R in action
Data analysis and graphics with R
Robert I. Kabakoff

R в действии[1]
Роберт И. Кабаков
Перевод Полины А. Волковой [2]

Краткое содержание

Часть 1. Начало работы

Глава 1. Знакомство с R

Глава 2. Создание набора данных

Глава 3. Начало работы с диаграммами

Глава 4. Основы управления данными

Глава 5. Более сложные способы управления данными

Часть 2. Базовые методы

Глава 6. Базовые диаграммы

Глава 7. Основные методы статистической обработки данных

[1] *Анализ и графическое представление данных в программе R – Прим. автора*

[2] *Учтены опечатки в англоязычной версии, перечисленные на форуме и сайте издательства – Прим. пер.*

Содержание

Часть 1. Начало работы

Глава 1. Знакомство с R

1.1. Зачем нужна R?

1.2. Получение и установка R

1.3. Работа в R

1.4. Пакеты

1.5. Пакетная обработка

1.7. Работа с большими массивами данных

1.8. Учимся на примере

1.9. Резюме

Глава 2. Создание набора данных

2.1. Что такое набор данных?

2.2. Типы структуры данных

2.3. Ввод данных

2.4. Аннотирование наборов данных

2.5. Полезные функции для работы с объектами

2.6. Резюме

Глава 3. Начало работы с диаграммами

3.1. Работа с диаграммами

3.2. Простой пример

3.3. Графические параметры

3.4. Добавление текста, настройка параметров осей и условных обозначений

3.5. Объединение диаграмм

3.6. Резюме

Глава 4. Основы управления данными

4.1. Рабочий пример

4.2. Создание новых переменных

4.3. Перекодировка переменных

4.4. Переименование переменных

4.5. Пропущенные значения

4.6. Данные в формате даты

4.7. Преобразования данных из одного типа в другой

4.8. Сортировка данных

4.9. Объединение наборов данных

4.10. Разделение наборов данных на составляющие

4.11. Использование команд SQL для преобразования таблиц данных

4.12. Резюме

Глава 5. Более сложные способы управления данными

5.1. Задача по управлению данными, которую нужно решить

5.2. Числовые и текстовые функции

5.3. Решение нашей задачи по управлению данными

5.4. Управление выполнением команд

5.5. Написанные пользователем функции

5.6. Агрегирование и переформатирование данных

5.7. Резюме

Часть 2. Базовые методы

Глава 6. Базовые диаграммы

6.1. Столбчатые диаграммы

6.2. Круговые диаграммы

6.3. Гистограммы

6.4. Диаграммы плотности ядра

6.5. Ящики с усами

6.6. Точечные диаграммы

6.7. Резюме

Глава 7. Основные методы статистической обработки данных

7.1. Описательные статистики

7.2. Таблицы частот и таблицы сопряженности

7.3. Корреляции

7.4. Тесты Стьюдента

7.5. Непараметрические тесты межгрупповых различий

7.6. Визуализация групповых различий

7.7. Резюме

Предисловие

Что толку в книжке, если в ней нет ни картинок, ни разговоров?

- Алиса. *«Алиса в Стране Чудес» Перевод Н. Демуровой*

Оно чудесно и наделено сокровищами, способными удовлетворить всех от мала до велика, но не предназначено для робких духом.

- Кью. *Сериал «Звездный путь: следующее поколение»*

Когда я начал писать книгу, я потратил достаточно много времени в поисках хорошего эпиграфа. В итоге я остановился на этих двух. R – это потрясающе гибкие программа и язык для исследования, визуализации и понимания данных. Я выбрал цитату из «Алисы в Стране Чудес», чтобы передать суть современного статистического анализа – интерактивного процесса, состоящего из исследования, визуализации и интерпретации.

Вторая цитата отражает широко распространенное мнение о том, что работе в R сложно научиться. Я надеюсь показать вам, что это не должно быть так. R – мощная программа с таким большим числом доступных аналитических и графических функций (по последним подсчетам их более 50 000), что она может в одинаковой степени навести ужас и на новичков, и на опытных пользователей. Однако в этом мнимом безумии есть поэзия и логика. Вооружившись руководствами и инструкциями, вы сможете ориентироваться в огромном разнообразии возможностей, выбрав те инструменты, которые нужны для того, чтобы уверенно, эффективно и элегантно выполнить вашу задачу.

Я впервые познакомился с R несколько лет назад, когда хотел получить новую должность консультанта по статистике. Многообещающий работодатель перед интервью спросил меня, владею ли я R. Следуя обычным советам специалистов по подбору персонала, я немедленно сказал «да» и стал учиться работать в этой программе. Я был опытным статистиком и исследователем с 25 годами опыта программирования в SAS и SPSS, свободно владевшим несколькими языками программирования. Чего же тут сложного? Знаменитые последние слова.

По мере того как я пытался выучить язык программирования (как можно быстрее, ведь день собеседования приближался с угрожающей быстротой), я находил или тома, посвященные глубинной структуре языка или многочисленные трактаты об отдельных продвинутых статистических методах, написанных специалистами в данной области для своих коллег. Онлайн-помощь была написана очень лаконично и служила скорее справочником, чем учебным пособием. Каждый раз, когда мне казалось, что я освоил общую логику и возможности R, находилось что-то новое, заставлявшее почувствовать себя невежественным и ничтожным.

Для того чтобы освоить программу, я подошел к этому с точки зрения исследователя, которому нужно обрабатывать данные. Я пытался понять, что нужно сделать, чтобы успешно обработать, проанализировать и понять данные, включая:

- доступ к данным (получение данных из разных источников);
- редактирование данных (замена или удаление пропущенных значений, преобразование признаков в более удобный вид);
- аннотирование данных (чтобы помнить, что представляет собой каждый их фрагмент);
- получение общих сведений о данных (вычисление описательных статистик для того, чтобы охарактеризовать данные);
- визуализация данных (поскольку картинка на самом деле ст**о**ит тысячи слов);
- моделирование данных (нахождение зависимостей и тестирование гипотез);
- оформление результатов (подготовка таблиц и диаграмм достаточного для публикации качества).

Затем я постарался понять, как я могу использовать R, чтобы выполнить каждую из этих задач. Поскольку я лучше всего учусь, обучая, я со временем создал сайт (www.statmethods.net), на котором рассказал все, что я узнал.

Затем, около года назад, Марьян Бейс (Marjan Bace), издатель, позвонила и спросила, не хочу ли я написать книжку про R. К этому времени я уже написал 50 статей в научных журналах, четыре

технических руководства, многочисленные главы в книгах и целую книгу по методологии исследования, так чего же тут могло быть сложного? Рискую повториться – знаменитые последние слова.

Книгу, которую вы держите в руках, я мечтал иметь много лет назад. Я постарался написать для вас путеводитель по R, который позволит быстро овладеть всей мощью этой замечательной программы с открытым кодом без разочарования и раздражения, которые пришлось испытать мне. Надеюсь, вам понравится.

P.S. Мне предложили ту должность, но я отказался. Однако знакомство с R развернуло мою карьеру в совершенно неожиданном направлении. Жизнь может быть забавной штукой.

Благодарности

Многие люди приложили все усилия, чтобы сделать эту книгу лучше:

- в первую очередь это Марьян Бейс (Marjan Bace), глава издательства Маннинг (Manning), которая предложила мне написать эту книгу;
- Себастьян Стирлинг (Sebastian Stirling), редактор-консультант по аудитории (development editor), который провел многие часы в телефонных беседах со мной, помогая выстроить материал, прояснить основные идеи и в целом сделать текст более интересным. Он также помог мне на многих этапах подготовки книги к изданию;
- Карен Тегмейер (Karen Tegtmeyer), редактор-рецензент (review editor), которая помогла найти рецензентов и координировала процесс рецензирования;
- Мари Пиргис (Mary Piergies), которая помогала следить за процессом подготовки книги к печати, и ее команду: Лиз Велч (Liz Welch), Сьюзан Харкинс (Susan Harkins) и Рахель Шредер (Rachel Schroeder);
- Пабло Доминик Васелли (Pablo Dominguez Vaselli), корректор, который помог обнаружить ошибки и свежим опытным взглядом проверил программный код;
- Рецензенты, которые потратили много времени на внимательное чтение текста, находили опечатки и делали ценные замечания: Крис Вилльямс (Chris Williams), Чарльз Мальпас (Charles Malpas), Анжела Стейплз (Angela Staples), Даниэль Рейс Перейра (Daniel Reis Pereira), Д.Х. Ван Райн (D.H. van Rijn), Кристиан Маркуардт (Christian Marquardt), Амос Фоларин (Amos Folarin), Стюарт Джеффрис (Stuart Jefferys), Дрор Берел (Dror Berel), Патрик Брин (Patrick Breen), Элизабет Островски (Elizabeth Ostrowski), Атеф Оуни (Atef Ouni), Карл Феноллоза (Carles Fenollosa), Рикардо Питробон (Ricardo Pietrobon), Самуэль МакКвилин (Samuel McQuillin), Ландон Кокс (Landon Cox), Августин Циглер (Austin Ziegler), Рик Вагнер (Rick Wagner), Райн Кокс (Ryan Cox), Сумит Пал (Sumit Pal), Филипп К. Джанер (Philipp K. Janert), Дипак Вохра (Deepak Vohra) и Софи Мормеде (Sophie Mormede);
- Многих участников программы раннего доступа издательства Маннинг (Manning Early Access Program – MEAP).

Каждый из перечисленных людей сделал эту книгу лучше и полнее.

Я также хотел бы поблагодарить многочисленных разработчиков программы, которые и сделали R такой мощной платформой для анализа данных. Этот список включает не только основную команду разработчиков, но и многочисленных самоотверженных людей, которые создали и поддерживают дополнительные пакеты, значительно расширяющие возможности R. В приложении F перечислены авторы всех пакетов, упомянутых в этой книге. Отдельно я хотел бы упомянуть Джона Фокса (John Fox), Хадли Викхама (Hadley Wickham), Франка Е. Харрела-мл. (Frank E. Harrell), Дипаян Саркар (Deepayan Sarkar) и Вильяма Ревилла (William Revelle), работами которых я восхищаюсь. Я старался как следует отразить их вклад, а ответственность за все ошибки и искажения, непреднамеренно допущенные в этой книге, лежит исключительно на мне.

На самом деле мне следовало бы начать эту книгу с благодарности моей жене и другу Кэрол Линн (Carol Lynn). Хотя она особенно не интересуется статистикой или программированием, она по много раз читала каждую главу и делала бесчисленные исправления и предложения. Никаким другим способом нельзя выразить свою любовь к другому человеку лучше, чем прочесть ради него текст по многомерной статистике. Столь же важно, что она переживала многие вечера и выходные,

которые я проводил в работе над этой книгой, с тактом, поддержкой и сочувствием. И за что это мне так повезло?

Есть еще два человека, которых я хочу поблагодарить. Один из них мой отец, любовь которого к науке вдохновляла меня и помогла понять ценность полученных данных. Другой человек – Гари К. Бургер (Gary K. Burger), мой руководитель в магистратуре. Гари заинтересовал меня статистикой и преподаванием, в то время как я собирался стать врачом. Это все он.

О книге

Если вы выбрали эту книгу, скорее всего у вас есть какие-то данные, которые нужно собрать в единое целое, преобразовать, исследовать, моделировать, визуализировать или представить коллегам. Если это так, то R создан для вас! R стал всемирно известным языком программирования для статистического анализа, предсказаний и визуализации данных. В этой программе реализовано множество методов анализа данных, от самых простых до самых сложных и современных.

Эта программа с открытым кодом работает под разными операционными системами, включая Windows, Mac OS X и Linux. Она развивается постоянно, новые методы появляются ежедневно. Кроме того, R поддерживается большим и разнородным сообществом ученых и программистов, которые охотно помогут новичку советами.

Хотя программа R, возможно, больше известна за способность создавать красивые и сложные диаграммы, она может справиться с любой статистической задачей. Базовая версия содержит сотни функций для статистического анализа, управления данными и построения диаграмм. Однако некоторые особенно мощные методы реализованы в дополнительных пакетах, созданных отдельными авторами.

Эта широта возможностей имеет свою цену. Для новичков бывает сложно понять, что такое R и как в нем работать. Даже самые опытные пользователи R с удивлением обнаруживают какие-то возможности, о которых они не подозревали.

«R в действии» представляет собой руководство-путеводитель по R, позволяя в общих чертах ознакомиться с самой программой и ее возможностями. В книге описаны наиболее полезные функции базовой версии и более 90 наиболее часто используемых дополнительных пакетов. На всем протяжении книги акцент делается на практическое применение – на то, чтобы вы, руководствуясь прочитанным, могли проанализировать ваши данные и изложить результаты коллегам. По окончании чтения этой книги вы будете иметь хорошее представление о том, как R работает, и где можно получить дальнейшую информацию. Вы научитесь применять разнообразные методы для визуализации данных и обретете достаточно умений, чтобы справиться как с простыми, так и со сложными задачами анализа данных.

Кому следует прочесть эту книгу

Книга «R в действии» предназначена для любого человека, который имеет дело с данными. Опыт в статистическом программировании не требуется. Хотя эта книга доступна и новичкам, в ней содержится достаточно нового и полезного материала, чтобы удовлетворить запросы даже опытных специалистов по R.

Пользователи, не владеющие познаниями в области статистики, которые хотят использовать R для управления данными, их обобщения и представления в графическом виде, смогут легко понять главы 1—6, 11 и 16. Главы 7 и 10 подразумевают, что вы прослушали вводный курс статистики, а главы 8, 9 и 12—15 потребуют более глубоких познаний в этой области. Однако я старался написать каждую главу так, чтобы в ней было что-то интересное и полезное и для новичков и для опытных статистиков.

Структура книги

Эта книга создана как путеводитель по программе R, с акцентом на методы, которые можно сразу применить для управления данными, их визуализации и осмысления. Книга состоит из 16 глав, сгруппированных в четыре части: «Начало работы», «Базовые методы», «Методы средней сложности» и «Методы повышенной сложности». Дополнительные темы рассмотрены в восьми приложениях.

Глава 1 начинается с обзора программы в целом и характеристик, которые делают ее столь полезной для обработки данных. В главе рассказано, как установить программу и как расширить ее возможности путем установки доступных в Сети дополнительных пакетов. Оставшаяся часть главы посвящена описанию интерфейса программы и рассказу о том, как запускать ее в интерактивном и пакетном режимах.

В главе 2 описаны многие методы импорта данных в программу. Первая половина главы посвящена характеристике типов данных в R и тому, как вводить данные с клавиатуры. Во второй половине главы обсуждаются способы импорта данных из текстовых файлов, веб-страниц, электронных таблиц, других статистических программ и баз данных.

Многие пользователи изначально выбрали R, потому что они хотят создавать диаграммы, так что мы сразу переходим к этой теме в главе 3. Вам не понадобится долго ждать. Мы обсуждаем, как создавать диаграммы, изменять их и сохранять в разных форматах.

Глава 4 посвящена основам управления данных, включая сортировку, объединение и разбиение наборов данных, а также преобразование, перекодировку и удаление переменных.

Глава 5 основана на главе 4 и содержит описание функций (математических, статистических, текстовых) и управляющих конструкций (циклы, выполнение при условии) для управления данными. Затем мы обсуждаем, как написать вашу собственную функцию в R и как сгруппировать данные различными способами.

В главе 6 рассказано, как создавать наиболее распространенные одномерные диаграммы, такие как столбчатая и круговая диаграмма, диаграмма распределения плотности, «ящик с усами» и точечная диаграмма. Все эти диаграммы полезны для того, чтобы понять, как распределены значения одной переменной.

Глава 7 начинается с описания того, как находить общие характеристики данных, включая использование описательных статистик и сводных таблиц. Затем мы рассматриваем основные способы изучения взаимосвязи между двумя переменными, включая корреляцию, тест Стьюдента, тест хи-квадрат и непараметрические методы.

Глава 8 посвящена применению регрессионных методов для моделирования взаимосвязи между числовой переменной-откликом (outcome variable) и набором из одной или нескольких независимых переменных (predictor variables). Подробно рассмотрены методы подгонки этих моделей, оценки их адекватности и интерпретации их значений.

В главе 9 рассмотрены основные типы экспериментального дизайна при дисперсионном анализе и его разновидностях. В этой ситуации нас обычно интересует, как комбинации разных типов воздействия или разных условий влияют на числовую переменную-отклик. Также описаны методы оценки адекватности анализа и визуализации результатов.

Детальное описание анализа мощности статистических тестов – предмет главы 10. Она начинается с обсуждения проблемы проверки гипотез, далее описано, как определить объем выборки, необходимый для выявления эффекта заданной величины при заданном уровне достоверности. Это поможет вам повысить вероятность достижения желаемого результата при планировании экспериментов.

Глава 11 – это продолжение главы 5. В ней рассказано, как создать диаграммы для визуализации связей между двумя и более переменными. Обсуждаются разные типы двух- и трехмерных диаграмм рассеяния, матриц диаграмм рассеяния, графиков, коррелограмм и мозаичных диаграмм.

В главе 12 представлены аналитические методы, которые хорошо работают, когда данные происходят из неизвестных или смешанных типов распределения, когда размеры выборок малы, когда выбросы представляют собой проблему или когда разработка статистического теста на основании наблюдаемого распределения слишком сложна. Это метод повторной выборки

(resampling) и бутстреп-анализ (bootstrapping) – подходы, требующие большого объема вычислений и легко реализуемые в R.

Глава 13 описывает, как применять регрессионный анализ, рассмотренный в главе 8, к данным с распределением, отличным от нормального. Глава начинается с описания обобщенных линейных моделей. Затем более подробно рассматриваются случаи, когда нужно предсказать переменную-отклик, представленную или категориальными (логистическая регрессия), или счетными данными (пуассоновская регрессия).

Одна из сложностей, связанных с многомерными данными – это проблема снижения их размерности. В главе 14 описаны методы, с помощью которых большое число скоррелированных переменных преобразуется в меньший набор независимых переменных (анализ главных компонент), а также методы обнаружения скрытой структуры в имеющемся наборе переменных (факторный анализ). Детально разобраны многочисленные этапы этих типов анализа.

В соответствии с нашим намерением описать актуальные методы анализа данных глава 15 посвящена современным подходам к решению распространенной проблемы пропущенных значений в данных. В R реализованы разнообразные изящные подходы к анализу неполных в силу разных причин данных. Здесь описаны лучшие из них, вместе с разъяснениями, когда стоит применять каждый из них, а какие – лучше избегать.

Глава 16 завершает обсуждение диаграмм рассмотрением некоторых наиболее сложных и полезных методов визуализации данных. Рассмотрена визуализация очень сложных данных с использованием панельной графики, даны основные сведения о новом пакете `ggplot2`, также кратко описаны способы работы с диаграммами в режиме реального времени.

В послесловии перечислены многие из лучших сайтов, которые следует посетить, чтобы научиться работать в R, влиться в сообщество пользователей R, получить ответы на возникшие вопросы и отслеживать изменения в этом стремительно развивающемся программном продукте.

И, последнее, но не менее важное, восемь приложений (от A до H) содержат дополнительные сведения по таким полезным темам как пользовательский интерфейс, настройка и обновление программы, экспорт данных, получение результатов высокого полиграфического качества, использование R для матричной алгебры (по образцу MATLAB) и работа с большими объемами данных.

Примеры

Для того чтобы сделать книгу так широко применимой, как это только возможно, я выбрал примеры из разных областей знаний, включая психологию, социологию, медицину, биологию, бизнес и технические науки. Ни один из примеров не требует специальных знаний в соответствующей области.

Наборы данных, используемые в этих примерах, были выбраны потому что они позволяют формулировать интересные вопросы и потому что они имеют небольшой размер. Это позволяет сосредоточиться на рассматриваемом методе и быстро понять происходящее. Когда учишься новым методам, меньше – значит лучше.

Наборы данных или поставляются с базовой установкой R, или доступны в составе дополнительных пакетов, которые можно скачать из Интернета. Программный код для каждого примера размещен на сайте http://www.manning.com/RinAction. Для того чтобы вы смогли получить все возможное от этой книги, я рекомендую выполнять примеры по мере того, как вы их читаете.

В заключение нужно вспомнить известную сентенцию, которая гласит, что если спросить двух статистиков, как анализировать определенный набор данных, получишь три разных ответа. Можно понимать этот афоризм по-разному – ведь каждый ответ приблизит вас к пониманию данных. Я не утверждаю, что предлагаемый мной способ анализа данных – лучший или единственный путь к решению конкретной задачи. Я предлагаю вам применить разные подходы к данным, используя знания, приобретенные во время чтения книги, и посмотреть, что вы сможете узнать. R – интерактивная программа, и лучший способ чему-то научиться в ней – это экспериментировать.

Принятые обозначения

В книге использованы следующие типографские обозначения:

- `моноширинный шрифт` использован для программного кода, который нужно вводить именно так, как указано в книге;
- `моноширинный шрифт` также использован внутри основного текста для обозначения фрагментов кода или ранее упомянутых объектов;
- *курсив* внутри программного кода – это указатель места заполнения. Его следует заменять подходящим текстом или значениями, соответствующими вашей задаче. Например, *путь_к_моему_файлу* должен быть заменен указанием пути к реальному файлу на вашем компьютере;
- R – это интерактивный язык, который информирует пользователя о готовности принять команду приглашением (> по умолчанию). Многие фрагменты программного кода в книге скопированы из интерактивных сессий. Если вы видите строки кода, которые начинаются с >, не набирайте этот символ приглашения к вводу команды;
- пояснения к программному коду приведены в виде внутритекстовых комментариев. В дополнение к этому некоторые пояснения обозначены красными пронумерованными кружками, такими как ❶, которые отсылают к объяснению ниже по тексту; для того чтобы сэкономить место или сделать текст более понятным, мы иногда добавляли в вывод результатов интерактивных сессий дополнительные пробелы или удаляли текст, который напрямую не относился к обсуждаемой теме.

Об авторе

Доктор наук Роберт Кабаков – заместитель президента по исследовательской работе (Vice President of Research) в группе исследований менеджмента (Management Research Group – MRG), международной фирме, специализированной на организационном развитии и консалтинге. У него за спиной более 20 лет опыта в сфере исследовательских и статистических консультаций в областях заботы о здоровье, финансовых операций, производства, бихевиоризма, управления и академической науки. Прежде чем присоединиться к MRG, Р. Кабаков был профессором психологии в юго-восточном университете Нова (Nova Southeastern University) во Флориде, где он преподавал количественные методы и статистическое программирование в магистратуре. В последние два года он поддерживает сайты Quick-R и учебного пособия по R.

Об иллюстрации с обложки

На обложке книги «R в действии» изображен «Мужчина из Задара». Эта иллюстрация позаимствована из альбома, посвященного хорватским национальным костюмам середины XIX века. Альбом составлен Николой Арсеновичем (Nikola Arsenovic) и опубликован в 2003 году музеем этнографии, расположенном в хорватском городе Сплит. Иллюстрация получена благодаря любезной помощи библиотекаря музея. Этот музей расположен в римской части средневекового центра города: недалеко от развалин дворца императора Диоклетиана, датированных примерно 304 годом нашей эры. Альбом состоит из красочных изображений людей из разных регионов Хорватии с описанием их костюмов и образа жизни.

Задар – это древнеримский город на севере далматского побережья Хорватии. Ему более 2000 лет, в течение сотен лет он был крупным портом по пути из Константинополя на запад. Расположенный на полуострове в окружении небольших островков Адриатического моря, этот живописный город служит популярной туристической достопримечательностью, привлекая своими архитектурными сокровищами – развалинами римских времен, рвами и старыми каменными стенами. Персонаж с обложки облачен в синие шерстяные брюки и белую льняную рубашку, поверх которой он надел синий жилет и куртку, отделанные характерной для этой района красочной вышивкой. Красные шерстяные пояс и шляпа дополняют костюм.

Принятая манера одеваться и стиль жизни заметно изменились за последние 200 лет. Региональные различия, столь заметные в прошлом, сильно размылись. Сейчас по внешнему виду сложно различить жителей разных континентов, не говоря уж о разных деревнях или городах, расположенных всего лишь на расстоянии нескольких километров друг от друга. Возможно, разнообразие культур преобразовалось в разнообразие личной жизни – и уж, конечно, в более разнообразную и динамичную технологичную жизнь.

Издательство Маннинг отмечает изобретательность и инициативность компьютерных технологий обложками книг, на которых представлено разнообразие региональных культур два столетия назад. Эти культуры оживают в нашей памяти благодаря иллюстрациям из старых книг и коллекциям, таким как эта.

Часть 1. Начало работы

Испытайте R в действии! R – одно из наиболее популярных современных программных средств для анализа данных и их визуализации. Это бесплатная программа с открытым кодом, предназначенная для операционных систем Windows, Mac OS X и Linux. Благодаря этой книге вы приобретете навыки, необходимые для того, чтобы овладеть этой многофункциональной программой и эффективно использовать ее для обработки ваших собственных данных.

Книга разделена на четыре части. Первая часть посвящена установке программы в ее базовой версии, знакомству с интерфейсом, импорту данных и преобразованию их в удобный для дальнейшего анализа вид.

Глава 1 познакомит вас с программной средой R. Эта глава начинается с обзора программы R и ее особенностей, которые делают ее столь мощным программным средством для современного анализа данных. После краткого объяснения того, как скачать и установить программу, пользовательский интерфейс описан на ряде простых примеров. Затем вы научитесь тому, как увеличить функциональность базовой версии при помощи расширений системы команд (так называемых дополнительных пакетов), которые можно скачать бесплатно с сетевых хранилищ. В конце главы размещены примеры, которые позволят применить ваши новые умения.

Как только вы познакомились с интерфейсом R, возникает следующая задача – загрузить ваши данные в программу. В современном богатом информацией мире данные могут поступать из разных источников и в разных форматах. В главе 2 описано множество методов, которые можно использовать для импорта данных в R. Первая половина главы посвящена описанию форматов, в которых R хранит данные, и рассказу о том, как вводить данные вручную. Во второй части обсуждаются методы импорта данных из текстовых файлов, веб-страниц, электронных таблиц, других статистических программ и баз данных.

Исходя из последовательности действий при обработке данных, возможно, следующим пунктом имело бы смысл обсудить управление данными и устранение ошибок в них. Однако многие пользователи, впервые познакомившиеся с R, больше интересуются ее мощными графическими возможностями. Чтобы не игнорировать этот интерес и не заставлять вас ждать, в главе 3 мы немедленно переходим к графическому представлению данных. В этой главе обсуждается, как создавать диаграммы, изменять их параметры и сохранять диаграммы в разных форматах. Рассказано, как на диаграммах выбирать цвета, типы символов и линий, шрифты, делать заголовки, надписи и списки условных обозначений. В заключение описано, как объединить несколько диаграмм на одном изображении.

После того, как вам представилась возможность испытать графические возможности R, настало время вернуться к анализу данных. Они редко сразу поступают в готовом к использованию виде. Часто бывает необходимо потратить значительное количество времени, комбинируя данные из различных источников, устраняя ошибки (неправильно закодированные, несоответствующие, отсутствующие данные) и создавая новые (комбинированные, трансформированные, перекодированные) переменные, прежде чем вы сможете перейти к решению интересующих вас задач. В главе 4 описаны все основные способы управления данными в R, включая сортировку, слияние и разделение наборов данных, а также трансформацию, перекодировку и удаление переменных.

Глава 5 основана на материале, изложенном в главе 4. Здесь рассказано, как использовать числовые (арифметические, тригонометрические и статистические) и текстовые функции (разбиение

строк, объединение и замену) в управлении данными. Для иллюстрации многих из описанных функций в этом разделе использованы многочисленные примеры. Далее разобраны управляющие конструкции (циклы, исполняемые при определенных условиях команды). После прочтения этого раздела вы научитесь создавать собственные функции в R. Это позволит расширить возможности R, объединив многие команды в одну легко настраиваемую функцию. В заключение обсуждаются мощные методы реорганизации и группировки данных, которые часто бывают полезными при подготовке данных к дальнейшему анализу.

После прочтения главы 1 вы подробно познакомитесь с программированием в среде R. Вы приобретете навыки, необходимые для ввода данных и получения их из внешних источников, а также для устранения ошибок в данных. Кроме того, вы получите опыт создания, изменения параметров и сохранения различных типов диаграмм.

Глава 1. Знакомство с R

В этой главе:

- устанавливаем R;
- знакомимся с языком программирования R;
- запускаем программу.

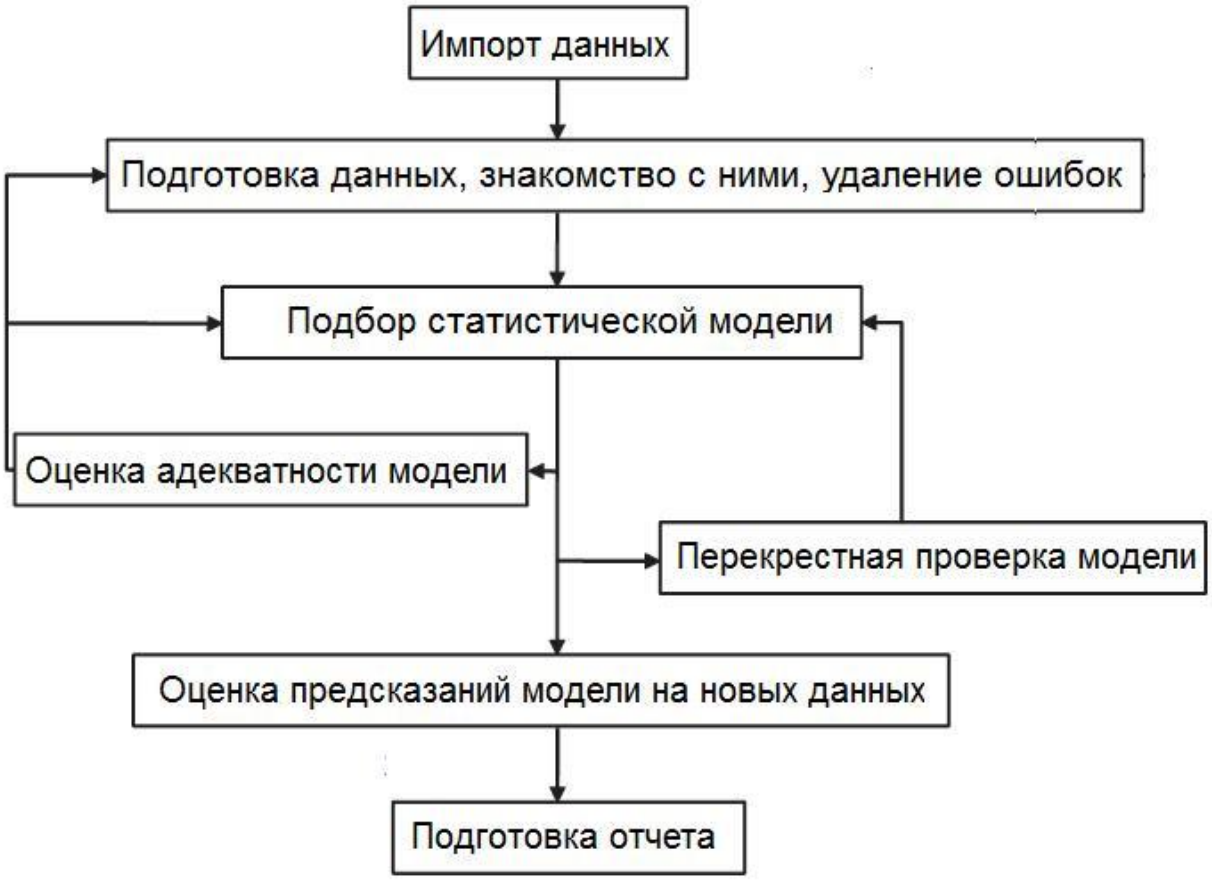
За последнее время методы анализа данных принципиально изменились. С появлением персональных компьютеров и Интернета объемы данных значительно возросли. Коммерческие компании обладают терабайтами данных о потребителях, правительственные, академические и частные исследовательские институты оперируют обширными архивными данными и материалами обследований по каждой теме исследований. Извлечение информации (не говоря уж о знаниях) из этих огромных объемов данных превратилось в самостоятельную отрасль деятельности. В то же время, представление информации в легко доступном и усвояемом виде стало более сложной задачей.

Развитие наук, посвященных анализу данных (статистика, психометрика, эконометрика, машинное самообучение) не отстают от взрывообразного роста объема данных. До эпохи персональных компьютеров и Интернета новые статистические методы разрабатывались учеными-теоретиками, которые публиковали свои результаты в виде статей в специализированных журналах. Могли пройти годы, прежде чем эти методы доходили до программистов и встраивались в широко доступные программы для статистической обработки данных. В наше время новые методики появляются *ежедневно*. Исследователи-статистики публикуют новые и усовершенствованные методы вместе с программным кодом, который их реализует, на легко доступных веб-сайтах.

Появление персональных компьютеров изменило подход к анализу данных еще одним образом. Когда анализ данных проводился в вычислительных центрах, компьютерное время было дорого и труднодоступно. Аналитики заранее тщательно устанавливали все параметры анализа. Когда анализ завершался, вывод результата занимал десятки или сотни страниц. Аналитик должен был просмотреть все их, оставляя нужное и отсеивая лишнее. Многие распространенные статистические программы были изобретены в ту эпоху, и до сих пор в некоторой степени придерживаются этого алгоритма.

С появлением дешевого и легкого доступа к анализу данных, предоставляемого персональными компьютерами, произошла смена парадигмы. Вместо того чтобы требовать предварительной установки всех параметров анализа, процесс стал в значительной степени интерактивным. При этом результат каждого этапа анализа служит данными для следующего этапа. Схема типичного анализа данных приведена на рис. 1.1. На любом этапе анализа могут быть произведены трансформация данных, вставка пропущенных значений, добавление или удаление переменных, после чего продолжается анализ данных. Этот процесс завершается, когда аналитик считает, что он или она полностью исследовал(а) данные и ответил(а) на все относящиеся к делу вопросы, на которые можно было ответить.

Рис. 1.1. Этапы типичного анализа данных



Появление персональных компьютеров (а в особенности мониторов с высоким разрешением) также повлияло на то, как мы понимаем результаты и представляем их. Изображение *действительно может* стоить тысячи слов, а люди весьма успешно извлекают информацию из визуальных образов. Современные методы обработки данных все больше опираются на графические способы представления результатов.

Итак, современному исследователю необходимо получать данные из разных источников (системы управления базами данных, текстовые файлы, статистические программы и электронные таблицы), объединять фрагменты данных, аннотировать их и удалять ошибки, анализировать их при помощи новейших методов, представлять результаты в легко поддающемся интерпретации и наглядном виде, а также включать результаты в привлекательные отчеты, которые могут быть распространены среди заинтересованных лиц и общественности.

1.1. Зачем использовать R?

R – это язык программирования и среда для статистических вычислений и графического анализа, сходный с языком S, первоначально разработанным в Белл Лабз (Bell Labs). Это программа для анализа данных с открытым кодом, которая поддерживается большим и активным исследовательским сообществом по всему миру. Однако существует много распространенных программ для статистической и графической обработки данных (таких как Microsoft Excel, SAS, IBM SPSS, Stata и Minitab). Зачем переходить на R?

У R есть много особенностей, которые позволяют рекомендовать именно его:

- большинство коммерческих статистических программ стоят тысячи, если не десятки тысяч долларов. R – это бесплатная программа! Если вы учитель или студент, выгода очевидна;
- R – это мощная статистическая программа, в которой реализованы все способы анализа данных;

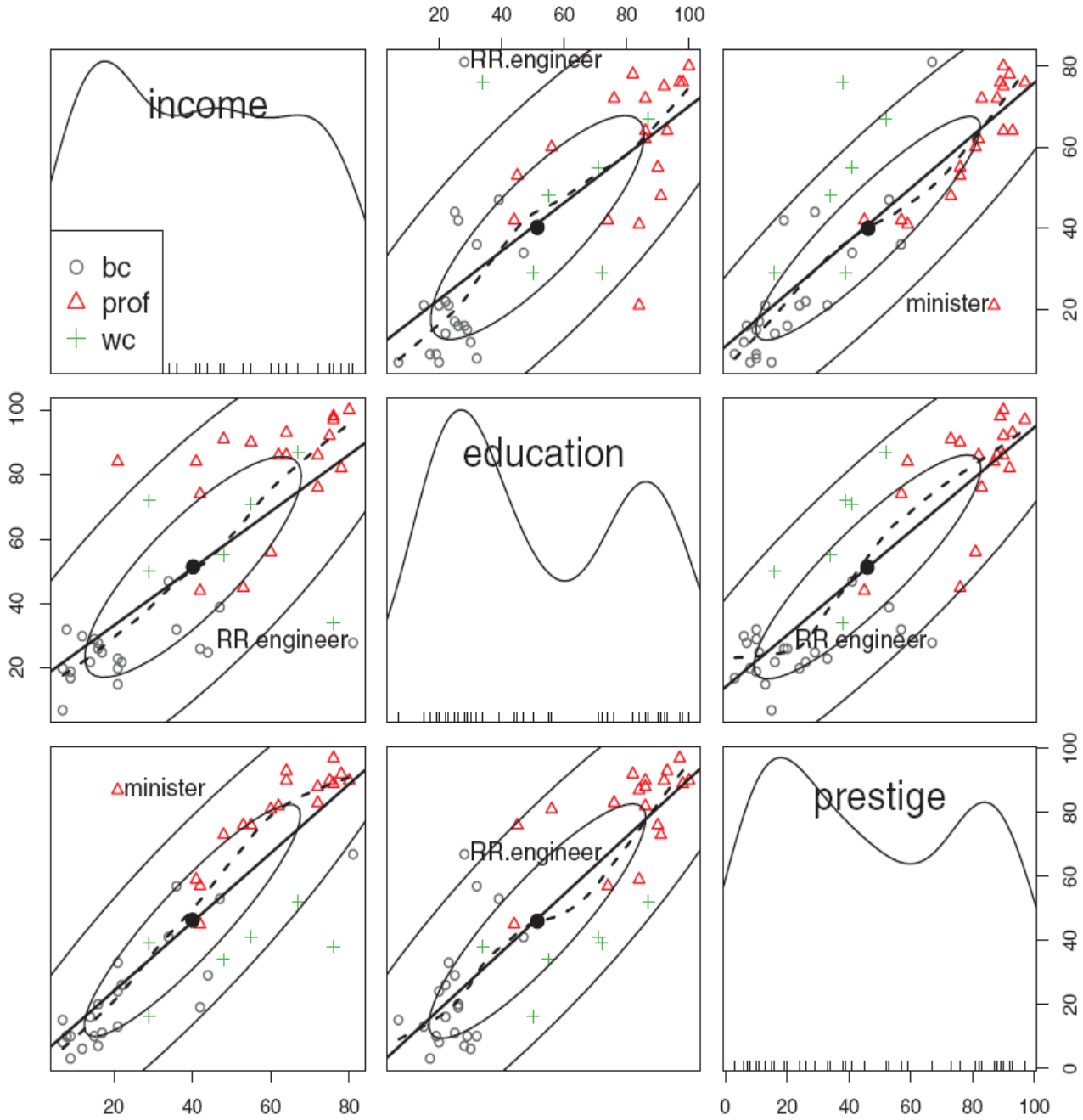
- R имеет современные графические возможности. Если вам нужно визуализировать сложные данные, то учтите, что в R реализованы самые разнообразные и мощные методы анализа данных из доступных;
- получение данных из разных источников в пригодном для использования виде может быть сложной задачей. R может импортировать данные из самых разных источников, включая текстовые файлы, системы управления базами данных, другие статистические программы и специализированные хранилища данных. R может также записывать данные в форматах всех этих систем;
- R представляет собой не имеющую аналогов платформу для простого написания программ, реализующих новые статистические методы;
- в R реализованы сложные статистические процедуры, еще недоступные в других программах. На самом деле, новые функции становятся доступными для скачивания еженедельно. Если вы работаете в SAS, представьте, насколько невероятно было бы получать новую версию программы каждые несколько дней;
- если вы не хотите учить новый язык, существует множество графических пользовательских интерфейсов, в которых мощь R реализована в форме меню и диалогов;
- R работает на разных операционных системах, включая Windows, Unix и Mac OS X. Он скорее всего запустится на любом компьютере, который у вас есть (я даже видел руководства по установке R на iPhone, что впечатляет, но вряд ли является хорошей идеей).

Демонстрацию графических возможностей R можно видеть на рис. 1.2. Эта диаграмма, созданная при помощи одной строки программного кода, описывает зависимости между доходом, образованием и престижем для рабочих, конторских и специальных профессий. Это матрица, составленная из диаграмм рассеяния, с обозначением разных групп при помощи цветов и символов, двумя типами аппроксимирующих линий – линейная регрессия и так называемое локально-взвешенное сглаживание (Locally Weighted Scatterplot Smoothing – LOESS), обозначающими границы доверительных интервалов эллипсами и двумя типами изображения плотности распределения точек на диаграмме (оценка плотности ядра – kernel density estimation и график-щетка[3] - rug plot). Кроме того, самый большой выброс на каждой диаграмме рассеяния автоматически подписан. Не беспокойтесь, если эти термины незнакомы вам. Мы подробно рассмотрим их в следующих главах. Сейчас просто поверьте мне, что это – замечательный график (и что статистики, которые читают это, просто в восторге).

-

[3] *Изобретение А.Б. Шипунова, до этого русскоязычный эквивалент отсутствовал – Прим. пер.*

Рис. 1.2. Взаимосвязь между уровнем дохода (income), образования (education) и престижа (prestige) для рабочих (blue-collar: bc), конторских (white-collar: wc) и специальных (prof) профессий. Источник: внешняя библиотека команд `car` (команда `scatterplotMatrix`), созданная Джоном Фоксом (John Fox). Диаграммы, подобные этой, сложно создавать в других статистических программах. В R их можно сделать при помощи одной-двух строк программного кода.



Вот главные вещи, которые видны на этом графике:

- между уровнями образования, дохода и престижа работы существует линейная связь;
- в среднем рабочие профессии характеризуются более низким уровнем образования и престижа, а специальные профессии – более высоким. Конторские профессии занимают промежуточное положение;
- существует несколько интересных исключений. Инженеры на железных дорогах (RR.engineer) зарабатывают много, но имеют низкое образование. Быть министром (minister) престижно, но не прибыльно;
- уровень образования и (в меньшей степени) престижа профессии распределены бимодально, то есть низкие и высокие значения этих показателей встречаются чаще, чем средние.

Вы узнаете гораздо больше об этой диаграмме в главе 8. Сейчас важно понять, что в R можно создавать изящные, информативные, легко настраиваемые диаграммы простыми и логичными способами. Создание сходных диаграмм при помощи других статистических программ было бы сложным, долгим или невозможным.

К сожалению, для R свойственна пологая кривая обучения. Поскольку он может делать так много всего, объем документации и файлов помощи очень велик. Кроме того, из-за того, что многие из этих умений реализованы в дополнительных модулях, созданных независимыми исследователями, эта документация бывает разобщенной и трудной для обнаружения. На самом деле, научиться выполнять все виды анализа, реализованные в R, очень непросто.

Задача этой книги – сделать овладение R быстрым и простым. Мы рассмотрим достаточно много возможностей R, чтобы вы смогли начать обработку собственных данных, с указаниями на то, где можно получить дополнительную информацию. Давайте начнем с установки программы.

1. 2. Получение и установка R

R можно бесплатно скачать из «всеобъемлющего сетевого архива R» (Comprehensive R Archive Network – CRAN) по адресу http://cran.r-project.org. Предварительно скомпилированные загрузочные файлы доступны для Linux, Mac OS X и Windows. Следуйте инструкциям по установке программы в вашей операционной системе. Позже мы обсудим, как повышать функциональность программы при помощи дополнительных модулей, называемых пакетами – они также доступны в CRAN. В приложении Н рассказано, как обновлять программу.

1.3. Работа в R

R – это чувствительный к регистру клавиатуры интерпретируемый язык программирования. Вы можете или вводить команды по одной в ответ на приглашение на ввод команды (>), или запускать набор команд из исходного файла. Типы данных очень разнообразны: векторы, матрицы, таблицы данных и списки (совокупность нескольких объектов). Мы обсудим все эти типы данных в главе 2.

Б**о**льшая часть «умений» программы реализуется при помощи встроенных и создаваемых пользователем функций, и все объекты (наборы данных) хранятся в памяти программы до завершения интерактивной сессии. Основные функции доступны по умолчанию. Остальные функции содержатся в пакетах, которые могут активироваться в ходе работы с программой по мере необходимости.

Командные строки состоят из функций и присвоений. R использует символ <- для присвоений, вместо обычного знака равенства. Например, командная строка

```
x <- rnorm(5)
```

создает объект типа вектор, который называется `x` и содержит пять случайных элементов нормального распределения.

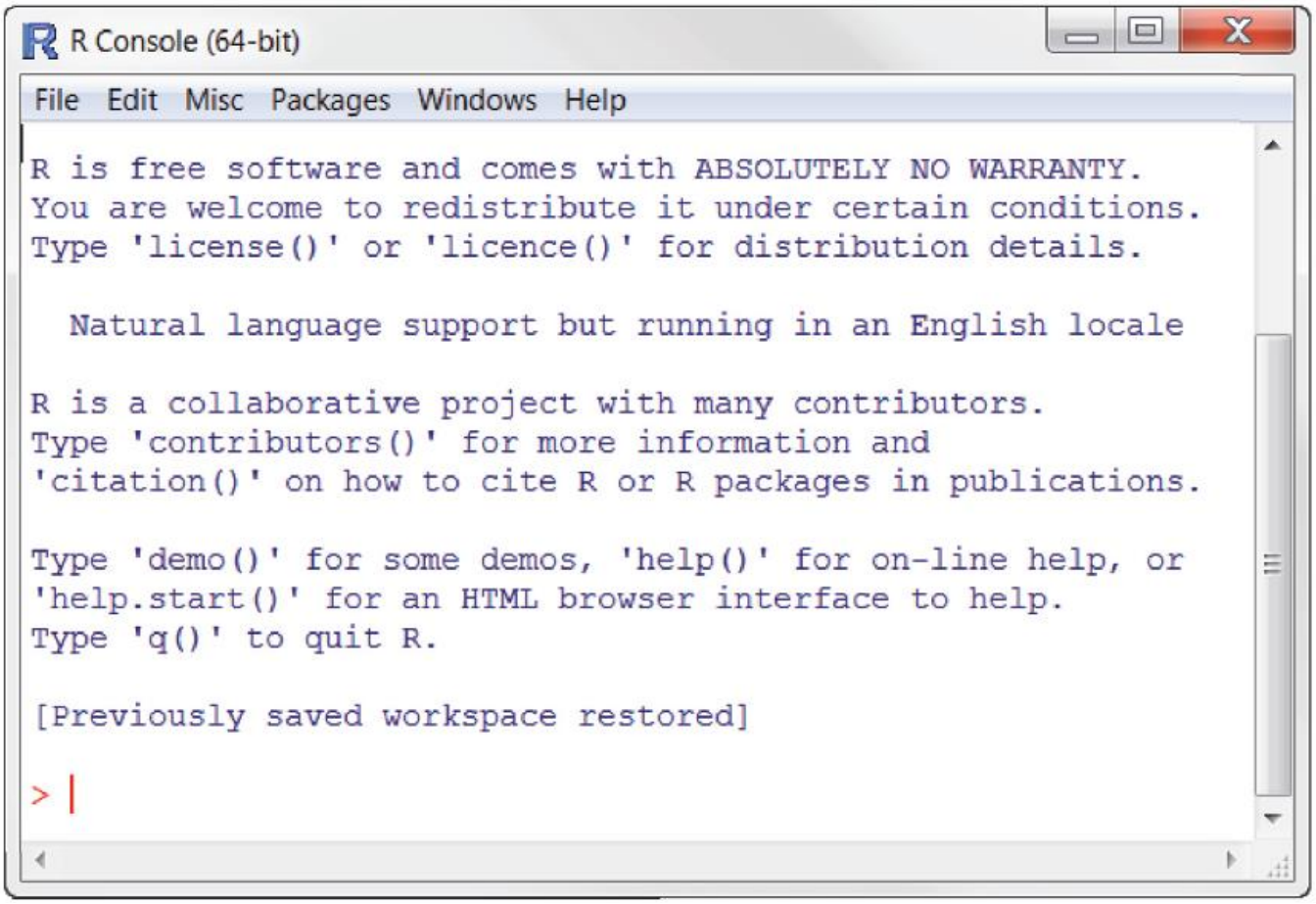
Примечание. *В R можно использовать знак равенства для присвоений. Однако немногие функции написаны таким образом. Поскольку это нестандартный синтаксис, в некоторых ситуаций он не будет работать, и программисты поднимут вас на смех. Можно писать присвоения в обратном порядке. Например, rnorm (5) -> x эквивалентно предыдущей командной строке. Опять же, так писать не принято, и я не рекомендую это делать.*

Комментарии предваряются символом #. Любой текст, который идет после #, игнорируется программой.

1.3.1. Начало работы

Если вы используете Windows, запустите R из стартового меню. В операционной системе Mac дважды щелкните на значок R в директории приложений. В Linux наберите «R» в командной строке терминала. Любое из этих действий запустит R (см. рис. 1.3 в качестве примера).

Рис. 1.3. Пример интерфейса R в операционной системе Windows



Для того чтобы освоиться с интерфейсом, давайте потренируемся на простом выдуманном примере. Представьте, что вы изучаете физическое развитие и собрали данные о возрасте и весе 10 младенцев первого года жизни (см. табл. 1.1). Вы интересуетесь распределением значений веса и их зависимостью от возраста.

Таблица 1.1. Возраст и вес 10 младенцев

Возраст (месяцы)	Вес (кг)	Возраст (месяцы)	Вес (кг)
01	4.4	09	7.3
03	5.3	03	6.0
05	7.2	09	10.4
02	5.2	12	10.2
11	8.5	03	6.1

Примечание. *Это выдуманные данные.*

Сначала вы вводите данные о возрасте и весе в виде векторов, используя функцию `c()`, которая преобразует свои аргументы в вектор или список. Затем вы вычисляете среднее арифметическое и стандартное отклонение для значений веса, а также коэффициент корреляции между возрастом и весом и графически изображаете взаимосвязь между этими двумя переменными, так, что вы можете изучить тренд визуально. Функция `q()`, как показано в приведенном ниже программном коде, завершает сессию и позволяет вам выйти из программы.

Программный код 1.1. Пример сессии в R

```
> age <- c(1,3,5,2,11,9,3,9,12,3)
> weight <- c(4.4,5.3,7.2,5.2,8.5,7.3,6.0,10.4,10.2,6.1)
> mean(weight)
[1] 7.06
> sd(weight)
[1] 2.077498
> cor(age,weight)
[1] 0.9075655
> plot(age,weight)
> q()
```

Из программного кода 1.1 видно, что средний вес этих 10 младенцев составляет 7.06 кг, а стандартное отклонение равно 2.08 кг, и что существует сильная линейная взаимосвязь между возрастом и весом (коэффициент корреляции равен 0.91). Эта взаимосвязь также видна на диаграмме рассеяния на рис. 1.4. Неудивительно, что по мере взросления младенцы в среднем становятся тяжелее.

Рис. 1.4. Диаграмма рассеяния веса младенцев (weight, в килограммах) в зависимости от их возраста (age, в месяцах)

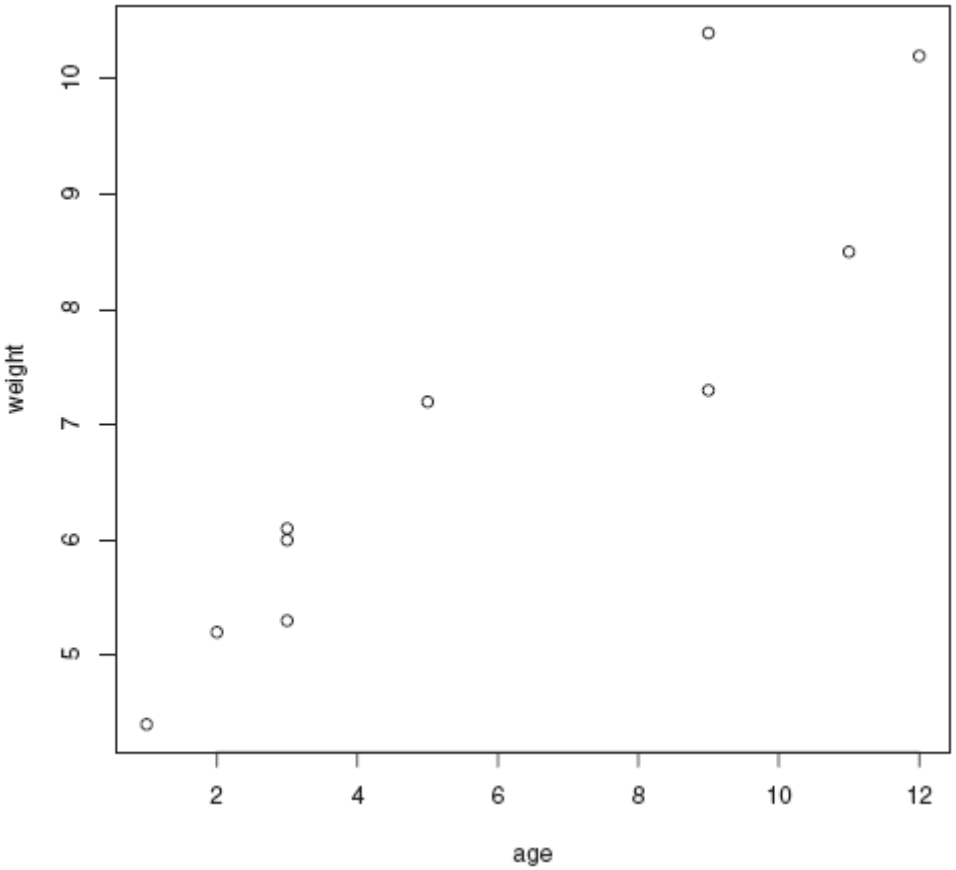
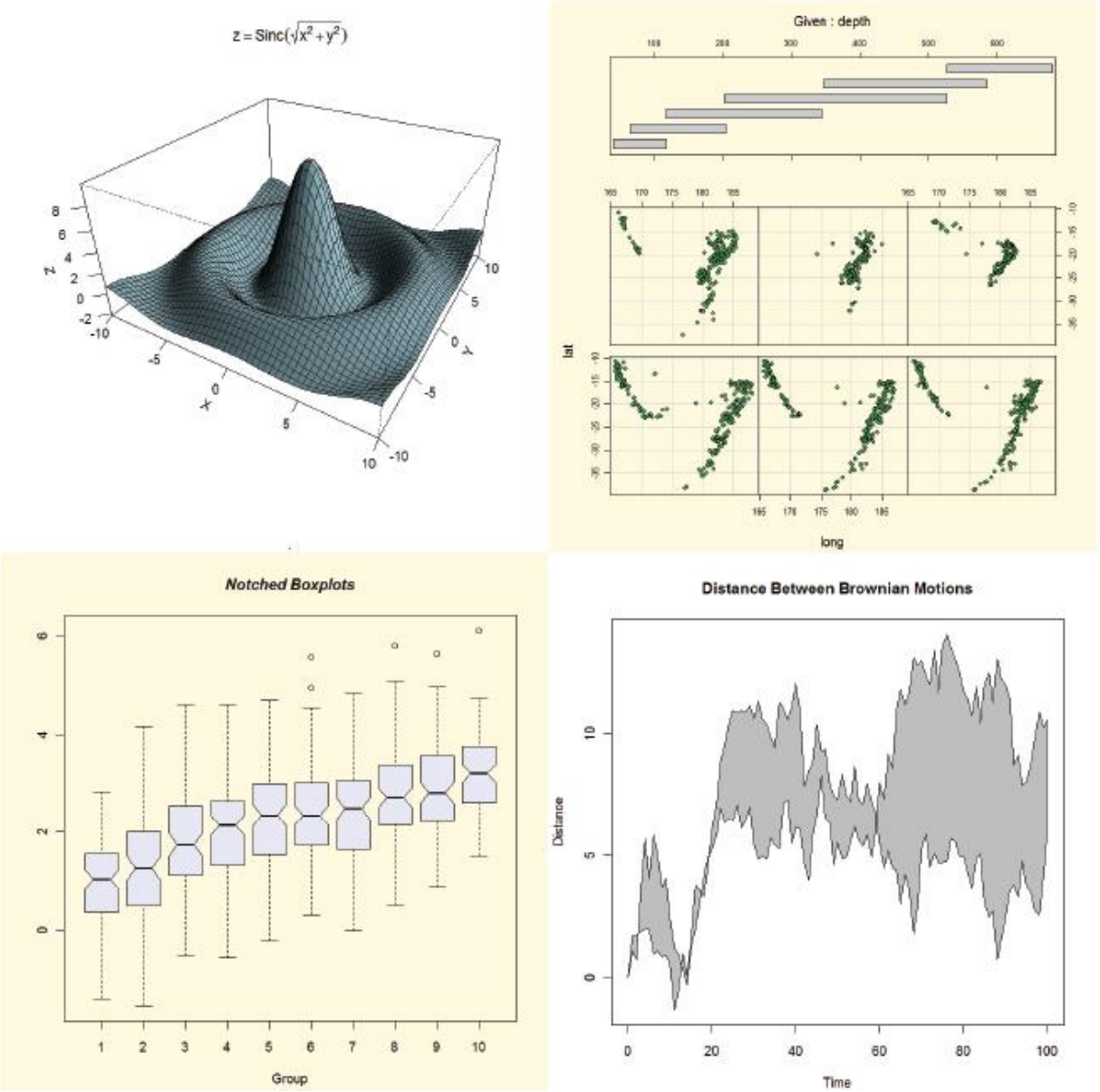


Диаграмма рассеяния на рис. 1.4 информативна, но выглядит не очень привлекательно. Из последующих глав вы узнаете, как модифицировать графики так, чтобы они удовлетворяли вашим потребностям.

Совет. *Чтобы получить представление о графических возможностях R, наберите в командной строке* `demo(graphics)`. *Некоторые из графиков, которые при этом появятся, представлены на рис. 1.5. Другие демонстрационные наборы графиков можно получить, напечатав* `demo(Hershey)`, `demo(persp)` *и* `demo(image)`. *Для того чтобы увидеть полный набор демонстрационных графиков, введите* `demo()` *без параметров.*

Рис. 1.5. Некоторые графики, которые появляются при вводе функции `demo()`



1.3.2. Как получить помощь

R обладает обширными справочными материалами. Научитесь ориентироваться в них, и это поможет вам при работе в программе. Встроенная система помощи содержит подробные разъяснения, ссылки на литературу и примеры для каждой функции из установленных пакетов.[4] Справку можно вызвать при помощи функций, перечисленных в табл. 1.2.

[4] *На английском языке. – Прим. пер.*

Таблица 1.2. Функции вызова справки в R

Функция	Действие
`help.start()`	Общая справка
`help ("foo")` или `?foo`	Справка по функции `foo` (кавычки необязательны)
`help.search("foo")` или `??foo`	Поиск в справке записей, содержащих `foo`
`example("foo")`	Примеры использования функции `foo` (кавычки необязательны)
`RSiteSearch("foo")`	Поиск записей, содержащих `foo` в онлайн-руководствах и заархивированных рассылках
`apropos("foo", mode="function")`	Список всех доступных функций, в названии которых есть `foo`
`data()`	Список всех демонстрационных данных, содержащихся в загруженных пакетах
`vignette()`	Список всех доступных руководств по загруженным пакетам
`vignette("foo")`	Список руководств по теме `foo`

Функция `help.start()` открывает окно браузера с перечнем доступных руководств разного уровня сложности, часто задаваемых вопросов и ссылок на источники. Функция `RSiteSearch()` осуществляет поиск на заданную тему в онлайн-руководствах и архивах рассылок и представляет результаты в окне браузера. Функция `vignette()` вызывает список всех вводных статей в формате PDF. Такие статьи написаны не для всех пакетов. Как вы могли убедиться, R предоставляет обширные справочные материалы, и умение ориентироваться в них пойдет вам на пользу. Можно пересчитать по пальцам сессии, во время которых я не использую справку, чтобы получить подробную информацию (опции или возвращаемые значения) о какой-нибудь функции.

1.3.3. Рабочее пространство

Рабочее пространство – это текущая рабочая среда R, которая включает в себя любые созданные пользователем объекты (векторы, матрицы, функции, таблицы данных или списки). В конце каждой сессии вы можете сохранить рабочее пространство, и оно автоматически загрузится при следующем запуске программы. Команды интерактивно вводятся в ответ на приглашение к их вводу. Можно использовать стрелки вверх и вниз для того, чтобы перемещаться между введенными ранее командами. Это позволяет вызвать предыдущую команду, отредактировать ее и вновь выполнить ее, нажав клавишу **Enter**.

Текущая рабочая директория – это та директория, где находятся файлы данных и куда по умолчанию сохраняются результаты. Функция `getwd()` позволяет узнать, какая директория в данный момент является рабочей. Вы можете назначить рабочую директорию при помощи функции `setwd()`. Если появляется необходимость импортировать файл, который находится не в рабочей директории, нужно написать полный путь к нему. Всегда заключайте в кавычки названия файлов и директорий.

Некоторые наиболее употребимые команды для управления рабочим пространством приведены в табл. 1.3.

Таблица 1.3. Функции, использующиеся для управления рабочим пространством в R

Функция	Действие
`getwd()`	Вывести на экран название текущей рабочей директории
`setwd("mydirectory")`	Назначить `mydirectory` текущей рабочей директорией
`ls()`	Вывести на экран список объектов в текущем рабочем пространстве
`rm(objectlist)`	Удалить один или несколько объектов
`help(options)`	Справка о возможных опциях
`options()`	Посмотреть или установить текущие опции
`history(#)`	Вывести на экран последние # команд (по умолчанию 25)
`savehistory("myfile")`	Сохранить историю команд в файл `myfile` (по умолчанию `.Rhistory`)
`loadhistory("myfile")`	Загрузить историю команд (по умолчанию `.Rhistory`)
`save.image("myfile")`	Сохранить рабочее пространство в файл `myfile` (по умолчанию `.Rdata`)
`save(objectlist, file="myfile")`	Сохранить определенные объекты в файл
`load("myfile")`	Загрузить сохраненное рабочее пространство в текущую сессию (по умолчанию `.Rdata`)
`q()`	Выйти из программы. Появится вопрос, нужно ли сохранить рабочее пространство.

Чтобы увидеть эти команды в действии, рассмотрим следующий программный код.

Программный код 1.2. Пример действия команд, используемых для управления рабочим пространством R

```
setwd("C:/myprojects/project1")
options()
options(digits=3)
x <- runif(20)
summary(x)
hist(x)
savehistory()
save.image()
q()
```

Сначала назначена текущая рабочая директория, выведены на экран действующие значения параметров и указано, что числа должны выводиться с тремя знаками после десятичного разделителя. Затем создан вектор из 20 случайных чисел и для него вычислены основные статистики и построена гистограмма. Наконец, история команд сохранена в файл `.Rhistory`, рабочее пространство (включая объект `x`) сохранено в файл `.Rdata` и сессия завершена.

Обратите внимание на то, то при указании пути к директории в команде `setwd()` использованы прямые слеши (`/`). R воспринимает обратный слеш (`\`) как символ выхода из программы. Даже если вы используете R на базе Windows, используйте прямые слеши в указании путей к файлам и директориям. Отметим также, что команда `setwd()` не будет создавать указанную директорию, если она не существует. При необходимости вы можете использовать функцию `dir.create()` для создания директории, а уже затем – `setwd()` для того, чтобы назначить эту директорию рабочей.

Разумно иметь отдельную директорию для каждого проекта. Я обычно начинаю любую сессию R с указания нужной рабочей директории при помощи функции `setwd()`, а затем ввожу команду `load()` без аргументов. Это обозначает окончание предыдущей сессии и помогает не смешивать данные и установочные параметры разных проектов. В операционных системах Windows и Mac OS X все еще проще. Нужно всего лишь найти файл с сохраненным рабочим пространством и дважды щелкнуть на него. Это действие запустит R, загрузит сохраненное рабочее пространство и назначит директорию, где находился этот файл, рабочей.

1.3.4. Ввод и вывод

По умолчанию запуск R начинает интерактивную сессию, где ввод осуществляется с клавиатуры, а результаты выводятся на экран. Однако также можно запустить команды из так называемого скрипта (файла, который содержит функции R), а вывод возможен напрямую в разные устройства.

Ввод

Функция `source("filename")` запускает скрипт. Если не прописан путь к файлу, подразумевается, что он находится в текущей рабочей директории. Например, команда `source("myscript.R")`, запускает серию команд R, которые записаны в файле `myscript.R`. Принято, чтобы файлы скриптов имели расширение `.R`, но это не обязательное условие.

Текстовый вывод

Функция `sink("filename")` выводит все результаты в файл `filename`. По умолчанию, если этот файл уже существует, то новая версия записывается поверх старой. Параметр `append=TRUE` позволяет добавлять новый текст в файл, а не записывать его вместо старого текста. Параметр `split=TRUE` позволяет выводить результаты и на экран, и в текстовый файл. Функция `sink()` без аргументов – команда выводить результаты снова только на экран.

Графический вывод

Хотя команда `sink()` управляет выводом текста, она не оказывает никакого воздействия на вывод графики. Для того чтобы управлять выводом изображений, используйте одну из функций, указанных в табл. 1.4. Функция `dev.off()` направляет вывод графики обратно на экран.

Таблица 1.4. Функции для сохранения графиков

Функция	Вывод (формат графического файла)
`pdf("filename.pdf")`	PDF
`win.metafile("filename.wmf")`	Windows metafile
`png("filename.png")`	PNG
`jpeg("filename.jpg")`	JPEG
`bmp("filename.bmp")`	BMP
`postscript("filename.ps")`	PostScript

Давайте рассмотрим все это в одном примере. Представьте, что у вас есть три файла-скрипта, в которых содержится программный код R (`script1.R`, `script2.R` и `script3.R`). Команда `source("script1.R")` запустит программный код, содержащийся в файле `script1.R`, и выведет результаты на экран.

Если вы наберете следующие команды:

```
sink("myoutput", append=TRUE, split=TRUE)
pdf("mygraphs.pdf")
source("script2.R")
```

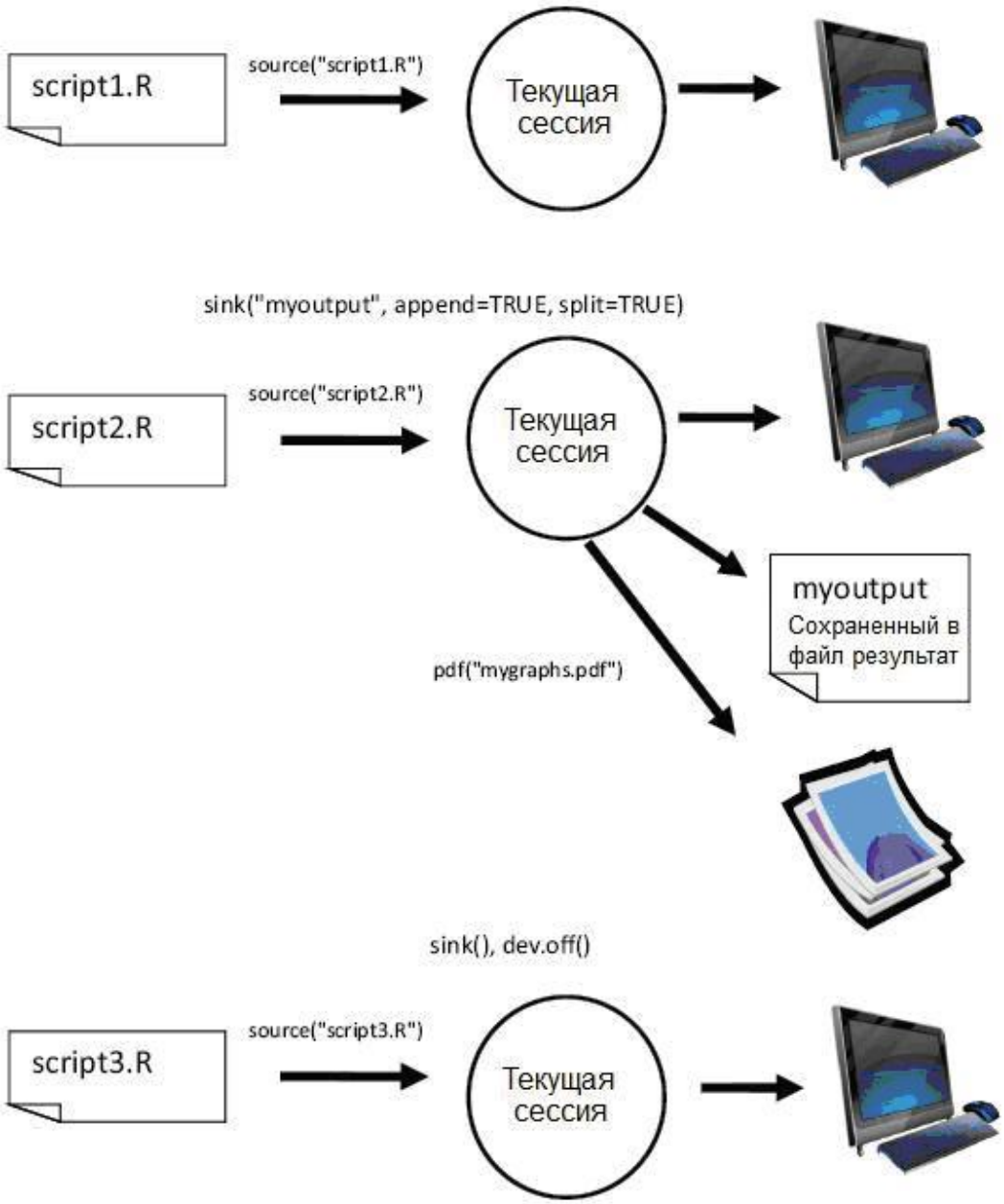
будет выполнен программный код из файла `script2.R`, и результаты вновь появятся на экране. Кроме того, текстовый вывод будет добавлен к содержимому файла `myoutput`, а графический вывод – сохранен в файле `mygraphs.pdf`.

Наконец, если вы введете такие команды:

```
sink()
dev.off()
source("script3.R")
```

будет выполнен программный код из файла `script3.R` и результаты появятся на экране. Ни текстовый, ни графический вывод не будут сохранены в файлы. Вся последовательность действий представлена на рис. 1.6.

Рис. 1.6. Ввод при помощи функции `source()` и вывод посредством функции




```
sink()
```

R предоставляет достаточно много возможностей для контроля ввода данных и вывода результатов. В разделе 1.5 вы узнаете, как запускать программу в пакетном режиме (batch mode).

1.4. Пакеты

R в базовой установке уже обладает обширными возможностями. Однако некоторые его наиболее впечатляющие «умения» реализованы в дополнительных модулях, которые можно скачать и установить. Существует более 2 500 созданных пользователями модулей, называемых *пакетами* (packages), которые вы можете скачать с http://cran.r-project.org/web/packages. В них заключены почти безграничные возможности – от анализа геостатистических данных до масс-спектроскопии белков и анализа психологических тестов! Мы используем многие из этих дополнительных пакетов по ходу повествования.

1.4.1. Что такое пакеты?

Пакеты – это собрания функций R, данных и скомпилированного программного кода в определенном формате. Директория, в которой пакеты хранятся на вашем компьютере, называется библиотекой. Функция `.libPath()` показывает, где расположена ваша библиотека, а функция `library()` выводит на экран названия всех имеющихся в библиотеке пакетов.

R поставляется уже со стандартным набором пакетов (включая `base`, `datasets`, `utils`, `grDevices`, `graphics` и `methods`). В них уже заключены разнообразные функции и наборы данных, доступных по умолчанию. Другие пакеты нужно скачивать и устанавливать. После установки, они загружаются во время сессии по мере необходимости. Команда `search()` выводит на экран названия загруженных и готовых к использованию пакетов.

1.4.2. Установка пакета

В R существует множество функций, предназначенных для управления пакетами. Для того чтобы установить пакет, используйте команду `install.packages()`. Эта команда, введенная без аргументов, вызовет список зеркал сайта CRAN. После выбора зеркала, вы увидите список всех доступных пакетов. Выберите один из них, и он будет скачан и загружен. Если вы знаете название пакета, который хотите установить, то можете сделать это, сообщив функции это название в виде аргумента. Например, пакет `gclus` содержит функции для создания усовершенствованных диаграмм рассеяния. Этот пакет можно скачать и установить при помощи команды `install.packages("gclus")`.

Пакет нужно установить только один раз. Однако, как и любые другие программы, пакеты часто обновляются своими разработчиками. Используйте команду `update.package()` для обновления всех установленных пакетов. Для того чтобы получить подробную информацию об установленных пакетах, можно использовать команду `installed.packages()`. Она выводит на экран список всех имеющихся пакетов с номерами их версий, названиями пакетов, от которых они зависят и другой информацией.

1.4.3. Загрузка пакета

Когда вы устанавливаете пакет, он скачивается с сайта CRAN и загружается в вашу библиотеку. Для того чтобы использовать этот пакет в текущей сессии программы, вам нужно загрузить его при помощи команды `library()`. Например, для того чтобы использовать пакет `gclus`, введите команду `library(gclus)`. Разумеется, прежде чем загрузить пакет, необходимо его установить. В течение сессии достаточно загрузить пакет один раз. При необходимости можно настроить рабочее

пространство так, чтобы часто используемые пакеты автоматически загружались в начале каждой сессии. Настройка начала работы подробно описана в приложении B.

1.4.4. Как получить больше информации о пакете

После загрузки пакета становятся доступны новые функции и наборы данных. Небольшие наборы данных поставляются вместе с демонстрационным программным кодом, что позволяет испробовать новые возможности. Система справки содержит описание каждой функции (с примерами) и информацию о каждом встроенном наборе данных. Функция `help(package="`*`название_пакета`*`")` выводит короткое описание этого пакета и алфавитный указатель всех входящих в него функций и наборов данных. Использование команды `help()` для названия каждой из этих функций или наборов данных позволит выяснить новые детали. Эту же информацию можно скачать с сайта CRAN в виде руководства в формате PDF.

Распространенные ошибки при программировании в R:

Существует ряд распространенных ошибок, которые часто допускают и новички, и опытные программисты на языке R. Если программы выдает сообщение об ошибке, проверьте, не сделали ли вы что-то из нижеперечисленного:

- использовали неправильный регистр: help(), Help() и HELP() – это три разные функции (только первая будет работать);
- забыли поставить кавычки там, где они необходимы: install.packages("gclus") работает, а install.packages(gclus) выдает сообщение об ошибке;
- забыли поставить скобки при обращении к функции: к примеру, нужно набирать help(), а не help. Даже если аргументы отсутствуют, скобки все равно необходимы;
- использовали \ в указании пути к файлу в операционной системе Windows: R воспринимает обратный слеш как символ выхода. setwd("c:\mydata") порождает сообщение об ошибке. Используйте вместо этого setwd("c:/mydata") или setwd("c:\\mydata");
- ввели функцию из пакета, который еще не загрузили. Функция order.clusters() содержится в пакете gclus. Если вы попробуете использовать ее до того, как загрузили этот пакет, возникнет сообщение об ошибке.

Сообщения об ошибках в R могут быть непонятными, однако появление многих таких сообщений можно предотвратить, если внимательно следовать вышеперечисленным правилам.

1.5. Пакетная обработка

Как правило, вы работаете в R интерактивно, то есть набираете команды в ответ на приглашение на ввод и видите результат выполнения каждой команды. В какой-то момент у вас может появиться желание запустить R в повторяющемся стандартном автоматическом режиме. Например, если один раз в месяц вам нужно сдавать один и тот же отчет, вы можете написать программу на языке R и запускать ее в пакетном режиме.

Способ запуска программы в пакетном режиме зависит от типа операционной системы. В Linux или Mac OS X это можно делать при помощи ввода следующей команды в окне терминала:

```
R CMD BATCH options infile outfile,
```

где *`infile`* – имя файла с программным кодом R, который должен быть выполнен, *`outfile`* – имя файла, в который будут записаны результаты, а в файле `options` перечислены опции, которые определяют параметры выполнения кода. Принято, чтобы *`infile`* имел расширение `.R`, а *`outfile`* – расширение *`.Rout`*.

В Windows используйте команду

```
"C:\Program Files\R\R-2.13.0\bin\R.exe" CMD BATCH
➥--vanilla --slave "c:\my projects\myscript.R",
```

в которой прописаны пути к файлу R.exe и к файлу с программным кодом. Дополнительная информация по этой теме, включая использование опций в командной строке, содержится в пособии «Введение в R» («Introduction to R»)[5] на сайте CRAN (http://cran.r-project.org).

[5] *На английском языке. – Прим. пер.*

1.6. Использование вывода в качестве ввода – повторное использование результатов

Одна из наиболее полезных характеристик R – возможность легко сохранить результат анализа данных и использовать его как ввод для дальнейшего анализа. Давайте рассмотрим это на примере с использованием встроенного набора данных. Если вы не понимаете, что за статистические методы здесь применены, не волнуйтесь. Здесь для нас важен общий принцип.

Для начала рассчитаем простую линейную регрессию, описывающую зависимость расхода топлива в милях на галлон (miles per gallon – mpg) от веса машины (weight – wt) для встроенного набора данных `mtcars`. Это делается при помощи следующей функции:

```
lm(mpg~wt, data=mtcars).
```

Результаты выводятся на экран, но не сохраняются.

Затем выполним ту же операцию, сохранив результаты в виде объекта:

```
lmfit <- lm(mpg~wt, data=mtcars).
```

Это присвоение создало объект типа список с названием `lmfit`, в котором записаны подробные результаты анализа (включая предсказанные значения, остатки, коэффициенты регрессии и так далее). Хотя при этом на экране ничего не появилось, результаты можно просматривать и подвергать дальнейшей обработке.

Команда `summary(lmfit)` выводит на экран основные характеристики (summary) результатов, при помощи команды `plot(lmfit)` можно построить диагностические графики. Функция `cook<-cooks.distance(lmfit)` рассчитывает статистики воздействия (influence statistics), а функция `plot(cook)` отображает их графически. Для того чтобы предсказать расход топлива, зная вес машины, по новым данным, нужно использовать `predict(lmfit, mynewdata)`.

Для того чтобы узнать, какие значения возвращает функция, загляните в раздел «значение» (Value) онлайн-справки для этой функции[6]. В командной строке можно набрать `help(lm)` или `?lm`. Так вы узнаете, что было сохранено, когда вы присвоили результаты функции объекту.

1.7. Работа с большими массивами данных

Программисты часто спрашивают меня, может ли R обрабатывать большие массивы данных. Как правило, они работают со значительными объемами данных, собранных в ходе исследований Интернета, климата или генетики. Поскольку R удерживает объекты в памяти, обычно объем оперативной памяти (RAM) является лимитирующим фактором. Например, на моем компьютере с операционной системой Windows и 2 Гб оперативной памяти, произведенном пять лет назад, я могу с легкостью обрабатывать данные из 10 миллионов элементов (100 признаков 100 000 объектов). На iMac с 4 Гб оперативной памяти я, как правило, могу без особых затруднений обрабатывать данные, состоящие из 100 миллионов элементов.

1.8. Учимся на примере

Закончим эту главу примером, который объединяет многие высказанные идеи. Вот задание:

1. Откройте общий файл справки и загляните в раздел «Введение в R».
2. Установите пакет `vcd` (пакет для визуализации категориальных данных, который мы будем более подробно рассматривать в главе 11).
3. Посмотрите на список всех функций и наборов данных, содержащихся в пакете.
4. Загрузите пакет и прочтите описание набора данных `Arthritis`.
5. Выведите этот набор данных на экран (набрав его название в командной строке).

---

[6] *На английском языке, как и все прочие упомянутые справочные материалы. – Прим. пер.*

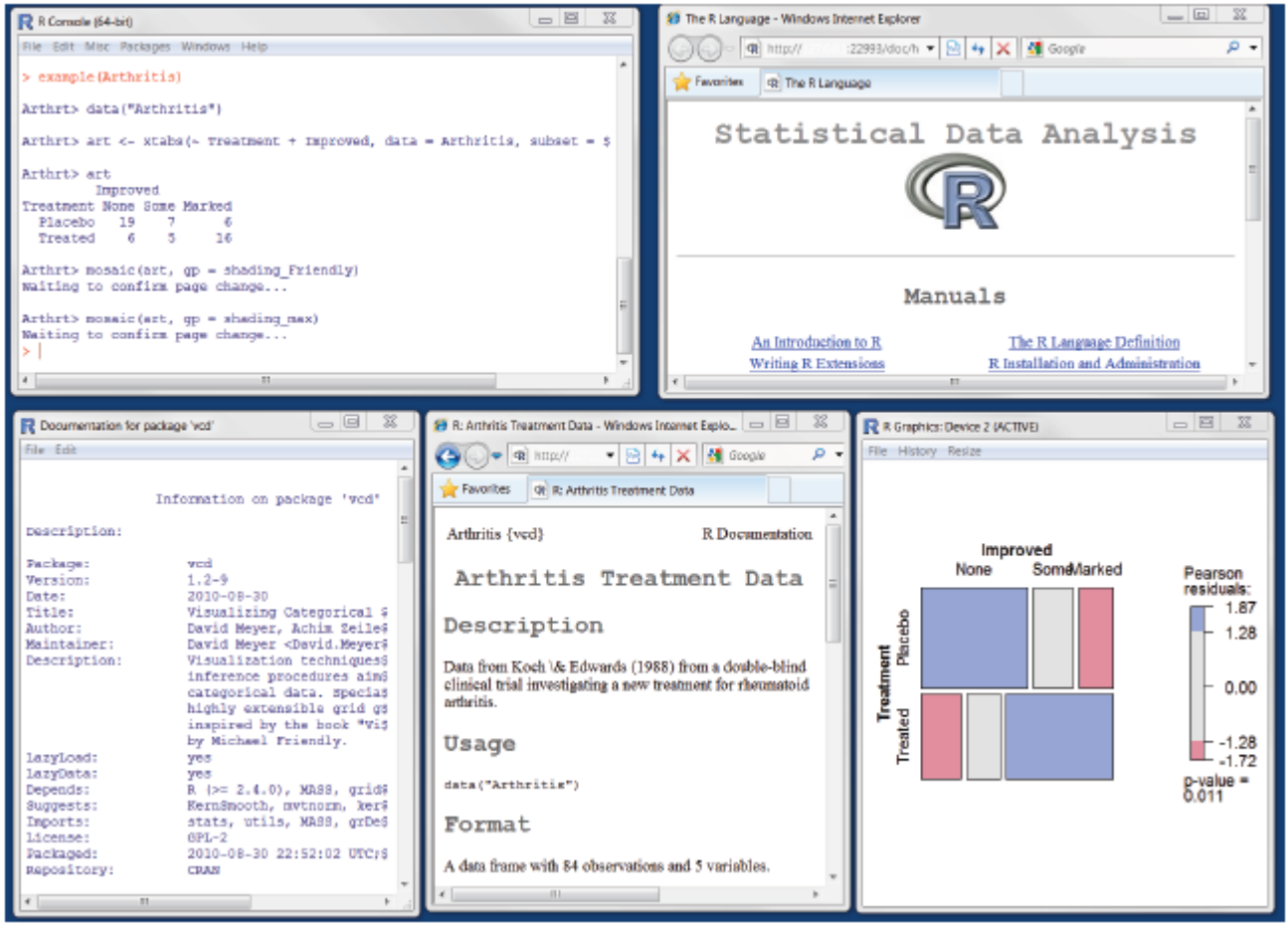
6. Запустите пример, который прилагается к набору данных `Arthritis`. Не беспокойтесь, если вы не понимаете результаты. В общих чертах они показывают, что страдающие артритом пациенты, которые получали лекарство, выздоравливали гораздо быстрее по сравнению с теми, которые получали плацебо.
7. Выйдите из программы.

Программный код, который вам понадобится, приведен ниже вместе с некоторыми результатами, представленными на рис. 1.7.

Программный код 1.3. Работа с новым пакетом

```
help.start()
install.packages("vcd")
help(package="vcd")
library(vcd)
help(Arthritis)
Arthritis
example(Arthritis)
q()
```

Рис. 1.7. Результат действия программного кода 1.3, включающий (слева направо) примера обработки набора данных `Arthritis`, общую помощь, сведения о пакете `vcd`, сведения о наборе данных `Arthritis` и диаграмму, иллюстрирующую зависимость результата лечения артрита от способа лечения.



Как показывает это упражнение, вы можете сделать многое при помощи короткого программного кода.

1.9. Резюме

В этой главе мы рассмотрели некоторые сильные стороны R, которые делают эту программу привлекательной для студентов, статистиков и аналитиков, которые хотят понять, что значат их

данные. Мы обсудили, как устанавливать программу и как расширить ее возможности путем установки дополнительных пакетов. Мы исследовали основные характеристики интерфейса, запустили программу интерактивно и в пакетном режиме, а также создали несколько графиков для примера. Вы также узнали, как сохранять результаты работы в текстовых и графических файлах. Поскольку R может вызывать затруднения, мы потратили немного времени на то, чтобы понять, как получить доступ к обширной справочной информации. Надеюсь, вы почувствовали, какой мощной может быть эта бесплатная программа.

Теперь, когда у вас установлена готовая к использованию программа R, настало время поработать с вашими данными. В следующей главе мы рассмотрим типы данных, с которыми работает R, и узнаем, как импортировать данные из текстовых файлов, других программ и систем управления базами данных.

Глава 2. Создание набора данных

В этой главе:

- изучаем структуры данных в R;
- вводим данные;
- импортируем данные;
- аннотируем наборы данных.

Первый этап любого анализа данных – создание набора данных, в котором содержится информация для изучения, в подходящем формате. В R эта задача распадается на следующие:

- выбор типа данных;
- ввод или импорт данных в выбранном формате.

Первая часть этой главы (разделы 2.1—2.2) содержит описание многочисленных типов данных, используемых в R. В частности, в разделе 2.2 обсуждаются векторы, факторы, матрицы, таблицы данных и списки. Знакомство с этими типами данных (и обозначениями, используемыми для доступа к отдельным их элементам) очень поможет вам в понимании того, как работает R. Возможно, вы захотите уделить этому разделу достаточно много времени.

Вторая часть главы (раздел 2.3) посвящена разным способам импорта данных в R. Данные можно вводить вручную или импортировать из внешнего источника. Таким источником могут быть текстовые файлы, электронные таблицы, статистические программы и системы управления базами данных. Например, я обычно работаю с данными, которые изначально содержатся в базах данных языка структурированных запросов (Structured Query Language – SQL). Хотя иногда я получаю данные из старомодных дисковых операционных систем (Disk Operating System – DOS) и из современных баз данных SAS и SPSS. Вероятно, вам достаточно будет использовать один или два метода из описанных в этом разделе, так что просто выберите тот, что вам подходит.

После того, как набор данных создан, его, как правило, нужно аннотировать, добавив подписи для переменных и кодов данных. Третья часть главы (раздел 2.4) посвящена аннотированию наборов данных и обзору некоторых полезных функций для работы с наборами данных (раздел 2.5). Давайте начнем с самого начала.

2.1. Что такое набор данных?

Набор данных – это, как правило, прямоугольный массив данных, в котором ряды соответствуют наблюдениям, а столбцы – признакам. В табл. 2.1 представлен воображаемый набор данных о пациентах.

Таблица 2.1. Набор данных о пациентах

PatientID	AdmDate	Age	Diabetes	Status
1	10/15/2009	25	Type1	Poor
2	11/01/2009	34	Type2	Improved
3	10/21/2009	28	Type1	Excellent
4	10/28/2009	52	Type1	Poor

Примечание: PatientID – порядковый номер пациента; AdmDate (admission date) – дата поступления: месяц/день/год; Age – возраст; Diabetes – тип диабета (Type1 – первый тип, Type2 – второй тип); Status – состояние (Poor – плохое; Improved – улучшенное; Excellent – превосходное).

Представители разных профессий по-разному называют строки и столбцы в наборе данных. Статистики называют их наблюдениями (observation) и переменными (variable), аналитики, которые работают с базами данных, говорят о записях (record) и полях (field), а те, кто работает в области нахождения в данных скрытой информации (data mining) и машинного обучения (machine learning), называют их образцы (example) и свойства (attribute). На протяжении этой книги мы будем использовать термины *наблюдение* и *переменная*.

Нужно различать структуру набора данных (в данном случае – прямоугольный массив данных) и типы данных, которые его составляют. В наборе данных, приведенном в табл. 2.1 `PatientID` – это название строки или наблюдения, `AdmDate` – переменная в формате даты, `Age` – непрерывная (или количественная – continuous) переменная, `Diabetes` – номинальная (nominal) переменная и `Status` – это порядковая (или шкальная – ordered) переменная.

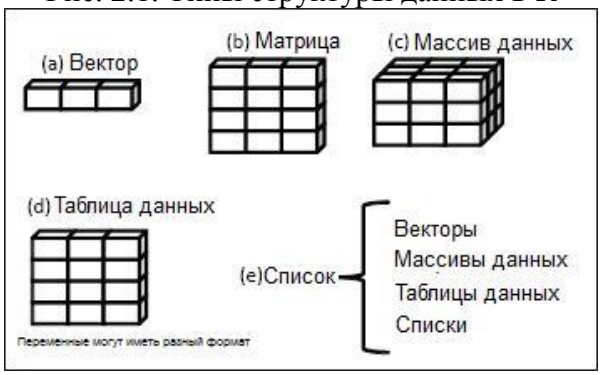
R работает с самыми разными структурами данных, включая скаляры, векторы, массивы данных, таблицы данных и списки. Табл. 2.1 будет прочитана в R как таблица данных. Такое большое разнообразие поддерживаемых структур данных сообщает языку R большую гибкость в работе с данными.

Типы данных в R бывают числовыми (numeric), текстовыми (character), логическими (TRUE/FALSE, правда/ложь), комплексными (мнимое число) и необработанными (байты). Переменные `PatientID`, `AdmDate` и `Age` будут прочитаны R как числовые, а `Diabetes` и `Status` – как текстовые. Еще вам нужно будет «сообщить» программе, что `PatientID` – это названия наблюдений, что в `AdmDate` записаны даты и что `Diabetes` и `Status` – это номинальная и порядковая переменные соответственно. В R имена строк называются `rownames`, а категориальные (номинальные и порядковые) переменные – `factor` (фактор). Мы рассмотрим все по порядку в следующем разделе. Про даты вы узнаете из главы 3.

2.2. Типы структуры данных

R работает с самыми разными структурами данных, включая скаляры, векторы, матрицы, массивы данных, таблицы данных и списки. Они различаются типами данных, способом создания, сложностью устройства, а также способом обозначать и извлекать их отдельные элементы. Эти структуры данных схематически изображены на рис. 2.1.

Рис. 2.1. Типы структуры данных в R



Некоторые определения:

Существует несколько присущих только R терминов, которые приводят в замешательство новых пользователей.

В R *объектом* (object) называется все, что может быть представлено в виде переменных, включая константы, разные типы данных, функции и даже диаграммы. У объектов есть вид (определяет, в каком виде объект хранится в памяти) и класс (который указывает общим функциям типа print, как с ним обращаться).

Таблица данных (data frame) – это тип структуры данных в R, аналогичный тому виду, в котором хранятся данные в обычных статистических программах (например, в SAS, SPSS и STATA). Столбцы – это переменные, а строки – это наблюдения. В одной таблице данных могут содержаться переменные разных типов (например, числовые и текстовые). Таблицы данных – это основной тип структуры данных.

Факторы – это номинальные или порядковые переменные. В R они хранятся и обрабатываются особым образом. Вы узнаете больше о факторах из раздела 2.2.5.

Большинство других терминов должны быть вам знакомы, они используются в статистике и в вычислениях.

Давайте рассмотрим все типы структуры данных по порядку, начиная с векторов.

2.2.1. Векторы

Векторы (vector) – это одномерные массивы данных, которые могут содержать числовые, текстовые или логические данные. Для создания вектора применяется функция объединения `c()`. Вот примеры векторов каждого типа:

```
a <- c(1, 2, 5, 3, 6, -2, 4)
b <- c("one", "two", "three")
c <- c(TRUE, TRUE, TRUE, FALSE, TRUE, FALSE)
```

Здесь `a` – числовой вектор, `b` – текстовый вектор, `c` – логический вектор. Обратите внимание на то, что все данные в векторе должны быть одного типа (числовые, текстовые или логические). Нельзя смешивать данные разных типов в одном векторе.

Примечание. *Скаляры – это векторы, состоящие из одного элемента, например, `f <- 3`, `g <- "US"` и `h <- TRUE`. Они используются для обозначения констант.*

Отдельные элементы вектора можно вызывать при помощи числового вектора, состоящего из номеров элементов, в квадратных скобках. Например, `a[c(2, 4)]` обозначает второй и четвертый элементы вектора `a`. Вот еще примеры:

```
> a <- c(1, 2, 5, 3, 6, -2, 4)
> a[3]
[1] 5
> a[c(1, 3, 5)]
[1] 1 5 6
> a[2:6]
[1]  2  5  3  6 -2
```

Знак двоеточия в последнем примере использован для создания последовательности чисел. Например, `a <- c(2:6)` – это то же самое, что `a <- c(2, 3, 4, 5, 6)`.

2.2.2. Матрицы

Матрица (matrix) – это двумерный массив данных, в котором каждый элемент имеет одинаковый тип (числовой, текстовый или логический). Матрицы создают при помощи функции `matrix`. Общий формат таков:

```
mymatrix <- matrix(vector, nrow=number_of_rows, ncol=number_of_columns,
                   byrow=logical_value, dimnames=list(
                   char_vector_rownames, char_vector_colnames))
```

где *`vector`* содержит элементы матрицы, `nrow` и `ncol` определяют число строк и столбцов в матрице, а `dimnames` содержит названия строк и столбцов (их указывать не обязательно), которые

хранятся в виде текстовых векторов. Параметр `byrow` определяет, как должна заполняться матрица – по строкам (`byrow=TRUE`) или по столбцам (`byrow=FALSE`). По умолчанию матрица заполняется по столбцам. Приведенный ниже программный код иллюстрирует применение функции `matrix`.

Программный код 2.1. Создание матриц

```
> y <- matrix(1:20, nrow=5, ncol=4)        ←❶ Создаем таблицу 5×4
> y
     [,1] [,2] [,3] [,4]
[1,]    1    6   11   16
[2,]    2    7   12   17
[3,]    3    8   13   18
[4,]    4    9   14   19
[5,]    5   10   15   20
> cells    <- c(1,26,24,68)                ←❷ Таблица 2×2 заполнена по рядам

> rnames   <- c("R1", "R2")
> cnames   <- c("C1", "C2")
> mymatrix <- matrix(cells, nrow=2, ncol=2, byrow=TRUE,
                     dimnames=list(rnames, cnames))
> mymatrix

   C1 C2
R1  1 26
R2 24 68
> mymatrix <- matrix(cells, nrow=2, ncol=2, byrow=FALSE,   ↰
                     dimnames=list(rnames, cnames))        ❸
> mymatrix                                  Таблица 2×2 заполнена по столбцам
  C1 C2
R1  1 24
R2 26 68
```

Сначала вы создаете матрицу 5×4 ❶. Затем вы делаете матрицу 2×2 с названиями строк и столбцов и заполняете ее по строкам ❷. Наконец, вы создаете матрицу 2×2 и заполняете ее по столбцам ❸.

Вы можете обозначать строки, столбцы и элементы матрицы при помощи индексов и квадратных скобок. Например, `X[i,]` обозначает *i*-тую строку матрицы `X`, `X[,j]` – обозначает ее *j*-тый столбец, а `X[i, j]` соответствует элементу этой матрицы, расположенному на пересечении этой строки и этого столбца. В качестве индексов *i* и *j* можно использовать числовые векторы, чтобы обозначить сразу несколько строк или столбцов, как это показано ниже.

Программный код 2.2. Использование индексов при работе с матрицами

```
> x <- matrix(1:10, nrow=2)
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
> x[2,]
  [1]  2  4  6  8 10
> x[,2]
[1] 3 4
> x[1,4]
[1] 7
> x[1, c(4,5)]
[1] 7 9
```

Сначала создана матрица 2×5, содержащая цифры от 1 до 10. По умолчанию матрица заполнена цифрами по столбцам. Затем выбраны все элементы во второй строке, а затем – все элементы во втором столбце. Затем выбран элемент, который находится в первой строке и в четвертом столбце. Наконец, выбраны элементы первой строки, которые находятся в четвертом и пятом столбцах.

Матрицы имеют два измерения и, как и векторы, могут состоять только из одного типа данных. Если есть больше двух измерений, нужно использовать массивы данных (раздел 2.2.3), данные разных типов можно хранить в таблицах (раздел 2.2.3).

2.2.3. Массивы данных

Массивы данных (array) сходны с матрицами, но могут иметь больше двух измерений. Массивы данных создаются при помощи функции `array` по такому образцу:

`myarray <- array(`*`vector, dimensions, dimnames`*`)`

где *`vector`* содержит сами данные, *`dimensions`* – это числовой вектор с указанием размерности для каждого измерения и *`dimnames`* – это необязательный список названий измерений. Ниже в качестве примера представлен программный код, при помощи которого создан трехмерный (2×3×4) массив чисел.

Программный код 2.3. Создание массива данных

```
> dim1 <- c("A1", "A2")
> dim2 <- c("B1", "B2", "B3")
> dim3 <- c("C1", "C2", "C3", "C4")
> z <- array(1:24, c(2, 3, 4), dimnames=list(dim1, dim2, dim3))
> z
, , C1
   B1 B2 B3
A1  1  3  5
A2  2  4  6
, , C2
   B1 B2 B3
A1  7  9 11
A2  8 10 12
, , C3
   B1 B2 B3
A1 13 15 17
A2 14 16 18
, , C4
   B1 B2 B3
A1 19 21 23
A2 20 22 24
```

Очевидно, что массивы данных – это просто расширенные матрицы. Они могут быть полезны при написании программ для реализации новых статистических методов. Как и в матрицах, все элементы массива данных должны иметь одинаковый тип. Система обозначений элементов здесь такая же, как для матриц. В приведенном выше примере элемент `z[1,2,3]` – это 15.

2.2.4. Таблицы данных

Таблица данных (data frame) – это более широко применимый по сравнению с матрицей объект, поскольку разные столбцы могут содержать разные типы данных (числовой, текстовый и т.д.). Таблица данных – это самая часто используемая структура данных в R.

Набор данных про пациентов (табл. 2.1) состоит из числовых и текстовых данных. Эти данные нужно представить в виде таблицы данных, а не матрицы, поскольку здесь есть данные разных типов.

Таблица данных создается при помощи функции `data.frame()`:

`mydata <- data.frame(`*`col1, col2, col3,…`*`),`

где – *`col1, col2, col3,…`* это векторы любого типа (текстового, числового или логического), которые станут столбцами таблицы. Названия каждому столбцу можно присвоить при помощи функции `names()`. Проиллюстрируем сказанное при помощи примера программного кода.

Программный код 2.4. Создание таблицы данных

```
> patientID <- c(1, 2, 3, 4)
> age <- c(25, 34, 28, 52)
> diabetes <- c("Type1", "Type2", "Type1", "Type1")
> status <- c("Poor", "Improved", "Excellent", "Poor")
> patientdata <- data.frame(patientID, age, diabetes, status)
> patientdata
  patientID age diabetes    status
1         1  25    Type1      Poor
2         2  34    Type2  Improved
3         3  28    Type1 Excellent
4         4  52    Type1      Poor
```

Каждый столбец должен содержать данные только одного типа, при этом в одной таблице данных могут быть столбцы с данными разного типа. Поскольку таблицы данных очень близки к тому, что аналитики называют наборами данных, при обсуждении таблиц данных мы будем использовать термины *столбцы* и *переменные* в качестве синонимов.

Существует несколько способов обозначить элементы таблицы данных. Можно использовать индексы, как мы делали раньше (например, для матриц) или можно указывать номера столбцов. Приведенный ниже программный код на примере созданной раньше таблицы данных `patientdata` демонстрирует оба способа.

Программный код 2.5. Обозначение элементов таблицы данных

```
> patientdata[1:2]
  patientID age
1         1  25
2         2  34
3         3  28
4         4  52
> patientdata[c("diabetes", "status")]
  diabetes    status
1    Type1      Poor
2    Type2  Improved
3    Type1 Excellent
4    Type1      Poor
> patientdata$age
[1] 25 34 28 52
```

Знак $ из третьего примера раньше нам не встречался ❶. Он используется, чтобы обозначить определенную переменную в таблице данных. Например, если вы захотите сделать сводную таблицу типов диабета в зависимости от состояния больного, вы можете использовать следующий программный код:

```
> table(patientdata$diabetes, patientdata$status)

        Excellent Improved Poor
  Type1         1        0    2
  Type2         0        1    0
```

←❶ Обозначаем переменную age (возраст)в таблице данных patientdata

Поскольку добавление `patientdata$` перед названием каждой переменной может быстро надоесть, существуют команды для быстрого вызова переменной. Для упрощения программного кода можно использовать функции `attach()` и `detach()` или `with()`.

Attach, detach и with

Функция `attach()` добавляет указанную таблицу данных к траектории поиска R. Когда указывается имя переменной, программа ищет эту переменную в таблицах данных, включенных в траекторию поиска. В качестве примера можно взять таблицу данных `mtcars` из главы 1, используя следующий программный код, чтобы узнать основные статистики расхода топлива (`mpg`), а также изобразить значения этой переменной на диаграмме в зависимости от рабочего объема цилиндров двигателя (`disp`) и веса машины (`wt`).

```
summary(mtcars$mpg)
plot(mtcars$mpg, mtcars$disp)
plot(mtcars$mpg, mtcars$wt)
```

Это можно также записать в виде

```
attach(mtcars)
  summary(mpg)
  plot(mpg, disp)
  plot(mpg, wt)
detach(mtcars)
```

Функция `detach()` удаляет таблицу данных из траектории поиска. Отметим, что эта функция ничего не делает с самим объектом. Ввод этой команды необязателен, но он полезен при программировании, и про него не следует забывать. Я буду иногда пренебрегать этим мудрым советом в следующих главах, чтобы сделать приводимые фрагменты программного кода короче и проще.

Ограничение этого метода становится очевидным, если у нас есть несколько объектов с одинаковыми названиями. Рассмотрим такой программный код:

```
> mpg <- c(25, 36, 47)
> attach(mtcars)
The following object(s) are masked _by_ '.GlobalEnv':    mpg
> plot(mpg, wt)
Error in xy.coords(x, y, xlabel, ylabel, log) :
  'x' and 'y' lengths differ
> mpg
[1] 25 36 47
```

Когда мы добавили к траектории поиска таблицу данных `mtcars`, в рабочем пространстве уже имелся объект с названием `mpg`. В подобных случаях преимущество получает объект, который был создан первым, а это не то, чего мы хотели. Команда `plot()` не выполняется, потому что `mpg` теперь состоит из трех элементов, а `disp` – из 32 элементов. Функции `attach()` и `detach()` лучше всего использовать, когда вы работаете с одной таблицей данных, и вероятность то, что у вас будет несколько объектов с одинаковыми именами, мала. В любом случае обращайте внимание на предупреждения о том, что объекты удалены (“objects are masked”) из рабочего пространства.

Альтернативный подход заключается в использовании функции `with()`. Предыдущий пример можно записать так:

```
with(mtcars, {
  summary(mpg, disp, wt)
  plot(mpg, disp)
  plot(mpg, wt)
})
```

В этом случае написанное внутри фигурных скобок относится к таблице данных `mtcars`. Теперь нам не придется заботиться о конфликте названий. Если нужно выполнить только одну команду (например, `summary(mpg)`), фигурные скобки необязательны.

Ограничение функции `with()` заключается в том, что она не действует за пределами фигурных скобок. Рассмотрим следующий пример:

```
> with(mtcars, {
   stats <- summary(mpg)
   stats
  })
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  10.40   15.43   19.20   20.09   22.80   33.90
> stats
Error: object ‘stats’ not found
```

Если требуется создать объекты, которые будут существовать вне конструкции `with()`, используйте специальный символ присвоения `<<-` вместо обычного (`<-`). Этот прием позволит сохранить созданный объект в рабочем пространстве вне конструкции `with()`. Это можно показать на примере такого программного кода:

```
> with(mtcars, {
   nokeepstats <- summary(mpg)
   keepstats <<- summary(mpg)
})
> nokeepstats
Error: object ‘nokeepstats’ not found
> keepstats
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    10.40   15.43   19.20   20.09   22.80   33.90
```

В большинстве руководств по R рекомендуется использовать `with()`, а не `attach()`. Я думаю, что выбор зависит от индивидуальных предпочтений и должен основываться на том, чего вы хотите достичь. В этой книге мы будем использовать обе команды.

Названия строк

В примере с данными о больных столбец patientID использовался для обозначения отдельных людей в наборе данных. В R названия строк могут быть назначены при помощи параметра `rowname` функции создания таблицы данных. Например, программный код

```
patientdata <- data.frame(patientID, age, diabetes, status,row.names=patientID)
```

назначает `patientID` переменной, которая будет использоваться для обозначения строк при выводе данных и создании диаграмм в R.

2.2.5. Факторы

Как уже узнали, переменные бывают номинальными, порядковыми или непрерывными. Номинальные данные – это категориальные данные, которые нельзя упорядочить. Переменная `Diabetes` – это пример номинальных данных. Даже если мы обозначим `Type 1` (тип 1) единицей, а `Type 2` (тип 2) – двойкой, все равно эти цифры нельзя будет сравнивать в терминах «больше – меньше». Порядковые данные можно упорядочить, но не оценить количественно. Переменная `Status` – хороший пример порядковых данных. Понятно, что у больного с плохим (`poor`) самочувствием дела идут не так хорошо, как у больного, чье состояние улучшилось (`improved`), но не ясно, насколько. Непрерывные переменные могут принимать любое значение в пределах определенного диапазона. Их значения можно упорядочить и понять, насколько одно из них больше другого. Возраст, выраженный в годах, является непрерывной переменной и может принимать такие значения как 14.5 или 22.8, а также любые значения между этими двумя. Вы знаете, что пятнадцатилетний подросток старше четырнадцатилетнего на один год.

Категориальные (номинальные и порядковые) данные называются в R факторами. Факторы очень важны в R, поскольку они определяют, как данные будут проанализированы и графически представлены. Вы будете видеть примеры этого на протяжении всей книги.

Функция `factor()` хранит категориальные данные в виде вектора из целых чисел в диапазоне от одного до k (где k – число уникальных значений категориальной переменной) и в виде внутреннего вектора из цепочки символов (исходных значений переменной), соответствующим этим целым числам.

К примеру, представьте, что у вас есть вектор
`diabetes <- c("Type1", "Type2", "Type1", "Type1").`
Команда `diabetes <- factor(diabetes)` преобразует этот вектор в (1, 2, 1, 1) и устанавливает внутреннее соответствие 1=Type1 и 2=Type2 (присвоение числовых значений происходит в алфавитном порядке). Любой анализ, который вы будете проводить с вектором `diabetes`, будет воспринимать эту переменную как номинальную и выбирать статистические методы, подходящие для этого типа данных.

Для векторов, которые представлены порядковыми данными, для функции `factor()` нужно добавлять параметр `ordered=TRUE`. Примененная к вектору
`status <- c("Poor", "Improved", "Excellent", "Poor")`
команда `status <- factor(status, ordered=TRUE)` преобразует этот вектор в вид (3, 2, 1, 3) и установит внутреннее соответствие как `1=Excellent, 2=Improved, 3=Poor`. Во время любой обработки этого вектора он будет воспринят как порядковая переменная с применением соответствующих статистических методов.

По умолчанию, уровни фактора присваиваются значениям вектора в алфавитном порядке. Это сработало для фактора `status`, поскольку порядок “Excellent”, “Improved”, “Poor” имеет смысл. Если бы вместо “Poor” стояло “Ailing” (чахнущий), то возникло бы затруднение, поскольку тогда порядок был бы такой: “Ailing”, “Excellent”, “Improved”. Сходная проблема возникла бы, если бы нам был нужен такой порядок: “Poor”, “Improved”, “Excellent”. Для упорядоченных факторов редко подходит алфавитный порядок уровней, предлагающийся по умолчанию.

Установку по умолчанию можно изменить, указав параметр `levels`. Например,

```
status <- factor(status, order=TRUE,
                 levels=c("Poor", "Improved", "Excellent"))
```

присвоит уровни значениям вектора следующим образом: 1=Poor, 2=Improved, 3=Excellent. Проверьте, что все присвоенные уровни соответствуют реальным значениям данных. Все значения данных, которые не были указаны, будут обозначены как отсутствующие.

Приведенный ниже программный код показывает как назначение факторов и упорядоченных факторов влияет на анализ данных.

Программный код 2.6. Использование факторов

```
> patientID <- c(1, 2, 3, 4)    ←❶ Вводим данные как векторы
> age <- c(25, 34, 28, 52)
> diabetes <- c("Type1", "Type2", "Type1", "Type1")
> status <- c("Poor", "Improved", "Excellent", "Poor")
> diabetes <- factor(diabetes)
> status <- factor(status, order=TRUE)
> patientdata <- data.frame(patientID, age, diabetes, status)
> str(patientdata)              ←❷ Смотрим структуру объекта
'data.frame':   4 obs. of  4 variables:
 $ patientID: num  1 2 3 4
 $ age      : num  25 34 28 52
 $ diabetes : Factor w/ 2 levels "Type1","Type2": 1 2 1 1
 $ status   : Ord.factor w/ 3 levels "Excellent"<"Improved"<..: 3 2 1 3
> summary(patientdata)          ←❸ Смотрим сводную статистику для объекта
   patientID          age          diabetes       status
 Min.   :1.00   Min.   :25.00   Type1:3   Excellent:1
 1st Qu.:1.75   1st Qu.:27.25   Type2:1   Improved :1
 Median :2.50   Median :31.00             Poor     :2
 Mean   :2.50   Mean   :34.75
 3rd Qu.:3.25   3rd Qu.:38.50
 Max.   :4.00   Max.   :52.00
```

Сначала вы вводите данные как векторы ❶. Затем вы указываете, что `diabetes` – это фактор, а `status` – это упорядоченный фактор. Наконец, вы объединяете данные в таблицу. Функция `str(`*`object`*`)` выводит информацию об объекте (в нашем случае – это таблица данных) ❷. Ясно видно, что `diabetes` – это фактор, а `status` – это упорядоченный фактор, также указано, как он закодирован внутри программы. Обратите внимание, что функция `summary()` обрабатывает переменные по-разному ❸. Для непрерывной переменной `age` вычислены минимум (minimum, Min.), максимум (maximum, Max.), среднее (Mean) и квартили (first and third quartiles: 1st Qu., 3rd Qu.), а для категориальных переменных `diabetes` и `status` подсчитана частота встречаемости каждого значения.

2.2.6. Списки

Списки – это самый сложный тип данных в R. Фактически, список – это упорядоченный перечень объектов (компонентов). Список может объединять разные (возможно, не связанные между собой) объекты под одним именем. К примеру, список может представлять собой сочетание векторов, матриц, таблиц данных и даже других списков. Список можно создать при помощи функции

`list()`:
`mylist <- list(object1, object2, …)`,
где объекты – это любые структуры данных, которые мы обсуждали до этого. Объектам в списке можно присваивать имена:
`mylist <- list(name1=object1, name2=object2, …)`.
Пример работы со списками приведен ниже.

Программный код 2.7. Создание списка

```
> g <- "My First List"
> h <- c(25, 26, 18, 39)
> j <- matrix(1:10, nrow=5)
> k <- c("one", "two", "three")
> mylist <- list(title=g, ages=h, j, k)      ← Создаем список
> mylist                                     ← Выводим на экран весь список
$title
[1] "My First List"
$ages
[1] 25 26 18 39
[[3]]
     [,1] [,2]
[1,]    1    6
[2,]    2    7
[3,]    3    8
[4,]    4    9
[5,]    5   10
[[4]]
[1] "one"   "two"   "three"
> mylist[[2]]                                ← Выводим на экран второй объект списка
[1] 25 26 18 39
> mylist[["ages"]]
[[1] 25 26 18 39
```

В приведенном примере вы создаете список из четырех компонентов: тестовая строка, числовой вектор, матрица и текстовый вектор. В виде списка можно сохранять любое число объектов.

Можно обозначать элементы списка, указав их номер или название внутри двойных квадратных скобок. В этом примере и `mylist[[2]]`, и `mylist[["ages"]]` обозначают один и тот же числовой вектор из четырех элементов. Списки – это важный тип структуры данных в R по двум причинам. Во-первых, они позволяют вам без труда упорядочить и вызвать на экран разрозненную информацию. Во-вторых, результаты многих команд представляют собой списки. В этом случае пользователь извлекает из таких списков нужную информацию. Вы сможете увидеть многочисленные примеры функций, которые возвращают списки, в следующих главах.

Информация для программистов

Для многих профессиональных программистов некоторые аспекты языка R кажутся необычными. Вот несколько особенностей языка R, о которых вам следует помнить:

- точка (.) в названиях объектов не имеет никакого специального значения. Однако знак доллара ($) имеет примерно такое же значение, как точка в других языках программирования, обозначая часть объекта. Например, A$x обозначает переменную x в таблице данных A;
- в R нет возможности создавать многострочные или блочные комментарии. Каждую строку комментария нужно начинать со знака #. При отладке программных кодов можно заключать участок кода, который должен быть пропущен программным интерпретатором, внутрь конструкции if(FALSE){…}. Замена FALSE на TRUE сделает возможным выполнение кода;

- присвоение значений несуществующему элементу вектора, матрицы, массива данных или списка расширит существующий объект, чтобы вместить новое значение. Рассмотрим следующий пример:
  > x <- c(8, 6, 4)
  > x[7] <- 10
  > x
  [1] 8 6 4 NA NA NA 10
  В результате присвоения размер вектора x увеличился с трех до семи элементов;
- в R нет скаляров. Они представлены в виде вектора, состоящего из одного элемента;
- нумерация в R начинается с 1, а не с 0. В приведенном выше векторе элемент x[1] – это 8.

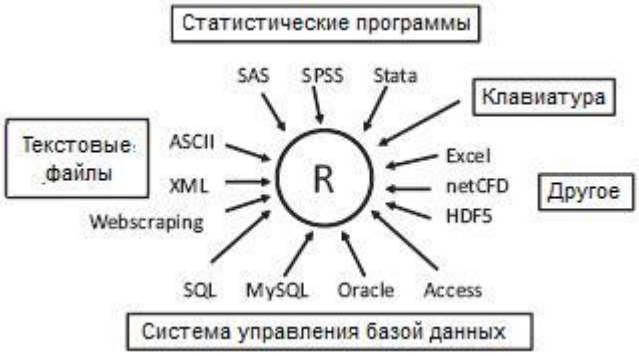
Дальнейшую информацию можно получить, заглянув в превосходный блог Джона Кука (John Cook) «Программирование в R для тех, кто раньше работал с другими языками» (www.johndcook.com/R_language_for_programmers.html) $$ Это и все последующие руководства, упомянутые в книге, написаны на английском языке. Прим. пер.$$.

Программисты, которым нужно руководство по стилю программирования, также могут заглянуть в «Руководство по стилю программирования в R» от Google (http://google-styleguide.googlecode.com/svn/trunk/google-r-style.html).

2.3. Ввод данных

Теперь, когда у нас есть структуры данных, нужно наполнить их данными! Обычно аналитики сталкиваются с данными, которые поступают из разных источников и в разных форматах. Задача состоит в том, чтобы импортировать данные в программу, проанализировать их и представить отчет о результатах. В R реализованы разные способы импорта данных. Исчерпывающее руководство по импорту данных в R доступно в Интернет по адресу: http://cran.r-project.org/doc/manuals/R-data.pdf.

Рис. 2.2. Источники, из которых можно импортировать данные в R



Как видно на рис. 2.2, в R можно вводить данные с клавиатуры, импортировать из текстовых файлов, из Microsoft Excel и Access, из распространенных статистических программ, специализированных форматов, а также из разных систем управления базами данных. Поскольку никогда нельзя угадать, откуда вы получите данные, мы рассмотрим здесь все источники по порядку. Вам нужно прочесть только про те, которые вы собираетесь использовать.

2.3.1 Ввод данных с клавиатуры

Наверное, самый простой способ введения данных – это ввод с клавиатуры. Функция `edit()` откроет текстовый редактор, куда вы сможете внести свои данные. Вот пошаговая инструкция:

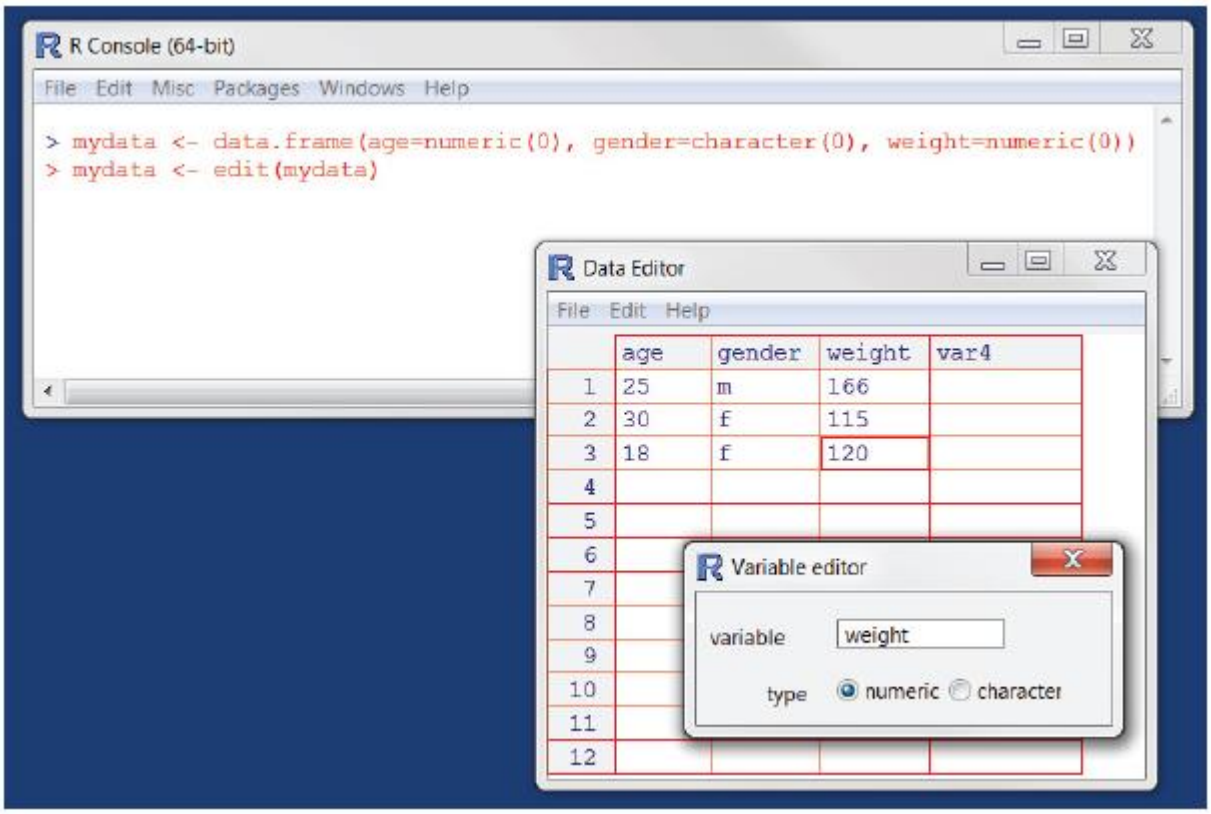
1. Создайте пустую таблицу данных (или матрицу), указав названия и типы переменных.
2. Откройте текстовый редактор с этим объектом, введите ваши данные и сохраните результат в виде объекта с данными.

В приведенном ниже примере вы создадите таблицу данных с названием `mydata` с тремя переменными: `age` (возраст, числовая), `gender` (пол, текстовая) и `weight` (вес, числовая). Затем вы откроете текстовый редактор, внесете данные и сохраните результат.

```
mydata <- data.frame(age=numeric(0),
  gender=character(0), weight=numeric(0))
mydata <- edit(mydata)
```

Присвоения типа `age=numeric(0)` создают пустую (без данных) переменную заданного формата. Обратите внимание на то, что результат редактирования вновь присвоен исходному объекту. Функция `edit()` работает с копией объекта. Если вы не присвоите результат ее работы какому-либо объекту, все ваши изменения пропадут! Результат работы функции `edit()` под Windows показан на рис. 2.3.

Рис. 2.3. Редактирование данных при помощи встроенного текстового редактора под Windows



На рисунке видно, что я взял на себя смелость добавить немного данных. Если щелкнуть на название столбца, можно изменить название и тип соответствующей переменной. Можно добавлять дополнительные переменные, щелкая на названия неиспользованных столбцов. После того, как вы закрываете текстовый редактор, результаты сохраняются в виде выбранного объекта (в данном случае объект `mydata`). Повторное введение функции `mydata <- edit(mydata)` позволяет редактировать введенные данные и добавлять новые. Сокращенная версия команды `mydata <- edit(mydata)` – это просто `fix(mydata)`.

Этот метод хорошо работает для маленьких наборов данных. Для наборов данных большего размера вам, возможно, захочется использовать описанные ниже методы: импорт данных из существующих текстовых файлов, электронных таблиц Excel, статистических программ или систем управления базами данных.

2.3.2. Импорт данных из текстового файла с разделителями

Импорт данных из текстовых файлов с разделителями возможен при помощи команды `read.table()`, функции, которая сохраняет данные в виде таблицы. Вот пример использования функции:

```
mydataframe <- read.table(file, header=logical_value,
  sep="delimiter", row.names="name")
```

где *file* – это ASCII (American Standard Code for Information Interchange – американский стандартный код обмена информацией) файл с разделителями, *header* – это логическое значение, обозначающее, содержит ли первая строка названия переменных (TRUE – да, FALSE – нет), sep указывает, каким символом разделены элементы данных, а row.names – необязательный параметр, для указания столбца (столбцов), в которых содержатся названия строк.

Например, программный код

```
grades <- read.table("studentgrades.csv", header=TRUE, sep=",",
  row.names="STUDENTID")
```

позволяет прочесть файл с разделителями-запятыми, который называется studentgrades.csv, из текущей рабочей директории и сохранить его в виде таблицы данных с названием grades. В этом файле названия переменных содержались в первой строке, а названия строк – в столбце с названием STUDENTID.

Обратите внимание на то, что использование параметра sep позволяет импортировать файлы с любыми символами в качестве разделителей. Файлы с символом табуляции в качестве разделителя можно импортировать, указав sep="\t". По умолчанию используется sep="", обозначающий один или несколько пробелов, символов табуляции, символов новой строки или возврата каретки.

По умолчанию текстовые переменные преобразуются в факторы. Это не всегда уместно (например, для переменной, которая содержит комментарии респондента). Такое преобразование можно заблокировать разными способами. Добавление параметра stringsAsFactors=FALSE не позволит преобразовывать в факторы все текстовые переменные. В качестве альтернативы можно использовать параметр colClasses для того, чтобы указать формат (например, логический, числовой, текстовый, фактор) каждого столбца.

У функции read.table() есть много дополнительных параметров, при помощи которых можно контролировать импорт данных. Подробнее об этом можно прочесть, введя help(read.table).

Примечание. *Многие примеры в этой главе основаны на импорте уже имеющихся на компьютере файлов. В R также реализовано несколько алгоритмов получение данных из Интернета. Например, вместо имени файла можно использовать функции file(), gzfile(), bzfile(), xzfile(), unz() и url(). Функция file() позволяет получать доступ к файлам, буферу обмена и СИ-уровневому стандартному вводу (англ. C-level standard input). Функции gzfile(), bzfile(), xzfile()и unz() дают возможность доступа к сжатым файлам. Функция url() предоставляет доступ к файлам в Интернете через полный URL-адрес, который включает http://, ftp:// или file://. Для HTTP или FTP можно назначить программы-посредники (proxy). Полные URL-адреса (заключенные в прямые кавычки) можно использовать также вместо названий файлов. Более подробную информацию можно получить, введя help(file).*

2.3.3. Импорт данных из Excel

Лучший способ прочесть файл в формате Excel – это сохранить его в формате текстового файла с разделителями и импортировать в R, как это описано выше. Под Windows для доступа к файлам Excel также можно использовать пакет RODBC. В первой строке электронной таблицы должны содержаться названия переменных (столбцов).

Прежде всего, скачайте и установите пакет RODBC.

```
install.packages("RODBC")
```

Теперь вы можете использовать следующий программный код для импорта данных:

```
library(RODBC)
channel <- odbcConnectExcel("myfile.xls")
mydataframe <- sqlFetch(channel, "mysheet")
odbcClose(channel)
```

Здесь myfile.xls – это файл Excel, mysheet – это название нужного листа из рабочей книги Excel, channel – это вспомогательный объект RODBC, созданный функцией

odbcConnectExcel() и mydataframe – это получившаяся таблица данных. Этот пакет можно также использовать для импорта данных из Microsoft Access. Подробности изложены в файле справки: help(RODBC).

В Excel 2007 используются файлы формата XLSX, которые фактически представляют собой сжатый набор XML файлов. Для импорта электронных таблиц в этом формате можно использовать пакет xlsx. Убедитесь, что перед первым использованием вы скачали и установили этот пакет. Функция read.xlsx() осуществляет импорт нужного листа XLSX файла в таблицу данных. Проще всего использовать эту функцию по такой схеме: read.xlsx(*file*, *n*), где *file* – это путь к файлу книги Excel 2007, а *n* – число листов, которые нужно импортировать. Например, под Windows программный код

```
library(xlsx)
workbook <- "c:/myworkbook.xlsx"
mydataframe <- read.xlsx(workbook, 1)
```

импортирует первый лист книги myworkbook.xlsx, хранящейся на диске C: и сохраняет его в виде таблицы данных mydataframe. Пакет xlsx может не только импортировать листы XLSX файлов. Он также может создавать файлы этого формата и управлять ими. Программистам, перед которыми стоит задача разработать интерфейс для обмена данными между R и Excel, следует обратить внимание на этот относительно новый пакет.

2.3.4. Импорт данных из XML файлов

В последнее время все больше данных появляется в виде файлов формата XML. В R есть несколько пакетов для работы с такими файлами. Например, пакет XML, созданный Дункан Темпл Ланг (Duncan Temple Lang) позволяет пользователям читать, записывать и преобразовывать XML файлы. Обсуждение этого формата не входит в нашу задачу. Заинтересованным в работе с XML файлами пользователям R рекомендую обратиться к прекрасной справочной документации пакета на сайте www.omegahat.org/RSXML.

2.3.5. Извлечение данных из веб-страниц

При извлечении данных из веб-страниц (webscrapping) пользователи получают информацию с веб-страниц, доступных в Интернете, и сохраняют их в виде объектов R для дальнейшего анализа. Один способ сделать это – скачать веб-страницу при помощи функции readLines() и дальше работать с ней при помощи функций grep() и gsub(). Для извлечения нужной информации из сложно устроенных веб-страниц можно использовать пакеты RCurl и XML. Более полную информацию, включая примеры, можно найти в руководстве “Извлечение данных из веб-страниц с использованием readLines и RCurl” (Webscraping using readLines and RCurl), доступном на сайте “Программирование в R” (Programming with R: www.programmingr.com).

2.3.6. Импорт данных из SPSS

Наборы данных в формате SPSS можно импортировать в R при помощи функции read. spss() из пакета foreign. Другой вариант – использовать функцию spss.get() из пакета Hmisc. Это интерфейсная функция, которая автоматически определяет многие параметры функции read.spss(), что делает преобразование файлов проще и эффективнее.

Во-первых, скачайте и установите пакет Hmisc (пакет foreign входит в базовую комплектацию):

```
install.packages("Hmisc")
```

Затем используйте следующий программный код для импорта данных:

```
library(Hmisc)
mydataframe <- spss.get("mydata.sav", use.value.labels=TRUE)
```

Здесь `mydata.sav` – это файл данных SPSS, который нужно импортировать, параметр `use.value.labels=TRUE` означает, что переменные с подписанными значениями будут преобразованы в факторы с соответствующими уровнями, а `mydataframe` – это получившаяся таблица данных.

2.3.7. Импорт данных из SAS

В R есть ряд функций для импорта SAS файлов, включая `read.ssd()` в пакете `foreign` и `sas.get()` в пакете `Hmisc`. К сожалению, если вы используете последнюю версию SAS (SAS 9.1 и выше), скорее всего, эти функции не будут работать, поскольку структура файлов SAS была изменена. Я рекомендую два способа решения этой проблемы.

Можно сохранить набор данных SAS в виде текстового файла с разделителями-запятыми при помощи команды `PROC EXPORT` и прочесть получившийся файл, как это описано в разделе 2.3.2. Вот пример:

Программа SAS:

```
proc export data=mydata
     outfile="mydata.csv"
     dbms=csv;
run;
```

Программа R:

```
mydata <- read.table("mydata.csv", header=TRUE, sep=",")
```

В качестве альтернативы можно использовать коммерческий продукт, который называется Stat Transfer (описан в разделе 2.3.12). Эта программа замечательно справляется с сохранением данных SAS (с учетом формата переменных) в виде таблиц данных R.

2.3.8. Импорт данных из Stata

Импортировать данные из Stata в R очень просто. Нужно использовать примерно такой программный код:

```
library(foreign)
mydataframe <- read.dta("mydata.dta")
```

Здесь `mydata.dta` – это набор данных, а `mydataframe` – это итоговая таблица данных R.

2.3.9. Импорт данных из netCDF

Данные в формате netCDF (network Common Data Form – сетевая общедоступная форма данных) создаются в программе Unidata с открытым программным кодом. Это машинно-независимые форматы данных для создания и распространения массивов научных данных. Этот формат часто используется для хранения геофизических данных. Пакеты `ncdf` и `ncdf4` создают в R интерфейс высокого уровня для доступа к netCDF файлам данных.

Пакет `ncdf` читает файлы, созданные в программе Unidata при помощи библиотеки netCDF (версия 3 или раньше) под Windows, Mac OS X и Linux. Пакет `ncdf4` работает и с четвертой версией библиотеки netCDF, но он пока не доступен под Windows.

Рассмотрим следующий программный код:

```
library(ncdf)
nc <- nc_open("mynetCDFfile")
myarray <- get.var.ncdf(nc, myvar)
```

В этом примере все данные из переменной `myvar` файла netCDF `mynetCDFfile` сохранены в R в виде массива данных `myarray`.

Учтите, что пакеты `ncdf` и `ncdf4` в последнее время были значительно изменены и могут работать не так, как их предыдущие версии. Кроме того, названия функций в этих пакетах различаются. Все подробности можно узнать из онлайн-документации.

2.3.10. Импорт данных из HDF5

HDF5 (Hierarchical Data Format – иерархический формат данных) – это программная технология, которая позволяет управлять чрезвычайно большими и сложными наборами данных. Пакет hdf5 можно использовать для сохранения объектов R в таком виде, который может быть прочитан программой, работающей с данными в формате HDF5. Потом эти файлы можно будет опять открыть в R. Это экспериментальный пакет, который подразумевает наличие на компьютере пользователя установленной библиотеки HDF5 (версия 1.2 или выше). В настоящее время поддержка файлов HDF5 в R сильно ограничена.

2.3.11. Импорт данных из систем управления базами данных

R может взаимодействовать с самыми разными системами управления базами данных (database management systems, DBMS), включая Microsoft SQL Server, Microsoft Access, MySQL, Oracle, PostgreSQL, DB2, Sybase, Teradata и SQLite. Некоторые пакеты предоставляют доступ через оригинальные драйверы баз данных, тогда как другие обеспечивают доступ к данным через ODBC (Open Database Connectivity interface – открытый интерфейс взаимодействия с базами данных) или JDBC (Java Database Connectivity – Java интерфейс взаимодействия с базами данных). Использование R для доступа к данным, хранящимся во внешних DBMS, может быть эффективным способом работы с большими наборами данных (см. приложение G). Это увеличивает эффективность и языка структурированных запросов (Structured Query Language – SQL), и R.

Интерфейс ODBC

Наверное, самый распространенный способ доступа к DBMS в R – это использование пакета `RODBC`, который позволяет R подключиться к любой DBMS, у которой есть драйвер OBMS. К таким системам относятся все перечисленные выше DBMS.

Первый шаг – это установка и конфигурация подходящего ODBC драйвера для конкретной операционной системы и базы данных – они не являются частью R. Если нужные драйверы еще не установлены на вашем компьютере, их можно найти в Интернете.

После установки и конфигурации драйверов установите пакет RODBC. Это можно сделать при помощи команды `install.packages("RODBC")`. Основные функции этого пакета перечислены в табл. 2.2.

Табл. 2.2. Функции пакета RODBC

Функция	Описание
odbcConnect(*dsn*,uid="",pwd="")	Открывает соединение с базой данных ODBC
sqlFetch(*channel*,*sqltable*)	Сохраняет таблицу базы данных ODBC в виде таблицы данных
sqlQuery(*channel*,*query*)	Посылает запрос в базу данных ODBC и выводит на экран результаты
sqlSave(*channel*,*mydf*,tablename =*sqtable*,append=FALSE)	Сохраняет или обновляет (append=TRUE) таблицу данных в виде таблицы в базе данных ODBC
sqlDrop(*channel*,*sqtable*)	Удаляет таблицу из базы данных ODBC
close(*channel*)	Закрывает соединение

Пакет RODBC обеспечивает двустороннее соединение между R и соединенной при помощи ODBC базой данных SQL. Это значит, что вы можете не только загружать данные в R из присоединенной базы данных, но и изменять содержимое базы данных при помощи R. Допустим, вам нужно сохранить две таблицы (Crime – преступление и Punishment – наказание) из DBMS в виде двух таблиц данных с названиями crimedat и pundat соответственно. Это можно сделать при помощи такого программного кода:

```
library(RODBC)
myconn <-odbcConnect("mydsn", uid="Rob", pwd="aardvark")
crimedat <- sqlFetch(myconn, Crime)
pundat <- sqlQuery(myconn, "select
 from Punishment")
close(myconn)
```

Здесь вы загружаете пакет RODBC и открываете соединение с базой данных ODBC через зарегистрированное название источника (mydsn) с идентификатором пользователя (user identifier – UID: rob) и паролем (aardvark). Результат соединения обрабатывается функцией sqlFetch, которая сохраняет таблицу Crime в виде таблицы данных R crimedat. Затем вы запускаете оператор SQL select для таблицы Punishment и сохраняете результат в таблице данных pundat. Наконец, вы закрываете соединение.

Функция sqlQuery() открывает много возможностей, поскольку в нее можно вставить любой действующий оператор SQL. Эта гибкая система позволяет выбирать отдельные переменные, создавать подмножества данных, создавать новые переменные, а также перекодировать и переименовывать существующие переменные.

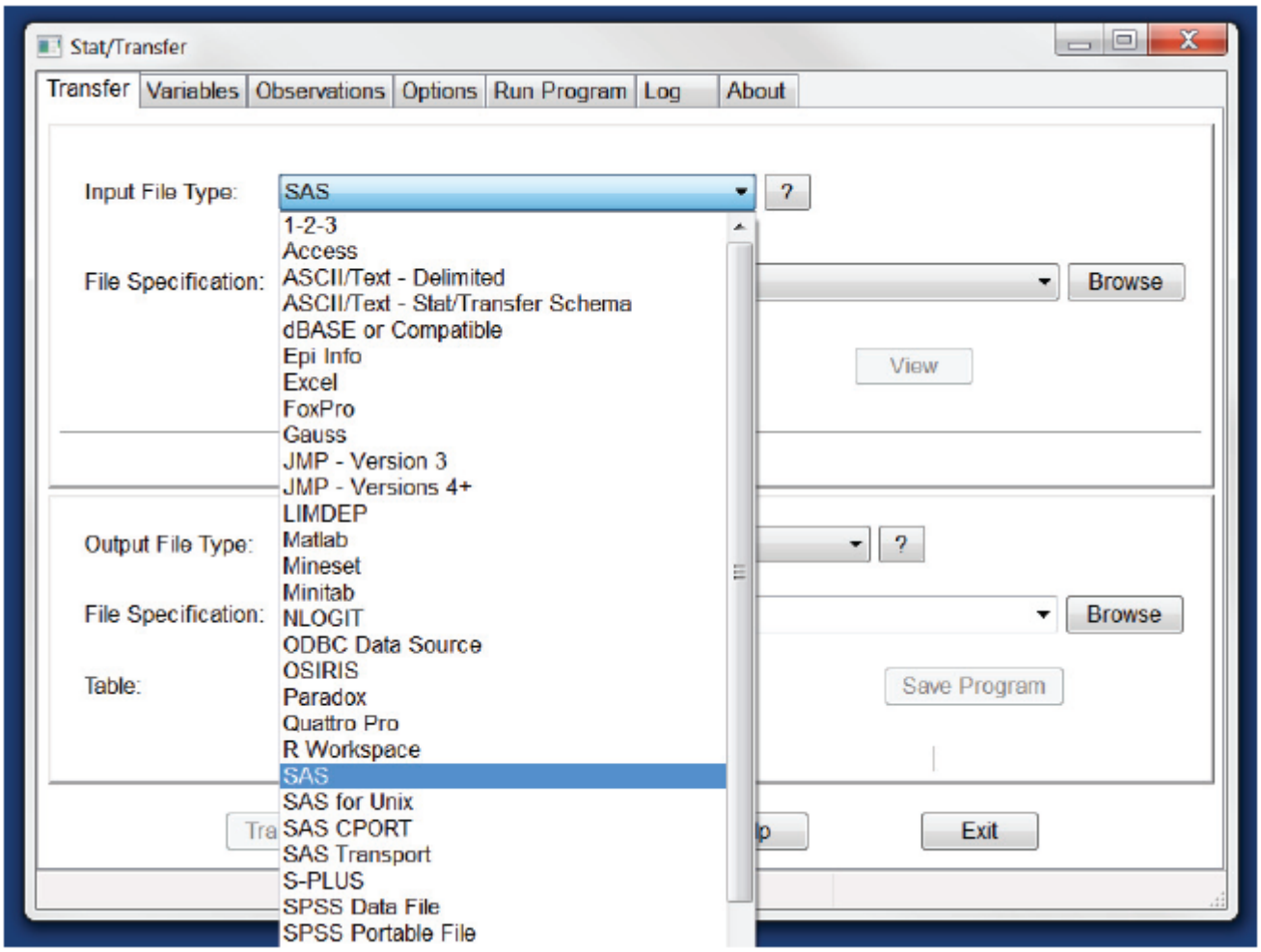
Пакеты, связанные с DBI

Пакет DBI предоставляет широко применимый и последовательный пользовательский интерфейс для DBMS. Созданный на его основе пакет RJDBC предоставляет доступ к DBMS через драйвер JDBC. Убедитесь, что перед его использованием на вашем компьютере установлены драйверы JDBC для соответствующей операционной системы и базы данных. Другие полезные пакеты, созданные на базе DBI – это RMySQL, ROracle, RPostgreSQL и RSQLite. Эти пакеты предоставляют драйверы, предназначенные для конкретных баз данных, однако они могут работать не на всех операционных системах. Ознакомьтесь с документацией CRAN (http://cran.r-project.org) для того, чтобы узнать подробности.

2.3.12. Импорт данных при помощи Stat/Transfer

Прежде чем мы закончим обсуждение импорта данных, стоит упомянуть коммерческую программу, которая может значительно упростить эту задачу. Stat/Transfer (www.stattransfer.com) – это непревзойденная программа, которая может преобразовывать данные в любой из 34 форматов, включая R (см. рис. 2.4).

Рис. 2.4. Основной диалог программы Stat/Transfer под Windows



Существуют версии программы для Windows, Mac и Unix, которые поддерживают форматы данных самых последних версий всех статистических программ, которых мы до этого рассматривали, а также такие DBMS как Oracle, Sybase, Informix и DB/2.

2.4. Аннотирование наборов данных

Исследователи обычно аннотируют наборы данных, чтобы результаты было легче интерпретировать. Обычно аннотация заключается в добавлении поясняющих подписей к названиям переменных и расшифровке кодировки, использованной для категориальных признаков. Например, для переменной `age` вы можете захотеть добавить более ясную подпись “Age at hospitalization (in years)” (Возраст в момент госпитализации (в годах)). Для переменной `gender`, со значениями 1 и 2, у вас может появиться желание поставить в соответствие цифрам надписи “male” (мужской) и “female” (женский).

2.4.1. Подписи для переменных

К сожалению, возможности R по работе с подписями для переменных ограничены. Один подход заключается в том, чтобы использовать подпись для переменной в качестве ее названия, а обозначать

эту переменную ее номером. Рассмотрим наш предыдущий пример с данными о пациентах. Во втором столбце с названием `age` был записан возраст пациентов, в котором они впервые попали в больницу. Программный код

```
names(patientdata)[2] <- "Age at hospitalization (in years)"
```

переименовывает `age` в `"Age at hospitalization (in years)"`. Конечно, это новое название слишком длинное, чтобы писать его много раз. Вместо этого вы можете обозначать эту переменную как `patientdata[2]`, при этом название `"Age at hospitalization (in years)"` будет появляться всюду, где изначально было название `age`. Конечно, это не самый лучший подход к делу, может быть, разумнее остановиться на более удобном названии (например, `admissionAge`).

2.4.2. Расшифровка значений переменных

Функцию `factor()` можно использовать для добавления расшифровки к значениям категориальных переменных. Продолжим рассматривать наш пример. Допустим, что у вас есть переменная `gender`, в которой значение male закодировано как 1, а female – как 2. Подписи к значениям этой переменной можно создать при помощи следующего программного кода:

```
patientdata$gender <- factor(patientdata$gender,
                             levels = c(1,2),
                             labels = c("male", "female"))
```

Здесь `levels` указывают реальные значения переменной, а `labels` представляет собой текстовый вектор, в котором содержатся нужные подписи.

2.5. Полезные функции для работы с объектами

Закончим эту главу коротким обзором полезных функций для работы с объектами (см. табл. 2.3).

Табл. 2.3. Полезные функции для работы с объектами

Функция	Описание
`length(`*`object`*`)`	Число элементов/компонентов объекта.
`dim(`*`object`*`)`	Число измерений объекта.
`str(`*`object`*`)`	Структура объекта.
`class(`*`object`*`)`	Класс или тип объекта.
`mode(`*`object`*`)`	Способ хранения (вид) объекта.
`names(`*`object`*`)`	Названия частей объекта.
`c(`*`object, object,...`*`)`	Объединяет объекты в вектор.
`cbind(`*`object, object, ...`*`)`	Объединяет объекты в виде столбцов.
`rbind(`*`object, object, ...`*`)`	Объединяет объекты в виде строк.
`object`	Выводит на экран весь объект.
`head(`*`object`*`)`	Выводит на экран первую часть объекта.
`tail(`*`object`*`)`	Выводит на экран последнюю часть объекта.
`ls()`	Выводит на экран список имеющихся объектов.
`rm(`*`object, object, ...`*`)`	Удаляет один или более объектов. Команда `rm(list = ls())` удалит почти все объекты из рабочего пространства.
`newobject` `<- edit(`*`object`*`)`	Редактирует объект и сохраняет результат в виде нового объекта.
`fix(`*`object`*`)`	Редактирует сам объект.

Мы уже обсудили большинство из приведенных функций. Команды `head()` и `tail()` используются для быстрого просмотра больших наборов данных. Например, `head(patientdata)` выводит на экран первые шесть строк таблицы данных, тогда как функция `tail(patientdata)` выводит последние шесть. Мы рассмотрим функции `length()`, `cbind()` и `rbind()` в следующей главе. Здесь они собраны все вместе для справки.

2.6. Резюме

Одна из наиболее сложных задач при анализе данных – это их подготовка к анализу. В этой главе мы значительно продвинулись в этом направлении, познакомившись с разными структурами данных в R и с многочисленными способами импорта данных как с клавиатуры, так и из внешних источников. В частности, в последующих главах нам вновь и вновь потребуется понимание того, что такое вектор, матрица, таблица данных и список. Приобретенные вами умения обозначать элементы этих структур с использованием квадратных скобок особенно пригодятся при выборе, разбиении на подмножества и преобразовании данных.

Как вы успели узнать, в R реализовано множество функций для доступа к внешним источникам данных, включая текстовые файлы с разделителями, веб-файлы, статистические программы, электронные таблицы и базы данных. Хотя цель этой главы состояла в том, чтобы познакомить вас со способами импорта данных в R, экспорт данных из R во все эти внешние форматы также возможен. Экспорт данных обсуждается в приложении C, способам работы с большими наборами данных (объем которых измеряется в гигабайтах или терабайтах) посвящено приложение G.

Как только вы загрузили свои данные в R, вам, скорее всего, нужно будет преобразовать их в более удобный для работы вид (честно говоря, я чувствую себя виноватым). В главе 4 мы рассмотрим способы создания новых переменных, преобразования и перекодирования имеющихся переменных, объединения наборов данных и выбора отдельных наблюдений.

Однако прежде чем мы обратимся к задачам управления данными, давайте посвятим некоторое время их графической обработке. Многие читатели выбрали R из-за интереса к ее графическим возможностям, и я больше не хочу заставлять их ждать. В следующей главе мы сразу перейдем к созданию диаграмм. Особое внимание будет уделено общим методам управления и настройки диаграмм, которые будут применяться на протяжении всей книги.

Глава 3. Начало работы с диаграммами

В этой главе:

- создаем и сохраняем диаграммы;
- изменяем тип символов и линий, цвета и оси;
- добавляем текст и названия;
- определяем число измерений диаграммы;
- объединяем несколько диаграмм в одну.

Много раз я показывал клиентам тщательно оформленные результаты статистического анализа в виде чисел и текста, только для того, чтобы увидеть, как у них от недоумения лезут глаза на лоб. Те же самые люди с восторгом восклицали “Ага!”, когда я представлял ту же информацию в графическом виде. Часто я мог увидеть закономерности или выявить ошибки в данных, разглядывая диаграммы – те закономерности и ошибки, которые я совершенно не замечал, выполняя более формализованный статистический анализ.

Люди прекрасно приспособлены для выявления взаимосвязей в графически представленных данных. При помощи хорошо организованной диаграммы можно сопоставить тысячи элементов информации, выявив закономерности, которые не так-то легко обнаружить другими методами. Это одна из причин, по которым достижения в статистической графике оказали такое большое влияние на анализ данных. Исследователям нужно *смотреть* на свои данные, и это одна из областей, где R блестяще себя показал.

В этой главе мы обсудим основные методы работы с диаграммами. Мы начнем с создания и сохранения диаграмм. Затем мы узнаем, как изменять общие для всех типов диаграмм параметры. Сюда входят названия диаграмм, оси, подписи, цвета, линии, символы и текстовые аннотации. Мы сосредоточимся на общих методах, которые можно применить ко всем типам диаграмм.

3.1. Работа с диаграммами

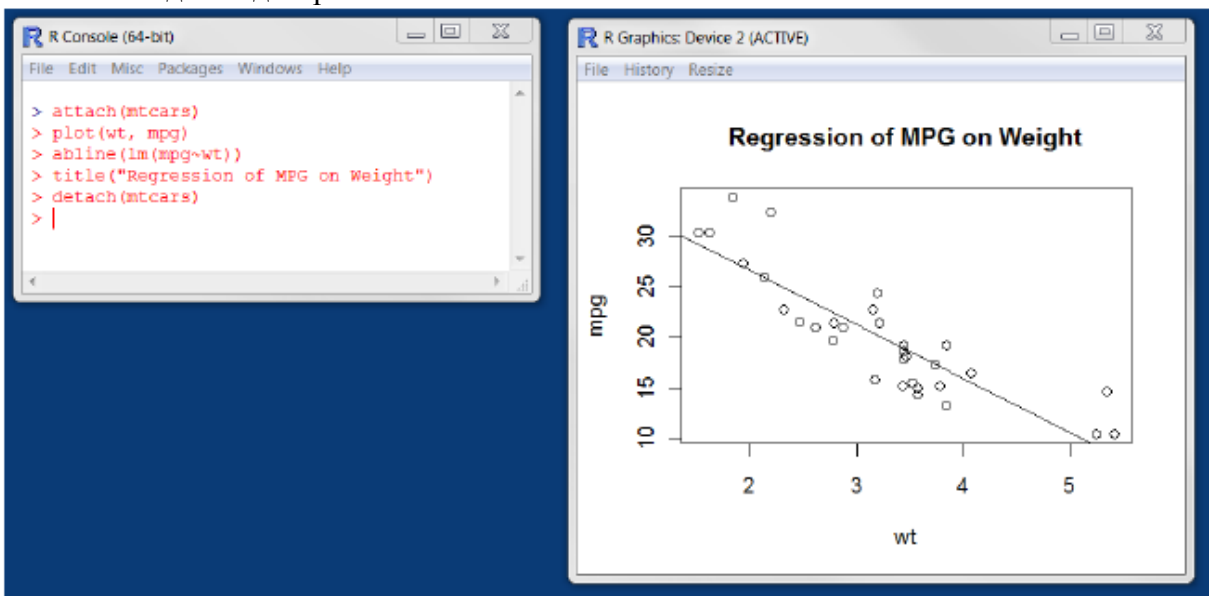
R – это изумительная программа для построения диаграмм. Я специально использую слово «построение». В стандартной интерактивной сессии, вы создаете диаграмму, вводя по одной команде и добавляя элементы диаграммы, пока не получите то, что хотели.

Рассмотрим следующие пять строк:

```
attach(mtcars)
plot(wt, mpg)
abline(lm(mpg~wt))
title("Regression of MPG on Weight")
detach(mtcars)
```

Первая команда добавляет в траекторию поиска таблицу данных `mtcars`. Вторая команда открывает окно графики и создает диаграмму рассеяния, на которой вес автомобиля отложен на горизонтальной оси, а расход топлива – на вертикальной. Третья команда добавляет регрессионную прямую. Четвертая команда добавляет название. Последняя команда удаляет таблицу данных из траектории поиска. В R диаграммы обычно создаются в таком интерактивном стиле (см. рис. 3.1).

Рис. 3.1. Создание диаграммы



Диаграммы можно сохранять при помощи программного кода или меню графического пользовательского интерфейса. Для того чтобы сохранить диаграмму при помощи кода, разместите создающие диаграмму команды между командами, которые назначают место вывода и закрывают вывод. Например, такой программный код позволяет сохранить диаграмму в формате PDF под названием `mygraph.pdf` в текущей рабочей директории:

```
pdf("mygraph.pdf")
 attach(mtcars)
 plot(wt, mpg)
 abline(lm(mpg~wt))
 title("Regression of MPG on Weight")
 detach(mtcars)
dev.off()
```

В дополнении к `pdf()`, можно использовать функции `win.metafile()`, `png()`, `jpeg()`, `bmp()`, `tiff()`, `xfig()` и `postscript()`, чтобы сохранять диаграммы в других форматах. Учтите, что формат Windows metafile доступен только под Windows. Более подробно сохранение диаграмм описано в разделе 1.3.4 первой главы.

Способ сохранения диаграмм при помощи графического пользовательского интерфейса различается в зависимости от оперативной системы. Под Windows нужно, когда активировано окно графики, выбрать в меню **Файл** $$ В русифицированной версии R. – Прим. пер. $$ > **Сохранить как**, а затем в появившемся диалоге выбрать нужный формат графического файла и директорию для сохранения. Под Mac нужно выбрать **File** > **Save as** в строке меню, когда активировано окно графики Quartz. В этом случае диаграмму можно сохранить в единственном формате – PDF. Под Unix диаграммы можно сохранять только при помощи кода. В приложении А мы рассмотрим другие графические интерфейсы для каждой операционной системы, которые имеют больше возможностей.

Новая диаграмма, которая создается при помощи команды высокого уровня, таких как `plot()`, `hist()` или `boxplot()`, обычно заменяет предыдущую диаграмму. Как же создать более одной диаграммы и иметь доступ к каждой из них? Есть несколько способов.

Во-первых, можно открыть новое окно графики *перед* тем как создавать новую диаграмму:

```
dev.new()
 команды для построения диаграммы 1
dev.new()
 команды для построения диаграммы 2
и т.д.
```

Каждая новая диаграмма будет появляться в последнем открытом окне.

Во-вторых, можно иметь доступ к нескольким диаграммам сразу через пользовательский интерфейс. В операционной системе Mac можно перемещаться между диаграммами в любое время при помощи кнопок **Back** и **Forward** в меню Quartz. Под Windows эта операция состоит из двух этапов. После того как открыто *первое* окно графики, выберите в меню **История команд** > **Запись**. Затем используйте пункты меню **Предыдущий** и **Следующий** для перемещения между созданными диаграммами.

В-третьих и в-последних, можно использовать функции `dev.new()`, `dev.next()`, `dev.prev()`, `dev.set()` и `dev.off()` для того, чтобы одновременно открыть несколько окон графики и выбирать, какую диаграмму открыть в каком окне. Эти команды работают под всеми операционными системами. Введите `help(dev.cur)`, чтобы узнать больше об этом подходе.

R создает привлекательные диаграммы при минимальных затратах усилий с нашей стороны. Однако можно использовать графические параметры, чтобы назначать шрифты, цвета, типы линий, создавать оси, вспомогательные линии и аннотации. Подобная гибкость открывает перед вами широкие возможности оптимизации диаграмм.

В этой главе мы начнем с простой диаграммы, а затем обсудим, как ее можно изменять и улучшать, чтобы диаграмма соответствовала вашим задачам. Затем мы рассмотрим усовершенствованные примеры, которые иллюстрируют более сложные методы изменения параметров диаграмм. Особое внимание будет уделено тем методам, которые могут быть использованы для самых разных типов диаграмм. Описанные здесь способы будут применимы для всех диаграмм, описанных в этой книге, за исключением созданных при помощи пакета `lattice` (глава 16). В этом пакете реализованы особые методы настройки параметров диаграмм. В других главах мы рассмотрим остальные типы диаграмм и обсудим, в каких случаях они могут быть особенно полезны.

3.2. Простой пример

Давайте начнем с простого вымышленного набора данных, представленного в табл. 3.1. Он описывает реакцию пациента на два лекарства в пяти дозировках.

Табл. 3.1. Реакция пациента на два лекарства в пяти дозировках

Дозировка	Реакция на лекарство А	Реакция на лекарство В
20	16	15
30	20	18
40	27	25
45	40	31
60	60	40

Эти данные можно ввести при помощи следующего программного кода:

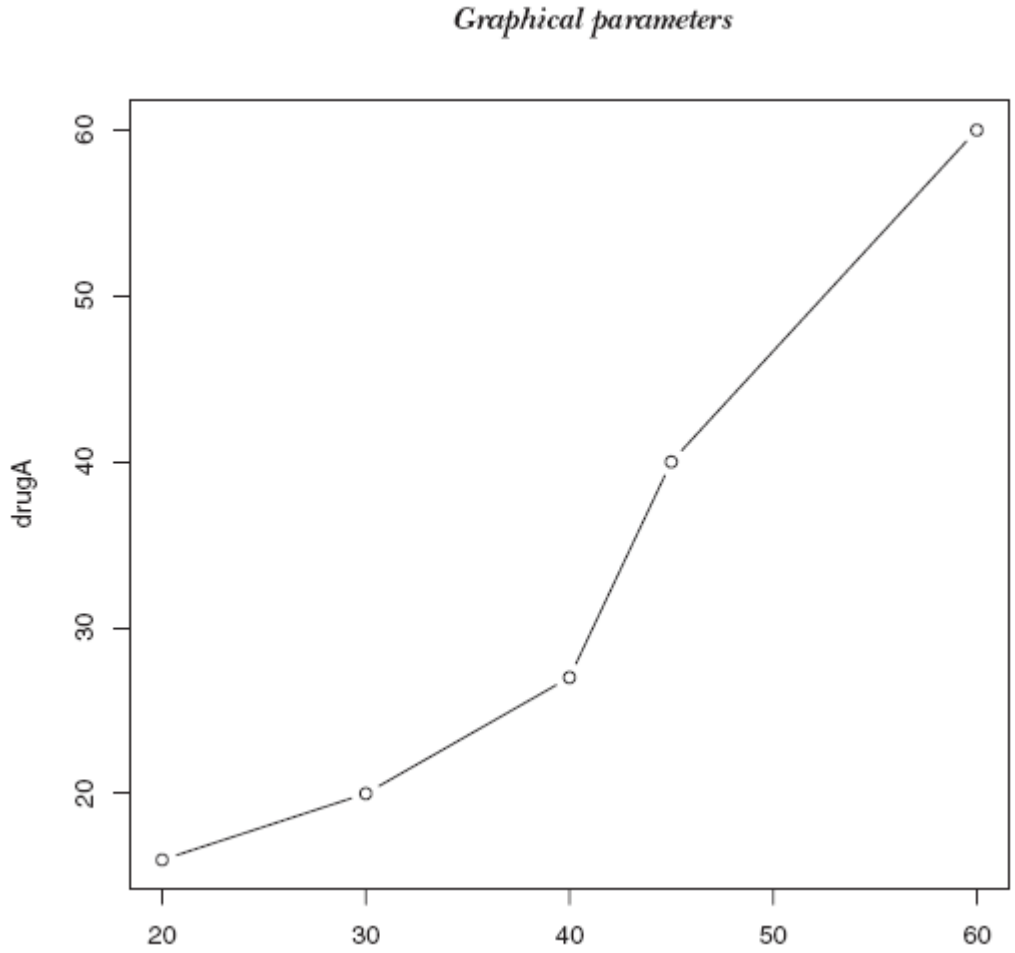
```
dose  <- c(20, 30, 40, 45, 60)
drugA <- c(16, 20, 27, 40, 60)
drugB <- c(15, 18, 25, 31, 40)
```

Простой линейный график, изображающий зависимость реакции пациента от дозы лекарства А, можно создать так:

```
plot(dose, drugA, type="b")
```

`plot()` – это общая функция, которая строит диаграммы в R (то, что получится в результате применения, зависит от типа объекта, к которому функция применена). В этом случае `plot(x, y, type="b")` располагает x на горизонтальной оси, а y – на вертикальной, изображает точки с координатами (x, y) и соединяет их линями. Параметр `type="b"` означает, что на графике должны быть показаны и точки, и линии. Введите `help(plot)`, чтобы узнать о других параметрах. Получившийся график показан на рис. 3.2.

Рис. 3.2. График зависимости реакции пациента от дозы лекарства А



Линейные графики подробно обсуждаются в главе 11. Давайте теперь изменим внешний вид этого графика.

3.3. Графические параметры

Многие характеристики диаграмм (шрифты, цвета, оси, названия) можно изменять при помощи опций, которые называются *графические параметры*.

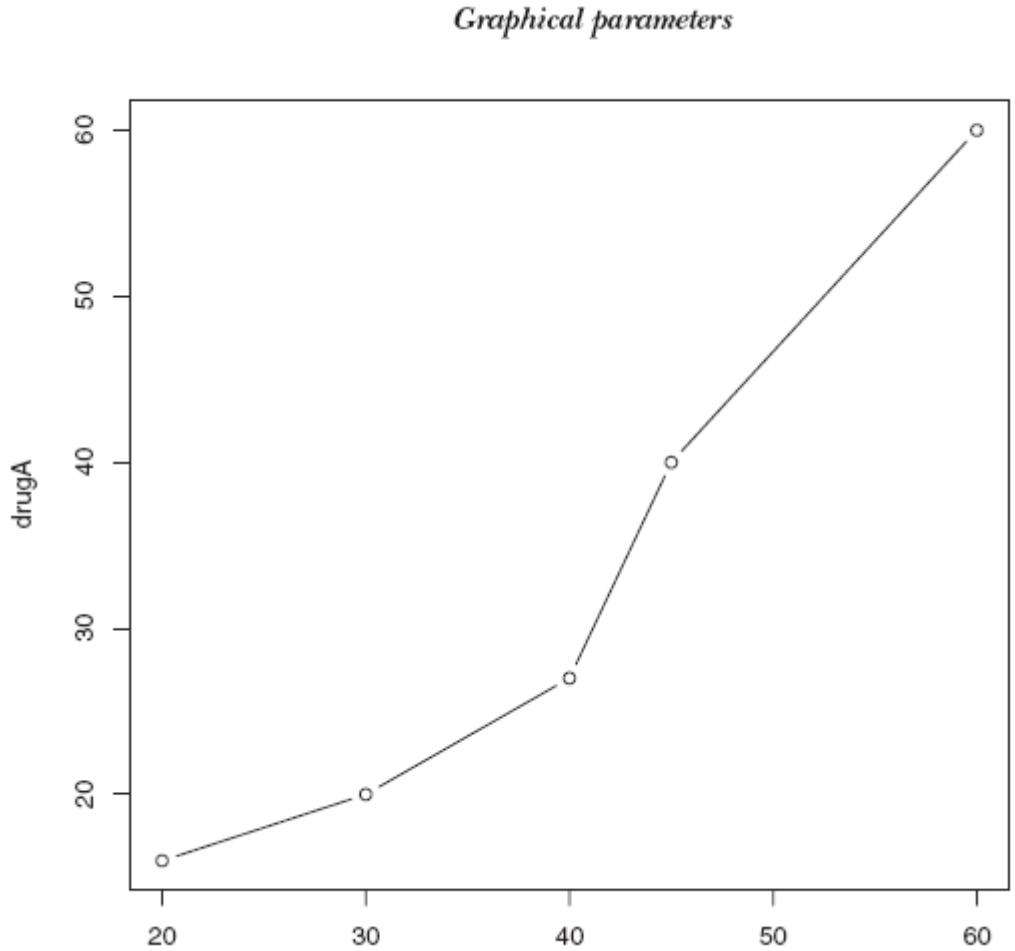
Один способ назначить эти параметры – использовать функцию `par()`. Значения параметров, заданные таким способом, будут действовать на протяжении всей сессии, пока вы не измените их. Формат применения функции таков: `par(название_параметра=значение, название_параметра=значение, ...)`. Функция `par()` без аргументов выводит на экран действующие значения графических параметров. Добавление аргумента `no.readonly=TRUE` позволяет увидеть только те графические параметры, которые можно менять.

Продолжая наш пример, представим, что вам захотелось для обозначения отдельных пациентов использовать заполненный треугольник вместо пустого кружка и соединить символы пунктирной линией, а не сплошной. Это можно сделать при помощи следующего программного кода:

```
opar <- par(no.readonly=TRUE)
par(lty=2, pch=17)
plot(dose, drugA, type="b")
par(opar)
```

Получившийся график показан на рис. 3.3.

Рис. 3.3. График зависимости реакции пациента от дозы лекарства А с измененными типом линии и символами



Первая команда создает копию текущих параметров. Вторая команда назначает тип линии – пунктирная (`lty=2`) вместо сплошной по умолчанию и тип символа – заполненный треугольник (`pch=17`). Затем вы создаете график и восстанавливаете исходные значения параметров. Типы линий и символов рассмотрены в разделе 3.3.1.

Можно использовать столько функций `par()`, сколько нужно, так что команда может быть также записана в виде

```
par(lty=2)
par(pch=17)
```

Второй способ задать графические параметры – это включить записи типа `название_параметра=значение` внутрь графической функции высокого уровня. В этом случае заданные параметры будут действовать только для конкретной диаграммы. Можно было бы построить тот же график при помощи следующего программного кода:

```
plot(dose, drugA, type="b", lty=2, pch=17)
```

Не во всех графических функциях высокого уровня можно изменять все возможные графические параметры. Познакомьтесь со справкой по каждой функции для построения диаграмм (например, `?plot`, `?hist` или `?boxplot`), чтобы узнать, какие графические параметры можно назначать таким образом. В оставшейся части раздела 3.3.1. описаны многие важные графические параметры, которые вы можете менять.

3.3.1. Символы и линии

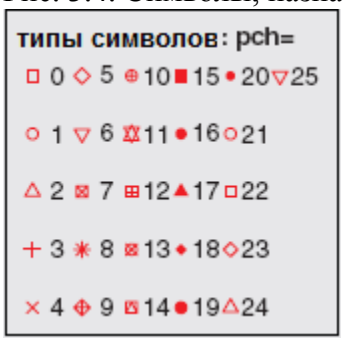
Как вы поняли, графические параметры можно использовать для того, чтобы указывать тип символов и линий на диаграммах. Соответствующие параметры перечислены в табл. 3.2.

Таблица 3.2. Параметры для указания типов символов и линий

Параметр	Описание
`pch`	Определяет тип символа (см. рис. 3.4)
`cex`	Определяет размер символа. `cex` – это число, обозначающее, как символы должны быть масштабированы по отношению к размеру по умолчанию. 1 = размер по умолчанию, 1.5 – на 50% крупнее, 0.5 – на 50% мельче и т.д.
`lty`	Определяет тип линии (см. рис. 3.5)
`lwd`	Определяет толщину линии по сравнению с толщиной линии по умолчанию (1). Например, `lwd=2` делает линию в два раза толще, чем по умолчанию.

Параметр `pch=` определяет тип символов, которые используются на диаграмме. Возможные значения приведены на рис. 3.4.

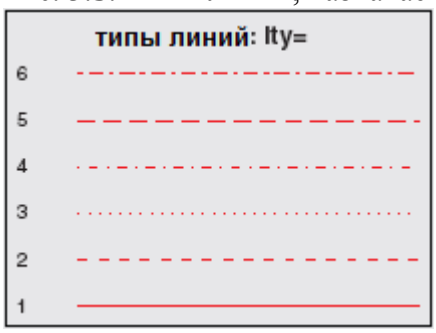
Рис. 3.4. Символы, назначаемые при помощи параметра `pch`



Для символов с 21 по 25 можно отдельно указывать цвет контура (`border=`) и заполнения (`bg=`).

Используйте параметр `lty=` для обозначения нужного типа линии. Значения параметра показаны на рис. 3.5.

Рис. 3.5. Типы линий, назначаемые при помощи параметра `lty`

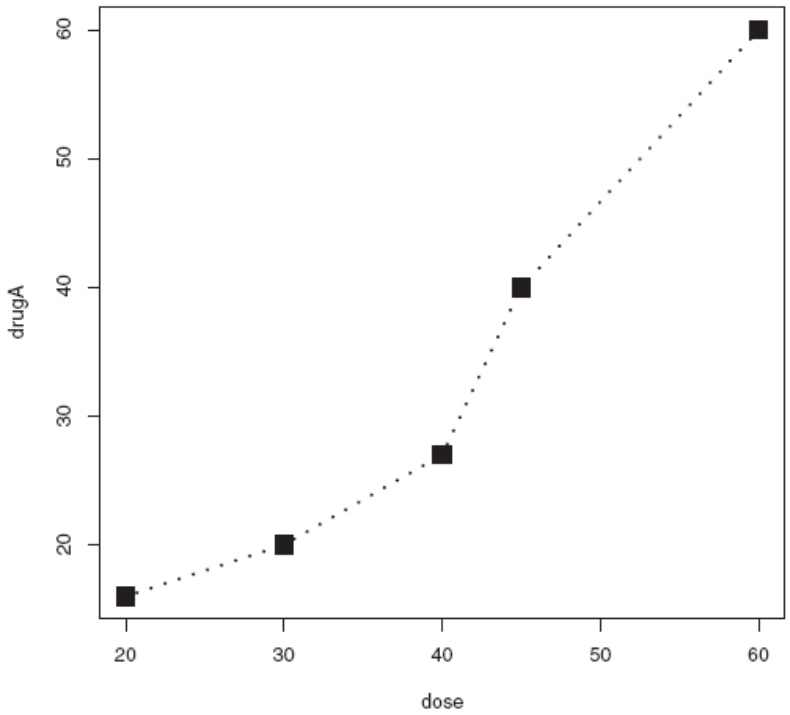


Программный код, объединяющий все эти параметры

```
plot(dose, drugA, type="b", lty=3, lwd=3, pch=15, cex=2)
```

создаст график, на котором точечная линией в три раза шире, чем по умолчанию, соединяет наблюдения, представленные в виде заполненных квадратов в два раза большего размера, чем по умолчанию. Результат представлен на рис. 3.6.

Рис. 3.6. График зависимости реакции пациента от дозы лекарства А с измененными типом и шириной линии, а также типом и размером символов



Теперь давайте посмотрим, как назначать цвета.

3.3.2. Цвета

В R есть несколько связанных с цветами параметров. В табл. 3.3 приведены некоторые из самых распространенных.

Табл. 3.3. Параметры для назначения цвета

Параметр	Описание
`col`	Цвет элементов на графике. Для некоторых функций (такие как `lines` и `pie`) можно указывать вектор из значений, которые используются по очереди. Например, если `col=c("red", "blue")`, и изображено три линии, первая будет красной, вторая – синей и третья – красной.
`col.axis`	Цвет значений осей.
`col.lab`	Цвет подписей осей.
`col.main`	Цвет заголовков.
`col.sub`	Цвет подзаголовков.
`fg`	Цвет графика.
`bg`	Цвет фона.

В R цвета можно обозначать номером, названием, в шестнадцатеричной системе, а также в системах RBG или HSV. Например, `col=1`, `col="white"`, `col="#FFFFFF"`, `col=rgb(1,1,1)` и `col=hsv(0,0,1)` – взаимозаменяемые способы обозначить белый цвет. Функция `rgb()` определяет цвета по значениям красного, зеленого и синего, а `hsv()` основана на значениях оттенка и насыщенности. Более полную информацию об этих функциях можно получить из файла справки.

Функция `colors()` выводит на экран список всех доступных цветов. Эйрл Ф. Глин (Earl F. Glynn) создал прекрасную онлайн-таблицу цветов R, доступную по адресу http://research.stowers-institute.org/efg/R/Color/Chart. В R также реализован ряд функций, которые позволяют создавать вектора из близких цветов. К таким функциям относятся `rainbow()`, `heat.colors()`, `terrain.colors()`, `topo.colors()` и `cm.colors()`. Например, `rainbow(10)` создает 10 соседних “радужных” цветов. Оттенки серого создаются функцией `gray()`. В этом случае вы задаете оттенки серого в виде вектора чисел от 0 до 1. Команда `gray(0:10/10)` создаст 10 оттенков серого. Попробуйте программный код

```
n <- 10
mycolors <- rainbow(n)
pie(rep(1, n), labels=mycolors, col=mycolors)
mygrays <- gray(0:n/n)
pie(rep(1, n), labels=mygrays, col=mygrays)
```

чтобы увидеть, как это работает. Примеры применения цветовых параметров будут встречаться вам на всем протяжении этой главы.

3.3.3. Характеристики текста

Графические параметры также используются для определения размера, шрифта и стиля текста. Параметры, определяющие размер шрифта, приведены в табл. 3.4. Параметры, при помощи которых можно указать тип шрифта, перечислены в табл. 3.5.

Табл. 3.4. Параметры, определяющие размер шрифта

Параметр	Описание
`cex`	Число, определяющее, как отображаемый на диаграмме текст будет масштабирован относительно размера по умолчанию (1). 1.5 – на 50% больше, 0.5 – на 50% меньше и т.д.
`cex.axis`	Размер значений на осях по отношению к `cex`.
`cax.lab`	Размер подписей по осям по отношению к `cex`.
`cex.main`	Размер заголовков по отношению к `cex`.
`cex.sub`	Размер подзаголовков по отношению к `cex`.

Например, на всех диаграммах, созданных после команды

```
par(font.lab=3, cex.lab=1.5, font.main=4, cex.main=2)
```

в 1.5 раза более крупные, чем по умолчанию, подписи осей будут выделены курсивом, а названия будут в два раза крупнее, чем по умолчанию, и еще выделены полужирным курсивом.

Таблица 3.5. Параметры, определяющие семейство, размер и стиль шрифта

Параметр	Описание
`font`	Число, которое определяет шрифт для текста на диаграмме. 1=обычный, 2=полужирный, 3=курсив, 4=полужирный курсив, 5=символы (в кодировке Adobe).
`font.axis`	Шрифт значений на осях.
`font.lab`	Шрифт для подписей по осям.
`font.main`	Шрифт для заголовков.
`font.sub`	Шрифт для подзаголовков.
`ps`	Размер точки в шрифте (приблизительно 0.3 мм).
`family`	Семейство шрифтов. Стандартные значения – `serif`, `sans` и `mono`.

Размер и стиль шрифта установить просто, дело с семейством шрифтов обстоит немного сложнее. Это происходит потому, что отображение `serif`, `sans` и `mono` зависит от устройства. Например, под Windows `mono` отображается как TT Courier New, `serif` – как TT Times New Roman, а `sans` – как TT Arial (TT обозначает True Type) Если вы удовлетворены таким отображением семейств шрифтов, то можете использовать параметры типа `family="serif"`, чтобы добиться желаемого результата. Если вы не удовлетворены, вам нужно создать новую систему соответствий. Под Windows вы можете назначать эти соответствия при помощи функции `windowsFont()`. Например, после выполнения команды

```
windowsFonts(
  A=windowsFont("Arial Black"),
  B=windowsFont("Bookman Old Style"),
  C=windowsFont("Comic Sans MS")
)
```

вы сможете использовать A, B и C как названия семейств шрифтов. В этом случае `par(family="A")` назначит шрифт Arial Black (программный код 3.2 в разделе 3.4.2 содержит

пример изменения параметров текста). Обратите внимание, что функция `windowsFont()` работает только под Windows. Под Mac используйте вместо нее `quartzFonts()`.

Если диаграммы будут сохранены в формате PDF или PostScript, изменить семейство шрифтов сравнительно просто. Для формата PDF используйте команду `names(pdfFonts())`, чтобы узнать, какие шрифты вам доступны, и команду `pdf(file="myplot.pdf", family="название_шрифта")`, чтобы создать диаграмму. Для формата PostScript используйте `names(postscriptFonts())` и `postscript(file="myplot.ps", family="название_шрифта")`. Прочтите онлайн-справку для получения дальнейшей информации.

3.3.4. Размеры диаграммы и полей

Наконец, можно определять размер диаграммы и полей при помощи параметров, приведенных в табл. 3.6.

Таблица 3.6. Параметры для определения размеров диаграммы и полей

Параметр	Описание
`pin`	Размер диаграммы (ширина, высота) в дюймах $$ 1 дюйм = 2.5 см. – Прим. пер.$$
`mai`	Числовой вектор, задающий размеры полей, где c(низ, лево, верх, право) указаны в дюймах
`mar`	Числовой вектор, задающий размеры полей, где c(низ, лево, верх, право) указаны в числе строк. По умолчанию это c(5, 4, 4, 2) + 0.1.

Команда

```
par(pin=c(4,3), mai=c(1,.5, 1, .2))
```

позволяет создавать диаграммы размером 4 дюйма в ширину и 3 дюйма в высоту с шириной полей сверху и снизу по одному дюйму, слева 0.5 дюйма и справа 0.2 дюйма. Исчерпывающее онлайн-руководство по настройке параметров полей от Эйрл Глин размещено по адресу http://research.stowers-institute.org/efg/R/Graphics/Basics/mar-oma/.

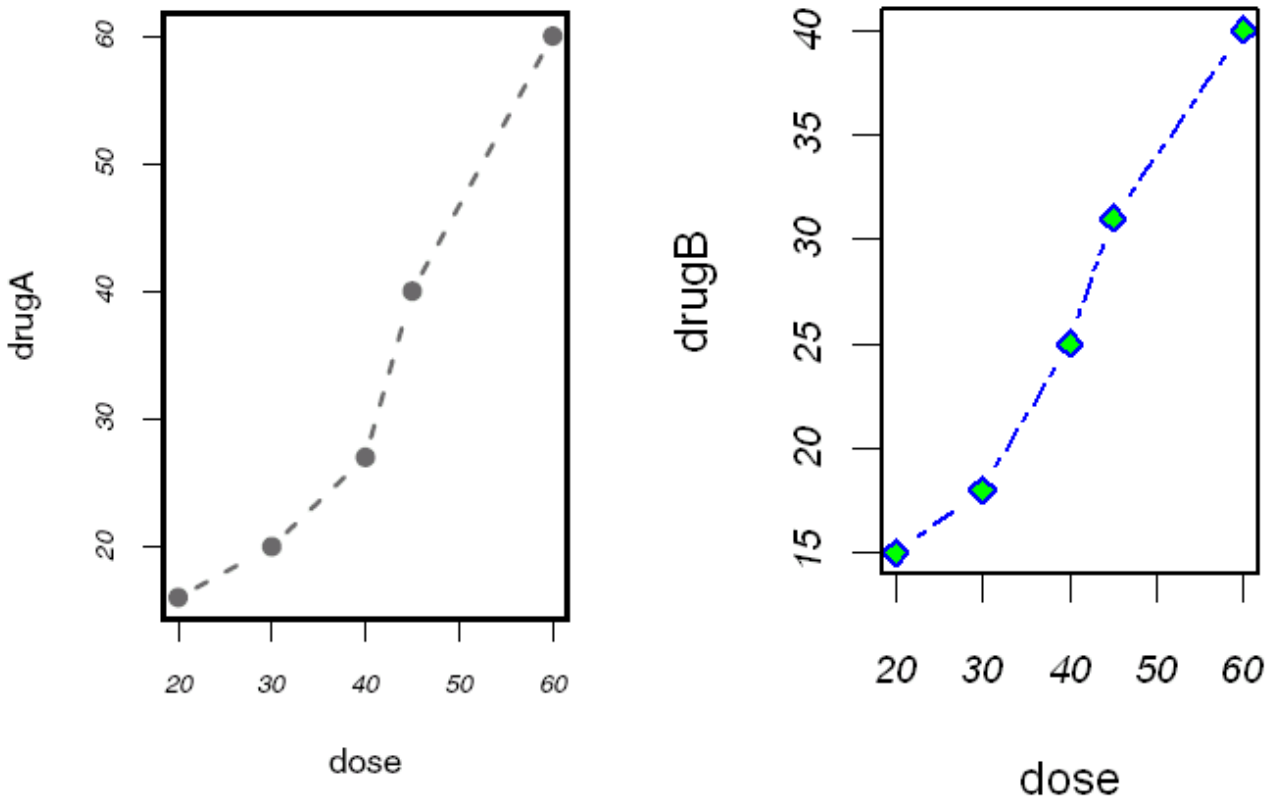
Давайте используем все параметры, которые мы успели обсудить, чтобы усовершенствовать наш простой пример. Представленный ниже программный код позволяет получить диаграммы, показанные на рис. 3.7.

Программный код 3.1. Использование графических параметров для определения внешнего вида диаграммы

```
dose  <- c(20, 30, 40, 45, 60)
drugA <- c(16, 20, 27, 40, 60)
drugB <- c(15, 18, 25, 31, 40)
opar <- par(no.readonly=TRUE)
par(pin=c(2, 3))
par(lwd=2, cex=1.5)
par(cex.axis=.75, font.axis=3)
plot(dose, drugA, type="b", pch=19, lty=2, col="red")
plot(dose, drugB, type="b", pch=23, lty=6, col="blue", bg="green")
par(opar)
```

Сначала вы вводите данные в виде векторов, затем сохраняете текущие графические параметры (чтобы можно было восстановить их позднее). Вы изменяете графические параметры так, чтобы графики были 2 дюйма в ширину и 3 дюйма в высоту. Кроме того, линии будут в два раза шире, а символы – в 1.5 крупнее, чем по умолчанию. Значения на осях даны курсивом, их размер составляет 75% от размера по умолчанию. Затем создан первый график с заполненными красными кружками и пунктирными линиями. Второй график содержит зеленые ромбы с синей каймой и синие линии. Наконец, вы восстанавливаете исходные графические параметры.

Рис. 3.7. Линейный график зависимости реакции пациента от дозы лекарств A и B



Обратите внимание на то, что параметры, заданные при помощи функции `par()` применяются к обоим графикам, а параметры, назначенные «внутри» функций `plot()`, действуют только для соответствующего графика. Посмотрев на рис. 3.7, вы можете заметить несколько недочетов. У графиков нет заголовков, вертикальные оси даны в разных масштабах, что затрудняет сравнение действия двух лекарств. Подписи по осям тоже могли бы быть более информативными.

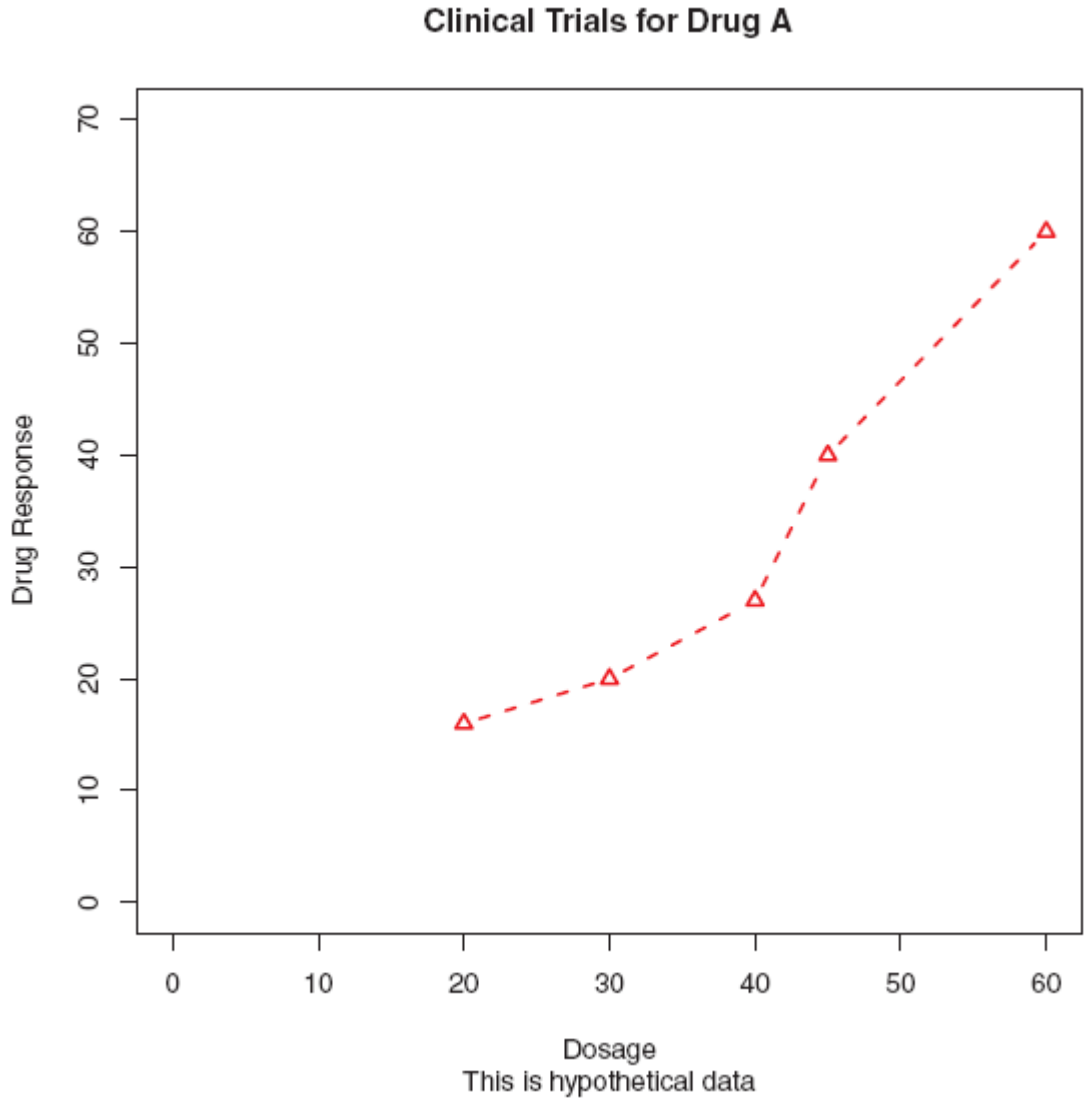
В следующем разделе мы перейдем к настройке параметров текстовых аннотаций (таких как заголовки и подписи) и осей. Чтобы получить больше информации об имеющихся графических параметрах, загляните в `help(par)`.

3.4. Добавление текста, настройка параметров осей и условных обозначений

Для многих графических функций высокого уровня (например, `plot`, `hist`, `boxplot`) возможен контроль не только графических параметров, но и параметров осей и надписей. К примеру, при помощи приведенного ниже программного кода можно разместить на диаграмме заголовок (`main`), подзаголовок (`sub`) и подписи осей (`xlab`, `ylab`), а также задать диапазон значений на осях (`xlim`, `ylim`). Результат представлен на рис. 3.8.

```
plot(dose, drugA, type="b",
     col="red", lty=2, pch=2, lwd=2,
     main="Clinical Trials for Drug A",
     sub="This is hypothetical data",
     xlab="Dosage", ylab="Drug Response",
     xlim=c(0, 60), ylim=c(0, 70))
```

Рис. 3.8. Линейный график зависимости реакции пациента от дозы лекарства А с заголовком, подзаголовком и модифицированными осями



Опять же, не все графические функции позволяют задавать эти параметры. Параметры, которые можно менять, указаны в файле справки для соответствующей функции. Для изменения параметров заголовков, осей, легенд и текстовых аннотаций на диаграммах вы можете использовать функции, которые описаны в этом разделе.

Примечание. *Некоторые графические функции высокого уровня по умолчанию выводят надписи и подписи на диаграммах. От них можно избавиться, указав* `ann=FALSE` *как один из аргументов команд* `plot()` *или* `par()`.

3.4.1. Заголовки

Для размещения заголовков и подписей осей на диаграмме используйте функцию `title()`. Формат ее применения таков:

```
title(main="основной_заголовок", sub="подзаголовок",
      xlab="подпись_по_оси_x", ylab="подпись_по_оси_y")
```

Графические параметры (такие как размер и тип шрифта, ориентация и цвет текста) тоже можно задать функции `title()`. К примеру, такой программный код позволяет получить диаграмму с красным заголовком, синим подзаголовком и зелеными подписями по осям, размер которых на 25% меньше, чем по умолчанию:

```
title(main="My Title", col.main="red",
      sub="My Sub-title", col.sub="blue",
      xlab="My X label", ylab="My Y label",
      col.lab="green", cex.lab=0.75)
```

3.4.2. Оси

Вместо осей, создаваемых на диаграммах по умолчанию, вы можете создать оси по своему усмотрению, используя функцию `axis()`. Формат ее применения таков (все параметры описаны в табл. 3.7):

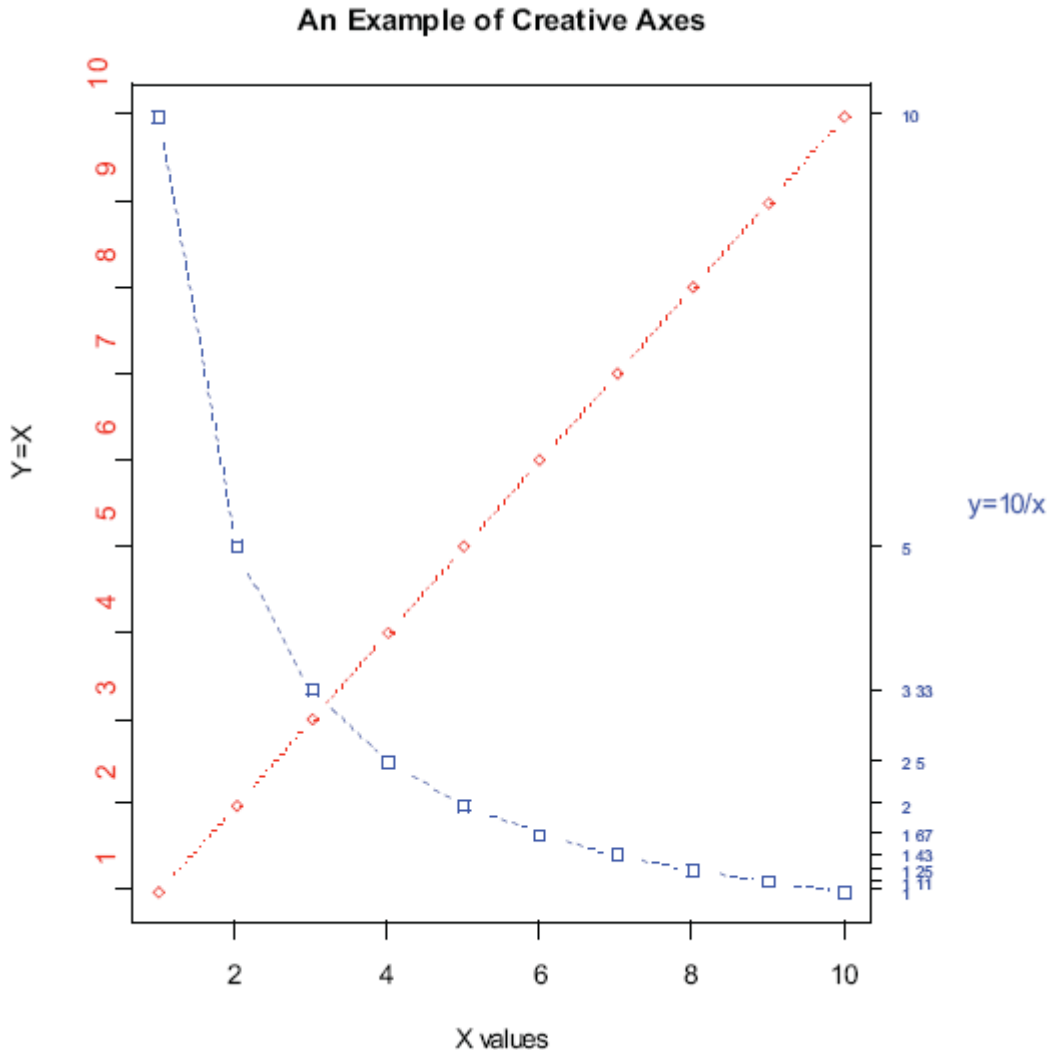
```
axis(side, at=, labels=, pos=, lty=, col=, las=, tck=, ...).
```

Табл. 3.7. Параметры осей

Параметр	Описание
`side`	Цифра, определяющая, с какой стороны диаграммы рисовать ось (1=низ, 2=лево, 3=верх, 4=право).
`at`	Числовой вектор, который задает положение делений на осях.
`labels`	Текстовый вектор, который содержит подписи под делениями осей (если вектор не задан, используются значения вектора `at`).
`pos`	Координата оси (то есть значение другой оси, в котором первая ось пересекает ее)
`lty`	Тип линии.
`col`	Цвет линии и делений оси.
`las`	Положение подписей делений по отношению к оси (0=параллельно, 2=перпендикулярно).
`tck`	Длина деления оси, выражается в виде доли от длины диаграммы (отрицательное число означает положение деления кнаружи от рамки диаграммы, положительное число – внутри рамки диаграммы, 0 – отсутствие делений, 1 – сетка); значение по умолчанию –0.01.
`(...)`	Остальные графические параметры.

Когда вы сами создаете оси, нужно предотвратить появление осей, которые создаются по умолчанию графической функцией высокого уровня. Аргумент `axes=FALSE` подавляет создание всех осей (даже рамку вокруг диаграммы, если вы не добавите аргумент `frame.plot=TRUE`). Аргументы `xaxt="n"` и `yaxt="n"` отменяют создание x- и y-осей соответственно (при этом рамка без делений остается). Приведенный ниже программный код – это слегка избыточный глуповатый пример, который демонстрирует применение всех аргументов, которые мы успели обсудить. График, который получается в результате, представлен на рис. 3.9.

Рис. 3.9. Пример модификации осей



Программный код 3.2. Пример настройки параметров осей

```
x <- c(1:10)                          ← Создаем данные
y <- x
z <- 10/x
opar <- par(no.readonly=TRUE)
par(mar=c(5, 4, 4, 8) + 0.1)
plot(x, y, type="b",                  ← Увеличиваем поля
     pch=21, col="red",
     yaxt="n", lty=3, ann=FALSE)
lines(x, z, type="b", pch=22, col="blue", lty=2)   ← Добавляем линию
зависимости 10/x от x

axis(2, at=x, labels=x, col.axis="red", las=2)←  Рисуем оси
axis(4, at=z, labels=round(z, digits=2),
     col.axis="blue", las=2, cex.axis=0.7, tck=-.01)← Добавляем подписи и
текст
mtext("y=1/x", side=4, line=3, cex.lab=1, las=2, col="blue")
title("An Example of Creative Axes",
      xlab="X values",
      ylab="Y=X")
par(opar)
```

Сейчас вам уже знакомы почти все команды из этого программного кода, за исключением `line()` и `mtext()`. Команда `plot()` начинает построение новой диаграммы. Используя вместо этого команду `line()`, вы можете добавить новые графические элементы к *уже существующей* диаграмме. Вам еще придется использовать эту функцию в разделе 3.4.4 для того, чтобы на одной диаграмме изобразить реакцию пациента на лекарства А и В. Функция `mtext()` нужна, чтобы размещать текст на полях диаграммы. Эта функция обсуждается в разделе 3.4.5, а функция `line()` более подробно разобрана в главе 11.

Вспомогательные деления на осях

Обратите внимание на то, что на всех диаграммах, которые вы создавали до сих пор, были только основные деления на осях, но не было вспомогательных. Для того чтобы разместить на осях вспомогательные деления, нужно воспользоваться функцией `minor.tick()` из пакета `Hmisc()`. Сначала установите пакет `Hmisc()`, если вы еще не сделали этого (см. раздел 1.4.2 главы 1). Добавить вспомогательные деления можно при помощи следующего программного кода:

```
library(Hmisc)
minor.tick(nx=n, ny=n, tick.ratio=n)
```

где `nx` и `ny` – число интервалов, на которое нужно разбить расстояние между основными делениями оси x и оси y соответственно. Текущую длину основных делений можно узнать при помощи команды `par("tck")`. Например, приведенный ниже программный код позволяет добавлять одно вспомогательное деление между соседними основными на оси x и два вспомогательных деления на оси y:

```
minor.tick(nx=2, ny=3, tick.ratio=0.5)
```

Длина этих вспомогательных делений составит 50% от длины основных делений. Пример использования вспомогательных делений приведен в следующем разделе (программный код 3.3, рис. 3.10).

3.4.3. Опорные линии

Функция `abline()` применяется для добавления опорных линий на нашу диаграмму. Формат ее применения таков:

```
abline(h=значение_y, v=значение_x)
```

При помощи этой функции также можно задавать другие графические параметры (например, цвет, тип и ширину линий). Например, команда

```
abline(h=c(1,5,7))
```

добавляет на диаграмму горизонтальные сплошные линии, пересекающие ось y в значениях 1, 5 и 7, тогда как команда

```
abline(v=seq(1, 10, 2), lty=2, col="blue")
```

создает вертикальные пунктирные синие линии, которые пересекают ось x в значениях 1, 3, 5, 7 и 9. Программный код 3.3 содержит команду для добавления опорной линии y=30 для нашего примера с лекарствами. Результат показан на рис. 3.10.

3.4.4. Легенда

Когда на диаграмме представлено больше одного набора данных или одной группы объектов, легенда помогает узнать, что изображает каждый столбик, фрагмент круговой диаграммы или линия. Легенду можно добавить на диаграмму при помощи функции `legend()`. Формат ее применения таков:

```
legend(location, title, legend, ...)
```

Распространенные аргументы перечислены в табл. 3.8.

Табл. 3.8. Аргументы функции `legend()`

Аргумент	Описание
`location`	Существует несколько способов определить положение легенды на диаграмме. Можно указать x,y координаты верхнего левого угла легенды. Можно использовать команду `locator(1)`; в этом случае вы используете мышку, чтобы указать положение легенды. Также можно обозначать положение легенды при помощи ключевых слов `bottom` (внизу), `bottomleft` (внизу слева), `left` (слева), `topleft` (сверху слева), `top` (сверху), `topright` (сверху справа), `right` (справа), `bottomright` (снизу справа) или `center` (в центре). Если вы применяете одно из этих ключевых слов, можно также использовать параметр `inset=` для того, чтобы указать, как далеко (в долях от размера диаграммы) нужно сдвинуть легенду внутрь диаграммы.
`title`	Текстовая строка – название легенды (необязательно)
`legend`	Текстовый вектор с расшифровкой значений легенды.
...	Другие аргументы. Если легенда расшифровывает цвета линий, укажите `col=` и приведите вектор с названиями цветов. Если легенда поясняет типы символов, укажите `pch=` и приведите вектор с номерами символов. Если легенда указывает значения линий разной ширины или стиля, используйте `lwd=` или `lty=` и вектор значений ширины или стиля линий. Для того чтобы включить в легенду закрашенные квадратики (часто применяется для столбчатых или круговых диаграмм), используйте `fill=` и вектор со значениями цветов.

Другие распространенные аргументы функции `legend()` – это `bty` (тип рамки), `bg` (цвет фона), `cex` (размер текста) и `text.col` (цвет текста). Легенду можно расположить горизонтально, а не вертикально при помощи аргумента `horiz=TRUE`. Чтобы получить более подробную информацию о легендах, читайте `help(legend)`. Особенно информативны приведенные в файле справки примеры.

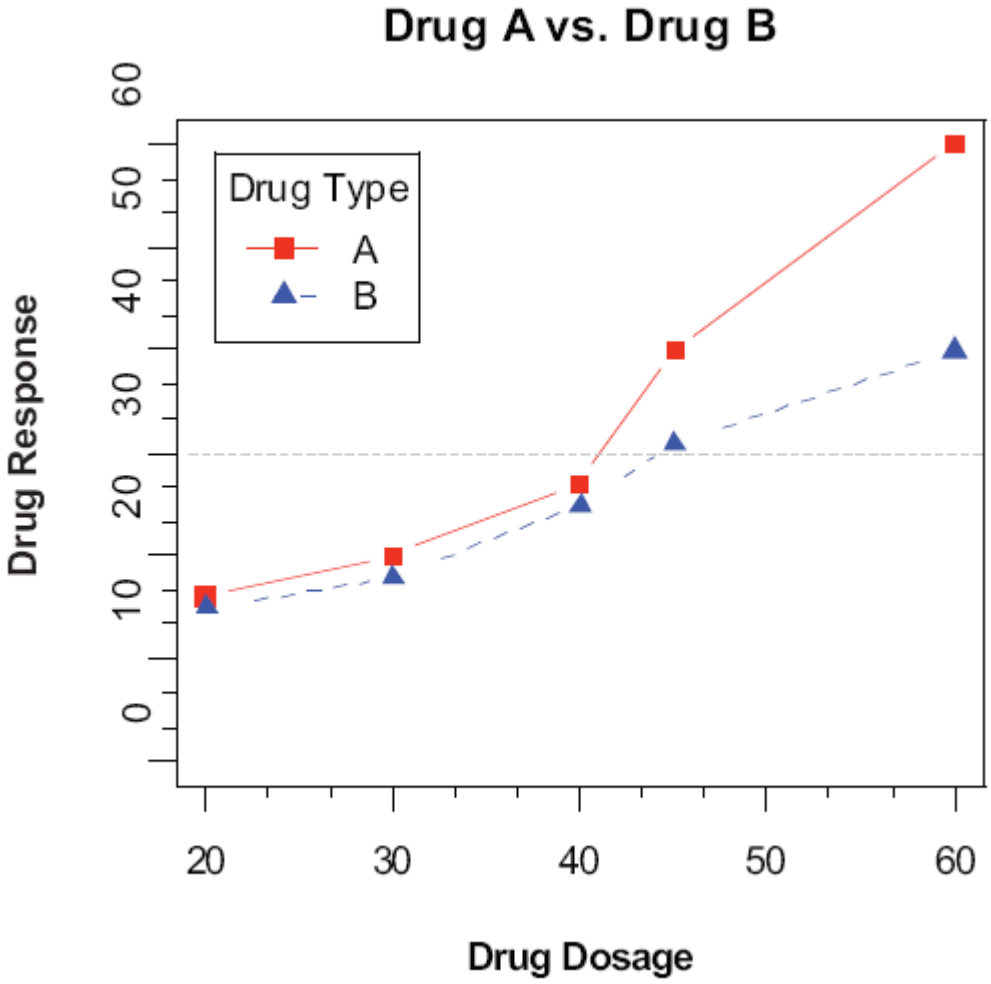
Давайте рассмотрим пример с использованием наших данных по лекарствам (программный код 3.3). Мы вновь используем много функций, которые успели рассмотреть к этому времени. Получившийся график представлен на рис. 3.10.

Программный код 3.3. Сопоставление реакции пациента на разные дозы лекарств A и B

```
dose  <- c(20, 30, 40, 45, 60)
drugA <- c(16, 20, 27, 40, 60)
drugB <- c(15, 18, 25, 31, 40)
opar <- par(no.readonly=TRUE)
par(lwd=2, cex=1.5, font.lab=2)
```
← Увеличиваем ширину линии, размер символов и подписей

```
plot(dose, drugA, type="b",      ← Создаем график
     pch=15, lty=1, col="red", ylim=c(0, 60),
     main="Drug A vs. Drug B",
     xlab="Drug Dosage", ylab="Drug Response")
lines(dose, drugB, type="b",
      pch=17, lty=2, col="blue")
abline(h=c(30), lwd=1.5, lty=2, col="gray")
library(Hmisc) ←  – Добавляем промежуточные деления на осях
minor.tick(nx=3, ny=3, tick.ratio=0.5)
legend("topleft", inset=.05, title="Drug Type", c("A","B"), ← Добавляем
условные обозначения
       lty=c(1, 2), pch=c(15, 17), col=c("red", "blue"))
par(opar)
```

Рис. 3.10. Аннотированное сопоставление лекарств A и B



Почти все параметры графика, представленного на рис. 3.10, могут быть изменены с использованием команд, описанных в этой главе. В дополнение к этому существует много способов назначения необходимых параметров. Последний тип аннотирования диаграммы, который нам осталось рассмотреть, это добавление текста. Этот аспект разобран в следующем разделе.

3.4.5. Аннотации

Текст можно добавлять на диаграмму при помощи команд `text()` и `mtext()`. Команда `text()` позволяет поместить текст внутри диаграммы, а функция `mtext()` размещает текст на одном из четырех полей диаграммы. Форматы применения таковы:

```
text(location, "текст", pos, ...)
```

```
mtext("текст", side, line=n, ...)
```

часто используемые аргументы перечислены в табл. 3.9.

Табл. 3.9. Аргументы для функций `text()` и `mtext()`

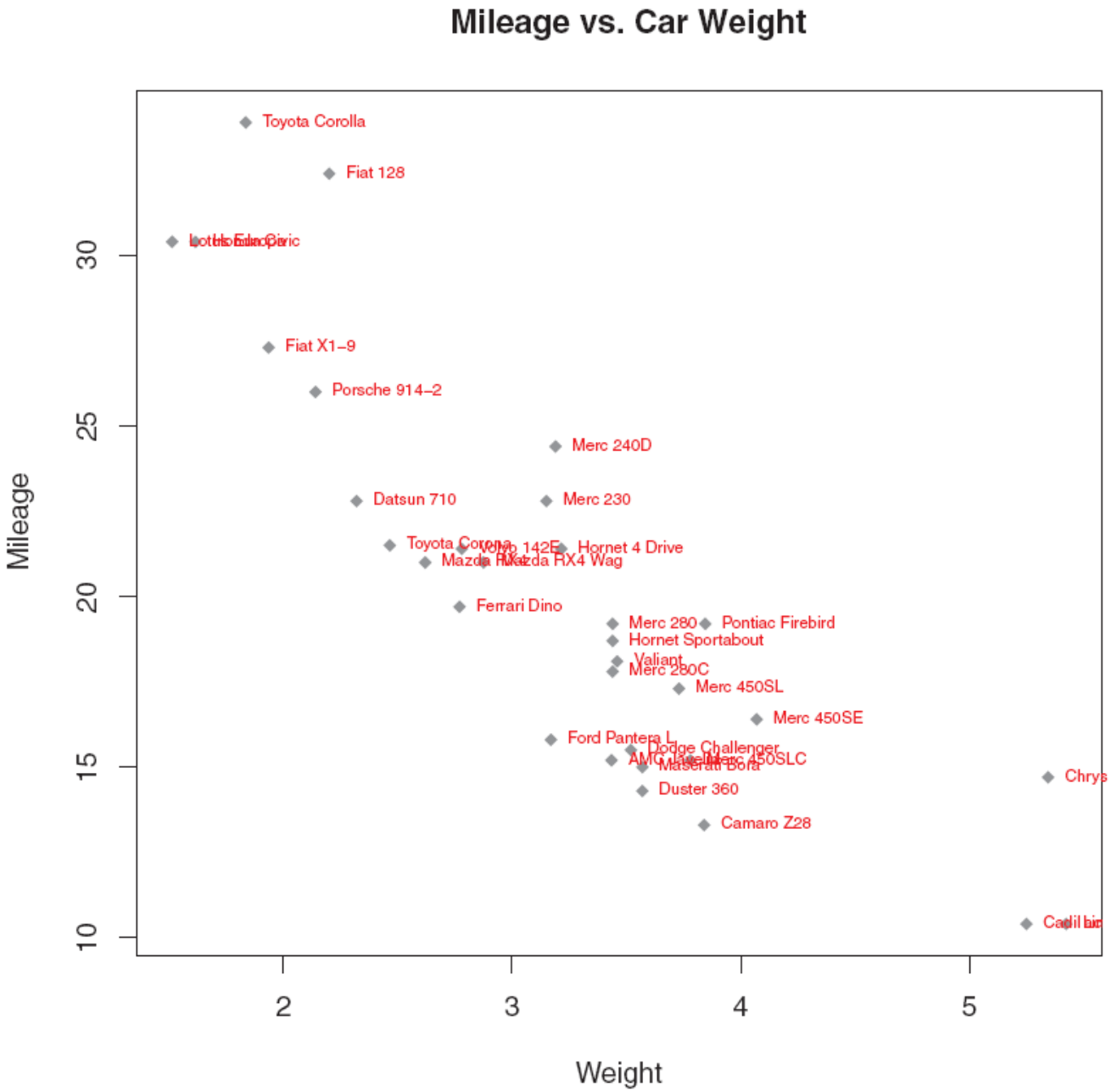
Функция	Описание
`location`	Положение можно указывать в виде x,y координат. Или же текст можно разместить на диаграмме в интерактивном режиме при помощи мышки, указав в качестве положения `locator(1)`
`pos`	Положение относительно точки, указанной предыдущим параметром. 1 – снизу, 2 – слева, 3 – сверху, 4 – справа. Если вы указали значения этого параметра, можно также использовать параметр `offset=` (в процентах от ширины буквы).
`side`	Указывает, на каком поле размещать текст, где 1 – нижнее, 2 – левое, 3 – верхнее, 4 – правое. При помощи параметра `line=` можно указать номер строки на полях (начиная с 0, ближайшей к диаграмме). Также можно указать `adj=0` для выравнивания по левому/нижнему краю или `adj=1` для выравнивания по верхнему/правому краю.

Другие часто используемые аргументы – это `cex`, `col` и `font` (для определения размера, цвета и стиля шрифта соответственно).

Функция `text()` обычно используется для того, чтобы подписывать точки на диаграмме и чтобы добавлять другие надписи. Необходимо указать положение надписей, как набор x,y-координат, и сами надписи в виде текстового вектора. Векторы x,y-координат и надписей должны иметь одинаковую длину. Ниже приведен пример, полученная диаграмма представлена на рис. 3.11.

```
attach(mtcars)
plot(wt, mpg,
     main="Mileage vs. Car Weight",
     xlab="Weight", ylab="Mileage",
     pch=18, col="blue")
text(wt, mpg,
     row.names(mtcars),
     cex=0.6, pos=4, col="red")
detach(mtcars)
```

Рис. 3.11. Пример диаграммы рассеяния (зависимость расхода топлива – mileage от веса машины – weight) с подписанными точками (марка машины)



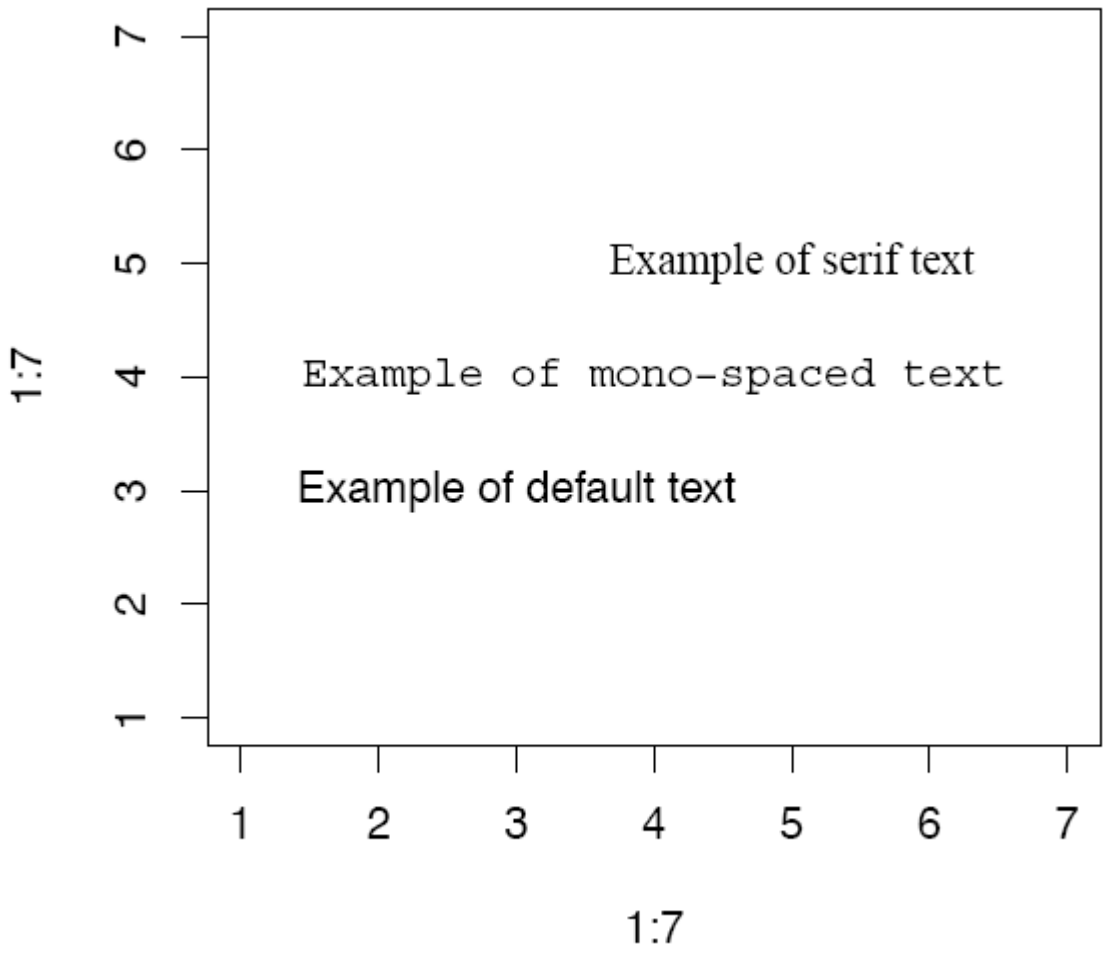
Здесь мы нанесли на диаграмму расход топлива в зависимости от веса машины для 32 марок автомобилей, представленных в таблице данных `mtcars`. Функция `text()` использована, чтобы добавить названия марок машин справа от каждой точки. Подписи уменьшены на 40% по сравнению с размером по умолчанию и выделены красным цветом.

В качестве второго примера приведен программный код, который можно использовать для демонстрации разных семейств шрифтов:

```
opar <- par(no.readonly=TRUE)
par(cex=1.5)
plot(1:7,1:7,type="n")
text(3,3,"Example of default text")
text(4,4,family="mono","Example of mono-spaced text")
text(5,5,family="serif","Example of serif text")
par(opar)
```

Результат, полученный в операционной системе Windows, показан на рис. 3.12. Здесь функция `par()` применена, чтобы увеличить размер шрифта для более удачного отображения текста.

Рис. 3.12. Примеры семейств шрифтов в операционной системе Windows



Эта диаграмма будет выглядеть по-разному в разных операционных системах, поскольку в них простой, моноширинный и серифный текст ассоциированы с разными семействами шрифтов. Как она выглядит на вашем компьютере?

Подписи с математическими символами

Наконец, вы можете размещать на диаграммах математические символы и формулы, используя правила, сходные с теми, что применяются в TEX. Подробная информация и примеры даны в `help(plotmath)`. Вы можете также попробовать ввести `demo(plotmath)`, чтобы увидеть, как это работает. Часть этой демонстрации возможностей функции представлена на рис. 3.13. Функция может быть использована, чтобы добавлять математические символы в названия, подписи осей или надписи внутри диаграммы или на ее полях.

Рис. 3.13. Часть результатов, которые создает функция `demo(plotmath)`

Арифметические операторы		Корни	
x + y	$x + y$	sqrt(x)	$\sqrt{x}$
x – y	$x - y$	sqrt(x, y)	$\sqrt[y]{x}$
x * y	xy	Соотношения	
x/y	x/y	x == y	$x = y$
x %+–% y	$x \pm y$	x != y	$x \uparrow y$
x%/%y	$x \surd y$	x < y	$x < y$
x %*% y	$x \times y$	x <= y	$x \prime\prime y$
x %.% y	$x \cdot y$	x > y	$x > y$
–x	$-x$	x >= y	$x \geq y$
+x	$+x$	x %~~% y	$x \oplus y$
Под/надстрочные индексы		x %=~% y	$x \cong y$
x[i]	x_i	x %==% y	$x \equiv y$
x^2	x^2	x %prop% y	$x \propto y$
Размещение		Гарнитура текста	
x * y	xy	plain(x)	x
paste(x, y, z)	xyz	italic(x)	*x*
Перечни		bold(x)	**x**
list(x, y, z)	x, y, z	bolditalic(x)	***x***
		underline(x)	<u>x</u>

Часто можно лучше разобраться в данных, сравнивая одновременно несколько диаграмм. Так что мы закончим эту главу обзором того, как поместить несколько диаграмм на один рисунок.

3.5. Объединение диаграмм

В R очень легко объединить несколько диаграмм в одну общую, используя функции `par()` или `layout()`. На этом этапе не нужно обращать внимание на типы объединяемых диаграмм, сейчас важно понять общие способы, которые используются для их объединения. Создание и интерпретация каждого типа диаграмм подробно рассмотрены в следующих главах.

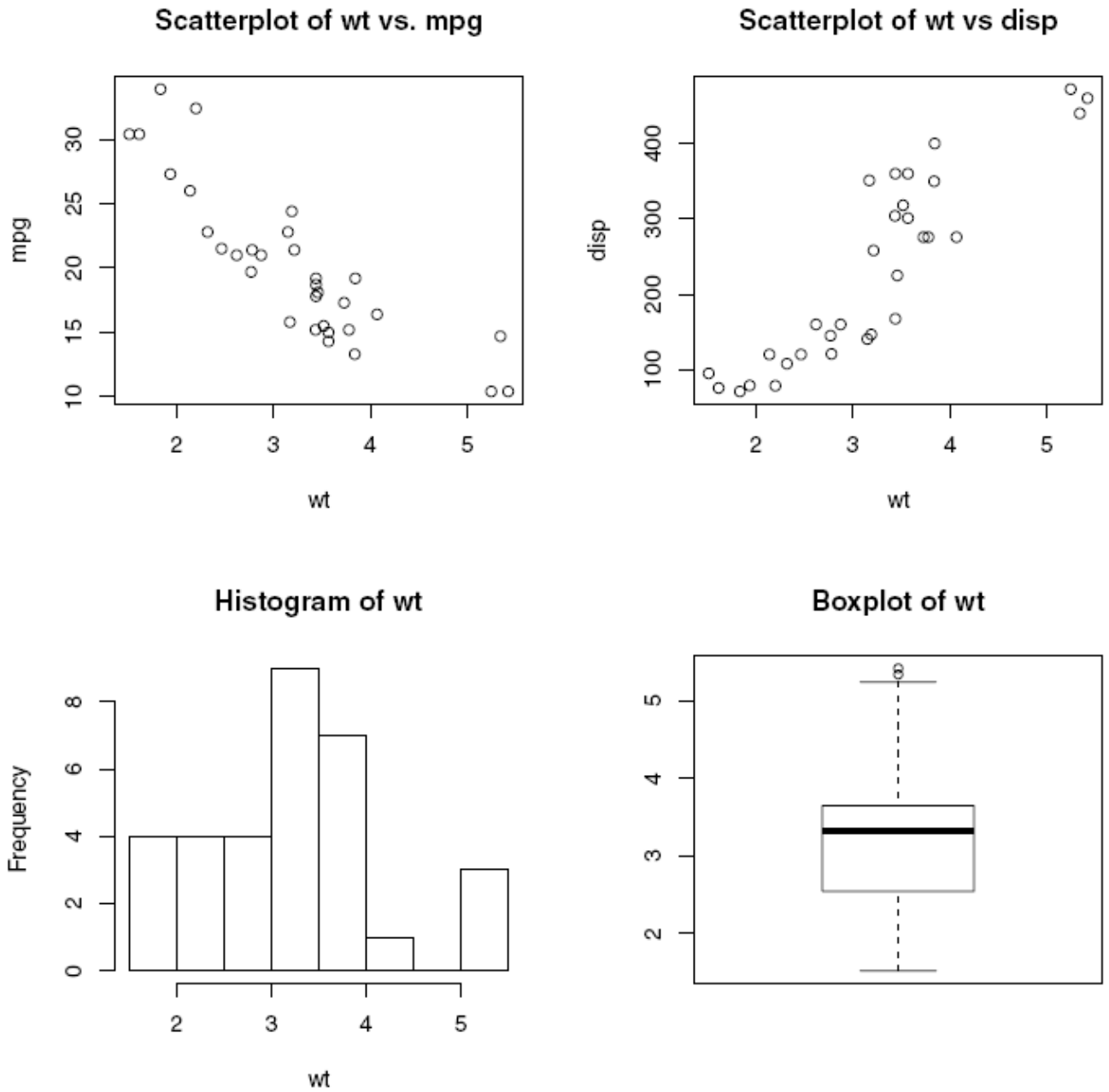
Вы можете добавить графический параметр `mfrow=c(`*`nrows, ncols`*`)` в функцию `par()`, чтобы создать матрицу из диаграмм размером *nrows*×*ncols*, которая будет заполнена по рядам. Для того чтобы заполнить эту матрицу по столбцам, нужно использовать параметр `mfcol=c(`*`nrows, ncols`*`)`.

Например, этот программный код позволяет создать четыре диаграммы и расположить их в две строки и два столбца:

```
attach(mtcars)
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2))
plot(wt,mpg, main="Scatterplot of wt vs. mpg")
plot(wt,disp, main="Scatterplot of wt vs disp")
hist(wt, main="Histogram of wt")
boxplot(wt, main="Boxplot of wt")
par(opar)
detach(mtcars)
```

Результат приведен на рис. 3.14.

Рис. 3.14. Четыре диаграммы, объединенные в одну, при помощи команды



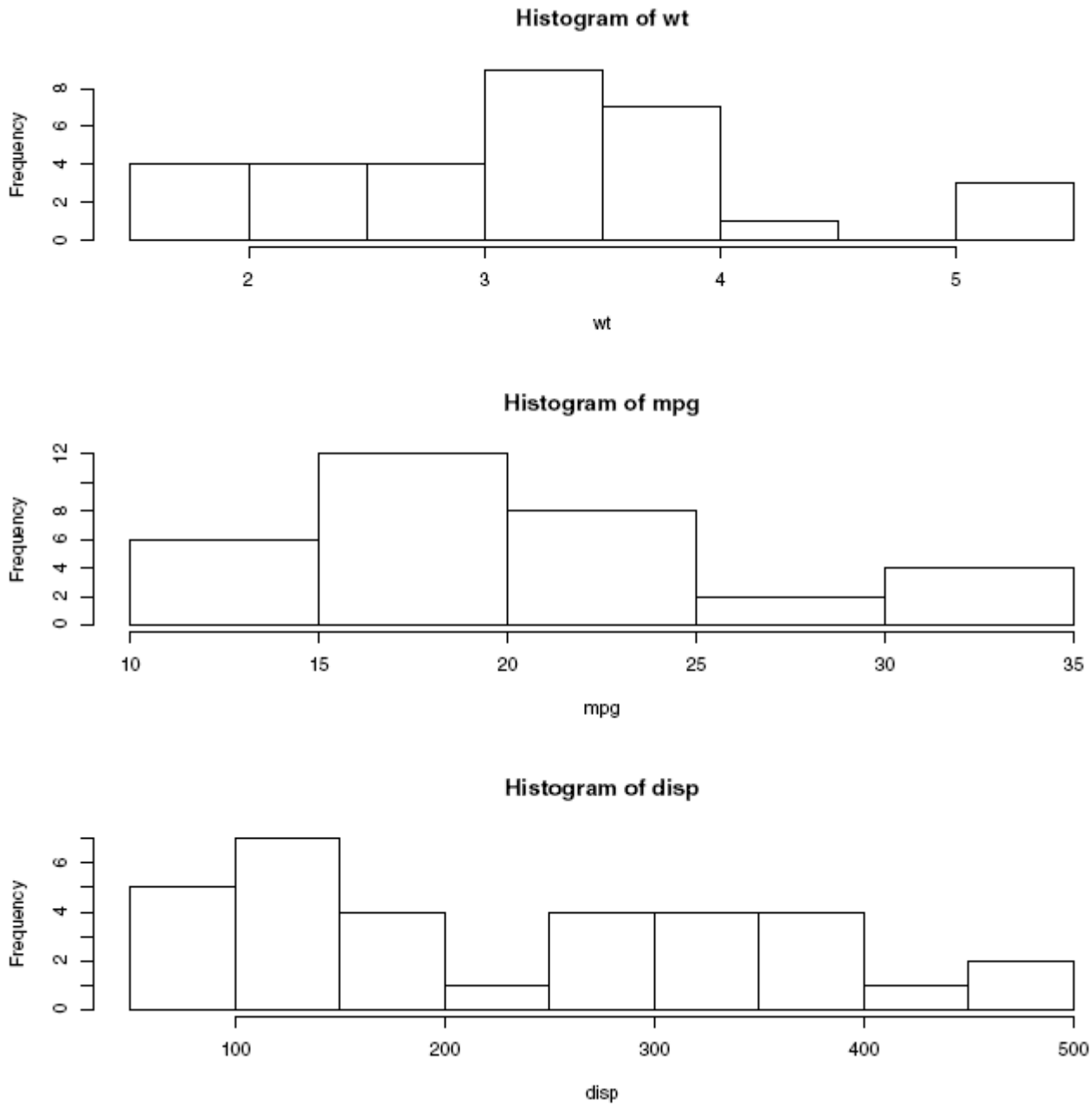

```
par(mfrow=c(2,2))
```

В качестве второго примера давайте расположим три диаграммы в три ряда и один столбец. Вот необходимый программный код:

```
attach(mtcars)
opar <- par(no.readonly=TRUE)
par(mfrow=c(3,1))
hist(wt)
hist(mpg)
hist(disp)
par(opar)
detach(mtcars)
```

Получившаяся диаграмма представлена на рис. 3.15. Обратите внимание на то, что функция высокого уровня `hist()` по умолчанию создает название гистограммы (используйте параметр `main="″`, чтобы предотвратить его появление или `ann=FALSE`, чтобы подавить все заголовки и подписи).

Рис. 3.15. Диаграмма, на которой три гистограммы объединены при помощи команды `par(mfrow=c(3,1))`

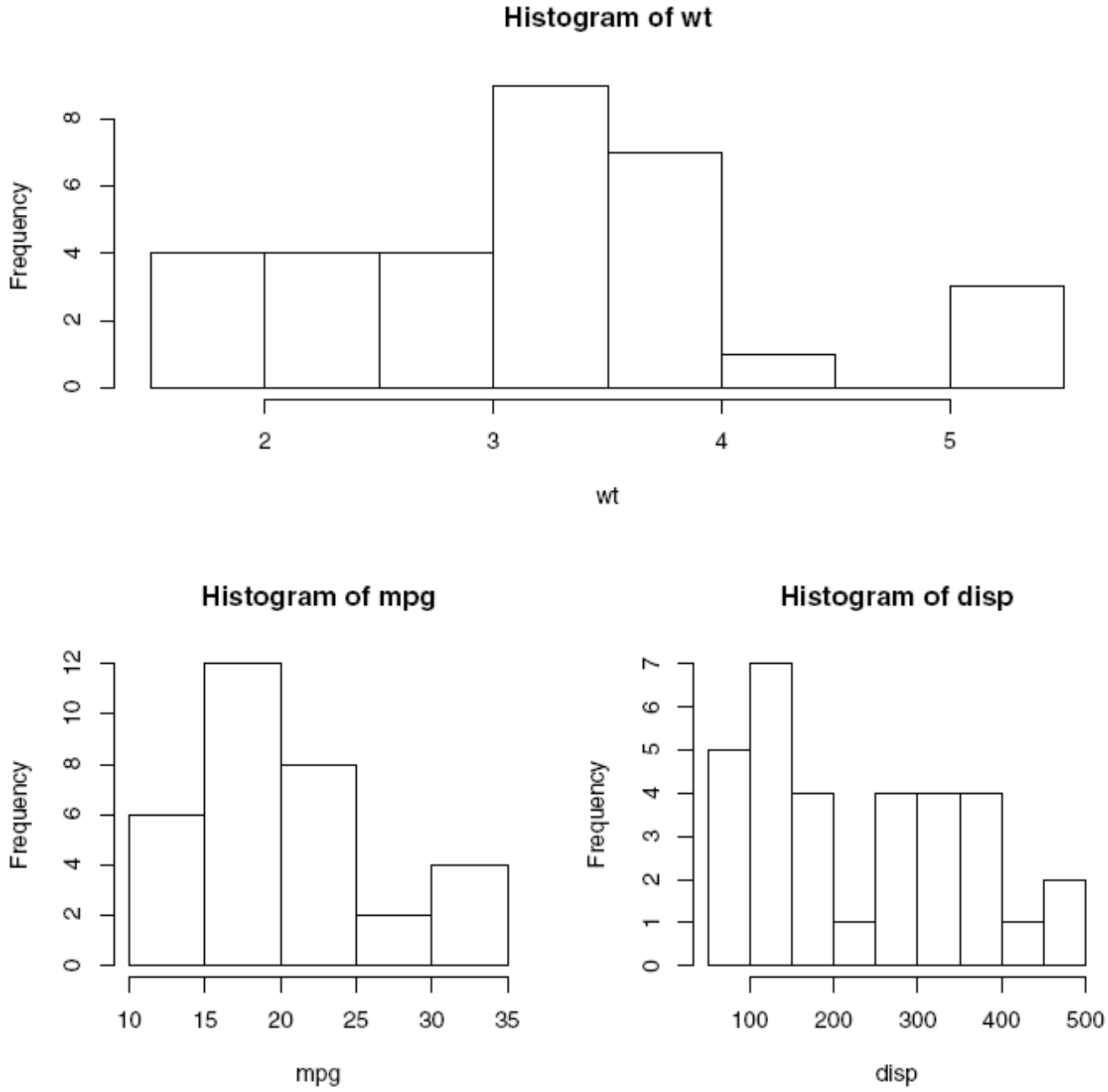


Функция `layout()` имеет формат применения `layout(`*`mat`*`)`, где *`mat`* – это матрица, в которой указаны положения нескольких совмещаемых диаграмм. При помощи приведенного программного кода одна диаграмма расположена в первом ряду, а две – во втором:

```
attach(mtcars)
layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE))
hist(wt)
hist(mpg)
hist(disp)
detach(mtcars)
```

получившаяся диаграмма представлена на рис. 3.16.

Рис. 3.16. Три диаграммы, объединенные в одну, при помощи функции `layout()` без указания ширины
диаграмм



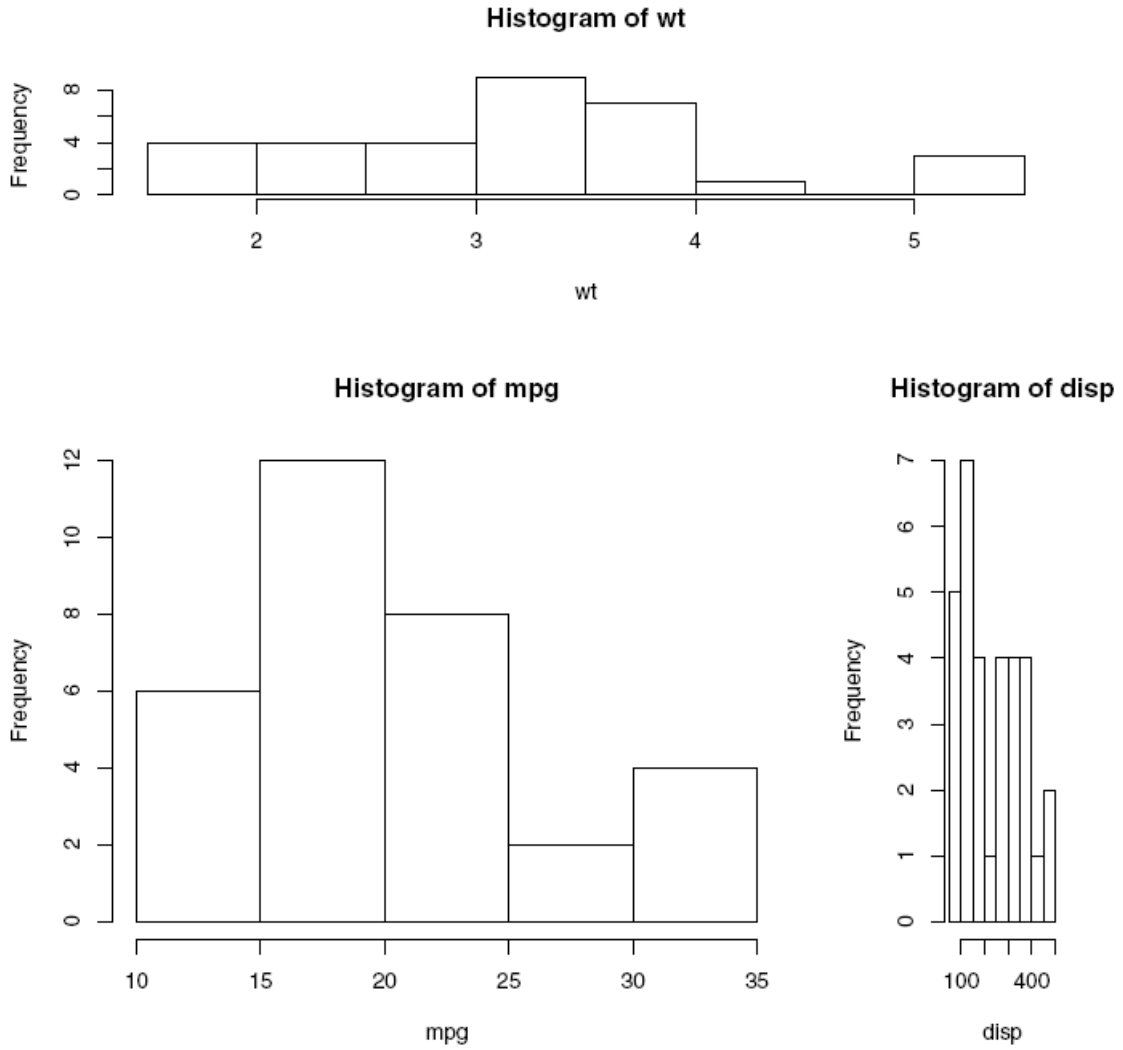
Для указания размера каждой диаграммы можно использовать дополнительные аргументы функции `layout()`:`widths=` и `heights=`. Эти аргументы нужно применять в виде
`widths=` вектор значений ширины каждого столбца
`heights=` вектор значений высоты каждой строчки.
Относительные размеры назначаются при помощи чисел. Абсолютные размеры (в сантиметрах) назначаются при помощи функции `1cm()`.

Приведенный ниже программный код также позволяет расположить одну диаграмму в первом ряду и две – во втором. Однако теперь высота диаграммы в первом ряду составляет одну треть от высоты диаграмм во втором ряду. Кроме того, ширина правой нижней диаграммы в четыре раза меньше, чем ширина левой нижней диаграммы:

```
attach(mtcars)
layout(matrix(c(1, 1, 2, 3), 2, 2, byrow = TRUE),
       widths=c(3, 1), heights=c(1, 2))
hist(wt)
hist(mpg)
hist(disp)
detach(mtcars)
```

Полученная диаграмма представлена на рис. 3.17.

Рис. 3.17.Три диаграммы, объединенные в одну, при помощи функции `layout()` с указанием ширины диаграмм.



Как вы можете видеть, функция `layout()` дает вам полный контроль над числом и расположением диаграмм в итоговом изображении, а также позволяет определять их относительные размеры. Для получения более полной информации читайте `help(layout)`.

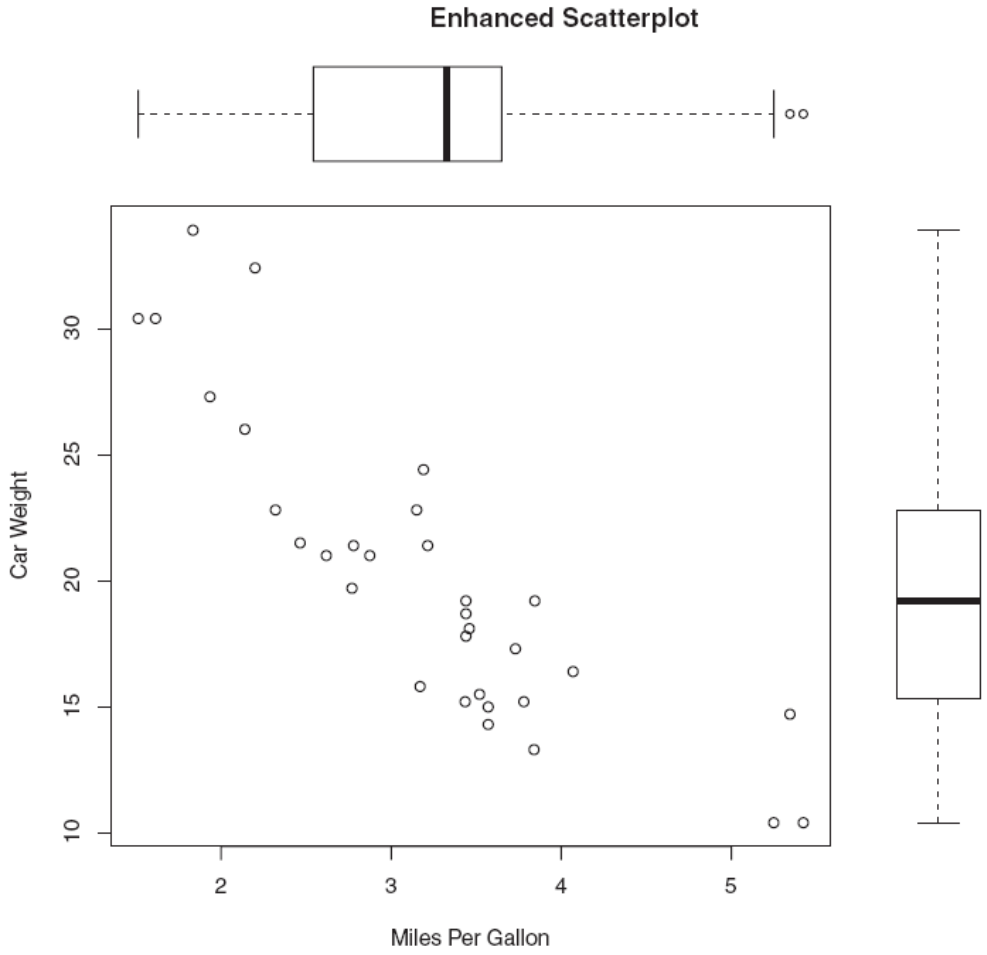
3.5.1. Полный контроль над расположением диаграмм

Иногда бывает нужно совместить и наложить несколько диаграмм, чтобы создать одну нужную. Для этого нужно тщательно контролировать расположение диаграмм. Этого можно достичь при помощи графического параметра `fig=`. В приведенном ниже программном коде две диаграммы типа ящик с усами добавлены к диаграмме рассеяния, в результате чего получилась одна усовершенствованная диаграмма (она показана на рис. 3.18).

Программный код 3.4. Полный контроль над расположением диаграмм

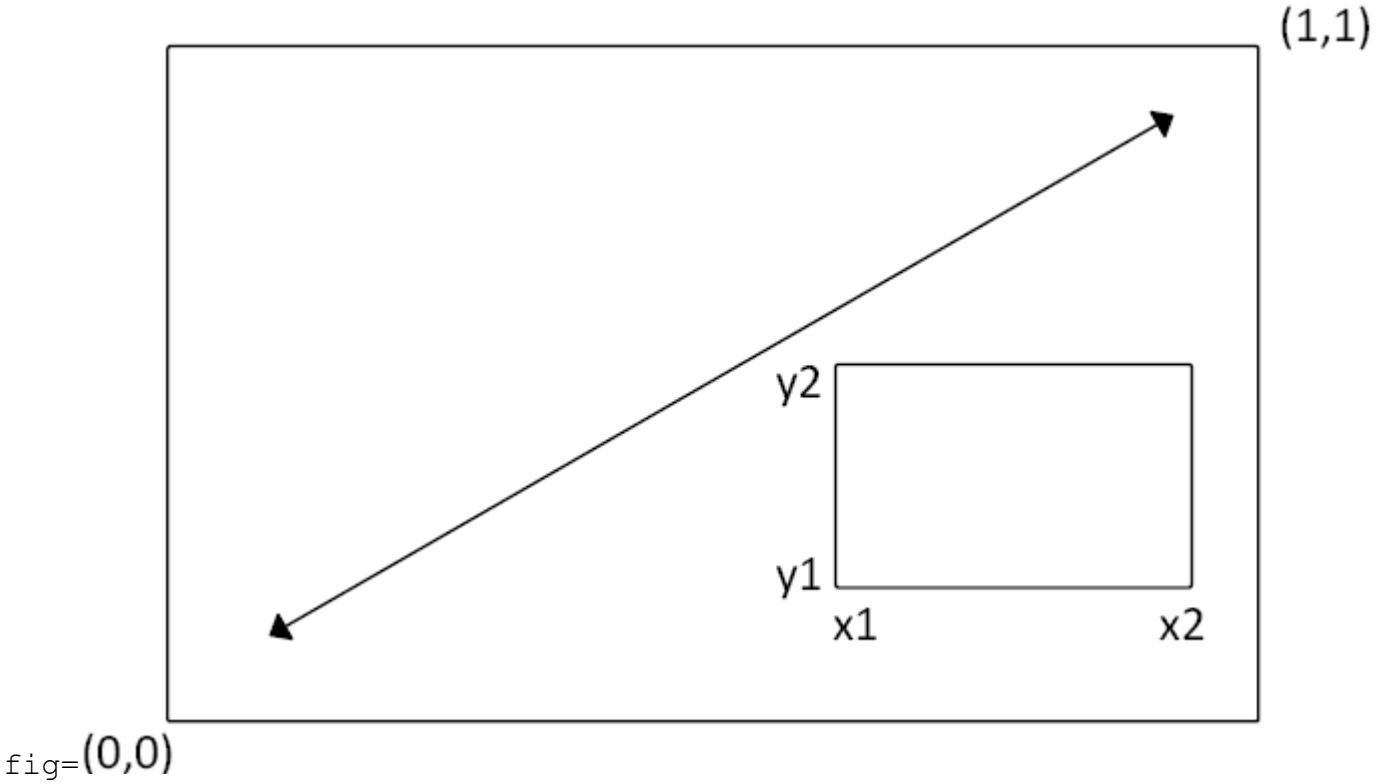
```
opar <- par(no.readonly=TRUE)
par(fig=c(0, 0.8, 0, 0.8))   ← Построение диаграммы рассеяния
plot(mtcars$wt, mtcars$mpg,
     xlab="Miles Per Gallon",
     ylab="Car Weight")
par(fig=c(0, 0.8, 0.55, 1), new=TRUE)
boxplot(mtcars$wt, horizontal=TRUE, axes=FALSE) ← Добавление «ящика с усами» сверху
par(fig=c(0.65, 1, 0, 0.8), new=TRUE)   ← Добавление «ящика с усами» справа
boxplot(mtcars$mpg, axes=FALSE)
mtext("Enhanced Scatterplot", side=3, outer=TRUE, line=-3)
par(opar)
```

Рис. 3.18. Диаграмма рассеяния с двумя ящиками с усами на полях



Для того чтобы понять, как была сделана эта диаграмма, представьте, что ее нижний левый угол имеет координаты 0,0, а верхний правый – 1,1 (рис. 3.19). Формат применения параметра `fig=` – это числовой вектор вида `c(x1, x2, y1, y2)`.

Рис. 3.19. Определение положения диаграммы при помощи графического параметра



Первое применение параметра `fig=` позволяет разместить диаграмму рассеяния на пространстве от 0 до 0.8 по оси x и от 0 до 0.8 по оси y. Верхний ящик с усами занимает пространство от 0 до 0.8 по оси x и от 0.55 до 0.8 по оси y. Верхний ящик с усами занимает пространство от 0.65 до 1 по оси x и от 0 до 0.8 по оси y. Параметр `fig=` строит новую диаграмму, поэтому при добавлении диаграммы на уже имеющуюся добавляйте параметр `new=TRUE`.

Я выбрал значение 0.55, а не 0.8, чтобы разместить верхний ящик с усами поближе к диаграмме рассеяния. По этой же причине я выбрал значение 0.65 для правого ящика с усами. Приходится пробовать разные варианты, чтобы добиться правильного взаимного расположения диаграмм.

Примечание. *Необходимая для отдельных диаграмм площадь может различаться в разных графических устройствах. Если появляется сообщение "Error in plot.new(): figure margins too large" (ошибка в plot.new(): поля диаграммы слишком велики), попробуйте изменить размеры отдельных диаграмм.*

3.6. Резюме

В этой главе мы узнали, как создавать диаграммы и сохранять их в разных форматах. Бо́льшая часть главы была посвящена тому, как изменять диаграммы, автоматически создаваемые R, чтобы они стали более полезными или привлекательными. Вы узнали, как изменять оси, шрифты, символы, линии и цвета на диаграммах, а также как добавлять названия, подписи, надписи, текст, легенды и опорные линии. Вы научились задавать размер диаграммы и полей, а также объединять несколько диаграмм в одну более наглядную.

В этой главе мы сконцентрировались на общих подходах, которые можно применить к любым диаграммам (кроме панельных диаграмм, обсуждаемых в главе 16). В дальнейших главах мы рассмотрим разные типы диаграмм. Например, в главе 7 рассмотрены способы графического

отображения одной переменной. Изображение взаимосвязей между переменными будет описано в главе 11. В главе 16 мы поговорим о панельной графике (диаграммах, которые позволяют отобразить взаимосвязи между переменными для каждого значения других переменных) и интерактивной графике. Интерактивные диаграммы позволяют при помощи мышки исследовать изображенные взаимосвязи в реальном времени.

В других главах мы будем обсуждать способы визуализации данных, которые особенно полезны для рассматриваемых статистических методов. Диаграммы – это основная часть современного анализа данных, и я приложу все усилия, чтобы приводить их при обсуждении каждого статистического метода.

В предыдущей главе мы обсуждали разные методы ввода и импорта данных в R. К сожалению, в реальности данные редко можно использовать в том виде, в котором вы их изначально получили. В следующей главе мы рассмотрим способы преобразования и обработки данных, которые применяются, чтобы сделать данные пригодными для анализа.