cartesian()`. Такая система координат образована двумя взаимно перпендикулярными прямыми линиями: осями координат X («ось абсцисс») и Y («ось ординат»). Точка пересечения этих осей называется *началом системы координат*. Положение точки в прямоугольной системе однозначно определяется двумя координатами: `x` (расстояние от точки до оси Y) и `y` (расстояние от точки до оси X). По сути, к декартовым относятся и три другие системы координат ggplot2, которые задаются функциями `coord_equal()`, `coord_flip()` и `coord_trans()` (см. табл. 6.1): во всех трех случаях координаты точки однозначно определяются ортогональными проекциями на оси X и Y.

У функции `coord_cartesian()` есть два основных аргумента: `xlim` и `ylim`, которые задают диапазоны значений координатных осей X и Y соответственно. С помощью этих аргументов можно достичь эффекта «увеличительного стекла», т. е. приблизить определенные участки графика для более тщательного изучения. Как показано на рис. 6.1, этот эффект отличается от того, что происходит в результате использования аргумента `limits` при вызове функций из семейств `scale_x_...` и `scale_y_...`, рассмотренных в разд. 5.3. При работе со `scale`–функциями все наблюдения, находящиеся за пределами нижней и верхней границ диапазона `limits`, принимают значение `NA` и не оказывают никакого влияния на внешний вид графика.

Код для рис. 6.1

```
ggplot(data = dreissena, aes(Length, sqrt(Infection))) +
       geom_point(alpha = 0.5) + geom_smooth()
ggplot(data = dreissena, aes(Length, sqrt(Infection))) +
       geom_point(alpha = 0.5) + geom_smooth() +
       coord_cartesian(xlim = c(10, 20), ylim = c(0, 40))
ggplot(data = dreissena, aes(Length, sqrt(Infection))) +
       geom_point(alpha = 0.5) + geom_smooth() +
       scale_x_continuous(limits = c(10, 20)) +
       scale_y_continuous(limits = c(0, 40))
```

Как правило, при построении статистических графиков и моделей значения независимой переменной откладывают по оси абсцисс, а значения зависимой переменной — по оси ординат. При этом предполагается, что значения независимой переменной измерены без ошибки, тогда как в отношении зависимой переменной имеется некоторая неопределенность. Однако иногда может возникнуть необходимость повернуть оси на 90 градусов так, чтобы значениям зависимой переменной соответствовала ось абсцисс, а значениям независимой переменной — ось ординат. Как видно на рис. 6.2, простой смены порядка перечисления переменных `x` на `y` при построении графика с помощью `qplot()` или `ggplot()` недостаточно: чтобы правильно отразить ошибку, с которой измерены значения зависимой переменной, необходимо воспользоваться функцией `coord_flip()` (эта функция имеет те же аргументы, что и `coord_cartesian()`).

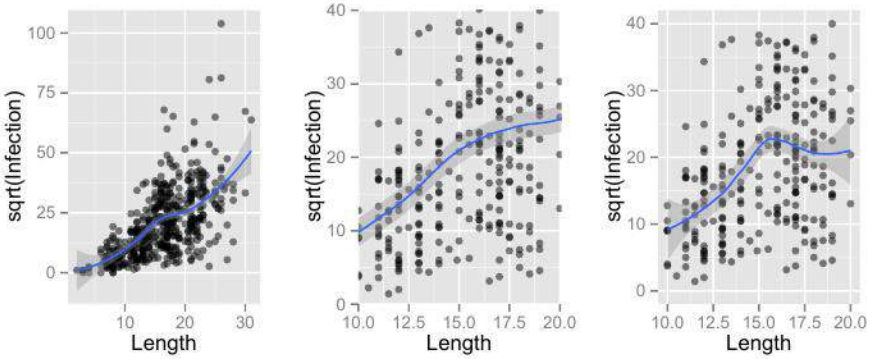


Рисунок 6.1. *Слева:* исходный график, построенный в декартовой системе координат. *В центре:* тот же график, однако диапазоны значений координатных осей были изменены при помощи аргументов `xlim` и `ylim` функции `coord_cartesian()`. Использование этих аргументов вызвало эффект «увеличительного стекла». *Справа:* диапазоны значений координатных осей были изменены при помощи аргумента `limits` функций `scale_x_continuous()` и `scale_y_continuous()`. Это привело к удалению из рассмотрения всех наблюдений, оказавшихся за пределами соответствующих диапазонов, и, как следствие, к изменению характера сглаживающей линии и ее доверительной области

Код для рис. 6.2

```
ggplot(data = dreissena, aes(Length, sqrt(Infection))) +
       geom_point(alpha = 0.5) + geom_smooth()
ggplot(data = dreissena, aes(sqrt(Infection), Length)) +
       geom_point(alpha = 0.5) + geom_smooth()
ggplot(data = dreissena, aes(Length, sqrt(Infection))) +
       geom_point(alpha = 0.5) + geom_smooth() + coord_flip()
```

Заметьте, что значения зависимой переменной на рис. 6.1 и 6.2 предварительно были преобразованы путем извлечения квадратного корня. Как мы уже знаем, преобразованные значения можно также изобразить, изменив масштаб соответствующей шкалы (для этого у функций из семейств `scale_x_...` и `scale_y_...` имеется аргумент `trans` — см. разд. 5.3). Еще один способ заключается в использовании функции `coord_trans()`. Основными аргументами этой функции являются `x` и `y`. Данные аргументы принимают названия функций, выполняющих преобразования значений независимой и зависимой переменных соответственно (см. рис. 6.3). Список этих функций был приведен ранее в табл. 5.4. Следует помнить, что в результате применения `coord_trans()` некоторые геометрические объекты, используемые для изображения данных и сводной статистической информации, иногда могут изменять свою форму (так, например, прямая линия может уже не быть прямой в преобразованной системе координат; см. справочный файл по этой функции — `?coord_trans`).

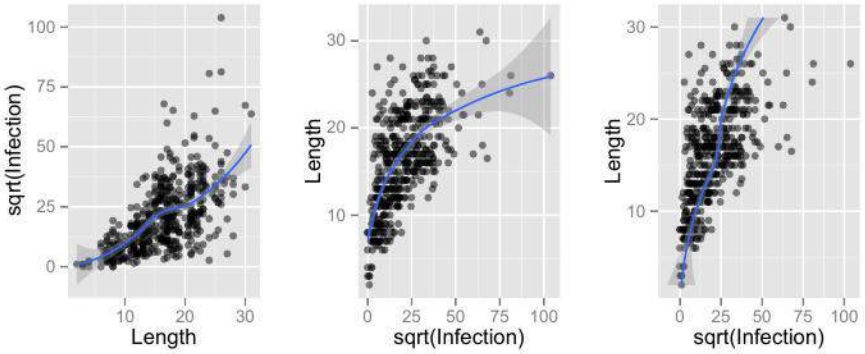


Рисунок 6.2. *Слева:* независимой переменной (`Length`) соответствует ось абсцисс, а зависимой переменной (`sqrt(Infection)`) — ось ординат. *В центре:* по оси абсцисс отложены значения `sqrt(Infection)`, а по оси ординат — значения `Length`. Для этого при вызове функции `ggplot()` сначала была указана переменная `sqrt(Infection)`, а затем — переменная `Length`. Однако с «точки зрения» `ggplot2` зависимой теперь стала переменная `Length`. *Справа:* применение функции `coord_flip()` позволяет сохранить исходную интерпретацию переменных, т. е. при повороте осей графика на 90 градусов зависимой по–прежнему остается переменная `sqrt(Infection)`

Код для рис. 6.3

```
p <- ggplot(data = dreissena) + geom_point(alpha = 0.5)
p + aes(Length, sqrt(Infection))
p + aes(Length, Infection) +
    scale_y_continuous(trans = "sqrt")
p + aes(Length, Infection) + coord_trans(y = "sqrt")
```

Наконец, рассмотрим вариацию декартовой системы координат, реализованную при помощи функции `coord_fixed()`. Варьируя значение аргумента `ratio` этой функции, можно задать определенное соотношение масштабов координатных осей. Так, при `ratio = 1` (значение, принятое по умолчанию) отрезок, соответствующий одному делению на оси X, будет иметь ту же длину, что и отрезок, соответствующий одному делению на оси Y. Значения `ratio > 0` приведут к тому, что деления на оси Y физически будут длиннее, чем деления на оси X, и наоборот (см. рис. 6.4).

Код для рис. 6.4

```
p <- ggplot(data = dreissena, aes(Length, sqrt(Infection))) +
     geom_point(alpha = 0.5)
p + coord_fixed(ratio = 1)
p + coord_fixed(ratio = 1.5)
p + coord_fixed(ratio = 0.2)
```

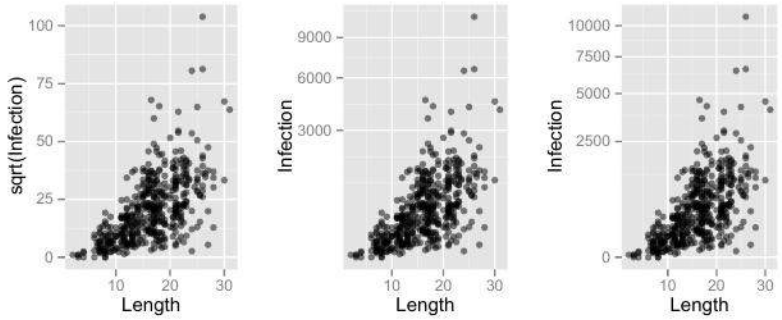


Рисунок 6.3. Иллюстрация трех подходов для изображения преобразованных значений переменных. *Слева:* по оси ординат отложены значения, полученные в результате извлечения квадратного корня из исходных значений переменной `Infection`. *В центре:* те же данные, но масштаб оси ординат изменен при помощи команды `scale_y_continuous(trans = "sqrt")`. *Справа:* те же данные, но масштаб оси ординат изменен при помощи команды `coord_trans(y = "sqrt")`

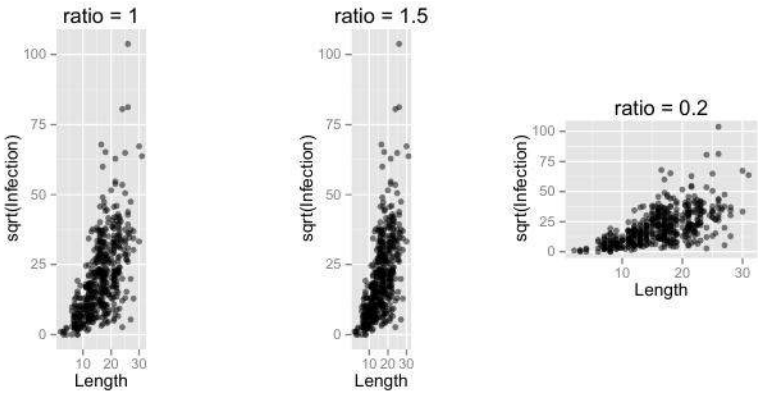


Рисунок 6.4. Примеры использования функции `coord_fixed()` с разными значениями аргумента `ratio`

6.2 Полярная система

В полярной системе координат любая точка задается двумя числами — *полярным радиусом* (обычно обозначается как r) и *полярным углом* (обозначается как θ или ϕ и измеряется в *радианах*). Радиальная координата соответствует расстоянию от точки до начала координат, а угловая координата — углу, на который необходимо повернуть против часовой стрелки полярную ось, для того чтобы попасть в эту точку. Использование полярной системы координат позволяет строить круговые диаграммы, «розы ветров», или лепестковые диаграммы, «радарные» диаграммы, а также

графики типа «бычий глаз». Функция `coord_polar()`, реализующая полярную систему координат в `ggplot2`, имеет следующие аргументы (см. примеры на рис. 6.5 и 6.6):

- `theta` — переменная, определяющая угловые координаты (по умолчанию это переменная X);
- `start` — точка отсчета (по умолчанию 0, но при желании это значение можно изменить);
- `direction` — направление поворота угла, т. е. по часовой стрелке (`1`) или против нее (`-1`).

Код для рис. 6.5

```
( p <- ggplot(data = dreissena, aes(factor(1),
              fill = Month)) + geom_bar(width = 1) )
p + coord_polar(theta = "y")
p + coord_polar(theta = "x")
```

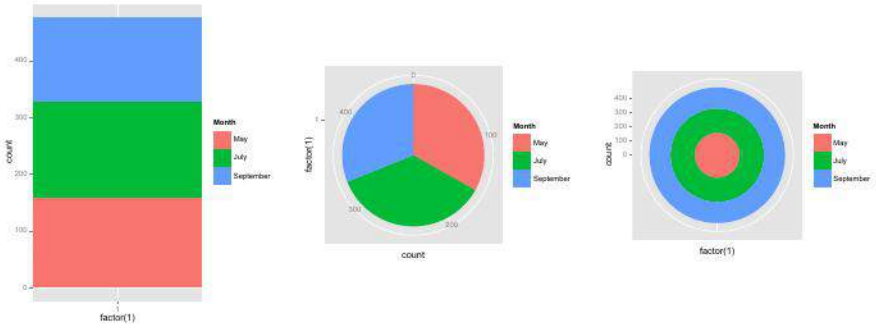


Рисунок 6.5. *Слева:* составная столбиковая диаграмма. *В центре:* круговая диаграмма, полученная путем представления той же столбиковой диаграммы в полярной системе, где угловые координаты определяются переменной Y. *Справа:* диаграмма типа «бычий глаз», полученная путем представления той же столбиковой диаграммы в полярной системе, где угловые координаты определяются переменной X

Код для рис. 6.6

```
library(dplyr) # для summarise() и %>%
dreiss <- dreissena %>% group_by(Lake, Day) %>%
       summarise(Length = mean(Length))
ggplot(dreissena, aes(x = Lake)) +
       geom_bar(width = 1, color = "black",
       fill = "lightblue") + coord_polar(theta = "x")
ggplot(dreiss, aes(x = Day, y = Length, color = Lake)) +
       geom_line(size = 1) + coord_polar(theta = "x")
ggplot(dreiss, aes(x = Day, y = Length, color = Lake)) +
       geom_line(size = 1) + coord_polar(theta = "y")
```

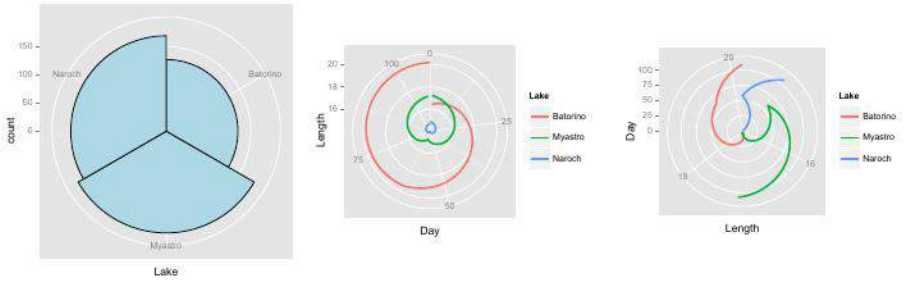


Рисунок 6.6. *Слева:* пример лепестковой диаграммы. *В центре:* пример визуализации временных рядов в полярной системе, где угловые координаты определяются переменной *X*. *Справа:* то же, что и в центре, но угловые координаты определяются переменной *Y*

6.3 Картографические проекции

Как известно, Земля имеет форму, близкую к сферической, в связи с чем изображение части ее поверхности на двухмерной плоскости требует применения той или иной процедуры *картографического проецирования*. Выполнение такого проецирования неизбежно сопровождается возникновением определенных искажений (длин, углов, площадей, форм). В связи с этим существует большое количество способов проецирования, уделяющих разное внимание соответствующим видам искажений. При построении карт средствами `ggplot2` тип картографической проекции определяется функцией `coord_map()`. Мы уже сталкивались с ней в разд. 4.9 (стр. 126) при рассмотрении функции `geom_map()`. Ниже приведено несколько дополнительных примеров использования функции `coord_map()` (рис. 6.7 и 6.8), которая имеет следующие аргументы:

- `projection` — тип картографической проекции (с полным списком доступных типов проекций можно ознакомиться в справочном файле функции `mapproject()` из пакета `mapproj`, к которой обращается `coord_map()`);
- `orientation` — ориентация проекции, которая задается числовым вектором из трех элементов: `latitude` (широта), `longitude` (долгота) и `rotation` (угол поворота). Для большинства типов проекций заданные по умолчанию значения — `c(90, 0, mean(range(x)))` — не подойдут, и их придется изменить;
- `xlim` и `ylim` — аргументы, позволяющие пользователю самостоятельно задать диапазоны значений осей `x` и `y` (в градусах долготы и широты соответственно);
- `...` — другие аргументы, подаваемые на функцию `mapproject()`.

Код для рис. 6.7

```
library(maps) # пакет с картами нескольких стран и всего мира
library(mapproj)
it <- map_data("italy") # карта Италии
itmap <- ggplot(it, aes(x = long, y = lat, group = group)) +
                geom_polygon(fill = "white",
                             colour = "black")
itmap + coord_map("mercator")
itmap + coord_map("orthographic")
itmap + coord_map("stereographic")
itmap + coord_map("conic", lat0 = 0)
```

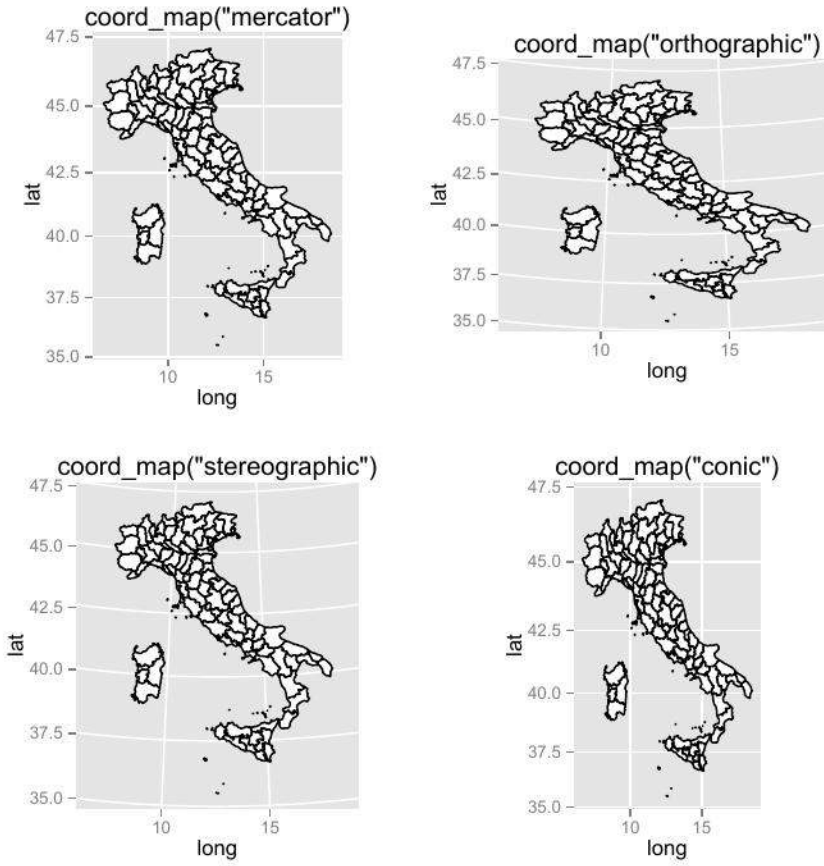


Рисунок 6.7. Примеры использования функции `coord_map()` с разными значениями аргумента `projection`

Код для рис. 6.8

```
world <- map_data("world")
worldmap <- ggplot(world, aes(x = long, y = lat,
                              group = group)) +
                  geom_path() +
                  scale_y_continuous(breaks = (-2:2) * 30) +
                  scale_x_continuous(breaks = (-4:4) * 45)
worldmap + coord_map("orthographic")
worldmap + coord_map("orthographic",
                     orientation = c(-90, 0, 0))
worldmap + coord_map("orthographic",
                     orientation = c(40, 30, 20))
```

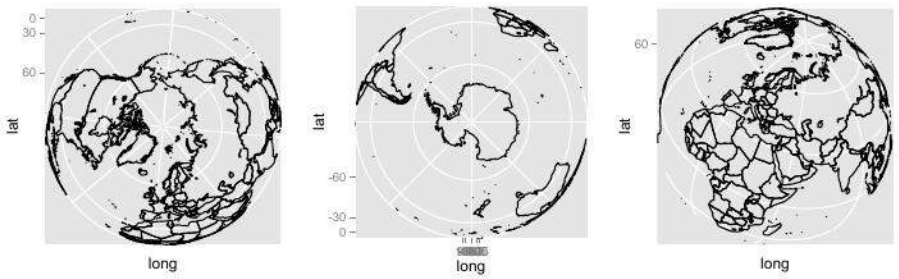


Рисунок 6.8. Примеры использования аргумента `orientation` функции `geom_map()`. *Слева:* ортографическая проекция с ориентацией, заданной по умолчанию (вид с северного полюса). *В центре:* вид с южного полюса. *Справа:* вид, полученный в результате применения произвольно выбранных значений аргумента `orientation`

Глава 7

Подробнее о категоризованных графиках

В этой главе мы продолжим начатое в разд. 2.4 обсуждение *категоризованных графиков*. Подобные графики применяются для визуализации данных, разбитых на отдельные подмножества, или категории. Эти подмножества размещаются в отдельных определенным образом упорядоченных «панелях» (англ. *panels*, *facets* или *multiples*), что позволяет компактно и очень эффективно визуализировать многомерные данные. В англоязычной литературе для обозначения категоризованных графиков применяют такие термины, как *«faceted plots»*, *«trellis plots»*, *«lattice plots»* и *«small multiples»*[1]. Стоит отметить, что категоризованные графики можно создавать при помощи еще одного популярного графического пакета для `R` — `lattice`. Подробнее об этом пакете можно узнать в книге, написанной его автором — Дипайяном Саркаром (Sarkar 2008[2]).

7.1 Два способа организации панелей

В `ggplot2` есть две функции, выполняющие разбиение данных на подмножества и последующее их изображение в виде категоризованных графиков (рис. 7.1):

- `facet_grid()` — разбивает данные в соответствии с уровнями *двух* задаваемых пользователем качественных переменных. Комбинирование уровней этих двух переменных приводит к формированию сетки (англ. *grid*), в ячейках которой располагаются панели с данными из соответствующих подмножеств.

[1] На русском языке о *«small multiples»* можно почитать в книге: Яу Н. (2013) Искусство визуализации в бизнесе. Как представить сложную информацию простыми образами. Манн, Иванов и Фербер.

[2] Sarkar D. (2008) Lattice: Multivariate Data Visualization with R. Springer. См. также сайт `http://lmdvr.r-forge.r-project.org/`.

- `facet_wrap()` — разбивает данные в соответствии с уровнями *одной* задаваемой пользователем качественной переменной. Поскольку есть только одна такая качественная переменная, то панели с наблюдениями из соответствующих подмножеств образуют одномерную «ленту». Однако для экономии места эта «лента» далее как бы сворачивается послойно в двухмерную конструкцию (*«to wrap»* значит *«заворачивать»*, *«сворачивать»*).

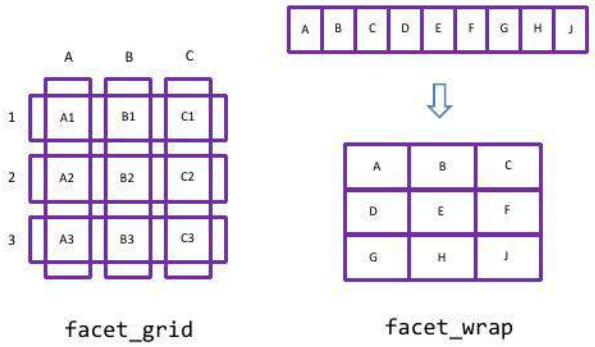


Рисунок 7.1. Организация панелей на категоризованных графиках, получаемых при помощи функций `facet_grid()` и `facet_wrap()`

7.2 Функция `facet_grid()`

Как было отмечено выше, функция `facet_grid()` служит для создания категоризованных графиков на двухмерной сетке. Структура этой сетки задается пользователем при помощи аргумента `facets` в виде `"facets = . ~ ."`. По левую и правую сторонам от тильды необходимо указать качественные переменные, по уровням которых происходит разбиение наблюдений на подмножества. Возможны следующие варианты:

- `"facets = . ~ A"` — наблюдения разбиваются на группы в соответствии с уровнями фактора `A`. Панели на графике выстраиваются в один ряд горизонтально (рис. 7.2). Такой подход полезен для сравнения наблюдений по их Y–координатам;

- `"facets = A ~ ."` — наблюдения также разбиваются на подмножества по уровням фактора `A`, но панели на графике выстраиваются в один ряд вертикально (рис. 7.3). Такой подход полезен для сравнения наблюдений по их X–координатам;

- `"facets = A ~ B"` — наблюдения разбиваются на подмножества по уровням двух факторов — `A` и `B`. Панели на графике располагаются в ячейках двухмерной сетки, в которой количество столбцов соответствует количеству уровней переменной `A`, а количество строк — количеству уровней переменной `B` (рис. 7.4). Если некоторые сочетания уровней переменных `A` и `B` в исходных данных отсутствуют, то соответствующие панели на графике будут изображены пустыми.

Допускается также указание нескольких переменных по ту или иную сторону от тильды (например, "facets = A + B ~ C"), однако на практике такой прием почти никогда не используется (особенно при работе с факторами с большим числом уровней).

Визуализация данных в виде категоризованных графиков с помощью функции facet_grid() напоминает представление данных в виде обычных в статистике таблиц сопряженности. Нередко в таких таблицах приводят также т. н. «маргинальные частоты», т. е. суммы частот отдельных ячеек таблицы по строкам и (или) по столбцам. Аналогичный прием может оказаться полезным и при работе с категоризованными графиками. Для отображения «маргинальных панелей» служит аргумент margins функции facet_grid(). Этот аргумент принимает либо логические значения (FALSE или TRUE), либо текстовые векторы с именами переменных, по уровням которых необходимо создавать маргинальные панели. Примеры использования аргумента margins представлены на рис. 7.5 и 7.6.

Код для рис. 7.2

```
qplot(Length, sqrt(Infection), data = dreissena) +
      facet_grid(facets = . ~ Lake)
```

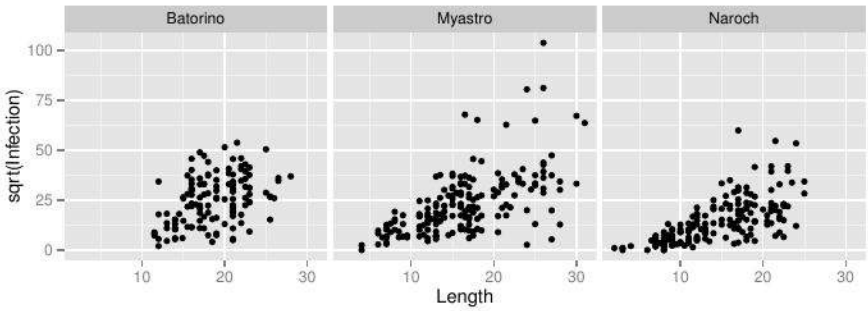


Рисунок 7.2. Пример категоризованного графика, на котором данные разбиты на группы в соответствии с уровнями одного фактора и панели упорядочены горизонтально

Код для рис. 7.3

```
qplot(Length, data = dreissena, geom = "histogram") +
      facet_grid(Lake ~ .)
# обратите внимание: имя аргумента facets при вызове
# функции facet_grid указывать необязательно
```

Код для рис. 7.4

```
qplot(Length, sqrt(Infection), data = dreissena) +
      facet_grid(Lake ~ Month)
```

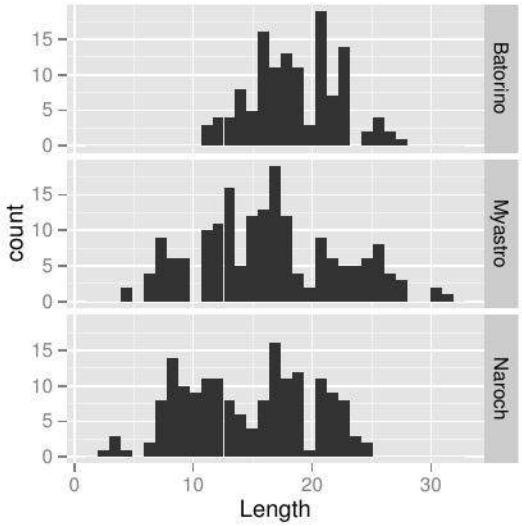


Рисунок 7.3. Пример категоризованного графика, на котором данные разбиты на группы в соответствии с уровнями одного фактора и панели упорядочены вертикально

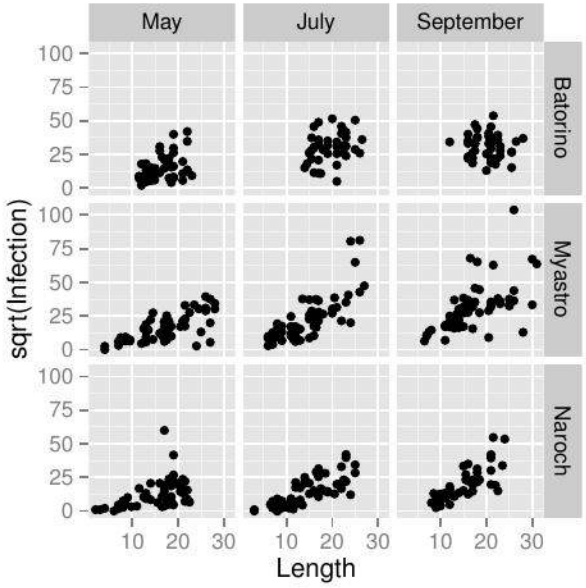


Рисунок 7.4. Пример категоризованного графика, на котором данные разбиты на группы в соответствии с уровнями двух факторов (`Lake` и `Month`)

Код для рис. 7.5

```
qplot(Length, sqrt(Infection),
      data = dreissena, alpha = I(0.4)) +
      facet_grid(facets = Lake ~ Month, margins = TRUE)
```

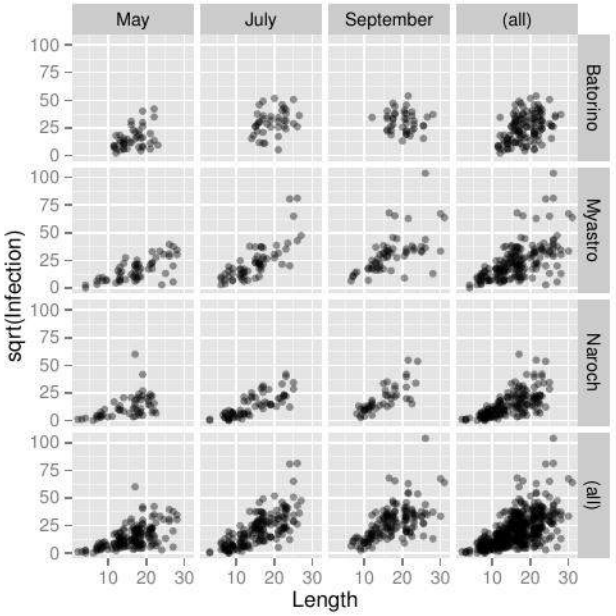


Рисунок 7.5. Пример категоризованного графика, на котором представлены «маргинальные панели» (обозначены как `(all)`). Для включения отображения этих панелей использован аргумент `margins = TRUE` функции `facet_grid()`

Код для рис. 7.6

```
qplot(Length, sqrt(Infection),
      data = dreissena, alpha = I(0.4)) +
      facet_grid(facets = Lake ~ Month,
                 margins = c("Month"))
```

Обратите внимание, например, на последний построенный нами график: по умолчанию все панели на нем имеют одинаковые X– и Y–оси. Использование одинаковых осей облегчает сравнение данных из разных групп. Однако в ряде случаев возникает необходимость использовать диапазоны значений, специфичные для данных в каждой конкретной группе. Добиться этого позволяет аргумент `scales` функции `facet_grid()`. Этот аргумент может принимать четыре возможных значения:

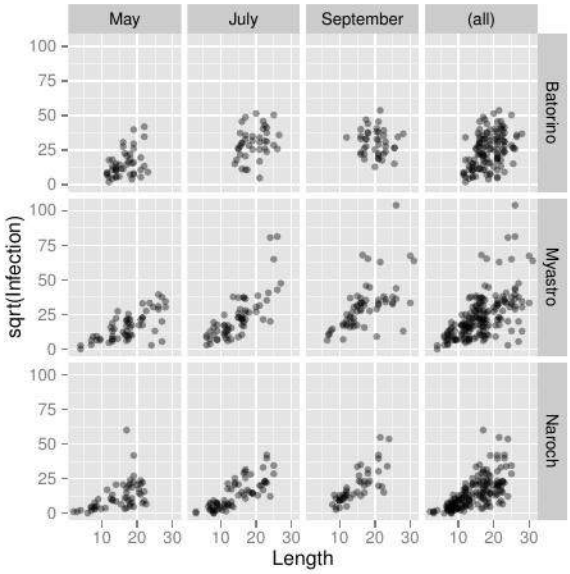


Рисунок 7.6. Пример категоризованного графика, на котором «маргинальные панели» представлены для уровней только одного фактора (`margins = c("Month")`)

- `"fixed"` — все панели имеют одинаковые координатные оси (принято по умолчанию);
- `"free_x"` — каждая панель имеет свой диапазон значений оси X;
- `"free_y"` — каждая панель имеет свой диапазон значений оси Y;
- `"free"` — каждая панель имеет свой диапазон значений как оси X, так и оси Y.

На рис. 7.7 приведен пример использования аргумента `scales` со значением `"free"`.

Код для рис. 7.7

```
qplot(Length, sqrt(Infection),
      data = dreissena, alpha = I(0.4)) +
      facet_grid(facets = Lake ~ Month, scales = "free")
```

С помощью аргумента `space`, который принимает те же значения, что и `scales`, мы можем еще больше подчеркнуть тот факт, что каждая панель имеет свои специфичные диапазоны координатных осей. Применение этого аргумента изменяет размер панели по высоте (`space = "free_y"`), ширине (`space = "free_x"`), или по обоим направлениям одновременно (`space = "free"`) пропорционально диапазонам значений соответствующих осей. На рис. 7.8 приведен пример изменения размера панелей по высоте.

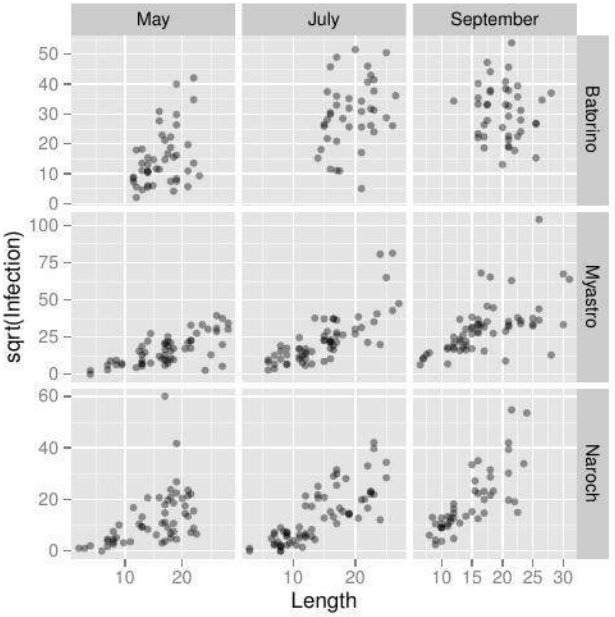


Рисунок 7.7. Пример категоризованного графика, на котором каждая панель имеет свой специфичный диапазон значений по обеим координатным осям

Код для рис. 7.8

```
qplot(Length, sqrt(Infection), data = dreissena,
      alpha = I(0.4)) + facet_grid(Lake ~ Month,
      scales = "free", space = "free")
```

Существует несколько способов изменить заголовки панелей на категоризованных графиках. Простейший из них состоит в том, чтобы изначально присвоить необходимые метки уровням качественных переменных, по которым происходит разбиение наблюдений на отдельные подмножества. Так, если бы на рис. 7.8 мы захотели заменить заголовки `May`, `July` и `September` на `M`, `J` и `S`, то для этого достаточно было бы выполнить команду `levels(dreissena[, "Month"]) <- c("M", "J", "S")`[3], а затем повторить выполнение приведенного выше кода для этого рисунка.

Однако для более тонкой настройки заголовков панелей рекомендуется использовать аргумент `labeller` функции `facet_grid()`. На этот аргумент необходимо подать одну из соответствующих вспомогательных функций из пакета `ggplot2`. По умолчанию на `labeller` подается функция `label_value` (без обычных круглых скобок!), что приводит к отображению меток уровней качественных переменных, по которым происходит

[3] Важно понимать, что тем самым будет изменена исходная таблица с данными. Если такой эффект нежелателен, то изменения стоит вносить в предварительно созданную копию этой таблицы.

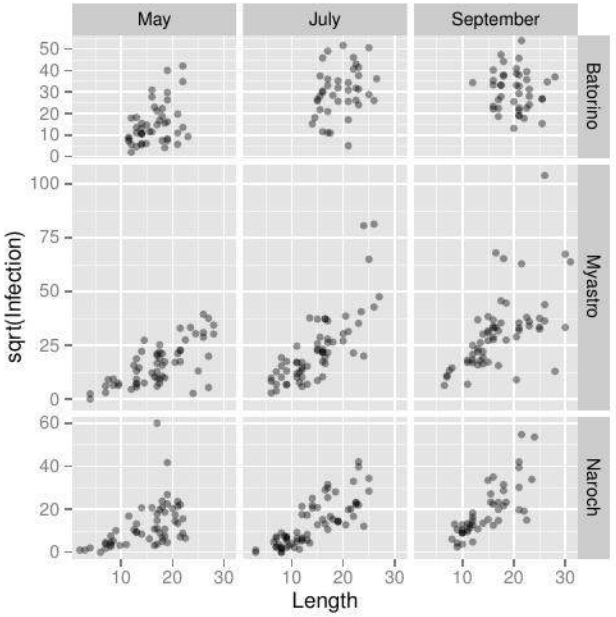


Рисунок 7.8. Пример категоризованного графика, на котором высота каждой панели пропорциональна диапазону значений оси Y

разбиение данных на подмножества. При необходимости указать в заголовке не только метки уровней, но и имена самих переменных на аргумент `labeller` следует подать функцию `label_both`. Пример использования этой функции приведен на рис. 7.9, где заголовки панелей включают как имя качественной переменной `Lake`, так и метки ее уровней (`Batorino`, `Myastro`, `Naroch`), отделенные двоеточием.

Код для рис. 7.9

```
qplot(Length, sqrt(Infection),
      data = dreissena, alpha = I(0.4)) +
      facet_grid(facets = . ~ Lake, labeller = label_both)
```

Иногда возникает необходимость привести в заголовках панелей математические выражения. Проще всего это можно сделать, добавив в таблицу с исходными данными новую качественную переменную, метки уровней которой представляют собой подлежащие парсингу выражения (с примерами использования таких выражений мы уже сталкивались в подразд. 4.8.8; подробнее см. справочные файлы по функциям `plotmath()` и `expression()`). Далее при построении категоризованного графика данные необходимо разбить на подмножества в соответствии с уровнями этой новой переменной, а при вызове функции `facet_grid()` — подать на аргумент `labeller` вспомогательную функцию `label_parsed` (рис. 7.10). В некоторых случаях математические выражения в заголовках панелей категоризованных графиков можно также создать, подав на `labeller` функ-

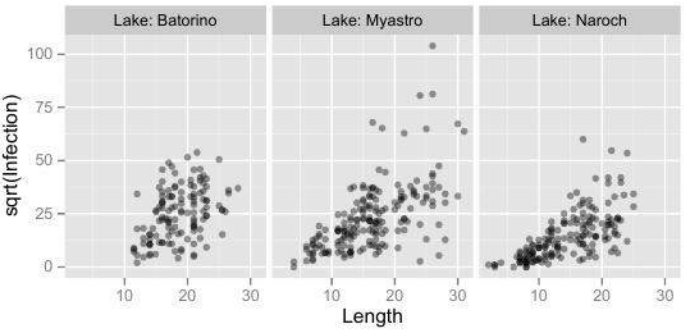


Рисунок 7.9. Пример категоризованного графика, заголовки панелей которого заданы при помощи аргумента `labeller` и вспомогательной функции `label_both`

цию `label_bquote()`, которая работает аналогично базовой R–функции `bquote()` (примеры здесь не приводятся; подробнее см. `?label_bquote` и `?bquote`).

Код для рис. 7.10

```
dreissena$dummy <- factor(dreissena$Lake,
     labels = c("x == frac(x[i], n)", "y == frac(y[i], n)",
                "z == frac(z[i], n)"))
qplot(Length, sqrt(Infection),
      data = dreissena, alpha = I(0.4)) +
facet_grid(facets = . ~ dummy, labeller = label_parsed)
```

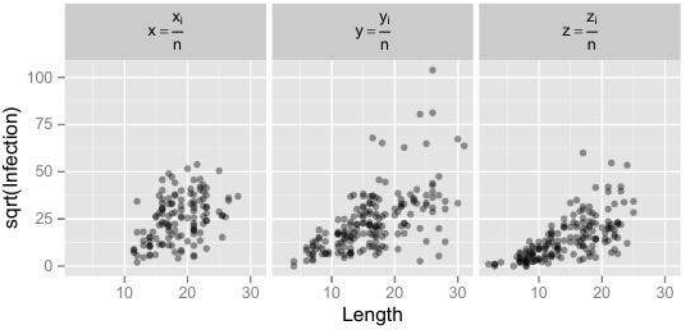


Рисунок 7.10. Пример категоризованного графика, заголовки панелей которого включают математические выражения, заданные при помощи аргумента `labeller` и вспомогательной функции `label_parse`

7.3 Функция `facet_wrap()`

Функция `facet_wrap()` служит для создания категоризованных графиков в соответствии с уровнями одной качественной переменной. Эту переменную указывают посредством аргумента `facets`, используя обычный для `R` синтаксис формул (например, `facets = ~ Month`). Одномерная «лента» из нескольких панелей, образующаяся в результате разбиения данных на отдельные подмножества, далее «укладывается» в несколько слоев (см. рис. 7.1). Количество «строк» и «столбцов» в итоговом графике можно контролировать при помощи аргументов `nrow` и `ncol` соответственно (рис. 7.11 и 7.12).

Код для рис. 7.11 и 7.12

```
qplot(Length, sqrt(Infection),
      data = dreissena, alpha = I(0.4)) +
      facet_wrap(facets = ~ Month, ncol = 1)
qplot(Length, sqrt(Infection),
      data = dreissena, alpha = I(0.4)) +
      facet_wrap(facets = ~ Month, nrow = 2)
```

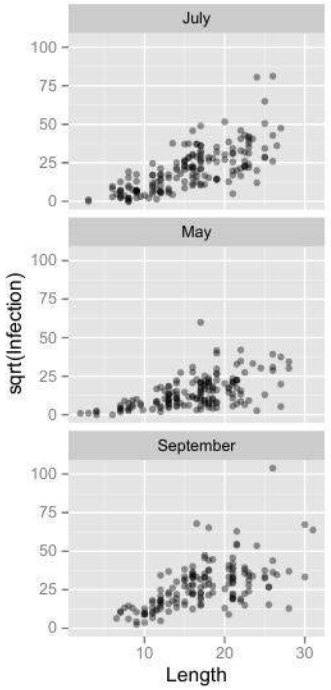


Рисунок 7.11. Пример использования функции `facet_wrap()` в сочетании с аргументом `ncol = 1`

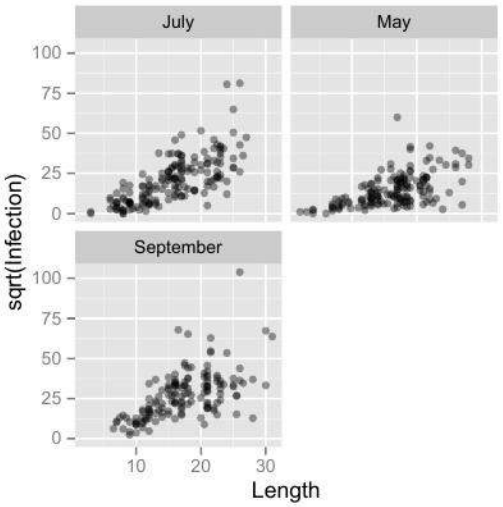


Рисунок 7.12. Пример использования функции `facet_wrap()` в сочетании с аргументом `nrow = 2`

Интересно, что хотя функция `facet_wrap()` предназначена для визуализации данных, разбитых на подмножество в соответствии с уровнями одной качественной переменной, на ее аргумент `facets` можно подать несколько переменных. В результате этого `R` «на лету» создаст новую качественную переменную, уровни которой будут представлять собой все возможные сочетания уровней соответствующих исходных переменных (рис. 7.13).

Код для рис. 7.13

```
qplot(Length, sqrt(Infection),
      data = dreissena, alpha = I(0.4)) +
      facet_wrap(facets = ~ Lake + Month)
```

Следует отметить, что, в отличие от `facet_grid()`, у функции `facet_wrap()` нет аргумента `labeller`, позволяющего создавать пользовательские заголовки панелей графика. Поэтому соответствующие заголовки панелей должны изначально присутствовать в таблице с данными и служить метками уровней качественной переменной, по которым происходит разбиение наблюдений на подмножества. В остальном аргументы функций `facet_wrap()` и `facet_grid()` одинаковы. Так, например, при помощи уже знакомого нам аргумента `scales` можно настраивать диапазоны значений, отображаемых по координатным осям. Аргумент `drop` (ранее не рассматривался) со значением `TRUE` позволяет отбросить отсутствующие в данных уровни анализируемой качественной переменной. Если же этому аргументу присвоить значение `FALSE`, то при отсутствии некоторых уровней соответствующие панели на графике окажутся пустыми.

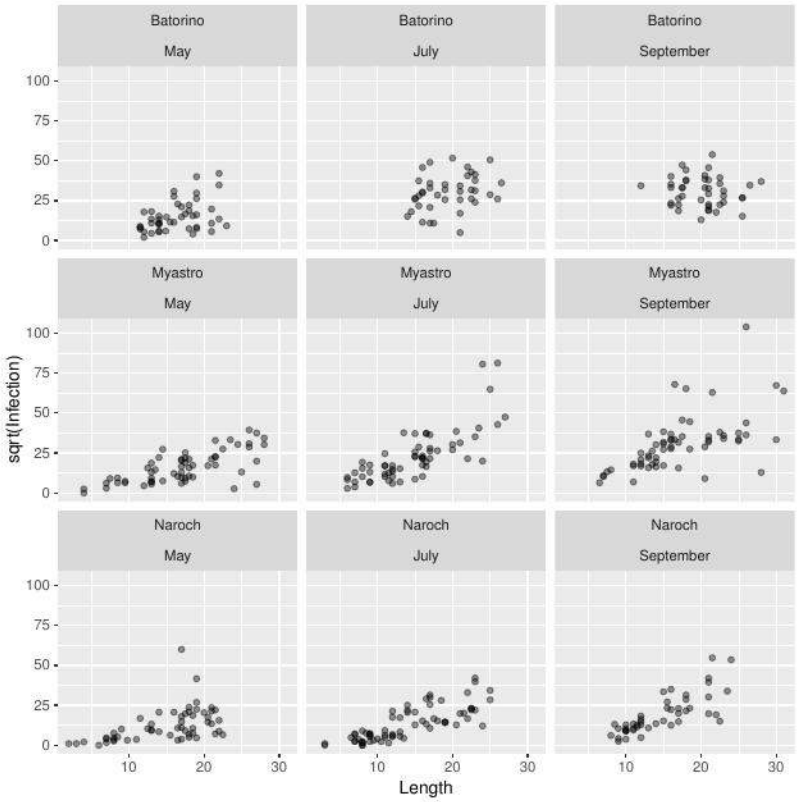


Рисунок 7.13. Пример использования функции `facet_wrap()` в сочетании с аргументом `facets`, на который были поданы две качественные переменные (`Lake` и `Month`)

Глава 8

Подготовка графиков к публикации

В предыдущих главах мы рассмотрели многочисленные функции из пакета `ggplot2`, предназначенные для построения статистических графиков. Внешний вид многих элементов графиков, получаемых с помощью этих функций, выбирается автоматически (например, цвет и размер шрифта подписей, цвет и длина отрезков, обозначающих деления осей, цвет фона основной части графика и т. д.). Однако нередко возникает необходимость выполнить более тонкую настройку отдельных деталей графика (например, в соответствии с требованиями журнала, в котором планируется опубликовать результаты исследования, или в соответствии с корпоративными требованиями, согласно которым график должен быть выполнен в определенных цветах, и т. д.). В данной главе вы узнаете об основных предназначенных для этого приемах. Кроме того, мы рассмотрим способы компоновки нескольких графиков на одном рисунке, а также средства экспорта графиков из среды `R`.

8.1 Стили

Внешний вид графиков в `ggplot2` определяется системой *стилей*, или *«тем»* (англ. *themes*). Есть несколько стандартных стилей, которые задаются следующими функциями:

- `theme_gray()` (синоним — `theme_grey()`) — принятый по умолчанию «серый» стиль, с которым мы уже неоднократно встречались в рассмотренных ранее примерах;
- `theme_dark()` — «темный» стиль;
- `theme_bw()` — черно–белый стиль;
- `theme_light()` — «светлый» стиль;
- `theme_miminal()` — минималистический стиль;
- `theme_classic()` — «классический» стиль;

- `theme_linedraw()` — стиль, похожий на `theme_light()`, однако с более выраженными линиями координатной сетки.

Для применения того или иного стандартного стиля к некоторому графику необходимо обычным образом (т. е. посредством оператора +) добавить соответствующую функцию к коду, определяющему этот график. Рекомендуется добавлять эту функцию в самом конце блока с соответствующим кодом. Поскольку «серый» стиль принят по умолчанию, то для его применения функцию `theme_gray()` в явном виде вызывать нет необходимости. Примеры эффектов перечисленных выше `theme`–функций приведены на рис. 8.1.

Код для рис. 8.1

```
p <- ggplot(data = dreissena, aes(Length, sqrt(Infection))) +
            geom_point(alpha = 0.5)
p + theme_gray() + ggtitle("theme_gray()")
p + theme_bw() + ggtitle("theme_bw()")
p + theme_light() + ggtitle("theme_light()")
p + theme_linedraw() + ggtitle("theme_linedraw()")
p + theme_minimal() + ggtitle("theme_minimal()")
p + theme_classic() + ggtitle("theme_classic()")
```

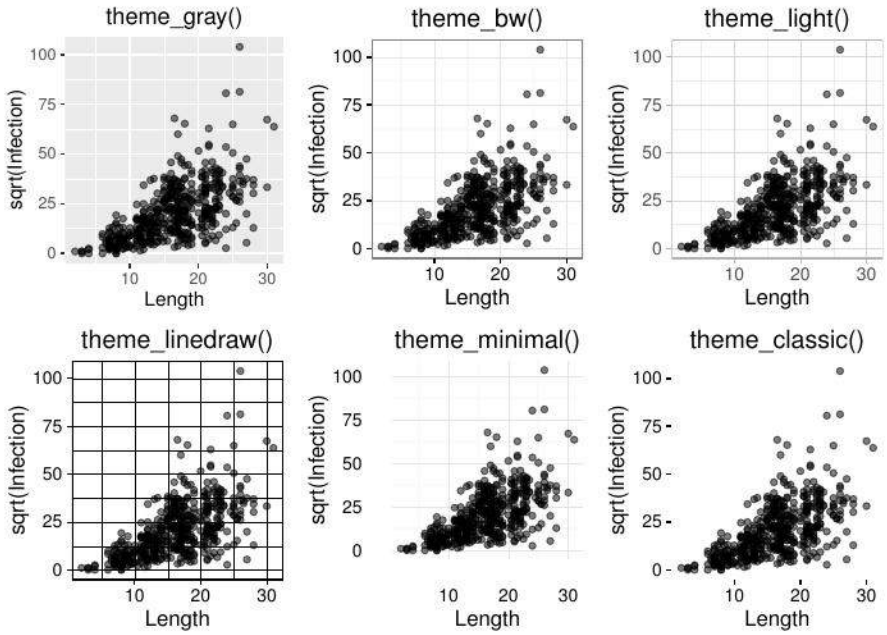


Рисунок 8.1. Стандартные стили `ggplot2`, примененные к одним и тем же данным

Пользователь имеет возможность создавать и свои собственные стили. Для этого служит функция `theme()`. У данной функции очень много аргументов, определяющих внешний вид разных элементов графика. Значения

этих аргументов задаются при помощи ряда вспомогательных функций с использованием следующего синтаксиса:

```
theme(аргумент_1 = вспомогательная_функция_1(...),
      аргумент_2 = вспомогательная_функция_2(...),
      ...)
```

К таким вспомогательным функциям относятся:

- `element_text()` — служит для формирования значений всех аргументов `theme()`, определяющих свойства текстовых элементов графика (например, заголовок графика, подписи осей, заголовок легенды). Функция `element_text()` имеет следующие аргументы:
  - `family` — семейство шрифта;
  - `face` — оформление шрифта (обычный — `"plain"`, наклонный — `"italic"`, полужирный — `"bold"` или наклонный полужирный — `"bold.italic"`);
  - `colour` (или `color`) — цвет;
  - `size` — размер;
  - `hjust` и `vjust` — горизонтальное и вертикальное смещение текста соответственно;
  - `angle` — угол наклона;
- `element_line()` — формирует значения всех аргументов `theme()`, которые определяют свойства используемых в оформлении графика линий (например, линии координатной сетки). Имеет следующие аргументы:
  - `colour` (или `color`) — цвет;
  - `size` — толщина;
  - `linetype` — тип линии;
  - `lineend` — внешний вид концов линии (круглые — `"round"`, усеченные квадратные — `"butt"` или квадратные — `"square"`);
- `element_rect()` — формирует значения всех аргументов `theme()`, которые определяют свойства используемых в оформлении графика прямоугольников (например, прямоугольной области, в пределах которой изображаются данные). Имеет следующие аргументы:
  - `fill` — цвет, которым закрашивается прямоугольник;
  - `colour` (или `color`) — цвет линии, окаймляющей прямоугольник;
  - `size` — толщина окаймляющей линии;
  - `linetype` — тип окаймляющей линии;
- `element_blank()` — служит для отключения действия *любого* из аргументов `theme()`. Например, выполнение команды `theme(plot.title = element_blank())` приведет к тому, что у графика не будет заголовка;

- `unit()` из базового пакета `grid` — формирует значения аргументов `theme()`, отвечающих за размер тех или иных элементов графика (например, длина отрезков, соответствующих делениям осей, или ширина зазора между легендой и основной частью графика). Эта функция имеет два основных аргумента: `x` — числовое значение, и `units` — единицы измерения. Например, если размер некоторого элемента графика должен составлять 1 см, то необходимо использовать команду `units(1, "cm")` (подробнее об этой функции можно узнать в ее справочном файле, доступном по команде `?unit`);

- `rel()` — функция из пакета `ggplot2`, которая, подобно `unit()`, задает размер элементов графика, однако выражает его в относительных единицах (см. примеры ниже).

В табл. 8.1 перечислены аргументы функции `theme()`, определяющие свойства разных «строительных блоков» графика, а также соответствующие им вспомогательные функции (заметьте, что в ряде случаев вспомогательные функции не требуются — значения аргументов задаются пользователем непосредственно):

Таблица 8.1. Аргументы функции `theme()` и соответствующие им вспомогательные функции

Аргумент	Элемент графика	Вспомогательная функция
`axis.title`	подписи осей	`element_text()`
`axis.title.x`	подпись оси X	`element_text()`
`axis.title.y`	подпись оси Y	`element_text()`
`axis.text`	подписи делений осей	`element_text()`
`axis.text.x`	подписи делений оси X	`element_text()`
`axis.text.y`	подписи делений оси Y	`element_text()`
`axis.ticks`	деления осей	`element_line()`
`axis.ticks.x`	деления оси X	`element_line()`
`axis.ticks.y`	деления оси Y	`element_line()`
`axis.ticks.length`	длина делений осей	`unit()`
`axis.line`	линии координатных осей	`element_line()`
`axis.line.x`	линия координатной оси X	`element_line()`
`axis.line.x`	линия координатной оси Y	`element_line()`
`legend.background`	фон легенды	`element_rect()`
`legend.margin`	пространство вокруг легенды	`unit()`
`legend.key`	прямоугольное пространство под ключами легенды	`element_rect()`
`legend.key.size`	размер пространства под ключами легенды	`unit()`
`legend.key.hight`	высота пространства под ключами легенды	`unit()`

Таблица 8.1: *продолжение*

Аргумент	Элемент графика	Вспомогательная функция
`legend.key.width`	ширина пространства под ключами легенды	`unit()`
`legend.text`	текст, используемый в легенде	`element_text()`
`legend.text.align`	выравнивание текста в легенде	от 0 (влево) до 1 (вправо)
`legend.title`	заголовок легенды	`element_text()`
`legend.title.align`	выравнивание заголовка легенды	от 0 (влево) до 1 (вправо)
`legend.position`	расположение легенды	`"none"` (нет легенды), `"left"` (слева), `"right"` (справа), `"top"` (вверху), `"bottom"` (внизу)
`legend.direction`	направление легенды	`"horizontal"` (горизонтальное), `"vertical"` (вертикальное)
`legend.box`	расположение блока с несколькими легендами	`"horizontal"` (горизонтальное), `"vertical"` (вертикальное)
`legend.box.just`	выравнивание каждой легенды в пределах общего блока	`"left"` (слева), `"right"` (справа), `"top"` (вверху), `"bottom"` (внизу)
`panel.background`	цвет фона основной части графика	`element_rect()`
`panel.border`	цвет линии, окаймляющей основную часть графика	`element_rect()`
`panel.margin`	ширина зазора между панелями категоризованного графика	`unit()`
`panel.margin.x`	ширина зазора между панелями категоризованного графика по горизонтали	`unit()`
`panel.margin.y`	ширина зазора между панелями категоризованного графика по вертикали	`unit()`
`panel.grid`	линии координатной сетки	`element_line()`

Таблица 8.1: *продолжение*

Аргумент	Элемент графика	Вспомогательная функция
`panel.grid.major`	основные линии координатной сетки	`element_line()`
`panel.grid.minor`	вспомогательные линии координатной сетки	`element_line()`
`panel.grid.major.x`	вертикальные основные линии координатной сетки	`element_line()`
`panel.grid.major.y`	горизонтальные основные линии координатной сетки	`element_line()`
`panel.grid.minor.x`	вертикальные вспомогательные линии координатной сетки	`element_line()`
`panel.grid.minor.y`	горизонтальные вспомогательные линии координатной сетки	`element_line()`
`panel.ontop`	возможность разместить основную часть графика поверх слоев с данными (имеет смысл только тогда, когда соответствующая область прозрачная или пустая)	`TRUE` или `FALSE`
`plot.background`	цвет фона вокруг основной части графика	`element_rect()`
`plot.title`	заголовок графика	`element_text()`
`plot.margin`	ширина пространства вокруг всего графика	`unit()`
`strip.background`	цвет фона под заголовками панелей категоризованного графика	`element_rect()`
`strip.text`	заголовки панелей категоризованного графика	`element_text()`
`strip.text.x`	заголовки панелей категоризованного графика по горизонтали	`element_text()`
`strip.text.y`	заголовки панелей категоризованного графика по вертикали	`element_text()`

Таблица 8.1: *продолжение*

Аргумент	Элемент графика	Вспомогательная функция
`strip.switch.pad.grid`	ширина зазора между заголовками панелей и координатными осями при активации аргумента `switch` функции `facet_grid()`	`unit()`
`strip.switch.pad.wrap`	ширина зазора между заголовками панелей и координатными осями при активации аргумента `switch` функции `facet_wrap()`	`unit()`

Демонстрация примеров работы каждого из перечисленных в табл. 8.1 аргументов функции `theme()` легко могла бы растянуться на десятки страниц. Поэтому с целью экономии места мы рассмотрим только наиболее часто встречающиеся задачи, которые можно решить с использованием этих аргументов[1].

Начнем с настройки внешнего вида основной части графика, т. е. прямоугольной области, в которой изображены данные. Предположим, что нам необходимо изменить цвет и толщину линии, окаймляющей эту область. Как следует из табл. 8.1, это можно сделать двумя способами: с помощью аргумента `panel.border` или аргумента `panel.backgound` функции `theme()`. Значения обоих этих аргументов формируются функцией `element_rect()`.

При использовании аргумента `panel.border` необходимо указать желаемые параметры окаймляющей линии (`colour` и `size`), а также присвоить значение `NA` параметру `fill` — иначе основная часть графика окажется залитой белым цветом (принят по умолчанию), что скроет данные и координатную сетку (см. рис. 8.2). При использовании же аргумента `panel.backgound` присваивать значение `NA` параметру `fill` нет необходимости — по умолчанию фон основной части графика будет закрашен «фирменным» для `ggplot2` светло–серым цветом и все элементы графика будут видны. На рис. 8.3 приведен пример использования аргумента `panel.backgound` для изменения этого цвета, а также примеры применения аргументов `panel.grid.major` и `panel.grid.minor` для настройки внешнего вида основных и вспомогательных линий координатной сетки.

Код для рис. 8.2

```
p <- ggplot(data = dplyr::filter(dreissena,
            Lake == "Naroch" & Site == "S9"),
            aes(Length, sqrt(Infection))) +
            geom_point(size = 3)
```

[1] Дополнительные примеры можно найти на странице с официальной документацией по функции `theme()`: `http://docs.ggplot2.org/current/theme.html`.

```
p + theme(panel.border =
          element_rect(size = 2, colour = "red"))
p + theme(panel.border =
          element_rect(fill = NA, size = 2, colour = "red"))
```

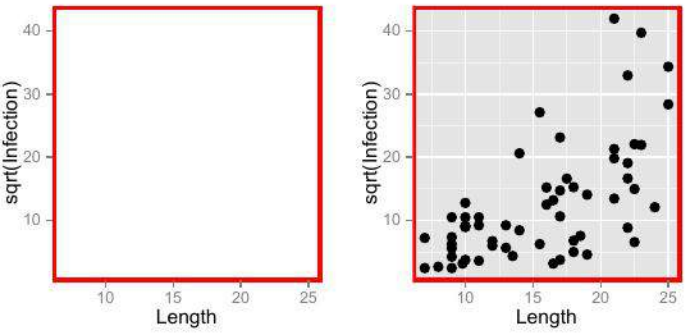


Рисунок 8.2. Настройка внешнего вида линии, окаймляющей основную часть графика. *Слева:* параметры линии заданы как `panel.border = element_rect(size = 2, colour = "red")`, в результате чего возник нежелательный эффект: основная часть графика оказалась залитой белым цветом и данные стали не видны. *Справа:* заливка основной части графика белым цветом отключена командой `panel.border = element_rect(fill = NA, size = 2, colour = "red")`

Код для рис. 8.3

```
# Объект p определен как на рис. 8.2
p + theme(panel.background =
          element_rect(fill = "lightblue",
                       color = "red", size = 2))
p + geom_point(color = "yellow", size = 3) +
              theme(panel.background =
    element_rect(fill = "black", color = "gray50"),
                    panel.grid.major =
    element_line(color = "gray30", size = 2),
                    panel.grid.minor =
    element_line(linetype = 2, color = "gray30", size = 1))
```

Для изменения размера и (или) цвета шрифта, используемого в заголовке графика и в подписях координатных осей X и Y, служат аргументы `plot.title`, `axis.title.x` и `axis.title.y` функции `theme()` соответственно. Примеры использования этих аргументов приведены на рис. 8.4. Похожим образом можно изменить также шрифт, используемый в подписях делений координатных осей: для этого служат аргументы `axis.text.x` и `axis.text.y`.

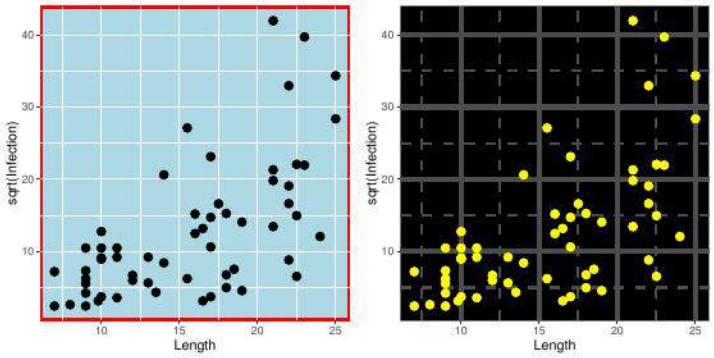


Рисунок 8.3. *Слева:* использование аргумента `panel.background` функции `theme()` для настройки внешнего вида окаймляющей линии и цвета заливки основной части графика. *Справа:* аргумент `panel.background` применен для закрашивания основной части графика черным цветом, а аргументы `panel.grid.major` и `panel.grid.minor` — для изменения внешнего вида линий координатной сетки

Код для рис. 8.4

```
# Объект p определен как на рис. 8.2
pp <- p + theme_minimal() +
      labs(title = "Заголовок графика",
           x = "Подпись оси X", y = "Подпись оси Y")
pp + theme(plot.title =
     element_text(size = rel(2), colour = "blue"),
           axis.title.x =
     element_text(size = 14, color = "red"),
           axis.title.y =
     element_text(size = 18, color = "green"))
```

Как мы уже неоднократно видели в предыдущих примерах, присваивание эстетических атрибутов геометрическим фигурам, которые используются для отображения данных на графике, обычно сопровождается автоматическим созданием легенды. Такой легенды вполне может оказаться достаточно. Однако часто требуется выполнить более тонкую настройку ее элементов. Для этого служат несколько аргументов функции `theme()`, в названиях которых есть приставка `legend`.

Так, с помощью аргумента `legend.position` можно изменить расположение легенды (рис. 8.5) или полностью ее отключить (`legend.position = "none"`). Заметьте, что аргумент `legend.position` способен также принимать вектор из двух чисел (от 0 до 1), задающих координаты расположения легенды (например, команда `theme(legend.position = c(0.5, 0.5))` изобразит легенду в центре основной части графика). Шрифт заголовка легенды, а также шрифт, используемый для обозначения отдельных ключей легенды, настраивают при помощи аргументов `legend.title` и `legend.text` соответственно (рис. 8.6).

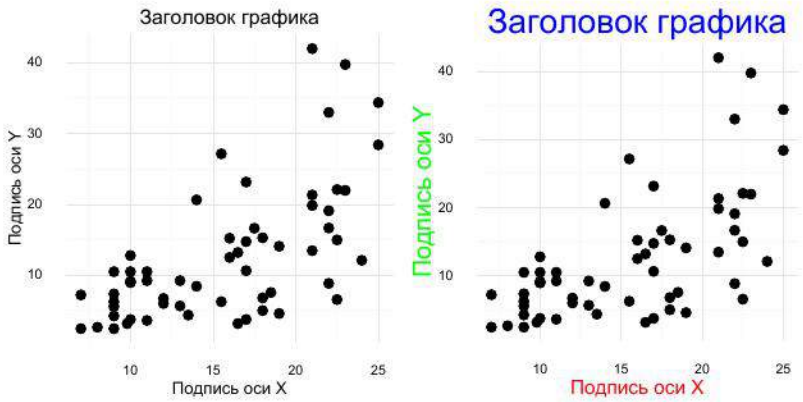


Рисунок 8.4. *Слева:* пример заголовка графика и подписей координатных осей, внешний вид которых задан в соответствии с автоматическими настройками. *Справа:* размер и цвет шрифта заголовка графика и подписей координатных осей изменены при помощи аргументов `plot.title`, `axis.title.x` и `axis.title.y` функции `theme()`. Заметьте, что для присвоения идентичных свойств подписям координатных осей можно просто воспользоваться аргументом `axis.title`, а не вызывать отдельно аргументы `axis.title.x` и `axis.title.y`

Код для рис. 8.5

```
(bp <- ggplot(data = dreissena, aes(Lake, Length,
              fill = Month)) +
       geom_boxplot() )
bp + theme(legend.position = "top")
```

Код для рис. 8.6

```
# Объект bp определен как на рис. 8.5
bpp <- bp + scale_fill_discrete(name = "Месяц:",
            labels = c("Май", "Июль", "Сентябрь"))
bpp + theme(legend.title =
            element_text(size = 18, colour = "blue"),
            legend.text = element_text(size = 14,
                                       colour = "red"))
```

Легенда и ее содержимое размещаются в пределах отдельного модуля прямоугольной формы. По умолчанию этот содержащий легенду прямоугольник невидим, однако при желании его можно сделать видимым, закрасив тем или иным цветом или выделив при помощи окаймляющей линии — для этого служит аргумент `legend.background` функции `theme()` (рис. 8.7).

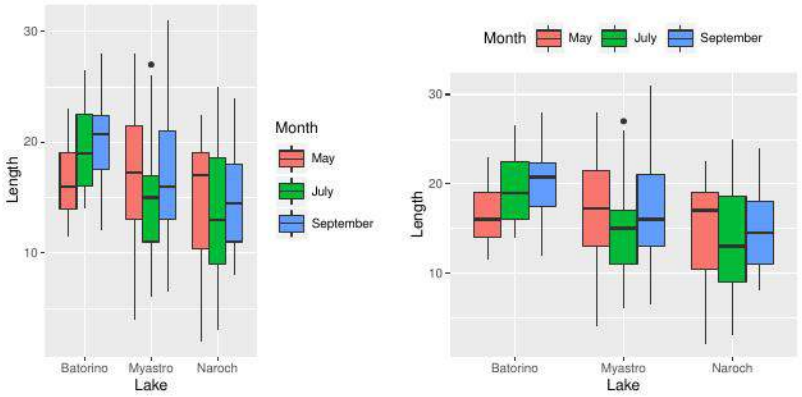


Рисунок 8.5. *Слева:* расположение и внешний вид легенды графика заданы в соответствии с автоматическими настройками. *Справа:* расположение легенды изменено при помощи аргумента `legend.position = "top"` функции `theme()`

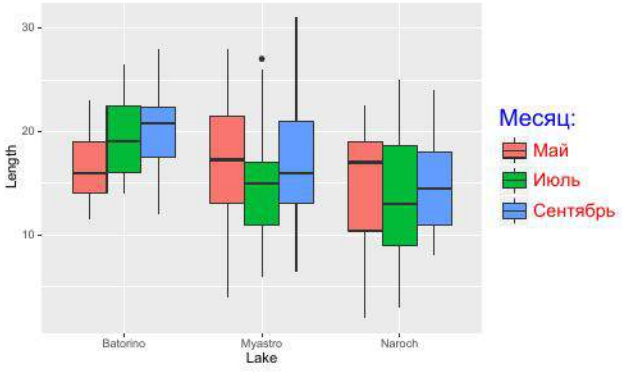


Рисунок 8.6. Пример изменения размера и цвета шрифтов, используемых в легенде графика

Код для рис. 8.7

```
# Объект bpp определен как на рис. 8.6
bpp + theme(legend.background =
      element_rect(colour = "darkblue", size = 2,
                   fill = "lightblue"))
bpp + theme(legend.background =
      element_rect(colour = "darkblue", size = 2,
      fill = "lightblue"), legend.margin = unit(1, "cm"))
```

Внешний вид ключей легенды настраивается с помощью аргументов функции `theme()`, содержащих в своем названии `"legend.key"`, — см. пример на рис. 8.8.

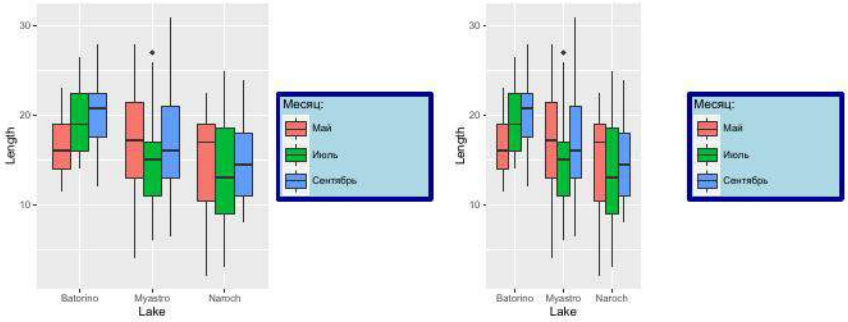


Рисунок 8.7. *Слева:* внешний вид прямоугольника, содержащего легенду графика, изменен с помощью аргумента `legend.background` функции `theme()`. *Справа:* расстояние между модулем легенды и основной частью графика изменено с помощью аргумента `legend.margin` функции `theme()`

Код для рис. 8.8

```
# Объект bpp определен как на рис. 8.6
bpp + theme(legend.key = element_blank(),
            legend.key.size = unit(2, "cm"))
```

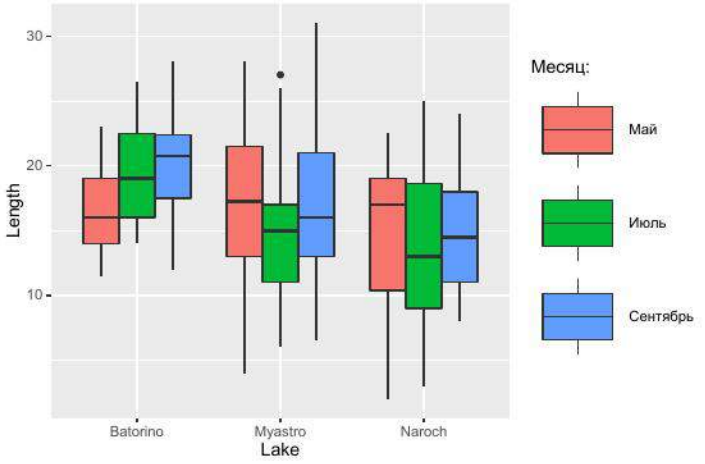


Рисунок 8.8. Размер ключей легенды увеличен с помощью аргумента `legend.key.size` функции `theme()`. Кроме того, задаваемый автоматически фон светло–серого цвета под каждым из ключей отключен при помощи аргумента `legend.key = element_blank()`

Наконец, рассмотрим, как можно настроить внешний вид заголовков панелей категоризованных графиков. Для этого служат аргументы функции `theme()`, в названии которых есть приставка `strip`. Примеры использования некоторых из этих аргументов приведены на рис. 8.9.

Код для рис. 8.9

```
ggplot(data = dreissena, aes(Lake, Length)) +
    geom_boxplot() + facet_wrap(~Month) +
    theme(strip.background =
          element_rect(fill = "darkblue",
          colour = "yellow", size = 1),
          strip.text =
          element_text(colour = "white",
          size = rel(2), face = 3))
```

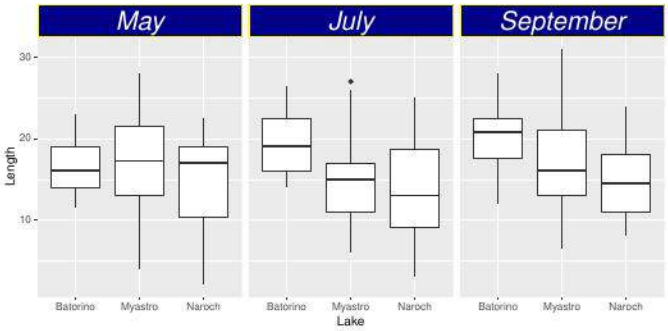


Рисунок 8.9. Цвет фона, а также ширина и цвет окаймляющей линии заголовков панелей категоризованного графика изменены с помощью аргумента `strip.background` функции `theme()`. Кроме того, с помощью аргумента `strip.text` изменен шрифт, которым выполнены эти заголовки

Как было отмечено в начале этой главы, с помощью функции `theme()` можно легко создавать свои собственные стили. В качестве примера рассмотрим следующий стиль, который вполне подойдет для подготовки графиков, предназначенных для публикации в научных журналах:

```
science_theme <-
    theme(axis.line.x =
          element_line(size = 1, color = "black"),
          axis.line.y =
          element_line(size = 1, color = "black"),
          axis.title =
          element_text(size = rel(1.5), face = "bold"),
          axis.text =
          element_text(size = rel(1.2), color = "black"),
          axis.ticks =
          element_line(colour = "black"),
          panel.background =
          element_rect(fill = "white", colour = NA),
          panel.border =
          element_rect(fill = NA, colour = NA),
          panel.grid.major =
```

```
        element_line(colour = "grey80", size = 0.5),
        panel.grid.minor = element_blank(),
        legend.title =
        element_text(face = "bold", size = rel(1.2)),
        legend.text = element_text(size = rel(1.2)),
        legend.key.size = unit(0.8, "cm"),
        legend.key = element_blank(),
        legend.position = "bottom",
        legend.margin = unit(0.5, "cm")
)
```

Как видно на рис. 8.10, вне зависимости от характера данных, к которым применяется этот новый стиль, мы всегда будем получать единообразно выглядящие графики.

Код для рис. 8.10

```
qplot(x = Lake, y = Length, data = dreissena,
      fill = Month, geom = "boxplot") +
      science_theme
qplot(x = Length, y = sqrt(Infection),
      data = dreissena, geom = "point",
      colour = Lake) + science_theme
```

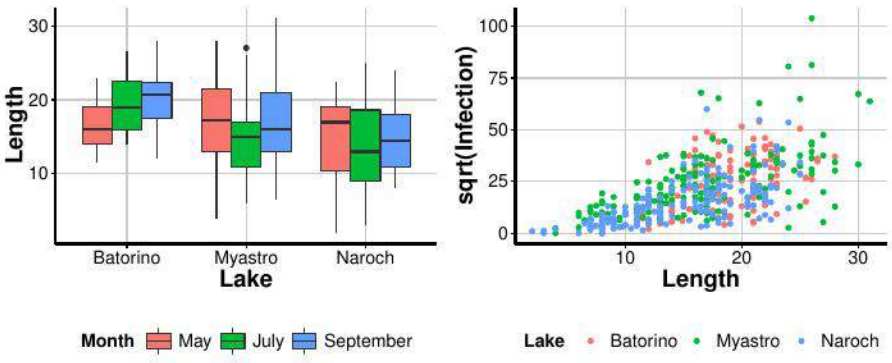


Рисунок 8.10. Пример применения пользовательского стиля к графикам разного типа

Создание собственного стиля может потребовать довольно кропотливой работы, сопровождающейся множеством проб и ошибок. К счастью, стандартных стилей `ggplot2` обычно оказывается вполне достаточно. Кроме того, при необходимости можно воспользоваться многочисленными готовыми стилями, созданными другими пользователями R и опубликованными в виде отдельных пакетов. Одним из наиболее популярных таких пакетов является `ggthemes`. На момент написания этой книги `ggthemes` содержал 14 стилей (рис. 8.11):

- `theme_base()`: напоминает стиль стандартных графиков R (см. также `theme_par()` ниже);
- `theme_calc()`: стиль графиков из LibreOffice Calc;
- `theme_economist()`: стиль графиков из журнала The Economist;
- `theme_excel()`: напоминает графики Microsoft Excel образца 2003 г.;
- `theme_few()`: стиль, созданный на основе рекомендаций из работы Few (2008[2]);
- `theme_fivethirtyeight()`: стиль графиков из известного блога `http://fivethirtyeight.com`;
- `theme_gdocs()`: стиль графиков Google Docs;
- `theme_pc()`: стиль графиков Highcarts JS[3];
- `theme_par()`: стиль на основе текущих значений графических параметров R, заданных функцией `par()`;
- `theme_pander()`: стиль графиков, реализованных в пакете `pander`[4];
- `theme_solarized()`: графики на основе цветовой палитры `solarized`[5];
- `theme_stata()`: стиль графиков статистической программы Stata;
- `theme_tufte()`: минималистический стиль графиков, описанный в известной книге Tufte (2001[6]);
- `theme_wsj()`: стиль графиков из журнала The Wall Street Journal.

Каждому из перечисленных стилей соответствует своя цветовая палитра (например, `scale_color_economist()`, `scale_color_wsj()`, `scale_fill_pander()` и т. д.). Кроме того, в состав пакета `ggthemes` входит ряд функций для определения формы символов, хорошо сочетающихся с тем или иным стилем (например, `scale_shape_calc()`, `scale_shape_tableau()`, `scale_shape_tremmel()` и т. д.). Подробнее об этих функциях можно узнать на странице `http://bit.ly/2bmbjm3`, а также в соответствующих справочных файлах.

Код для рис. 8.11

```
library(ggthemes)
p <- qplot(x = Length, y = sqrt(Infection),
           data = dreissena, geom = "point",
           colour = Lake)
p + theme_tufte() + ggtitle("theme_tufte()")
```

[2] Few S (2008) Practical Rules for Using Color in Charts. Perceptual Edge. Адрес доступа: `http://bit.ly/18SQJSn`.

[3] См. `http://www.highcharts.com`.

[4] См. `http://rapporter.github.io/pander`.

[5] См. `http://ethanschoonover.com/solarized`.

[6] Tufte E. (2001) The Visual Display of Quantitative Information. Graphics Pr.

```
p + theme_economist() + scale_colour_economist() +
    ggtitle("theme_economist()")
p + theme_solarized() + scale_colour_solarized("blue") +
    ggtitle("theme_solarized()")
p + theme_stata() + scale_colour_stata() +
    ggtitle("theme_stata()")
p + theme_excel() + scale_colour_excel() +
    ggtitle("theme_excel()")
p + theme_fivethirtyeight() +
    scale_color_fivethirtyeight() +
    ggtitle("theme_fivethirtyeight()")
```

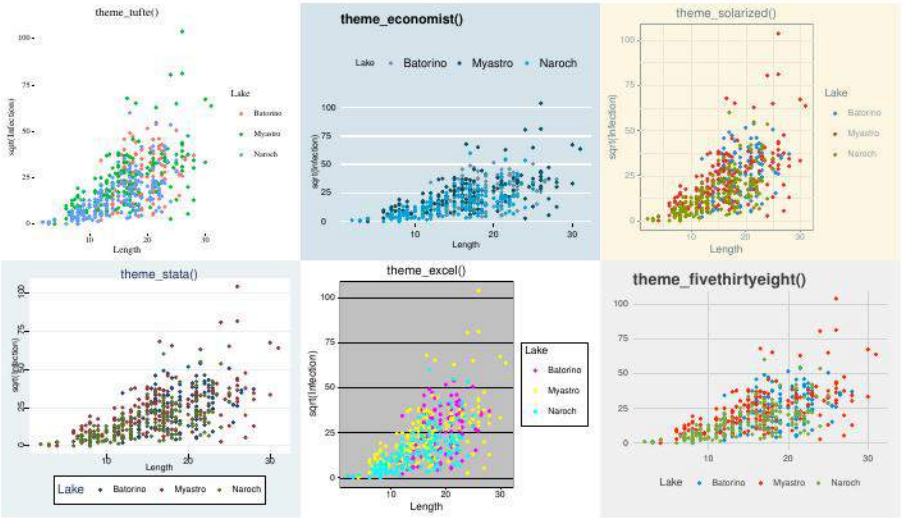


Рисунок 8.11. Примеры стилей, реализованных в пакете `ggthemes`

Еще одним полезным пакетом, с помощью которого можно изменять стиль `ggplot`–графиков, является `ggthemr`[7]. Идея, реализованная в `ggthemr`, очень проста: мы задаем стиль только один раз в начале скрипта, и дальше этот стиль применяется ко всем графикам автоматически без каких–либо дополнительных действий с нашей стороны. Данная операция выполняется с помощью функции `ggthemr()`.

Как мы уже знаем, стиль любого графика `ggplot2` определяется следующими компонентами:

- *цветовая палитра* для фона графика, его осей, линий координатной сетки, подписей и т. д.;
- *тип линий*, формирующих оси и координатную сетку;

[7] Этот пакет можно установить только из хранилища GitHub, для чего необходимо выполнить команду `devtools::install_github("cttobin/ggthemr")`.

- *ширина промежутков* между отдельными элементами графика (например, между осями и их подписями), а также ширина пустого пространства вокруг всего графика;
- *размер шрифта.*

В соответствии с перечисленными компонентами функция `ggthemr` имеет следующие основные аргументы:

- `palette` — цветовая схема, задаваемая при помощи одного из следующих текстовых значений: `"flat"`, `"flat dark"`, `"camoflauge"`, `"chalk"`, `"copper"`, `"dust"`, `"earth"`, `"fresh"`, `"grape"`, `"grass"`, `"greyscale"`, `"light"`, `"lilac"`, `"pale"`, `"sea"`, `"sky"`, `"solarized"`;
- `layout` — тип линий, формирующих оси и координатную сетку. Возможные значения: `"clean"`, `"clear"` (принято по умолчанию), `"minimal"`, `"plain"`, `"scientific"`;
- `spacing` — число (0, 1, 2 и т. д.), определяющее «степень сжатия» графика. Чем больше это число, тем больше пустого пространства будет добавлено вокруг графика (см. пример ниже);
- `text_size` — число, определяющее базовый размер шрифта;
- `type` — значение `"inner"` этого аргумента приведет к тому, что определенным фоновым цветом будет залита только основная часть графика. При значении `"outer"` этим цветом будет залито также и окружающее график пустое пространство;
- `line_weight` — число, задающее толщину линий осей и координатной сетки.

Как было отмечено выше, вызов функции `ggthemr()` необходимо выполнять в начале блока с кодом, который определяет один или несколько графиков `ggplot2`. Для отмены действия этой функции (т. е. для возвращения к исходному стандартному стилю `ggplot2`) необходимо вызвать функцию `ggthemr_reset()`. Примеры использования функции `ggthemr()` приведены на рис. 8.12 и 8.13. С другими особенностями работы с пакетом `ggthemr` можно ознакомиться по адресу `https://github.com/cttobin/ggthemr`.

Код для рис. 8.12

```
library(ggthemr)
p <- ggplot(dreissena, aes(Length, fill = Lake)) +
     geom_histogram()
ggthemr(palette = "flat")
p + ggtitle('pallete = "flat"') +
    theme(legend.position = "NA")
ggthemr(palette = "flat dark")
p + ggtitle('pallete = "flat dark"') +
    theme(legend.position = "NA")
ggthemr(palette = "camoflauge")
p + ggtitle('pallete = "camoflauge"') +
```

```
    theme(legend.position = "NA")
ggthemr(palette = "chalk")
p + ggtitle('pallete = "chalk"') +
    theme(legend.position = "NA")
ggthemr(palette = "fresh")
p + ggtitle('pallete = "fresh"') +
    theme(legend.position = "NA")
ggthemr(palette = "sea")
p + ggtitle('pallete = "sea"') +
    theme(legend.position = "NA")
ggthemr_reset() # отмена действия ggthemr()
```

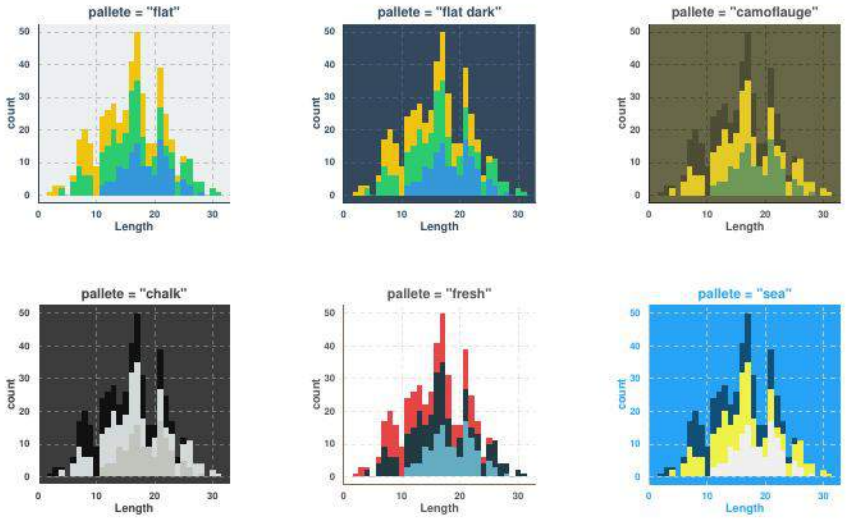
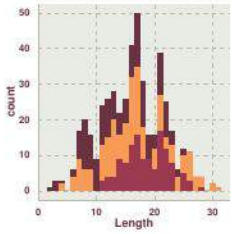
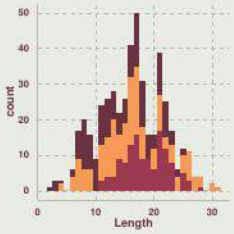


Рисунок 8.12. Примеры цветовых схем, реализованных в пакете ggthemr (легенды у приведенных графиков отключены для экономии места)

Код для рис. 8.13

```
# Объект p определен как на рис. 8.12
ggthemr(palette = "grape", type = "inner")
p + theme(legend.position = "NA")
ggthemr(palette = "grape", type = "outer")
p + theme(legend.position = "NA")
ggthemr(palette = "grape", type = "outer", spacing = 4)
p + theme(legend.position = "NA")
```





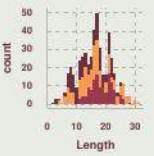


Рисунок 8.13. *Слева:* значение аргумента `type` функции `ggthemr()` равно `"inner"` (по умолчанию). *В центре:* аргументу `type` присвоено значение `"outer"`. *Справа:* пространство вокруг графика увеличено при помощи аргумента `spacing = 4`

8.2 Создание составных рисунков

Пусть у нас есть следующие две таблицы с данными:

```
t1 <- dplyr::filter(dreissena, Lake == "Naroch")
t2 <- dplyr::filter(dreissena, Lake == "Batorino")
```

```
head(t1)
  Month Day   Lake Site Length Infection
1   May   0 Naroch   S9    8.0         7
2   May   0 Naroch   S9    9.8        10
3   May   0 Naroch   S9   11.0        13
4   May   0 Naroch   S9   19.0        21
5   May   0 Naroch   S9   13.0        85
6   May   0 Naroch   S9   14.0       425
```

```
head(t2)
  Month Day     Lake Site Length Infection
1   May   1 Batorino   S3   14.9        36
2   May   1 Batorino   S3   14.0        30
3   May   1 Batorino   S3   13.0       331
4   May   1 Batorino   S3   14.0       110
5   May   1 Batorino   S3   12.0         4
6   May   1 Batorino   S3   14.0       171
```

Предположим, что для каждого из этих наборов данных мы хотим изобразить зависимость `sqrt(Infection)` от `Length` и представить полученные графики на одном составном рисунке. Используя базовые возможности `R`, это можно было бы сделать с помощью функции `par()` (результат приведен на рис. 8.14):

Код для рис. 8.14

```
par(mfrow = c(1, 2))
plot(t1$Length, sqrt(t1$Infection), main = "Lake Naroch",
     xlab = "Length", ylab = "sqrt(Infection)")
plot(t2$Length, sqrt(t2$Infection), main = "Lake Batorino",
     xlab = "Length", ylab = "sqrt(Infection)")
```

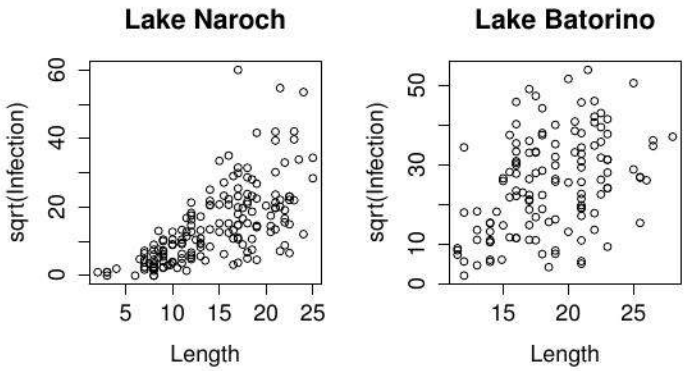


Рисунок 8.14. Пример двух графиков, скомпонованных при помощи базовой функции `par()`

Но как быть, если для построения графиков мы желаем воспользоваться пакетом `ggplot2`? В таком случае для получения результата, похожего на рис. 8.14, нам сначала пришлось бы объединить наборы данных `t1` и `t2` в одну таблицу, а затем построить категоризованный график, на котором данные разбиты на группы по уровням фактора `Lake` (см. рис. 8.15).

Код для рис. 8.15

```
tab <- rbind(t1, t2)
tab$Lake <- relevel(tab$Lake, ref = "Naroch")
qplot(data = tab, x = Length, y = sqrt(Infection),
      geom = "point", facets = ~ Lake)
```

Чтобы получить рис. 8.15, нам пришлось создать еще одну таблицу данных. Это не всегда будет хорошим решением, например при работе с данными большого объема. Кроме того, создание такой общей таблицы данных может оказаться невозможным, когда стоит задача скомпоновать `ggplot`–графики, основанные на совершенно разных наборах данных, которые не имеют общих переменных (как, например, таблица `dreissena` и классический набор данных `iris`). К сожалению, базовая функция `par()` неприменима для объединения `ggplot`–графиков, и поэтому вместо нее придется воспользоваться одним из двух описанных ниже способов.

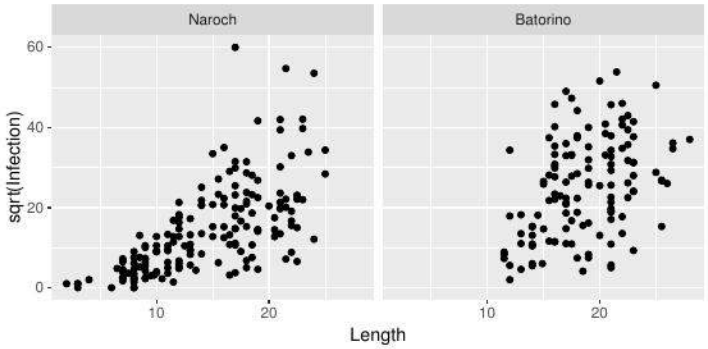


Рисунок 8.15. Пример двух графиков, скомпонованных при помощи опции `facets` (см. код для этого рисунка)

8.2.1 Использование окон просмотра

Первый способ объединения нескольких `ggplot`–графиков состоит в использовании функций из пакета `grid`[8], на основе которого построен сам пакет `ggplot2`. Наиболее важной для освоения концепцией здесь является «окно просмотра» (англ. *viewport*), которое представляет собой ограниченную прямоугольную область графического устройства[9]. По умолчанию эта область занимает все пространство графического устройства, однако ее можно разбить на более мелкие области и тем самым упорядочить на одном рисунке несколько графиков одновременно.

Для создания окон просмотра служит функция `viewport()`, основными аргументами которой являются следующие:

- `x` и `y` — координаты расположения окна. По умолчанию эти координаты выражаются в т. н. `npc` единицах (от *normalized parent coordinates*), которые изменяются от 0 до 1. Нижний левый угол окна просмотра имеет координаты (0, 0), верхний правый угол — (1, 1), а центр окна — (0.5, 0.5). При желании эти относительные единицы измерения можно изменить с помощью знакомой нам функции `unit()` (например, `unit(1, "cm")` или `unit(2, "inch")`);

- `width` и `height` — ширина и высота окна соответственно. По умолчанию каждый из этих параметров равен 1 (в `npc`–единицах);

- `layout` — объект класса `grid`, который содержит информацию о свойствах сетки, разбивающей окно на отдельные прямоугольные области. По умолчанию этому аргументу присвоено значение `NULL`;

- `layout.pos.row` и `layout.pos.col` — числовые векторы, определяющие количество строк и столбцов, занятых данным окном в его «родительском окне». По умолчанию оба аргумента равны `NULL`, т. е. данное окно «игнорирует» любое разбиение родительского окна на

[8] Подробнее см. книгу: Murrell P. (2005) R Graphics. Chapman & Hall/CRC Press.

[9] Подробнее о графических устройствах см. справочный файл, доступный по команде `?Devices`.

отдельные области. Если каждому из этих аргументов присвоено некоторое отличное от NULL скалярное значение, то данное окно будет размещено на пересечении соответствующих строки и столбца сетки, разбивающей родительское окно на области. Наконец, подав на эти аргументы векторы из двух чисел, мы можем указать несколько строк и столбцов в сетке родительского окна, которые должны быть заняты данным окном. Например, значения `layout.pos.row = c(1, 3)` и `layout.pos.col = c(2, 4)` приведут к заполнению ячеек сетки, находящихся на пересечении строк 1, 2 и 3 и столбцов 2, 3 и 4.

Разобраться с предназначением перечисленных аргументов функции `viewport()` помогут описанные ниже примеры[10].

Один из приемов по размещению нескольких графиков на одном рисунке состоит в том, что сначала при помощи функции `viewport()` создают одно большое родительское окно просмотра, в которое затем добавляют дочерние окна меньшего размера с соответствующими графиками. Однако этот прием является довольно трудоемким, так как требует от пользователя самостоятельного расчета координат и размеров всех дочерних окон. Более легкий способ заключается в разбиении родительского окна просмотра на сетку из нескольких дочерних окон с помощью функции `grid.layout()` (из того же пакета `grid`) и последующем заполнении этих окон необходимыми графиками. Все вычисления координат и размеров графиков будут при этом выполняться автоматически.

Предположим, что мы хотим объединить на одном рисунке следующие графики:

```
a <- qplot(data = t1, x = Length, y = sqrt(Infection),
           geom = c("point"), main = "Lake Naroch")
b <- qplot(data = t2, x = Length, fill = Month,
           geom = c("histogram"), main = "Lake Batorino")
c <- qplot(data = rbind(t1, t2), x = Month, y = Length,
           fill = Lake, geom = "boxplot", main = "Both lakes")
```

Пусть по нашей задумке график `a` должен располагаться вверху рисунка по всей его ширине, а графики `b` и `c` — в одном ряду под графиком `a`, занимая половину ширины рисунка каждый.

Начнем с создания родительского окна просмотра, разбитого на четыре (2×2) области одинакового размера:

```
parentvp <- viewport(layout = grid.layout(2, 2))
```

Следует отметить, что созданный нами объект `parentvp` содержит лишь описание структуры родительского окна. Для того чтобы подать эту структуру на графическое устройство, необходимо воспользоваться функцией `pushViewport()` из пакета `grid`:

```
pushViewport(parentvp)
```

[10] Обратите внимание на то, что список аргументов `viewport()` длиннее приведенного здесь. Подробнее см. справочный файл, доступный по команде `?viewport`.

Внешне при выполнении последней команды ничего не произойдет, однако на самом деле в графическом устройстве будет создано новое окно просмотра, разбитое на четыре дочерние области. Теперь мы можем начать заполнять эти области созданными ранее графиками в соответствии с нашей задумкой. Для этого необходимо воспользоваться методом `print()` для объектов класса `ggplot` в сочетании с аргументом `vp`, на который подается спецификация соответствующего дочернего окна просмотра (см. результат на рис. 8.16):

Код для рис. 8.16

```
print(a, vp = viewport(layout.pos.row = 1,
                       layout.pos.col = c(1:2)))
print(b, vp = viewport(layout.pos.row = 2,
                       layout.pos.col = 1))
print(c, vp = viewport(layout.pos.row = 2,
                       layout.pos.col = 2))
```

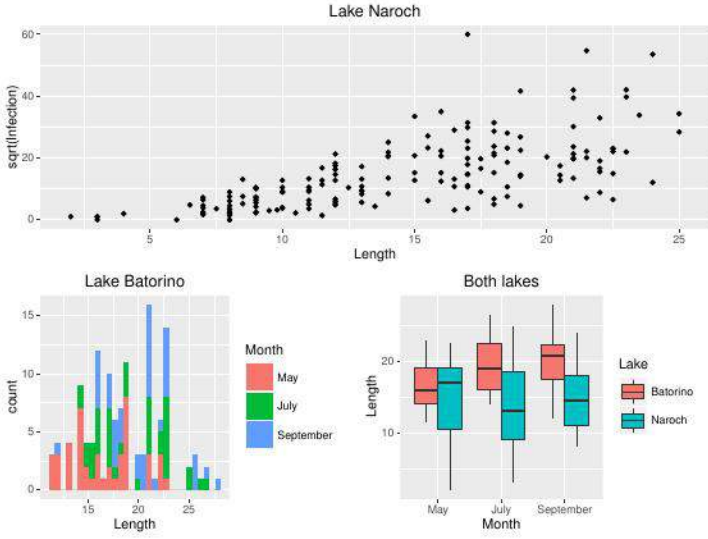


Рисунок 8.16. Пример графиков, скомпонованных на одном рисунке при помощи функций из пакета `grid`

Иногда возникает необходимость поверх того или иного графика изобразить другой, поясняющий график меньшего размера. Это можно без труда выполнить при помощи все той же функции `viewport()` в сочетании с методом `print()` для `ggplot`–графиков (см. пример на рис. 8.17).

Код для рис. 8.17

```
subplotvp <- viewport(width = 0.35, height = 0.35,
                      x = 0.65, y = 0.25)
c; print(a + theme_gray(base_size = 9), vp = subplotvp)
```

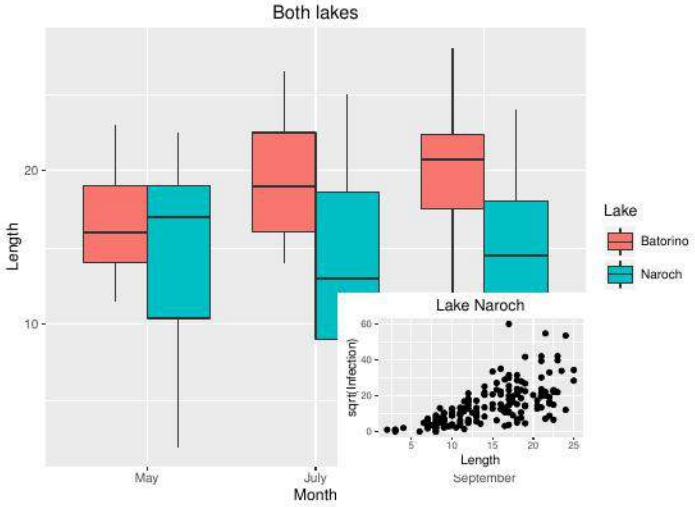


Рисунок 8.17. Пример использования функции `viewport()` для вставки поясняющего графика

8.2.2 Использование пакета `gridExtra`

Функции из рассмотренного выше пакета `grid` обеспечивают практически полный контроль над характером расположения графиков на составном рисунке. Однако эта гибкость не дается бесплатно: пользователю необходимо затратить приличные усилия на расчет координат расположения, размера и других параметров графиков для получения удовлетворительного результата. Для облегчения этой работы был создан пакет `gridExtra`, который помогает создавать составные рисунки с помощью более высокоуровневых команд. В частности, этот пакет позволяет сымитировать эффект базовой команды `par(mfrow = ...)` (см. рис. 8.14). Этот эффект достигается при помощи функции `grid.arrange()`, на которую подается список подлежащих компоновке объектов класса `ggplot`, а также значение аргумента `ncol` (или `nrow`), задающее количество строк (или столбцов) в сетке, определяющей структуру составного рисунка[11]. Пример использования этой функции приведен на рис. 8.18.

Код для рис. 8.18

```
library(gridExtra)
grid.arrange(a, b, c, ncol = 2)
```

В состав пакета `gridExtra` входит много интересных функций, позволяющих добавлять к составному рисунку разнообразные геометрические фигуры и объекты. В частности, полезными на практике являются функции `grid.table()` и `tableGrob()`. С помощью `grid.table()` в окне графического устройства можно изобразить текстовую информацию, пред-

[11] Большинство составных рисунков, представленных в этой книге, выполнено с использованием именно этой функции.

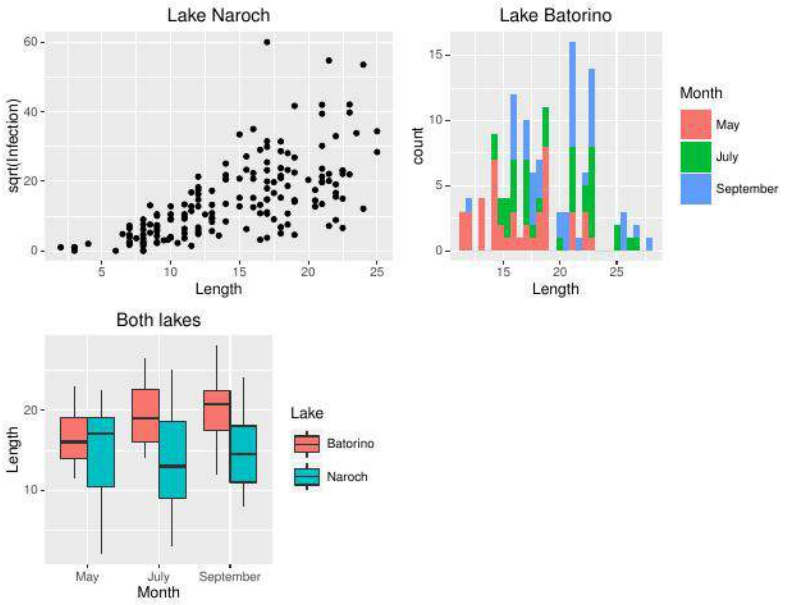


Рисунок 8.18. Пример составного рисунка, созданного с помощью функции `grid.arrange()` из пакета `gridExtra`

ставленную в табличном виде. Например, мы могли бы выполнить дисперсионный анализ модели, описывающей зависимость `Length` от `Lake` и `Month`, и вывести результаты этого анализа в виде опрятно оформленной таблицы (рис. 8.19):

Код для рис. 8.19

```
mod <- lm(Length ~ Lake * Month, data = rbind(t1, t2))
grid.table(round(anova(mod), 3))
```

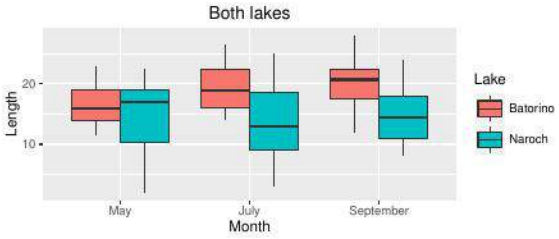
	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Lake	1	1231.107	1231.107	56.421	0
Month	2	124.266	62.133	2.848	0.06
Lake:Month	2	247.546	123.773	5.672	0.004
Residuals	290	6327.847	21.82	NA	NA

Рисунок 8.19. Пример таблицы, полученной при помощи функции `grid.table()` из пакета `gridExtra`

С помощью функции `tableGrob()` подобные таблицы можно сделать частью составного рисунка (см. пример на рис. 8.20).

Код для рис. 8.20

```
# Объект mod определен как на рис. 8.19,
# а объект c - как на рис. 8.16
tab <- tableGrob(round(anova(mod), 3))
grid.arrange(c, tab, ncol = 1)
```



	Df	Sum Sq	Mean Sq	F value	Pr(>F)
Lake	1	1231.107	1231.107	56.421	0
Month	2	124.266	62.133	2.848	0.06
Lake:Month	2	247.546	123.773	5.672	0.004
Residuals	290	6327.847	21.82	NA	NA

Рисунок 8.20. Пример использования функции `tableGrob()` из пакета `gridExtra` для добавления таблицы к составному рисунку

Внешний вид таблиц, получаемых с помощью `table.grid()` и `tableGrob()`, можно настроить, изменяя параметры функции `ttheme_default()`. Соответствующие примеры описаны на странице `http://bit.ly/2bttG2k`.

8.3 Экспорт графиков из среды R

Выполнить экспорт `ggplot`–графиков из среды `R` можно с помощью следующих способов:

- используя реализованные в `R` драйверы стандартных графических устройств (например, `pdf`, `png`, `postscript`, `bitmap` и т. д.);
- с помощью специальной функции `ggsave()` из пакета `ggplot2`;
- через соответствующее меню программы `RStudio` (при условии, что работа с `R` ведется именно из этой программы).

Работая со стандартными графическими устройствами, пользователь может сохранить `ggplot`–графики либо в растровом, либо в векторном формате[12]. В целом векторный формат (например, `pdf`) является более

[12] Подробнее об этих форматах см. статью в Википедии: `http://bit.ly/1N2sscF`.

предпочтительным, однако в случае со сложными векторными изображениями может потребоваться значительное время на их рендеринг. Поэтому иногда лучше сохранять графики в растровом формате (например, png). Типографское качество таких файлов достигается при разрешении 600 dpi («точек на дюйм») и выше (при этом чем выше разрешение, тем больше размер файла). Ниже приведены примеры кода для сохранения одного и того же графика в двух форматах — pdf и png:

```
# Обратите внимание - без вызова метода print()
# графики не будут сохранены

pdf(file = "dreissena.pdf", width = 6, height = 3)
print(qplot(Length, sqrt(Infection), data = dreissena))
dev.off()

png(file = "dreissena.png", width = 480, height = 320,
    units = "in", res = 600)
print(qplot(Length, sqrt(Infection), data = dreissena))
dev.off()
```

Если тот или иной график предполагается вставить в документ, подготовленный при помощи Microsoft Word или текстового редактора OpenOffice, то такой график можно сохранить в форматах windows (Windows metafile) или postrcript. Однако следует помнить, что ни один из этих форматов не поддерживает прозрачность цвета: если в своем графике вы используете, например, полупрозрачный цвет точек, то после сохранения файла эти точки просто исчезнут.

Если вы используете в своей работе LaTeX, то, согласно рекомендации Х. Уикхэма (Wickham, 2009[13]), в преамбулу документа стоит добавить команду \DeclareGraphicsExtensions{.png, .pdf}. В таком случае вам не придется в явном виде указывать расширение графического файла при вызове команды \includegraphics{} — LaTeX автоматически будет распознавать png– и pdf–файлы, отдавая предпочтение первым. В преамбулу документа стоит также добавить команду \graphicspath{{include/}}, с помощью которой можно задать путь к отдельной директории со всеми графическими файлами.

Как было отмечено выше, ggplot–графики можно также сохранить в виде отдельных файлов с помощью функции ggsave(), которая имеет следующие основные аргументы:

- filename — имя сохраняемого файла (включая расширение);
- path — путь к директории, в которую необходимо поместить сохраняемый файл (по умолчанию это рабочая директория R);
- plot — подлежащий сохранению график. По умолчанию сохраняется последний график, выведенный на графическое окно R;
- device — формат сохраняемого файла. По умолчанию будет предпринята попытка распознать его автоматически по расширению файла, указанного в filename. В настоящее время функция ggsave()

[13] Wickham H. (2009) ggplot2: Elegant Graphics for Data Analysis. Springer.

способна работать со следующими форматами: `eps`, `ps`, `tex` (`pictex`), `pdf`, `jpeg`, `tiff`, `png`, `bmp`, `svg` и `wmf` (только на компьютерах с ОС Windows);

- `width` и `height` — ширина и высота графика, выражаемые в единицах, которые задаются при помощи другого аргумента — `units` (`"in"`, `"cm"`, `"mm"`; по умолчанию это `"in"` — дюймы). Если `width` и `height` в явном виде не указаны, то размер графика будет равен размеру текущего графического окна `R`. Рекомендуется всегда указывать значения этих аргументов;

- `dpi` — разрешение растровых изображений. По умолчанию разрешение составляет 300 dpi, что достаточно для печати на большинстве принтеров. Для типографской печати рекомендуется увеличить этот параметр до 600 dpi. Для создания же веб–изображений достаточным будет разрешения в 72 dpi.

Пример использования функции `ggsave()` приведен ниже:

```
p <- qplot(Length, sqrt(Infection), data = dreissena)
ggsave(filename = "dreissena.png", plot = p, dpi = 600,
       units = "in", width = 6, height = 3)
```

Наконец, если вы работаете с `R` в программе `RStudio`, то для сохранения `ggplot`–графиков можно воспользоваться выпадающим меню `Export` на закладке `Plots`. Как видно на рис. 8.21, у пользователя есть возможность сохранять графики через это меню либо в формате `pdf` (*Save as PDF*), либо в каком–либо другом из доступных форматов (*Save as Image*). Выбрав соответствующую опцию, пользователь получит возможность указать дополнительные свойства сохраняемого файла (см. рис. 8.22 и рис. 8.23).

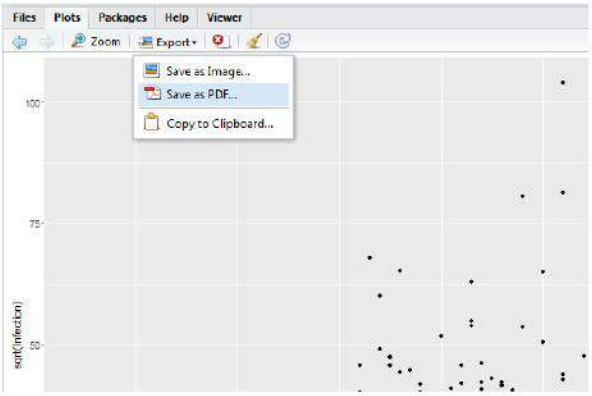


Рисунок 8.21. Экспорт графиков через меню `Export` программы `RStudio`

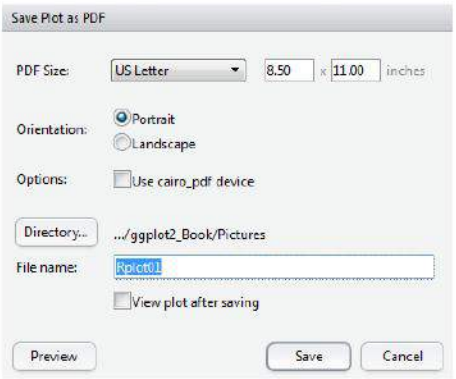


Рисунок 8.22. Окно настройки свойств графиков, сохраняемых в формате `pdf`

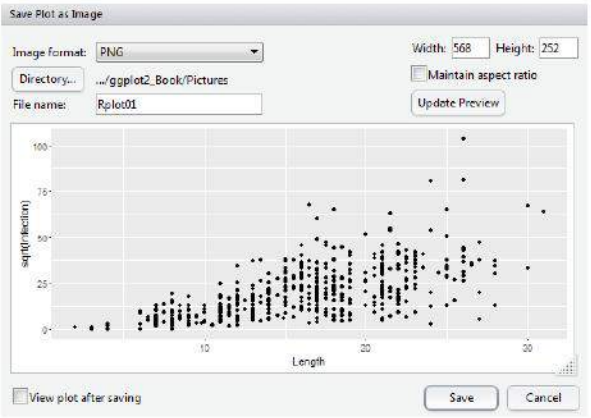


Рисунок 8.23. Окно настройки свойств графиков, которые сохраняются в форматах, отличных от `pdf`

Глава 9

Дополнительные ресурсы для изучения ggplot2

Цель этой книги заключалась в том, чтобы познакомить читателя с основами визуализации данных средствами пакета `ggplot2`. Однако в одной книге невозможно описать все нюансы работы с этим замечательным инструментом. Дальнейшее освоение как самого пакета `ggplot2`, так и многочисленных расширений, созданных на его основе, потребует изучения дополнительных источников информации. В этой главе перечислены некоторые из наиболее интересных и значимых таких источников.

9.1 Литература

Наиболее авторитетным и полным литературным источником по `ggplot2` на данный момент является книга, опубликованная автором этого пакета — проф. Х. Уикхэмом — в 2009 г. (Wickham H. (2009) ggplot2: Elegant Graphics for Data Analysis. Springer), а также ее значительно переработанное второе издание, которое вышло в июне 2016 г. После публикации Wickham (2009) на английском языке было издано еще несколько книг, либо полностью посвященных `ggplot2`, либо уделяющих этому пакету значительное внимание. В частности, можно отметить следующие работы:

- Teetor P. (2011) R Cookbook. O'Reilly Media.
- Chang W. (2013) R Graphics Cookbook. O'Reilly Media.
- Abedin J., Mittal H.V. (2014) R Graph Cookbook, 2nd ed. Packt Publishing.
- R Fundamentals & Graphics, Volume 1 (2014). RMS Books.
- Hilfiger J.J. (2015) Graphing Data with R: An Introduction. O'Reilly Media.
- Teutonico D. (2015) ggplot2 Essentials. Packt Publishing.
- Wickham H., Grolemund G. (2016) R for Data Science. O'Reilly Media.

Существует также несколько статей о ggplot2, опубликованных в научных журналах и изданиях, посвященных программному обеспечению:

- Wickham H. (2011) ggplot2. WIREs Computation Statistics 3: 180–185.
- Kahle D., Wickham H. (2013) ggmap: spatial visualization with ggplot2. The R Journal 5(1): 144–161.
- Ito K., Murphy D. (2013) Application of ggplot2 to pharmacometric graphics. Pharmacometrics & Systems Pharmacology 2: e79.

Печатной литературы на русском языке, посвященной ggplot2, почти нет. Помимо книги, которую вы сейчас держите в руках, некоторые примеры работы с этим пакетом можно найти в книге: Мастицкий С. Э., Шитиков В. К. (2015) Статистический анализ и визуализация данных с помощью R. ДМК Пресс.

9.2 Онлайн–ресурсы

В связи с большой популярностью ggplot2 в Сети существует огромное количество ресурсов, посвященных визуализации данных с помощью этого пакета. Ниже приведен список наиболее полезных из них:

- официальная документация по ggplot2: http://docs.ggplot2.org/current;
- ответы на вопросы по ggplot2 на сайте «StackOverflow»: http://bit.ly/2bMThIR;
- примеры работы с ggplot2 на сайте «R Cookbook» (этот сайт является приложением к одноименной книге, упомянутой в разд. 9.1): http://www.cookbook-r.com/Graphs;
- примеры работы с ggplot2 на сайте «Software and Programmer Efficiency Research Group (SAPE)»: http://bit.ly/2bxo9I5;
- примеры работы с ggplot2 на сайте «Statistical Tools for High–Throughput Data Analysis»: http://bit.ly/2cmNbQb;
- цикл статей по ggplot2 на сайте А. Матрунича: http://matrunich.com/tags/ggplot2;
- статья, посвященная работе с ggplot2 в блоге А. Огурцова «Биостатистика и язык R»: http://bit.ly/2bxcRYZ;
- шпаргалка по ggplot2, подготовленная компанией RStudio: http://bit.ly/1vUS8fi;
- еще одна шпаргалка: http://bit.ly/1KPrv4q;
- обучающий курс по ggplot2 на базе платформы «DataCamp»: http://bit.ly/1STIOJe и http://bit.ly/1RPjS0D;

- обучающий курс по ggplot2 на базе платформы «Statistics.com»: http://bit.ly/2c57Jsn;
- обучающий курс по ggplot2 на базе платформы «Udemy»: http://bit.ly/2c0DLXk;
- серия видеоуроков по ggplot2: http://bit.ly/2bxo3QE.

9.3 Расширения, созданные на основе ggplot2

Как было показано в предыдущих главах (в частности, см. главу 4), в состав пакета ggplot2 входит широкий набор «строительных блоков» и вспомогательных функций для визуализации данных. Однако некоторые типы графиков, специфичные, например, для определенных научных дисциплин, в базовой версии этого пакета недоступны. В связи с этим пользователями было создано несколько расширений на основе ggplot2[1]. Обновляемый каталог таких расширений и соответствующие примеры кода можно найти на сайте «ggplot2 extentions» (http://www.ggplot2-exts.org). В сентябре 2016 г. на этом сайте были перечислены следующие расширения:

- ggiraph (http://bit.ly/2bMz85S): позволяет превратить обычные ggplot–графики в интерактивные HTML–виджеты, которые можно вставить в любой веб–документ;
- ggstance (http://bit.ly/2bwZGlY): облегчает создание графиков с повернутыми осями (т. е. графиков, на которых ось y принимает горизонтальное положение) без вызова функции coord_flip() в явном виде;
- ggforce (http://bit.ly/2bwVRCq): содержит дополнительные геометрические объекты для изображения данных (например, дуги, окружности и др.), а также функции для преобразования данных;
- ggrepel (http://bit.ly/2cmuzzJ): позволяет добавлять к графику неперекрывающиеся текстовые метки наблюдений;
- ggraph (http://bit.ly/2bx39B9): содержит дополнительные геометрические объекты для визуализации графов и подобных им структур (например, дендрограмм);
- ggpmisc (http://bit.ly/2bDOPZN): набор функций, позволяющих выводить на график уравнения подогнанных к данным линейных регрессионных моделей, а также стандартные показатели качества таких моделей (например, R^2, AIC и т. д.);
- geomnet (http://bit.ly/2c3cPov): еще один пакет для визуализации графов;

[1] Фактически речь идет об отдельных пакетах, использующих «строительные блоки» ggplot2.

- ggExtra (http://bit.ly/2bMDeuL, а также http://bit.ly/2bx4GXP): набор функций для изображения гистограмм, кривых плотности вероятности и диаграмм размахов по краям основного ggplot–графика. Такие краевые диаграммы помогают более полно описать характер распределения анализируемых данных;

- gganimate (http://bit.ly/20wCyw0): позволяет создавать анимации из отдельных ggplot–графиков;

- plotROC (http://bit.ly/2bMz55m): пакет для визуализации ROC–кривых;

- ggtheme (http://bit.ly/1A2ziZU): дополнительные стили для ggplot–графиков. Особенности работы с этим пакетом были рассмотрены нами в разд. 8.1;

- ggspectra (http://bit.ly/2c3f6Af): набор функций для визуализации данных, получаемых в ходе спектрометрических исследований;

- ggnetwork (http://bit.ly/2c3fWwS): еще один пакет для визуализации графов;

- ggtech (http://bit.ly/2bDT1ZA): пакет с дополнительными стилями, имитирующими цветовые схемы и шрифты таких компаний, как Google, Facebook, Twitter, и т. д.;

- ggradar (http://bit.ly/2bx9w7E): пакет для создания «радарных диаграмм»;

- ggTimeSeries (http://bit.ly/2cmBUPW): набор функций для визуализации временных рядов;

- ggtree (http://bit.ly/2bMDzfo): набор функций для визуализации филогенетических деревьев (включая круговые дендрограммы);

- ggseas (http://bit.ly/2bvajv0): пакет для визуализации сезонной компоненты временных рядов.

Существует также несколько интересных расширений ggplot2, не указанных на сайте «ggplot2 extentions»:

- cowplot (http://bit.ly/1Knxlj1): пакет для подготовки ggplot–графиков к публикации в научных изданиях;

- ggdendro (http://bit.ly/2bV7odW): пакет для визуализации дендрограмм и деревьев решений;

- GGally (http://bit.ly/2bxb2GK): набор функций, облегчающих комбинирование нескольких стандартных геометрических объектов ggplot2 на одном графике;

- ggfortify (http://bit.ly/2bMFjVQ): позволяет изобразить результаты подгонки статистических моделей, выполненной с помощью целого ряда пакетов R;

- ggmcmc (http://bit.ly/2bMFrll): пакет для визуализации результатов оценки параметров статистических моделей, выполненной методом Монте–Карло;
- plotly (http://bit.ly/2bxm3rO): содержит функцию ggplotly(), которая превращает статичные ggplot–графики в интерактивные HTML–виджеты;
- ggRandomForests (http://bit.ly/2bMEeKN): позволяет визуализировать результаты моделирования, выполненного с помощью метода «случайный лес»;
- ggsci (http://bit.ly/2bxnuGB): набор стилей для ggplot2, имитирующих графики из нескольких известных научных изданий;
- ggtern (http://www.ggtern.com): пакет для создания «треугольных диаграмм» (англ. *ternary plots*).

Предметный указатель

Книги издательства «ДМК Пресс» можно заказать
в торгово-издательском холдинге «Планета Альянс» наложенным платежом,
выслав открытку или письмо по почтовому адресу:
115487, г. Москва, проспект Андропова, д. 38.
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.alians-kniga.ru**.
Оптовые закупки: тел. **(499) 782-38-89**.
Электронный адрес: **books@alians-kniga.ru**.

Мастицкий Сергей Эдуардович

Визуализация данных с помощью ggplot2

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com

Корректор *Синяева Г. И.*
Верстка *Мастицкий С. Э.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.
Гарнитура «Петербург». Печать офсетная.
Усл. печ. л. 20,8. Тираж 200 экз.

Сайт издательства: www.дмк.рф