Глава 4. Основы управления данными

В этой главе:

- работаем с датами и пропущенными значениями;
- понимаем, как преобразовывать один тип данных в другой;
- создаем и перекодируем переменные;
- сортируем, объединяем и разделяем наборы данных;
- выбираем и исключаем из анализа переменные.

Во второй главе мы обсуждали разнообразные методы импорта данных в R. К сожалению, преобразование наших данных в прямоугольную форму матрицы или таблицы данных – это только первый шаг на пути их подготовки к анализу. Перефразируя капитана Кирка из серии “Вкус Армагеддона” сериала “Звездный путь” (и раз и навсегда оправдывая мое помешательство), “Данные – это запутанная штука – очень и очень запутанная штука”. В моей собственной работе не менее 60% времени, отведенного на анализ данных, я трачу на подготовку данных к этому анализу. Я выйду за пределы частного опыта и скажу, что это, вероятно, в той или иной степени справедливо для большинства исследователей. Давайте рассмотрим пример.

4.1. Рабочий пример

Одна из задач, которую я решаю по долгу службы – это как мужчины и женщины различаются по стилю руководства организациями. Обычные вопросы могут быть следующими:

- различаются ли мужчины и женщины на руководящих позициях по степени лояльности к вышестоящему начальству?
- зависит ли это от страны, или выявленные гендерные различия носят универсальный характер?

Один из способов ответить на эти вопросы – взять начальников из разных стран и ранжировать их менеджеров по степени лояльности, используя вопросы вроде этого:
этот менеджер спрашивает мое мнение перед принятием кадровых решений
1 – абсолютно не согласен;
2 – не согласен;
3 – бывает по-разному;
4 – согласен;
5 – полностью согласен.

В результате можно получить данные вроде тех, что представлены в табл. 4.1. Каждая строка – это оценка, которую дал менеджеру его или ее начальник.

Таблица 4.1. Гендерные различия в стиле руководства

Maneger (менеджер)	Date (дата)	Country (страна)	Gender (пол)	Age (возраст)	q1	q2	q3	q4	q5
1	10/24/08	US	M	32	5	4	5	5	5
2	10/28/08	US	F	45	3	5	2	5	5
3	10/01/08	US	F	25	3	5	5	5	2
4	10/12/08	US	M	39	3	3	4		
5	05/01/09	US	F	99	2	2	1	2	1

Здесь, каждый менеджер оценен своим начальником по пяти параметрам (q1 – q5), связанным с лояльностью к вышестоящим сотрудникам. Например, менеджер 1 – это 32-х летний мужчина,

работающий в США, который склонен подчиняться начальству, тогда как менеджер 5 – это женщина неизвестного возраста (99, вероятно, означает отсутствие информации), работающая в США и недостаточно лояльная к начальству. В таблице также указана дата проведения опроса.

Хотя набор данных может состоять из десятков переменных и тысяч наблюдений, мы оставили только десять столбцов и пять строк, чтобы упростить примеры. Кроме того, мы ограничили число вопросов, характеризующих лояльность менеджеров к начальству, пятью. В реальном исследовании обычно используют 10–20 вопросов, чтобы получить более надежные и обоснованные результаты. Вы можете создать таблицу данных, представленную в табл. 4.1, при помощи следующего программного кода:

Программный код 4.1. Создание таблицы данных `leadership`

```
manager <- c(1, 2, 3, 4, 5)
date <- c("10/24/08", "10/28/08", "10/1/08", "10/12/08", "5/1/09")
country <- c("US", "US", "UK", "UK", "UK")
gender <- c("M", "F", "F", "M", "F")
age <- c(32, 45, 25, 39, 99)
q1 <- c(5, 3, 3, 3, 2)
q2 <- c(4, 5, 5, 3, 2)
q3 <- c(5, 2, 5, 4, 1)
q4 <- c(5, 5, 5, NA, 2)
q5 <- c(5, 5, 2, NA, 1)
leadership <- data.frame(manager, date, country, gender, age,
                         q1, q2, q3, q4, q5, stringsAsFactors=FALSE)
```

Для того чтобы ответить на интересующие нас вопросы, нужно сначала решить несколько проблем управления данными. Вот их неполный список:

- нужно объединить пять параметров оценки (от q1 до q5), чтобы для каждого менеджера получить единый усредненный показатель лояльности к начальству;
- при анкетировании респонденты часто пропускают вопросы. Например, начальник, который оценивал менеджера 4, не ответил на вопросы 4 и 5. Нам потребуется как-то справиться с неполными данными. Также нам нужно будет обозначить значения вроде 99 как отсутствующие;
- набор данных может содержать сотни переменных, но нас, возможно, заинтересуют только некоторые из них. Для упрощения ситуации у нас может появиться желание создать новый набор данных, состоящий только из этих переменных;
- предыдущие исследования показали, что отношение к начальству может меняться с возрастом. Для того чтобы проверить это, мы можем захотеть перекодировать значения возраста в возрастные группы (например, молодые, люди среднего возраста и старшего возраста);
- отношение к начальству может меняться со временем. Мы можем захотеть сосредоточиться на периоде последнего глобального финансового кризиса. Для этого можно ограничиться данными, собранными, скажем, с 1 января по 31 декабря 2009 года.

В этой главе мы разберем каждую из перечисленных проблем, вместе с остальными главными задачами управления данными, такими как объединение и сортировка наборов данных. Затем, в главе 5, мы обратимся к более сложным темам.

4.2. Создание новых переменных

Обычно при анализе данных вам требуется создавать новые переменные и преобразовывать существующие. Это достигается при помощи присвоений в таком формате:

переменная <- выражение

Слово *выражение* означает самые разные операторы и функции. В табл. 4.2 перечислены арифметические операторы, которые используются при составлении формул в R.

Табл. 4.2. Арифметические операторы

Оператор	Описание
+	Сложение
–	Вычитание
*	Умножение
/	Деление
^ или **	Возведение в степень
x%%y	Остаток от деления x на y: 5%%2=1
x%/%y	Целая часть при делении x на y: 5%/%2=2

Представим, что у вас есть таблица данных под названием `mydata` с переменными `x1` и `x2`, а вы хотите создать новую переменную `sumx`, которая представляет собой сумму этих двух переменных, и новую переменную `meanx` – среднее арифметическое этих двух переменных. Если вы используете программный код

```
sumx  <-  x1 + x2
meanx <- (x1 + x2)/2
```

вы получите сообщение об ошибке, поскольку R «не знает» о том, что `x1` и `x2` – это часть таблицы данных `mydata`. Если вы используете вместо этого такой программный код

```
sumx  <-  mydata$x1 + mydata$x2
meanx <- (mydata$x1 + mydata$x2)/2
```

он будет выполнен, но в результате у вас будет исходная таблица данных (`mydata`) и два отдельных вектора (`sumx` и `meanx`). Это, скорее всего, не то, что вы хотели получить. Вы же хотели сделать новые переменные частью исходной таблицы данных. В приведенном ниже программном коде представлены три альтернативных способа достичь этой цели. Вы можете выбрать любой на ваше усмотрение, результат от этого не изменится.

Программный код 4.2. Создание новых переменных

```
mydata<-data.frame(x1 = c(2, 2, 6, 4),
                   x2 = c(3, 4, 2, 8))
mydata$sumx  <-  mydata$x1 + mydata$x2
mydata$meanx <- (mydata$x1 + mydata$x2)/2

attach(mydata)
mydata$sumx  <-  x1 + x2
mydata$meanx <- (x1 + x2)/2
detach(mydata)

mydata <- transform(mydata,
                    sumx  =  x1 + x2,
                    meanx = (x1 + x2)/2)
```

Лично я предпочитаю третий метод, с применением функции `transform()`. Она упрощает создание необходимого числа новых переменных и сохраняет результат в виде таблицы данных.

4.3. Перекодировка переменных

При перекодировке новые значения переменной определяются значениями этой и/или других переменных. Например, вы можете захотеть:

- преобразовать непрерывные данные в категориальные;
- заменить ошибочные данные правильными значениями;
- создать переменную типа сдал/не сдал на основе экзаменационных баллов.

Для перекодировки данных можно использовать один из знаков логических операций (табл. 4.3), то есть выражений, которые возвращают значения `TRUE/FALSE` (правда/ложь).

Табл. 4.3. Знаки логических операций

Знак	Описание
`<`	Меньше чем
`<=`	Меньше или равно
`>`	Больше чем
`>=`	Больше или равно
`==`	Тождественно равно
`!=`	Не равно
`!`*x*	Не *x*
x`|`*y*	*x* или *y*
x`&`*y*	*x* и *y*
`isTRUE(`*x*`)`	Проверяет, выполняется ли *x*

Предположим, что вы хотите перекодировать возраст менеджеров в нашем наборе данных из непрерывной переменной `age` в категориальную переменную `agecat` (`Young`, `Middle Aged`, `Elder` – молодой, средних лет, старшего возраста). Сначала нужно закодировать значение 99 как отсутствующее:

```
leadership$age[leadership$age  == 99] <- NA
```

Присвоение вида `переменная[условие] <- выражение` будет происходить только в том случае, если условие выполняется.

После того как пропущенные значения выявлены, для создания переменной `agecat` можно использовать следующий программный код:

```
leadership$agecat[leadership$age  > 75]   <- "Elder"
leadership$agecat[leadership$age >= 55 &
                  leadership$age <= 75]   <- "Middle Aged"
leadership$agecat[leadership$age  < 55]   <- "Young"
```

Названия таблицы данных входят в `leadership$agecat`, чтобы новые переменные сохранялись в этой таблице. Мы решили, что средний возраст варьирует от 55 до 75 лет, так что я не буду чувствовать себя слишком старым. Обратите внимание на то, что если бы мы сначала не закодировали значение 99 как пропущенное, менеджер 5 был бы ошибочно отнесен к категории людей старшего возраста.

Этот программный код можно записать в более компактном виде:

```
leadership <- within(leadership,{
                     agecat <- NA
                     agecat[age > 75]              <- "Elder"
                     agecat[age >= 55 & age <= 75] <- "Middle Aged"
                     agecat[age < 55]              <- "Young" })
```

Функция `within()` сходна с функцией `with()` (раздел 2.2.4), однако она позволяет модифицировать таблицу данных. Сначала создается переменная `agecat`, состоящая из пропущенных значений. Затем ее значения последовательно заменяются на соответствующие возрастные категории согласно условиям в квадратных скобках. Помните о том, что `agecat` – это текстовая переменная, скорее всего, вам захочется преобразовать ее в упорядоченный фактор, используя указания из раздела 2.2.5.

В нескольких пакетах содержатся полезные функции для перекодировки, в особенности просто перекодировать числовые и текстовые векторы и факторы при помощи функции `recode()` из пакета `car`. В пакете `doBy` представлена еще одна популярная функция `recodevar()`. Наконец, в R реализована функция `cut()`, которая разделяет числовые переменные на интервалы, возвращая фактор.

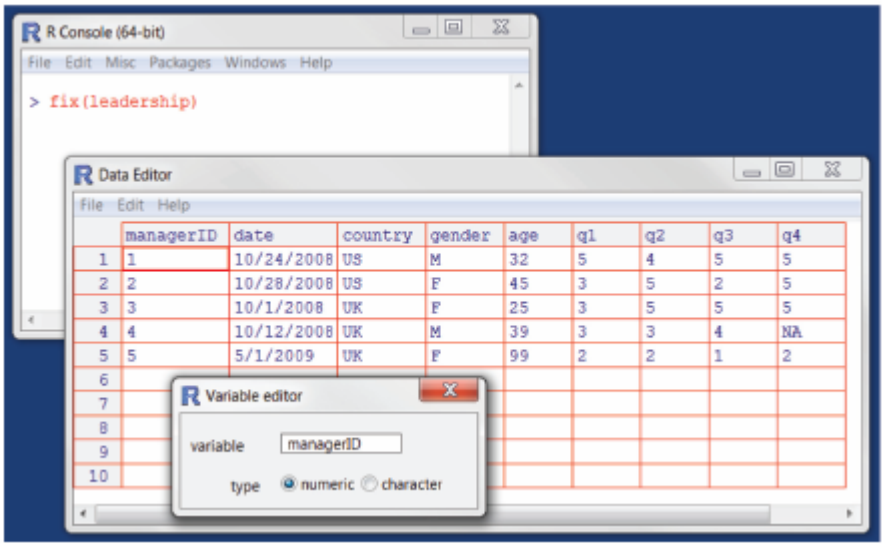
4.4. Переименование переменных

Если вас не устраивают названия переменных, вы можете поменять их в интерактивном режиме или при помощи программного кода. Допустим, вы хотите изменить название переменной `manager` на `managerID` и `date` – на `testDate`. Можно использовать команду

```
fix(leadership)
```

чтобы открыть интерактивный редактор. Затем нужно выбрать названия переменных и изменить их в диалоговом режиме (рис. 4.1).

Рис. 4.1. Интерактивное переименование переменных с использованием функции `fix()`



Для изменения названий переменных при помощи программного кода можно воспользоваться функцией `rename()` из пакета `reshape`. Формат применения этой функции таков:

```
rename(таблица_данных, c(старое_название="новое_название",
старое_название="новое_название",…))
```

Вот пример:

```
library(reshape)
leadership <- rename(leadership,
                     c(manager="managerID", date="testDate")
)
```

Пакет reshape не установлен по умолчанию, так что вам потребуется установить его перед первым использованием при помощи команды install.packages("reshape"). В этом пакете реализовано несколько эффективных функций для изменения структуры наборов данных. Мы испробуем некоторые из них в главе 5.

Наконец, переименовать переменные можно при помощи функции names(). Например, команда

```
names(leadership)[2] <- "testDate"
```

переименовывает date в testDate, как это видно из следующего программного кода:

```
> names(leadership)
 [1] "manager" "date"    "country" "gender"  "age"     "q1"      "q2"
 [8] "q3"      "q4"      "q5"
> names(leadership)[2] <- "testDate"
> leadership
  manager testDate country gender age q1 q2 q3 q4 q5
1       1 10/24/08      US      M  32  5  4  5  5  5
2       2 10/28/08      US      F  45  3  5  2  5  5
3       3  10/1/08      UK      F  25  3  5  5  5  2
4       4 10/12/08      UK      M  39  3  3  4 NA NA
5       5   5/1/09      UK      F  99  2  2  1  2  1
```

Сходным образом, команда

```
names(leadership)[6:10] <- c("item1", "item2", "item3", "item4", "item5")
```

меняет названия переменных от q1 до q5 на названия от item1 до item5.

4.5. Пропущенные значения

При любых исследованиях данные скорее всего будут неполными из-за пропущенных вопросов, барахлящего оборудования или ошибок в кодировке данных. В R пропущенные данные обозначаются символом NA (not available – нет в наличии). Недопустимые значения (например, деление на 0) обозначаются как NaN (not a number – не является числом). В отличие от таких программ как SAS, в R используется одно и то же обозначение для пропущенных значений в текстовых и числовых данных.

В R есть несколько функций, предназначенных для выявления пропущенных значений. Функция is.na() позволяет проверить данные на наличие пропущенных значений. Предположим, что у нас имеется вектор:

```
y <- c(1, 2, 3, NA)
```

тогда команда

```
is.na(y)
```

возвращает вектор c(FALSE, FALSE, FALSE, TRUE).

Обратите внимание, как функция is.na() работает с объектом. Она возвращает объект такой же размерности, где ячейки заменены символом TRUE, если значение было пропущено, и FALSE – если оно не было пропущено. В программном коде 4.3 показано, что получится, если эту команду применить к нашей таблице данных leadership.

Программный код 4.3. Использование функции `is.na()`

```
> is.na(leadership[,6:10])
        q1    q2    q3    q4    q5
[1,] FALSE FALSE FALSE FALSE FALSE
[2,] FALSE FALSE FALSE FALSE FALSE
[3,] FALSE FALSE FALSE FALSE FALSE
[4,] FALSE FALSE FALSE  TRUE  TRUE
[5,] FALSE FALSE FALSE FALSE FALSE
```

В этом примере `leadership[,6:10]` ограничивает таблицу данных столбцами с 6 по 10, а функция `is.na()` выявляет пропущенные значения.

Примечание. *Пропущенные значения считаются несравнимыми, даже сами с собой. Это значит, что вы не можете использовать знаки операций сравнения для выявления пропущенных значений. Например, логическое выражение `myvar == NA` никогда не будет правдой. Вместо этого для выявления пропущенных значений вам придется использовать специальные функции вроде тех, что были описаны в этом разделе.*

4.5.1. Перекодировка значений в отсутствующие

Как уже было сказано в разделе 4.3, для перекодировки значений в отсутствующие можно использовать знак присвоения. В приведенном примере пропущенные значения возраста были закодированы как 99. Перед тем как анализировать такой набор данных, вам нужно обозначить значение 99 как отсутствующее (иначе среднее значение возраста для этой выборки будет далеким от реальности!). Этого можно достичь при помощи перекодировки переменной:

```
leadership$age[leadership$age == 99] <- NA
```

Любое значение возраста, равное 99, будет заменено на `NA`. Перед анализом данных всегда проверяйте, что все пропущенные значения закодированы соответственно, в противном случае результаты анализа будут лишены смысла.

4.5.2. Исключение пропущенных значений из анализа

После того как вы выявили пропущенные значения, их нужно каким-то образом удалить, прежде чем продолжать анализ данных. Это необходимо, потому что арифметические операции и функции, которые применяются к данным с пропущенными значениями, в качестве результата также будут выдавать отсутствующие значения. Для примера рассмотрим следующий программный код:

```
x <- c(1, 2, NA, 3)
y <- x[1] + x[2] + x[3] + x[4]
z <- sum(x)
```

И `y`, и `z` будут иметь значение `NA`, поскольку третий элемент вектора `x` отсутствует.

К счастью, у большей части числовых функций есть параметр `na.rm=TRUE`, который удаляет пропущенные значения перед вычислениями и позволяет применить функцию к оставшимся элементам:

```
x <- c(1, 2, NA, 3)
y <- sum(x, na.rm=TRUE)
```

Теперь y равен 6.

Когда вы применяете функции к неполным данным, узнайте при помощи справки (введя, например, `help(sum)`), как эти функции обрабатывают пропущенные значения. Функция `sum()` – это лишь одна из многих функций, которые мы рассмотрим в главе 5. Подобные функции позволяют с легкостью осуществлять разнообразные преобразования данных.

При помощи функции `na.omit()` можно избавиться от *всех* наблюдений с пропущенными данными. Эта функция удаляет все строки, в которых есть хотя бы одно пропущенное значение. Давайте применим эту функцию к нашей таблице `leadership`.

Программный код 4.4. Использование функции `na.omit()` для удаления неполных наблюдений

Таблица данных с пропущенными данными

```
> leadership                    ↓
  manager     date country gender age q1 q2 q3 q4 q5
1       1 10/24/08      US      M  32  5  4  5  5  5
2       2 10/28/08      US      F  40  3  5  2  5  5
3       3 10/01/08      UK      F  25  3  5  5  5  2
4       4 10/12/08      UK      M  39  3  3  4 NA NA
5       5 05/01/09      UK      F  99  2  2  1  2  1
> newdata <- na.omit(leadership)  Таблица, в которой остались только строки
> newdata                       ↙              без пропущенных данных
  manager     date country gender age q1 q2 q3 q4 q5
1       1 10/24/08      US      M  32  5  4  5  5  5
2       2 10/28/08      US      F  40  3  5  2  5  5
3       3 10/01/08      UK      F  25  3  5  5  5  2
5       5 05/01/09      UK      F  99  2  2  1  2  1
```

Строка с пропущенными значениями была удалена из таблицы `leadership` перед сохранением результатов в виде таблицы `newdata`.

Удаление всех наблюдений с пропущенными значениями (так называемое построчное удаление) – один из способов обработки неполных наборов данных. В том случае, когда пропущено лишь несколько значений, сосредоточенных в небольшом числе строк, построчное удаление – хороший способ решить проблему пропущенных значений. Однако если пропущенные значения рассеяны по всей таблице данных или находятся в нескольких переменных, построчное удаление может уничтожить заметную часть ваших данных. Мы познакомимся с несколькими более сложными способами работы с пропущенными данными в главе 15. Теперь давайте поговорим о датах.

4.6. Данные в формате даты

Даты обычно вводятся в R в виде текстовых строк, а затем переводятся формат даты и хранятся в числовом виде. Для этого преобразования используется функция `as.Date()` в формате `as.Date(x, "исходный_вид")`, где x – это текстовые данные, а `исходный_вид` указывает, в каком формате представлены даты (табл. 4.4).

Табл. 4.4. Форматы данных

Символ	Значение	Пример
`%d`	День в виде числа (0-31)	01-31
`%a`	Сокращенное название дня недели	Mon
`%A`	Полное название дня недели	Monday[1]
`%m`	Порядковый номер месяца (00-12)	00-12
`%b`	Сокращенное название месяца	Jan
`%B`	Полное название месяца	January[2]
`%y`	Две последние цифры года	07
`%Y`	Все четыре цифры года	2007

По умолчанию даты вводятся в формате “год-месяц-день”. Команда

```
mydates <- as.Date(c("2007-06-22", "2004-02-13"))
```

преобразует текстовые данные в даты, используя этот формат по умолчанию. А этот программный код

```
strDates <- c("01/05/1965", "08/16/1975"
dates <- as.Date(strDates, "%m/%d/%Y")
```

читает даты в формате “месяц-день-год”.

В нашей таблице данных о лояльности менеджеров к начальству даты представлены в формате “месяц-день-две последние цифры года”. Так что команды

```
myformat <- "%m/%d/%y"
leadership$date <- as.Date(leadership$date, myformat)
```

позволяют использовать заданный формат для того, чтобы преобразовать столбец с текстовыми значениями в даты. Как только вы преобразовали переменную в формат даты, ее можно анализировать и отображать графически, используя разнообразные аналитические подходы, описанные в следующих главах. Для создания временн**ы**х отметок вам особенно пригодятся две функции. Функция `Sys.Date()` возвращает сегодняшнее число, а функция `date()` возвращает текущее число и время. Я пишу эти строки 12 декабря 2010 года в 16:28. Так что выполнение этих команд приводит к следующему результату:

```
> Sys.Date()
[1] "2010-12-01"
> date()
[1] "Wed Dec 01 16:28:21 2010"
```

Для вывода дат в заданном формате и для экспорта составных частей дат можно использовать функцию `format(x, format="`*`формат_вывода`*`")`:

```
> today <- Sys.Date()
> format(today, format="%B %d %Y")
[1] "December 01 2010"
> format(today, format="%A")
[1] "Wednesday"
```

[1] *Понедельник. – Прим. пер.*
[2] *Январь. – Прим. пер.*

Функция `format()` преобразовывает аргумент (в нашем случае дату) в заданный формат вывода (составленный в нашем случае из обозначений, расшифрованных в табл. 4.4). Здесь самое важное то, что до выходных осталось всего два дня!

Даты хранятся в памяти программы в виде числа дней, прошедших с первого января 1970 года. Более ранние даты представлены отрицательными числами. Это значит, что с датами можно выполнять арифметические действия. Например, программный код

```
> startdate <- as.Date("2004-02-13")
> enddate   <- as.Date("2011-01-22")
> days      <- enddate - startdate
> days
Time difference of 2535 days
```

позволяет узнать, сколько дней прошло с 13 февраля 2004 года по 22 января 2011 года.

Наконец, вы можете использовать функцию `difftime()`, чтобы вычислять длину временного отрезка в секундах, минутах, часах, днях или неделях. Предположим, что я родился 12 октября 1956 года. Сколько времени прошло с той поры?

```
> today <- Sys.Date()
> dob   <- as.Date("1956-10-12")
> difftime(today, dob, units="weeks")
Time difference of 2825 weeks
```

Оказывается, мне 2825 недель. Кто знал? Вопрос на засыпку: в какой день недели я родился?

4.6.1. Преобразование дат в текстовые переменные

Можно и перекодировать даты в текстовые значения, хотя это не так часто бывает нужно. Это можно сделать при помощи функции `as.character()`:

```
strDates <- as.character(dates)
```

Это позволяет применять к датам многие текстовые функции (разделение на подгруппы, замена, слияние и т.д.). Текстовые функции будут детально разобраны в главе 5.

4.6.2. Получение дальнейшей информации

Чтобы узнать больше о преобразовании текстовых переменных в даты, введите `help(as.Date)` и `help(strftime)`. Чтобы получить дополнительную информацию о форматах даты и времени, прочтите `help(ISOdatetime)`. Пакет `lubridate` содержит множество функций, которые упрощают работу с датами. В нем реализованы функции, которые обнаруживают данные в формате даты и времени и анализируют их структуру, извлекают отдельные составляющие таких данных (например, годы, месяцы, дни и т.д.), а также производят с ними арифметические операции. Если вам нужно произвести сложные вычисления с датами, воспользуйтесь пакетом `fCalendar`. В нем содержится несметное число функций для работы с данными. Эти функции позволяют одновременно работать с несколькими часовыми поясами и совершать сложные операции с календарем, распознавая рабочие, выходные и праздничные дни.

4.7. Преобразования данных из одного типа в другой

В предыдущем разделе мы обсуждали, как преобразовывать текстовые значения в формат даты и наоборот. В R реализован ряд функций, которые позволяют определить тип данных и преобразовать их в другой формат. Эти преобразования происходит в R сходным с другими языками программирования образом. Например, если добавить текстовые данные к числовому вектору, все значения вектора преобразуются в текстовый формат. Для проверки типа данных и преобразования их в другой формат можно использовать функции, перечисленные в табл. 4.5.

Табл. 4.5. Функции, используемые для изменения формата данных

Проверка	Преобразование
`is.numeric()`	`as.numeric()`
`is.character()`	`as.character()`
`is.vector()`	`as.vector()`
`is.matrix()`	`as.matrix()`
`is.data.frame()`	`as.data.frame()`
`is.factor()`	`as.factor()`
`is.logical()`	`as.logical()`

Функции типа `is.`*`тип_данных`*`()` возвращают `TRUE` или `FALSE`, тогда как функции типа `as.`*`тип_данных`*`()` преобразуют данные в соответствующий формат. Пример представлен в приведенном ниже программном коде.

Программный код 4.5. Преобразование данных из одного формата в другой

```
> a <- c(1,2,3)
> a
[1] 1 2 3
> is.numeric(a)
[1] TRUE
> is.vector(a)
[1] TRUE
> a <- as.character(a)
> a
[1] "1" "2" "3"
> is.numeric(a)
[1] FALSE
> is.vector(a)
[1] TRUE
> is.character(a)
[1] TRUE
```

Функции типа `is.`*`тип_данных`*`()` в сочетании с операторами, управляющими исполнением программы (такими как `если …, то…`), которые мы обсудим в главе 5, могут стать мощным инструментом, который позволяет обрабатывать данные разных типов по-разному. Кроме того, в R некоторые функции работают только с данными определенного типа (текстовые или числовые, матрица или таблица данных). Команда `as.`*`тип_данных`*`()` позволит преобразовать данные в нужный формат перед началом их анализа.

4.8. Сортировка данных

Иногда, просмотр отсортированных данных помогает лучше разобраться в них. К примеру, какие менеджеры наиболее лояльно относятся к начальству? Для сортировки таблицы данных в R используйте функцию `order()`. По умолчанию данные сортируются в порядке возрастания. Поставьте перед интересующей вас переменной знак минус, чтобы отсортировать ее значения в порядке убывания. Приведенный ниже пример иллюстрирует сортировку данных на примере данных об отношении к начальству.

Команда

```
newdata <- leadership[order(leadership$age),]
```

создает новый набор данных, в котором строки отсортированы, начиная с самого молодого менеджера, кончая самым старым. Команды

```
attach(leadership)
newdata <- leadership[order(gender, age),]
detach(leadership)
```

позволяют отсортировать строки так, чтобы сначала шли женщины, а потом – мужчины, а внутри каждого пола менеджеры были бы расположены от самых молодых к самым старым.

Наконец, команды

```
attach(leadership)
newdata <-leadership[order(gender, -age),]
detach(leadership)
```

сортируют строки сначала по полу, а потом – в порядке убывания возраста в пределах каждого пола.

4.9. Объединение наборов данных

Если ваши данные существуют в виде разрозненных фрагментов, их нужно объединить, прежде чем двигаться дальше. В этом разделе рассказано, как добавлять столбцы (переменные) и строки (наблюдения) к таблице данных.

4.9.1. Добавление столбцов

Для того чтобы объединить две таблицы данных в горизонтальном направлении, нужно использовать функцию `merge()`. В большинстве случаев, две таблицы объединяются по значениям одной или нескольких ключевых переменных. К примеру, команда

```
total <- merge(dataframeA, dataframeB, by="ID")
```

объединяет таблицы данных `dataframeA` и `dataframeB` по значениям переменной `ID`. Аналогично, команда

```
total <- merge(dataframeA, dataframeB, by=c("ID","Country"))
```

объединяет две таблицы данных по значениям переменных `ID` и `Country`. Подобное объединение таблиц данных в горизонтальном направлении часто используется для добавления переменных к таблице.

Примечание. *Если вы хотите объединить две матрицы или таблицы данных в горизонтальном направлении, и вам не нужно указывать значения переменной, по которым произойдет объединение, можете использовать функцию* `cbind()`*:*

```
total <- cbind(A, B)
```

Эта команда объединяет объекты A и B в горизонтальном направлении. Для того чтобы функция работала правильно, каждый объект должен иметь одинаковое число строк, расположенных в одинаковом порядке.

4.9.2. Добавление строк

Для того чтобы объединить две таблицы данных в вертикальном направлении, используйте функцию `rbind()`:

```
total <- rbind(dataframeA, dataframeB)
```

Объединяемые таблицы должны состоять из одинаковых переменных, но эти переменные не обязательно должны быть расположены в одной и той же последовательности. Если в таблице `dataframeA` есть переменные, которых нет в таблице `dataframeB`, тогда перед объединением этих таблиц нужно сделать одно из двух:

- удалить лишние переменные из таблицы `dataframeA`;
- создать дополнительные переменные в таблице `dataframeB` и присвоить им значения NA (пропущенные).

Слияние таблиц в вертикальном направлении обычно используется для добавления наблюдений в таблицу данных.

4.10. Разделение наборов данных на составляющие

В R есть обширные возможности для обозначения частей объектов. Эти возможности можно использовать для выбора и исключения переменных и/или наблюдений. В приведенных ниже разделах обсуждаются несколько способов выбора или удаления переменных и наблюдений.

4.10.1. Выбор переменных

Часто бывает так, что новый набор данных создается из небольшого числа переменных, выбранных из большего набора данных. В главе 2 вы узнали, как выбирать элементы таблицы данных при помощи команды типа `таблица_данных[номера_строк, номера_столбцов]`. Этот прием можно использовать для выбора отдельных переменных. К примеру, команда

```
newdata <- leadership[,c(6:10)]
```

позволяет выбрать переменные `q1`, `q2`, `q3`, `q4` и `q5` из таблицы данных `leadership` и сохранить их в таблице данных `newdata`. Не указывая номера строк (`,`), мы по умолчанию выбираем все строки.

При помощи команд

```
myvars <- c("q1", "q2", "q3", "q4", "q5")
newdata <-leadership[myvars]
```

можно выбрать те же самые переменные. В этом случае имена переменных (в кавычках) используются для обозначения переменных, которые должны быть выбраны.

Наконец, для выполнения той же задачи вы могли бы использовать команды

```
myvars <- paste("q", 1:5, sep="")
newdata <- leadership[myvars]
```

В этом примере для создания такого же вектора, как в предыдущем случае, использована функция `paste()`. Эту функцию мы подробно рассмотрим в главе 5.

4.10.2. Исключение переменных

Существует много причин для того, чтобы исключить переменные. Например, в том случае, если переменная содержит несколько пропущенных значений, вам может понадобиться удалить ее перед тем, как анализировать данные. Давайте рассмотрим несколько способов удаления переменных.

Переменные `q3` и `q4` можно удалить при помощи следующих команд

```
myvars <- names(leadership) %in% c("q3", "q4")
newdata <- leadership[!myvars]
```

Для того чтобы понять, как это работает, рассмотрим команды по частям:

1. `names(leadership)` создает текстовый вектор, содержащий названия переменных: `c("managerID","testDate","country","gender","age","q1", "q2","q3","q4","q5")`.
2. `myvars <- names(leadership) %in% c("q3", "q4")` возвращает логический вектор со значениями `TRUE` для каждого элемента вектора `names(leadership)`, который соответствует `q3` или `q4`, и со значениями `FALSE` в противном случае: `c(FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, FALSE)`.
3. Оператор «не» (!) меняет логические значения на противоположные: `c(TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, FALSE, FALSE, TRUE)`.
4. `leadership[c(TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, FALSE, FALSE, TRUE)]` выбирает столбцы, для которых значения логического вектора равны `TRUE`, так что `q3` и `q4` исключаются.

Зная, что `q3` и `q4` – это восьмая и девятая переменные, вы можете удалить их при помощи команды

```
newdata <- leadership[c(-8,-9)]
```

Это работает, поскольку минус перед номером столбца означает, что этот столбец должен быть удален.

Наконец, эти же два столбца можно удалить при помощи команды

```
leadership$q3 <- leadership$q4 <- NULL
```

Теперь вы назначаете эти столбцы неопределенными (`NULL`). Обратите внимание, что `NULL` – это не то же самое, что `NA` (отсутствующие значения).

Удаление переменных – действие, противоположное отбору переменных. Выбор между этими действиями зависит от того, какое из них легче осуществить. Если нужно удалить много переменных, может быть, проще выбрать остающиеся, и наоборот.

4.10.3. Выбор наблюдений

Выбор или удаление наблюдений (строк) – это, в большинстве случаев, залог успешной подготовки данных и их анализа. Несколько примеров содержатся в приведенном ниже программном коде.

Программный код 4.6. Выбор наблюдений

```
newdata <- leadership[1:3,]
newdata <- leadership[which(leadership$gender=="M" &
                            leadership$age > 30),]
attach(leadership)
newdata <- leadership[which(gender=='M' & age > 30),]
detach(leadership)
```

В каждом из этих примеров приведены номера строк, а место для номеров столбцов оставлено пустым (то есть выбраны все столбцы). В первом примере вы выбираете строки с первой по третью (первые три наблюдения).

Во втором примере вы выбираете всех мужчин старше 30 лет. Давайте рассмотрим эту строку программного кода по частям, чтобы понять его.

1. Логическое выражение `leadership$gender=="M"` создает вектор `c(TRUE, FALSE, FALSE, TRUE, FALSE)`.
2. Логическое выражение `leadership$age > 30` создает вектор `c(TRUE, TRUE, FALSE, TRUE, TRUE)`.
3. Логическое выражение `c(TRUE, FALSE, FALSE, TRUE, FALSE) & c(TRUE, TRUE, FALSE, TRUE, TRUE)` создает вектор `c(TRUE, FALSE, FALSE, TRUE, FALSE)`.
4. Функция `which()` возвращает номера элементов вектора, которые представлены значением `TRUE`. Таким образом, выражение `which(c(TRUE, FALSE, FALSE, TRUE, FALSE))` возвращает вектор `c(1, 4)`.
5. Команда `leadership[c(1,4),]` выбирает из таблицы данных первое и четвертое наблюдения, которые удовлетворяют нашим критериям (мужчины старше 30 лет).

В третьем примере использована функция `attach()`,чтобы вам не нужно было писать перед каждым именем переменной название таблицы данных.

В начале этой главы я предположил, что при анализе данных вы можете захотеть ограничиться наблюдениями, сделанными в период между 1 января и 31 декабря 2009 года. Как это можно осуществить? Вот одно из возможных решений:

```
leadership$date <- as.Date(leadership$date, "%m/%d/%y")
startdate <- as.Date("2009-01-01")
enddate   <- as.Date("2009-10-31")
newdata <- leadership[which(leadership$date >= startdate &
leadership$date <= enddate),]
```

Преобразуйте даты, которые исходно воспринимались программой как текстовые значения, в формат даты (мм/дд/гг). Затем назначьте начальную и конечную даты временного отрезка. Поскольку по умолчанию функция `as.Date()` уже создает даты в формате гггг/мм/дд, вам не нужно указывать формат. Наконец, выберите наблюдения, которые удовлетворяют заданному критерию, как вы делали в предыдущем примере.

4.10.4. Функция subset()

Примеры, приведенные в предыдущих двух разделах, важны, поскольку они помогают понять, как R интерпретирует логические векторы и операторы сравнения. Понимание принципа работы этих примеров поможет понять общие принципы действия программного кода в R. Теперь, после того, как вы освоили сложные способы, посмотрим, как сделать это проще.

Функция `subset()` – возможно, самый простой способ выбора переменных и наблюдений. Вот два примера:

```
newdata <- subset(leadership, age >= 35 | age < 24,
                  select=c(q1, q2, q3, q4))
newdata <- subset(leadership, gender=="M" & age > 25,
                  select=gender:q4)
```

В первом примере вы выбираете все строки, в которых значение переменной `age` больше или равно 35 *или* меньше 24, оставляя переменные с `q1` по `q4`. Во втором примере вы отбираете всех мужчин старше 25 лет, оставляя переменные с `gender` по `g4` (`gender`, `g4` и все столбцы, находящиеся между ними). Вы уже видели оператор двоеточие в выражениях типа `от:до` в главе 2. Здесь этот оператор позволяет оставить все переменные в таблице данных, начиная с переменной `от` и заканчивая переменной `до`.

4.10.5. Случайные выборки

Создание выборок из больш**и**х наборов данных – обычное дело при поиске структуры в данных или в машинном обучении. К примеру, вам может понадобиться получить две случайные выборки, чтобы создать предсказательную модель для одной из них и оценить эффективность этой модели на второй выборке. Функция `sample()` позволяет создавать случайные выборки (с замещением или без него) заданного объема из набора данных.

Случайную выборку из таблицы данных `leadership`, состоящую из трех элементов, можно создать при помощи команды

```
mysample <- leadership[sample(1:nrow(leadership), 3, replace=FALSE),]
```

Первый аргумент функции `sample()` – это вектор из элементов, которые могут попасть в выборку. Здесь вектор состоит из чисел от единицы до числа наблюдений в таблице данных. Второй аргумент – это число элементов, которые должны быть выбраны, а третий аргумент указывает на то, что выборка должна быть создана без замещения. Функция `sample()` возвращает случайно выбранные элементы, которые используются для отбора строк из таблицы данных.

Дополнительная информация

В R реализованы обширные возможности создания выборок, включая получение и проверку пробных выборок (пакет `sampling`) и анализ комплексных выборочных данных (пакет `survey`). Другие методы, основанные на создании случайных выборок, включая бутстреп-анализ и метод повторных выборок, описаны в главе 11.

4.11. Использование команд SQL для преобразования таблиц данных

До этого момента для преобразования данных мы использовали команды R. Однако многие аналитики данных перешли к R, предварительно овладев языком структурированных запросов (Structured Query Language – SQL). Было бы обидно не воспользоваться этими накопленными знаниями. Так что, прежде чем закончить этот раздел, позвольте мне кратко упомянуть о существовании пакета `sqldf`. (Если вы не знакомы с языком структурированных запросов, можете спокойно пропустить этот раздел).

После того как вы скачали и установили этот пакет (`install.packages("sqldf")`), вы можете использовать команду `sqldf()`, чтобы применить функцию языка структурированных запросов `SELECT` к таблицам данных. В приведенном ниже программном коде есть два примера.

Программный код 4.7. Использование команд SQL для преобразования таблиц данных

```
> library(sqldf)
> newdf <- sqldf("select * from mtcars where carb=1 order by mpg",
                  row.names=TRUE)
> newdf
                mpg cyl  disp  hp drat   wt qsec vs am gear carb
Valiant        18.1   6 225.0 105 2.76 3.46 20.2  1  0    3    1
Hornet 4 Drive 21.4   6 258.0 110 3.08 3.21 19.4  1  0    3    1
Toyota Corona  21.5   4 120.1  97 3.70 2.46 20.0  1  0    3    1
Datsun 710     22.8   4 108.0  93 3.85 2.32 18.6  1  1    4    1
Fiat X1-9      27.3   4  79.0  66 4.08 1.94 18.9  1  1    4    1
Fiat 128       32.4   4  78.7  66 4.08 2.20 19.5  1  1    4    1
Toyota Corolla 33.9   4  71.1  65 4.22 1.83 19.9  1  1    4    1
> sqldf("select avg(mpg) as avg_mpg, avg(disp) as avg_disp, gear
              from mtcars where cyl in (4, 6) group by gear")
  avg_mpg avg_disp gear
1    20.3      201    3
2    24.5      123    4
3    25.4      120    5
```

В первом примере мы выбрали все переменные (столбцы) из таблицы данных `mtcars`, оставив только те автомобили (строки), у которых есть один карбюратор (`carb`), отсортировав автомобили в возрастающем порядке по значениям переменной `mpg`, и сохранив результаты в виде новой таблицы данных `newdf`. Выражение `row.names=TRUE` позволяет перенести названия строк из исходной таблицы данных в новую. Во втором примере мы вывели на экран средние значения переменных `mpg` и `disp` для каждого числа передач (`gear`) и цилиндров (`cyl`).

Продвинутые пользователи SQL сочтут пакет `sqldf` полезным приложением для управления данными в R. Посетите домашнюю Интернет-страницу проекта (http://code.google.com/p/sqldf/) для того чтобы узнать больше об этом.

4.12. Резюме

В этой главе мы многое успели рассмотреть. Мы познакомились с тем, как R хранит пропущенные значения и даты, и исследовали разные способы работы с такими данными. Вы узнали, как определять тип объекта и как преобразовывать его в объекты другого типа. Вы использовали простые формулы для создания новых переменных и перекодировки уже имеющихся. Я показал, как сортировать данные и переименовывать столбцы. Вы узнали, как объединять ваши данные с другими как горизонтально (добавляя переменные), так и вертикально (добавляя наблюдения). Наконец, мы обсудили, как выбирать или удалять переменные и как выбирать наблюдения разными способами.

В следующей главе мы рассмотрим великое множество арифметических, текстовых и статистических функций, которые используются в R для создания и преобразования переменных. После знакомства со способами управления процессом выполнения команд, вы узнаете, как создать собственные функции. Мы также увидим, как можно использовать эти функции для объединения и обобщения ваших данных.

К концу пятой главы вы овладеете почти всеми способами управления сложными наборами данных. (И станете объектом зависти всех людей, которые анализируют данные!).

Глава 5. Более сложные способы управления данными

В этой главе:

- математические и статистические функции;
- текстовые функции;
- циклы и исполнение команд при условии;
- пользовательские функции;
- способы объединять и преобразовывать данные.

В четвертой главе мы рассмотрели основные способы управления данными в R. В этой главе мы сосредоточимся на более сложных подходах. Глава разделена на три основных раздела. Первый представляет собой краткий обзор многочисленных функций, которые используются в R для математических, статистических и текстовых преобразований. Для придания этому разделу большей актуальности мы начнем с описания задачи по преобразованию данных, которую можно решить с использованием этих функций. После рассмотрения самих функций мы познакомимся с одним из возможных решений этой задачи.

Затем мы обсудим, как создавать свои собственные функции для управления данными и их анализа. Сначала вы исследуете способы контроля за выполнением команд, включая циклы и выполнения команд при условии. Потом вы познакомитесь со структурой созданных пользователями функций и узнаете, как применять их.

Наконец, мы рассмотрим способы объединения, обобщения и преобразования данных. В качестве способа объединения данных можно использовать любую встроенную или написанную пользователем функцию, так что изученное в первых двух разделах этой главы вам действительно пригодится.

5.1. Задача по управлению данными, которую нужно решить

Прежде чем начинать обсуждение числовых и текстовых функций, давайте рассмотрим одну задачу по управлению данными. Группа студентов сдавала экзамены по математике, естественным наукам и английскому языку. Вы хотите объединить их баллы по трем предметам, чтобы получить единый показатель успеваемости для каждого студента. Кроме того, вы хотите поставить оценку A[3] первым по успеваемости 20% студентов, оценку B[4] – следующим по успеваемости 20% и так далее. Наконец, вы хотите отсортировать студентов в алфавитном порядке. Данные представлены в табл. 5.1.

Таблица 5.1. Результаты студенческих экзаменов

Студент	Математика	Естественные науки	Английский язык
John Davis	502	95	25
Angela Williams	600	99	22
Bullwinkie Moose	412	80	18
David Jones	358	82	15
Janice Markhammer	495	75	20
Cheryl Cushing	512	85	28
Reuven Ytzrhak	410	80	15
Greg Knox	625	95	30
Joel England	573	89	27
Mary Rayburn	522	86	18

При взгляде на эти данные немедленно обнаруживается ряд проблем. Во-первых, баллы, полученные за разные экзамены несопоставимы между собой. Их средние значения и стандартные отклонения сильно различаются, так что усреднять их не имеет смысла. Для вычисления единого показателя успеваемости необходимо преобразовать эти баллы так, чтобы их можно было сопоставлять между собой. Во-вторых, нам понадобится метод для определения места студентов в общем рейтинге успеваемости, чтобы поставить им итоговую оценку. В-третьих, для нормальной сортировки студентов в алфавитном порядке нужно будет разбить первый столбец на два – с именем и фамилией.

Каждая из перечисленных проблем может быть решена при разумном использовании числовых и текстовых функций в R. После рассмотрения всех функций в следующем разделе, мы сможем найти возможное решение для нашей задачи по управлению данными.

---

[3] *«Отлично» в американской системе оценивания. – Прим. пер.*

[4] *«Хорошо» в американской системе оценивания. – Прим. пер.*

5.2. Числовые и текстовые функции

Этот раздел представляет собой обзор функций R, которые могут быть использованы при управлении данными. Эти функции можно разделить на числовые (математические, статистические, вероятностные) и текстовые. После того как мы рассмотрим оба этих типа функций, я покажу вам, как применять их к столбцам (переменным) и строкам (наблюдениям) таблиц данных (см. раздел 5.2.6).

5.2.1. Математические функции

В таблице 5.2 перечислены наиболее распространенные математические функции вместе с короткими примерами.

Таблица 5.2. Математические функции

Функция	Описание
`abs(`*x*`)`	Модуль `abs(-4)` равно 4
`sqrt(`*x*`)`	Квадратный корень `sqrt(25)` равно 5 Это то же самое, что и 25^(0.5)
`ceiling(`*x*`)`	Наименьшее целочисленное значение, не меньшее, чем *x* `ceiling(3.457)` равно 4
`floor(`*x*`)`	Наибольшее целочисленное значение, не большее, чем *x* `floor(3.457)` равно 3
`trunk(`*x*`)`	Целое число, полученное при округлении *x* в сторону нуля `trunk(5.99)` равно 5
`round(`*x*`, digits=`*n*`)`	Округляет *x* до заданного числа знаков после запятой `round(3.475, digits=2)` равно 3.48
`signif(`*x*`, digits=`*n*`)`	Округляет *x* до заданного числа значащих цифр `signif(3.475, digits=2)` равно 3.5
`cos(`*x*`),sin(`*x*`),tan(`*x*`)`	Косинус, синус и тангенс `cos(2)` равно -0.416
`acos(`*x*`),asin(x),atan(`*x*`)`	Арккосинус, арксинус и арктангенс `acos(-0.416)` равно 2
`cosh(`*x*`),sinh(`*x*`),tanh(`*x*`)`	Гиперболические косинус, синус и тангенс `sinh(2)` равно 3.627
`acosh(`*x*`),asinh(`*x*`),atanh(`*x*`)`	Гиперболические арккосинус, арксинус и арктангенс `asinh(3.627)` равно 2

`log(`*x*`, base=`*n*`)`
`log(`*x*`)`
`log10(`*x*`)`
	Логарифм *x* по основанию *n*
	Для удобства:
	`log(x)` – натуральный логарифм
	`log10(x)` – десятичный логарифм
	`log(10)` равно 2.3026
	`log10(10)` равно 1

`exp(`*x*`)`	Экспоненциальная функция `exp(2.3026)` равно 10

В основном эти функции применяются для преобразования данных. К примеру, данные с положительно ассиметричным распределением перед дальнейшей обработкой обычно логарифмируют. Математические функции также используют при составлении формул, создании графиков (например, кривая зависимости x от sin(x)) и форматировании числовых значений перед выводом на экран.

В таблице 5.2 приведены примеры применения математических функций к скалярам (отдельным числам). Когда эти функции применяются к числовым векторам, матрицам или таблицам данных, они преобразуют каждое число по отдельности. Например, `sqrt(c(4, 16, 25))` возвращает вектор `c(2, 4, 5)`.

5.2.2. Статистические функции

Самые распространенные статистические функции перечислены в табл. 5.3. У многих из них есть дополнительные параметры, которые влияют на результат. Например,

```
y <- mean(x)
```

позволяет вычислить среднее арифметическое для всех элементов объекта x, а

```
z <- mean(x, trim = 0.05, na.rm=TRUE)
```

вычисляет усеченное среднее, исключив 5% самых больших и 5% самых маленьких значений в выборке и не принимая во внимание пропущенные значения. Используйте команду `help()`, чтобы узнать больше о каждой функции и ее аргументах.

Таблица 5.3. Статистические функции

Функция	Описание
`mean(`*x*`)`	Среднее арифметическое `mean(c(1,2,3,4))` равно 2.5
`median(`*x*`)`	Медиана `median(c(1,2,3,4))` равно 2.5
`sd(`*x*`)`	Стандартное отклонение `sd(c(1,2,3,4))` равно 1.29
`var(`*x*`)`	Дисперсия `var(c(1,2,3,4))` равно 1.67.
`mad(`*x*`)`	Абсолютное отклонение медианы `mad(c(1,2,3,4))` равно 1.48

`quantile(`*x*`, `*probs*`)`	Квантили, где *x* – числовой вектор, для которого нужно вычислить квантили, а *probs* – числовой вектор с указанием вероятностей в диапазоне [0; 1]
	`# 30-ая и 84-ая перцентили x`
	`y <- quantile(x, c(.3,.84))`

`range(`*x*`)`	Размах значений `x <- c(1,2,3,4)` `range(x)` равно `c(1,4)`. `diff(range(x))` равно 3
`sum(`*x*`)`	Сумма `sum(c(1,2,3,4))` равно 10
`diff(`*x*`, lag=`*n*`)`	Разность значений в выборке, взятых с заданным интервалом (`lag`). По умолчанию интервал равен 1. `diff(x)` равно `c(4, 18, 6)`
`min(`*x*`)`	Минимум `min(c(1,2,3,4))` равно 1
`max(`*x*`)`	Максимум `max(c(1,2,3,4))` равно 4
`scale(`*x*`, center=TRUE, scale=TRUE)`	Значения объекта *x*, центрованные (`center=TRUE`) или стандартизованные (`center=TRUE, scale=TRUE`) по столбцам. Пример дан в программном коде 5.6.

Для того чтобы увидеть все эти функции в действии, посмотрите на размещенный ниже программный код. В нем показаны два способа расчета среднего арифметического и стандартного отклонения для числового вектора.

Программный код 5.1. Вычисление среднего арифметического и стандартного отклонения

```
> x <- c(1,2,3,4,5,6,7,8)                    ←быстрый способ
> mean(x)
[1] 4.5
> sd(x)
[1] 2.449490
```

```
> n <- length(x)                              ← долгий способ
> meanx <- sum(x)/n
> css <- sum((x - meanx)^2)
> sdx <- sqrt(css / (n-1))
> meanx
[1] 4.5
> sdx
[1] 2.449490
```

Полезно посмотреть, как скорректированная сумма квадратов (`css`) вычисляется вторым способом.

1. `x` представляет собой вектор `c(1,2,3,4,5,6,7,8)`, а среднее арифметическое значение x равно 4.5 (`length(x)` возвращает число элементов, составляющих `x`);
2. выражение `x - meanx` позволяет вычесть 4.5 из каждого элемента `x`, в результате чего получается вектор `c(-3.5, -2.5, -1.5, -0.5, 0.5, 1.5, 2.5, 3.5)`;
3. выражение `(x - meanx)^2` возводит в квадрат каждый элемент `(x - meanx)` в результате чего получается вектор `c(12.25, 6.25, 2.25, 0.25, 0.25, 2.25, 6.25, 12.25)`;
4. команда `sum((x - meanx)^2)` вычисляет сумму всех элементов `(x - meanx)^2)`, равную 42.

Создание формул в R имеет много общего с языками матричных преобразований, таких как MATLAB (мы более детально рассмотрим решение задач матричной алгебры в приложении E).

Стандартизация данных

По умолчанию функция `scale()` стандартизирует заданный столбец матрицы или таблицы данных так, чтобы его среднее арифметическое было равно нулю, а стандартное отклонение – единице:
`newdata <- scale(mydata)`
Для преобразования каждого столбеца так, чтобы его среднее арифметическое и стандартное отклонение приобрели заданные значения, нужно использовать примерно такой программный код:
`newdata <- scale(mydata)*SD + M`
где `M` – это нужное значение среднее арифметического, а `SD` – стандартного отклонения.

Применение функции `scale()` к столбцам с нечисловыми данными вызывает сообщение об ошибке. Для того чтобы стандартизировать определенный столбец, а не всю матрицу или таблицу данных целиком, нужно использовать программный код вроде этого:
`newdata <- transform(mydata, myvar = scale(myvar)*10+50).`

Этот код преобразует переменную myvar так, что ее среднее арифметическое становится равным 50, а стандартное отклонение – 10. Мы будем использовать функцию scale() для решения описанной выше задачи по управлению данными в разделе 5.3.

5.2.3. Вероятностные функции

Вы можете задаться вопросом – почему вероятностные функции не были рассмотрены вместе со статистическими (это действительно вызвало у вас беспокойство, не правда ли?). Хотя вероятностные функции по определению статистические, они настолько своеобразны, что заслуживают вынесения в отдельный раздел. Вероятностные функции обычно используются для создания искусственных данных с известными параметрами и для вычисления вероятностей для созданных пользователями статистических функций.

В программе R вероятностные функции имеют вид
[dpqr]*сокращенное_название_распределения*()
где первые буквы означают параметры распределения данных:
d=плотность
p=функция распределения
q=функция, определяющая квантили
r=генератор случайных отклонений
Наиболее распространенные вероятностные функции перечислены в табл. 5.4.

Таблица 5.4. Типы распределений

Распределение	Сокращенное название
Бета	beta
Биномиальное	binom
Коши	Cauchy
Хи-квадрат (ассиметричное)	chisq
Экспоненциальное	exp
F	f
Гамма	gamma
Геометрическое	geom
Гипергеометрическое	hyper
Логнормальное	lnorm
Логистическое	logis
Мультиномиальное	multinom
Отрицательное биномиальное	nbinom
Нормальное	norm
Пуассоновское	pois
Wilcoxon signed rank	signrank
T	t
Равномерное	unif
Вейбулла	weibull
Wilcoxon rank sum	wilcox

Для того чтобы понять, как это работает, давайте рассмотрим функции, связанные с нормальным распределением. Если вы не указали значения среднего арифметического и стандартного отклонения, по умолчанию это будет стандартное нормальное распределение (среднее арифметическое равно 0, стандартное отклонение равно 1). Примеры функций плотности (dnorm), распределения (pnorm), квантилей (qnorm) и генератора случайных отклонений (rnorm) приведены в табл. 5.5.

Таблица 5.5. Функции нормального распределения

Задача	Решение

Как нарисовать кривую стандартного нормального распределения в диапазоне значений [-3, 3] (см. ниже)?	`x <- pretty(c(-3,3), 30)`
	`y <- dnorm(x)`
	`plot(x, y,`
	`  type = "l",`
	`  xlab = "Normal Deviate",`
	`  ylab = "Density",`
	`  yaxs = "i"`
	`)`

Какова площадь под кривой стандартного нормального распределения слева от z=1.96?	`pnorm(1.96)` равно 0.975
Каково значение 90-ой перцентили нормального распределения со средним значением 500 и стандартным отклонением 100?	`qnorm(.9, mean=500, sd=100)` равно 628.16
Как создать 50 случайных чисел, принадлежащих нормальному распределению со средним значением 50 и стандартным отклонением 10?	`rnorm(50, mean=50, sd=10)`

Не беспокойтесь, если приведенные функции имеют незнакомые вам параметры. Они подробно разобраны в главе 11; функция `pretty()` рассмотрена ниже в этой главе в табл. 5.7.

Назначение начального числа при генерации случайных чисел

Каждый раз при генерации псевдослучайных чисел используется новое начальное число, и, соответственно, получаются разные результаты. Для того чтобы сделать результаты воспроизводимыми, можно задать это начальное число в явном виде при помощи функции `set.seed()`. Пример приведен в следующем программном коде. Здесь функция `runif()` используется для генерации псевдослучайных чисел, принадлежащих к однородному распределению, в интервале от 0 до 1.

Программный код 5.2. Генерация псевдослучайных чисел, принадлежащих к однородному распределению

```
> runif(5)
[1] 0.8725344 0.3962501 0.6826534 0.3667821 0.9255909
> runif(5)
[1] 0.4273903 0.2641101 0.3550058 0.3233044 0.6584988
> set.seed(1234)
> runif(5)
[1] 0.1137034 0.6222994 0.6092747 0.6233794 0.8609154
> set.seed(1234)
> runif(5)
[1] 0.1137034 0.6222994 0.6092747 0.6233794 0.8609154
```

Установив начальное число вручную, вы можете получать воспроизводимые результаты. Это может пригодиться при создании примеров, к которым вы и другие люди будете возвращаться впоследствии.

Генерация многомерных данных, принадлежащих к нормальному распределению

В исследованиях с использованием искусственных данных и методов Монте Карло часто бывает необходимо генерировать данные, принадлежащие к многомерному нормальному распределению с заданными вектором средних значений и матрицей ковариации. Функция `mvnorm()` из пакета `MASS` позволяет легко справиться с этой задачей. Общий вид применения функции таков:
`mvnorm(n, mean, sigma)`
где *n* – это необходимый объем выборки, *mean* – вектор средних значений, а *sigma* – ковариационная (или корреляционная) матрица. В программном коде 5.3 вы создадите выборку из 500 наблюдений, принадлежащих к многомерному нормальному распределению из трех переменных со следующими параметрами:

Вектор средних значений	230.7	146.7	3.6
Ковариационная матрица	15360.8	6721.2	-47.1
	6721.2	4700.9	-16.5
	-47.1	-16.5	0.3

Программный код 5.3. Генерация данных, принадлежащих многомерному нормальному распределению

```
> library(MASS)
> options(digits=3)
> set.seed(1234)
```
←❶ Определяем случайное начальное число

```
> mean <- c(230.7, 146.7, 3.6)
```
←❷ Назначаем вектор средних значений и ковариационную матрицу

```
> sigma <- matrix(c(15360.8, 6721.2, -47.1,
                    6721.2, 4700.9, -16.5,
                     -47.1,  -16.5,   0.3), nrow=3, ncol=3)

> mydata <- mvrnorm(500, mean, sigma)
```
←❸ Генерируем данные

```
> mydata <- as.data.frame(mydata)
> names(mydata) <- c("y","x1","x2")

> dim(mydata)
```
←❹ Смотрим результаты

```
> head(mydata, n=10)
       y    x1   x2
1   98.8  41.3 4.35
2  244.5 205.2 3.57
3  375.7 186.7 3.69
4  -59.2  11.2 4.23
5  313.0 111.0 2.91
6  288.8 185.1 4.18
7  134.8 165.0 3.68
8  171.7  97.4 3.81
9  167.3 101.0 4.01
10 121.1  94.5 3.76
```

В программном коде 5.3 вы назначаете начальное число для генерации случайных чисел, что позволяет вам позже воспроизвести полученный результат ❶. Вы задаете вектор средних значений и ковариационную матрицу ❷ и генерируете 500 псевдослучайных чисел ❸. Для удобства результаты преобразованы из матрицы в таблицу данных, а переменным даны названия. Наконец, вы убеждаетесь в том, что ваши данные содержат 500 наблюдений и три переменных, и выводите на

экран первые 10 наблюдений ❹. Учтите, что поскольку корреляционная матрица также представляет собой ковариационную матрицу, вы не можете напрямую задать корреляционную структуру.

С помощью вероятностных функций в R вы можете генерировать искусственные данные, принадлежащие к распределениям с известными параметрами. Число статистических методов, которые используют искусственные данные, в настоящее время лавинообразно растет, и вы увидите несколько примеров их применения в следующих главах.

5.2.4. Текстовые функции

В то время как математические и статистические функции оперируют числовыми данными, текстовые функции извлекают информацию из текстовых данных или меняют формат текстовых данных для вывода на экран и составления отчетов. Например, вам может понадобиться объединить имя и фамилию человека в одной ячейке таблицы, убедившись в том, что они написаны с прописных букв. Некоторые из наиболее востребованных текстовых функций перечислены в табл. 5.6.

Таблица 5.6. Текстовые функции

Функция	Описание
`nchar(x)`	Подсчитывает число элементов *x* `x <- c("ab", "cde", "fghij")` `length(x)` равно 3 (см. табл. 5.7). `nchar(x[3])` равно 5.
`substr(x, start, stop)`	Отделяет или заменяет часть текстового вектора. `x <- "abcdef"` `substr(x, 2, 4)` равно "bcd". `substr(x, 2, 4) <- "22222"` (x теперь представляет собой `"a222ef"`).
`grep(pattern, x, ignore.case=FALSE, fixed=FALSE)`	Ищет определенные элементы в *x*. Если `fixed=FALSE`, то элемент представляет собой регулярное выражение. Если `fixed=TRUE`, тогда элемент представляет собой текстовую строку. Команда возвращает номера подходящих под условие элементов. `grep("A", c("b","A","c"), fixed=TRUE)` равно 2.
`sub(pattern, replacement, x, ignore.case=FALSE, fixed=FALSE)`	Находит определенный элемент в *x* и заменяет его заданным текстом. Если `fixed=FALSE`, то элемент – это регулярное выражение. Если `fixed=TRUE`, то элемент – это текстовая строка. `sub("\\s",".","Hello There")` возвращает надпись `Hello.There`. Обратите внимание на то, что `"\s"` – это регулярное выражение для поиска пробелов; используйте вместо него `\\s`, поскольку `"\"` в R – это символ выхода (см. раздел 1.3.3).
`strsplit(x, split, fixed=FALSE)`	Разбивает элементы текстового вектора *x* по значениям *split*. Если `fixed=FALSE`, то элемент – это регулярное выражение. Если `fixed=TRUE`, то элемент – это текстовая строка. `y <- strsplit("abc", "")` возвращает список, состоящий из одного компонента и трех элементов: `"a" "b" "c"`. и `unlist(y)[2]`, и `sapply(y, "[", 2)` возвращают "b".
`paste(..., sep="")`	Объединяет элементы, разделяя их указанным символом. `paste("x", 1:3, sep="")` возвращает `c("x1", "x2", "x3")`. `paste("x",1:3, sep="M")` возвращает

	`c("xM1","xM2" "xM3").`
	`paste("Today is", date())` возвращает
	`Today is Thu Jun 25 14:17:32 2011`

`toupper(x)`	Перевод в верхний регистр `toupper("abc")` возвращает "ABC".
`tolower(x)`	Перевод в нижний регистр `tolower("ABC")` возвращает "abc".

Обратите внимание на то, что функции `grep()`, `sub()` и `strsplit()` способны осуществлять поиск текстовых строк (`fixed=TRUE`) или регулярных выражений (`fixed=FALSE`, так по умолчанию). Регулярные выражения имеют простой и последовательный синтаксис для того, чтобы обозначить определенный элемент текста. Например, регулярное выражение
`^[hc]?at`
позволяет найти все строки, которые начинаются с 0 или с одной буквы `h` или `c`, за которыми следует `at`. Таким образом, это выражение позволит отыскать слова hat, cat и at, но не bat. Для того чтобы узнать больше, прочтите статью «регулярные выражения» в Wikipedia.[5]

5.2.5. Другие полезные функции

Функции, указанные в табл. 5.7, также весьма полезны для управления данными и их преобразований, однако их нельзя уверенно отнести ни к одной из названных выше категорий.

Таблица 5.7. Другие полезные функции

Функция	Описание
`length(x)`	Число элементов объекта *x*. `x <- c(2, 5, 6, 9)` `length(x)` равно 4.
`seq(from, to, by)`	Создание последовательности элементов. `indices <- seq(1,10,2)` `indices` равно `c(1, 3, 5, 7, 9)`.
`rep(x, n)`	Повторяет *x n* раз. `y <- rep(1:3, 2)` `y` равно `c(1, 2, 3, 1, 2, 3)`.
`cut(x, n)`	Преобразует непрерывную переменную *x* в фактор с *n* уровнями. Для создания упорядоченного фактора добавьте опцию `ordered_result = TRUE`.
`pretty(x, n)`	Создает «красивые» пограничные значения. Разделяет непрерывную переменную *x* на *n* интервалов, выбрав *n+1* одинаково отстоящее друг от друга округленное пограничное значение. Часто используется при построении диаграмм.

`cat(… , file = "myfile", append = FALSE)`	Объединяет объекты в … и выводит их на экран или в файл (если указано его название).
	`firstname <- c("Jane")`
	`cat("Hello" , firstname, "\n").`

На последнем примере в таблице показано, как используется символ выхода при выводе результатов на экран. Используйте `\n` для перехода на новую строку, `\t` – как символ табуляции, `\b` – как знак возврата к предыдущему символу и так далее (наберите `?Quotes` для получения дальнейшей информации).

[5] *см. более полный вариант статьи на английском языке: regular expressions. – Прим. пер.*

Например, программный код

```
name <- "Bob"
cat( "Hello", name, "\b.\n", "Isn\'t R", "\t", "GREAT?\n")
```

дает следующий результат

```
Hello Bob.
 Isn't R         GREAT?
```

Обратите внимание на то, что вторая строка смещена на один символ вправо. Когда команда cat объединяет объекты для вывода на экран, она разделяет их пробелами. Вот почему мы добавили символ возврата (`\b`) перед точкой. В противном случае у нас получилось бы «`Hello Bob .`»

Способы применения изученных на данный момент функций к числам, строкам и векторам интуитивно понятны и просты, но как же применять их к матрицам и таблицам данных? Это и станет темой следующего раздела.

5.2.6. Применение функций к матрицам и таблицам данных

Одно из интересных свойств функций R – это то, что их можно применять к разным типам объектов (скаляры, векторы, матрицы, массивы и таблицы данных). Приведенный ниже программный код послужит иллюстрацией этого.

Программный код 5.4. Применение функций к объектам

```
> a <- 5
> sqrt(a)
[1] 2.236068
> b <- c(1.243, 5.654, 2.99)
> round(b)
[1] 1 6 3
> c <- matrix(runif(12), nrow=3)
> c
       [,1]  [,2]  [,3]  [,4]
[1,] 0.4205 0.355 0.699 0.323
[2,] 0.0270 0.601 0.181 0.926
[3,] 0.6682 0.319 0.599 0.215
> log(c)
       [,1]   [,2]   [,3]   [,4]
[1,] -0.866 -1.036 -0.358 -1.130
[2,] -3.614 -0.508 -1.711 -0.077
[3,] -0.403 -1.144 -0.513 -1.538
> mean(c)
[1] 0.444
```

Обратите внимание на то, что в программном коде 5.4 среднее значение матрицы `c` равно скаляру (0.444). Функция `mean()` вычисляет среднее арифметическое для всех 12 элементов матрицы. А что если вам нужно вычислить средние значения для каждой из трех строчек или каждого из четырех столбцов?

В R есть функция `apply()`, которая позволяет применить любую функцию к любой части матрицы, массива или таблицы данных. Формат применения этой функции таков:

`apply(`*`x`*`, `*`MARGIN`*`, `*`FUN`*`, ...)`

где *`x`* – это объект, *`MARGIN`* – индекс, обозначающий часть объекта (столбцы или строки), *`FUN`* – функция и `...` – это любые параметры этой функции. Для матрицы или таблицы данных `MARGIN=1` обозначает строки, а `MARGIN=2` – столбцы. Взгляните на пример в программном коде 5.5.

Программный код 5.5. Применение функции к строкам (столбцам) матрицы

```
> mydata <- matrix(rnorm(30), nrow=6)   ←❶ Генерируем данные
> mydata
         [,1]   [,2]    [,3]   [,4]   [,5]
[1,]  0.71298  1.368 -0.8320 -1.234 -0.790
[2,] -0.15096 -1.149 -1.0001 -0.725  0.506
[3,] -1.77770  0.519 -0.6675  0.721 -1.350
[4,] -0.00132 -0.308  0.9117 -1.391  1.558
[5,] -0.00543  0.378 -0.0906 -1.485 -0.350
[6,] -0.52178 -0.539 -1.7347  2.050  1.569
> apply(mydata, 1, mean)   ←❷ Вычисляем средние значения для строк
[1] -0.155 -0.504 -0.511  0.154 -0.310  0.165
> apply(mydata, 2, mean)   ←❸ Вычисляем средние значения для столбцов
[1] -0.2907  0.0449 -0.5688 -0.3442  0.1906
> apply(mydata, 2, mean, trim=0.2) ←❹ Вычисляем усеченные средние
[1] -0.1699  0.0127 -0.6475 -0.6575  0.2312  значения для столбцов
```

Сначала вы создаете матрицу 6×5, состоящую из случайно выбранных элементов нормального распределения ❶. Затем вы вычисляете средние значения для каждой из шести строк ❷ и каждого из пяти столбцов ❸. Наконец, вы вычисляете усеченные средние значения для каждого столбца (в данном случае усреднены «центральные» 60% данных, а по 20% самых больших и самых маленьких значений были проигнорированы) ❹.

Поскольку *FUN* означает любую функцию в R, в том числе и ту, которую вы сами написали (см. раздел 5.4), функция `apply()` – это мощное средство. В то время как `apply()` применяется к строкам или столбцам массива данных, `lapply()` и `sapply()` применяют функцию к целому списку. Вы увидите пример применения функции `sapply()` (которая представляет собой дружественный к пользователю вариант функции `lapply()`) в следующем разделе.

Теперь у вас есть все знания для того, чтобы решить задачу по управлению данными, описанную в разделе 5.1, так что давайте попробуем сделать это.

5.3. Решение нашей задачи по управлению данными

В задаче из раздела 5.1 требовалось объединить результаты экзаменов по каждому предмету в единый балл успеваемости для каждого студента, поставить каждому студенту оценку от A до F в зависимости от позиции в общем рейтинге (верхние 20%, следующие 20% и т.д.) и отсортировать строки в списке по фамилии студентов, а затем и по имени. Решение этой задачи представлено в приведенном ниже программном коде.

Программный код 5.6. Решение учебной задачи

```
> options(digits=2)

> Student <- c("John Davis", "Angela Williams", "Bullwinkle Moose",
               "David Jones", "Janice Markhammer", "Cheryl Cushing",
               "Reuven Ytzrhak", "Greg Knox", "Joel England",
               "Mary Rayburn")
> Math <- c(502, 600, 412, 358, 495, 512, 410, 625, 573, 522)
> Science <- c(95, 99, 80, 82, 75, 85, 80, 95, 89, 86)
> English <- c(25, 22, 18, 15, 20, 28, 15, 30, 27, 18)
> roster <- data.frame(Student, Math, Science, English,
                       stringsAsFactors=FALSE)

> z <- scale(roster[,2:4])
> score <- apply(z, 1, mean)
> roster <- cbind(roster, score)
> y <- quantile(score, c(.8,.6,.4,.2))
> roster$grade[score >= y[1]] <- "A"
> roster$grade[score < y[1] & score >= y[2]] <- "B"
> roster$grade[score < y[2] & score >= y[3]] <- "C"
> roster$grade[score < y[3] & score >= y[4]] <- "D"
> roster$grade[score < y[4]] <- "F"
> name <- strsplit((roster$Student), " ")
> lastname <- sapply(name, "[", 2)
> firstname <- sapply(name, "[", 1)
> roster <- cbind(firstname,lastname, roster[,-1])
> roster <- roster[order(lastname,firstname),]
```

← Вычисляем объединенный показатель успеваемости

← Оцениваем студентов

← Извлекаем фамилии и имена

← Сортируем по фамилиям и именам

```
> roster
    Firstname   Lastname Math Science English score grade
6      Cheryl    Cushing  512      85      28  0.35     C
1        John      Davis  502      95      25  0.56     B
9        Joel    England  573      89      27  0.70     B
4       David      Jones  358      82      15 -1.16     F
8        Greg       Knox  625      95      30  1.34     A
5      Janice Markhammer  495      75      20 -0.63     D
3  Bullwinkle      Moose  412      80      18 -0.86     D
10       Mary    Rayburn  522      86      18 -0.18     C
2      Angela   Williams  600      99      22  0.92     A
7      Reuven    Ytzrhak  410      80      15 -1.05     F
```

Этот программный код очень концентрированный, поэтому давайте рассмотрим его по шагам.

Шаг 1. Дан исходный список студентов. Команда `options(digits=2)` сокращает до двух число знаков после запятой у всех выводимых на экран чисел, это упрощает их восприятие.

```
> options(digits=2)
> roster
              Student Math Science English
1         John Davis  502      95      25
2    Angela Williams  600      99      22
3   Bullwinkle Moose  412      80      18
4        David Jones  358      82      15
5  Janice Markhammer  495      75      20
6     Cheryl Cushing  512      85      28
7     Reuven Ytzrhak  410      80      15
8          Greg Knox  625      95      30
9        Joel England  573      89      27
10      Mary Rayburn  522      86      18
```

Шаг 2. Поскольку оценки по математике, естественным наукам и английскому языку выставлены по разным шкалам (их среднее и стандартное отклонение заметно различаются), вам необходимо сделать их сопоставимыми, прежде чем комбинировать. Один из способов выполнить эту задачу – стандартизировать переменные так, чтобы результат каждого теста был выражен в стандартных отклонениях, а не в исходных баллах. Это можно сделать при помощи функции `scale()`:

```
> z <- scale(roster[,2:4])
> z
        Math Science English
[1,]  0.013   1.078   0.587
 [2,]  1.143   1.591   0.037
 [3,] -1.026  -0.847  -0.697
 [4,] -1.649  -0.590  -1.247
 [5,] -0.068  -1.489  -0.330
 [6,]  0.128  -0.205   1.137
 [7,] -1.049  -0.847  -1.247
 [8,]  1.432   1.078   1.504
 [9,]  0.832   0.308   0.954
[10,]  0.243  -0.077  -0.697
```

Шаг 3. Можно рассчитать показатель успеваемости студентов, усреднив значения стандартизированных баллов для каждой строки посредством функции `mean()` и добавив их к списку при помощи функции `cbind()`:

```
> score <- apply(z, 1, mean)
> roster <- cbind(roster, score)
> roster
              Student   Math  Science English  score
1         John Davis  502      95      25    0.559
2    Angela Williams  600      99      22    0.924
3   Bullwinkle Moose  412      80      18   -0.857
4        David Jones  358      82      15   -1.162
5  Janice Markhammer  495      75      20   -0.629
6     Cheryl Cushing  512      85      28    0.353
7     Reuven Ytzrhak  410      80      15   -1.048
8          Greg Knox  625      95      30    1.338
9        Joel England  573      89      27    0.698
10      Mary Rayburn  522      86      18   -0.177
```

Шаг 4. Функция `quantile()` позволяет вычислить границы значений 20%-ных интервалов (перцентилей) показателя успеваемости. Вы видите, что нижняя граница для оценки A равна 0.74, для оценки B – 0.44 и т.д.

```
> y <- quantile(roster$score, c(.8,.6,.4,.2))
> y
  80%   60%   40%   20%
 0.74  0.44 -0.36 -0.89
```

Шаг 5. Используя логические операторы, вы можете перекодировать значения показателя успеваемости в новую категориальную переменную итоговой оценки. Для этого вы создаете переменную `grade` в таблице данных `roster`.

```
> roster$grade[score >= y[1]] <- "A"
> roster$grade[score < y[1] & score >= y[2]] <- "B"
> roster$grade[score < y[2] & score >= y[3]] <- "C"
> roster$grade[score < y[3] & score >= y[4]] <- "D"
> roster$grade[score < y[4]] <- "F"
> roster
              Student Math Science English  score grade
1         John Davis  502      95      25  0.559     B
2    Angela Williams  600      99      22  0.924     A
3   Bullwinkle Moose  412      80      18 -0.857     D
4        David Jones  358      82      15 -1.162     F
5  Janice Markhammer  495      75      20 -0.629     D
6     Cheryl Cushing  512      85      28  0.353     C
7     Reuven Ytzrhak  410      80      15 -1.048     F
8          Greg Knox  625      95      30  1.338     A
9       Joel England  573      89      27  0.698     B
10      Mary Rayburn  522      86      18 -0.177     C
```

Шаг 6. Нам понадобится использовать функцию `strsplit()`, чтобы разделить пробелом имена и фамилии студентов. Применение этой функции к вектору из строк порождает список:

```
> name <- strsplit((roster$Student), " ")
> name
[[1]]
[1] "John"  "Davis"
[[2]]
[1] "Angela"   "Williams"
[[3]]
[1] "Bullwinkle" "Moose"
[[4]]
[1] "David" "Jones"
[[5]]
[1] "Janice"     "Markhammer"
[[6]]
[1] "Cheryl"  "Cushing"
[[7]]
[1] "Reuven"  "Ytzrhak"
[[8]]
[1] "Greg" "Knox"
[[9]]
[1] "Joel"    "England"
[[10]]
[1] "Mary"    "Rayburn"
```

Шаг 7. Можно использовать функцию `sapply()` чтобы поместить первый элемент каждого компонента списка в вектор с именами, а второй – в вектор с фамилиями. "`[`" – это функция, которая выбирает часть объекта – в данном случае, первый или второй элемент листа `name`. Вам потребуется применить функцию `cbind()`, чтобы добавить вектора с именами и фамилиями к списку. Поскольку вам больше не нужна переменная с именем и фамилией каждого студента, можно от нее избавиться (при помощи -1 в индексе при названии списка).

```
> Firstname <- sapply(name, "[", 1)
> Lastname <- sapply(name, "[", 2)
> roster <- cbind(Firstname, Lastname, roster[,-1])
> roster
    Firstname   Lastname Math Science English  score grade
1        John      Davis  502      95      25  0.559     B
2      Angela   Williams  600      99      22  0.924     A
3  Bullwinkle      Moose  412      80      18 -0.857     D
4       David      Jones  358      82      15 -1.162     F
5      Janice Markhammer  495      75      20 -0.629     D
6      Cheryl    Cushing  512      85      28  0.353     C
7      Reuven    Ytzrhak  410      80      15 -1.048     F
8        Greg       Knox  625      95      30  1.338     A
9        Joel    England  573      89      27  0.698     B
10       Mary    Rayburn  522      86      18 -0.177     C
```

Шаг 8. Наконец, можно отсортировать список по именам и фамилиям студентов при помощи функции `order()`:

```
> roster[order(Lastname,Firstname),]
    Firstname   Lastname  Math  Science  English  score   grade
6      Cheryl    Cushing  512       85       28     0.35     C
1        John      Davis  502       95       25     0.56     B
9        Joel    England  573       89       27     0.70     B
4       David      Jones  358       82       15    -1.16     F
8        Greg       Knox  625       95       30     1.34     A
5      Janice Markhammer  495       75       20    -0.63     D
3  Bullwinkle      Moose  412       80       18    -0.86     D
10       Mary    Rayburn  522       86       18    -0.18     C
2      Angela   Williams  600       99       22     0.92     A
7      Reuven    Ytzrhak  410       80       15    -1.05     F
```

Вот и все! Пара пустяков!

Существует много других способов решить эту же задачу, но приведенный программный код позволяет вам понять идею применения этих функций. Теперь настала пора обратиться к функциям управления и функциям, которые пишутся самими пользователями.

5.4. Управление выполнением команд

Обычно команды в R выполняются последовательно, с начала программного кода до конца. Однако бывают случаи, когда некоторые команды нужно выполнить несколько раз, а другие – только при определенных условиях. В такой ситуации необходимо использовать специальные конструкции для управления выполнением команд.

В R реализованы стандартные конструкции такого типа, такие, как и в любом другом современном языке программирования. Сначала мы рассмотрим конструкции, обеспечивающие

выполнение команд при определенном условии, а затем – конструкции, используемые для циклического выполнения команд.

Во всех примерах этого раздела будут использоваться следующие сокращения:

- *statement* – простая или сложная команда (группа команд, заключенная в фигурные скобки {} и разделенная точкой с запятой);
- *cond* – выражение, которое может быть истинным или ложным;
- *expr* – выражение, которое соответствует числу или текстовой строке;
- *seq* – последовательность чисел или текстовых строк.

После обсуждения конструкций, управляющих вводом команд, вы узнаете, как создавать собственные функции.

5.4.1. Повторение и циклы

Циклические конструкции многократно выполняют одну и ту же команду или набор команд, пока не будет выполнено заданное условие. К таким конструкциям относят `for` и `while`.

for

Циклическая конструкция `for` повторно выполняет определенную команду, пока значение переменной будет содержаться в последовательности `seq`. Синтаксис таков:

```
for (var in seq) statement
```

В этом примере

```
for (i in 1:10)  print("Hello")
```

слово Hello будет напечатано 10 раз.

while

Циклическая конструкция `while` повторно выполняет команду, пока заданное условие не перестанет быть истинным. Общий вид применения конструкции таков:

```
while (cond) statement
```

Программный код

```
i <- 10
while (i > 0) {print("Hello"); i <- i - 1}
```

опять напечатает слово Hello 10 раз. Убедитесь, что утверждения внутри скобок так изменяют условие, что рано или поздно оно перестанет быть истинным – иначе цикл никогда не завершится! В предыдущем примере утверждение

```
i <- i - 1
```

вычитает 1 из объекта `i` при выполнении каждого цикла, так что после десятого цикла оно уже не будет больше 0. Если бы вы, наоборот, прибавляли бы по единице после каждого цикла, то R никогда бы не закончил приветствовать вас. Вот почему `while` может быть опаснее других циклических конструкций.

Циклы в R могут быть неэффективными и занимать много времени, когда вы имеете дело с большими объемами данных. При любой возможности лучше вместо циклов использовать встроенные числовые и текстовые функции в R вместе с функциями семейства `apply`.

5.4.2. Выполнение при условии

Выполнение при условии значит, что команды действуют только, если выполняется определенное условие. К таким конструкциям относятся `if-else`, `ielse` и `switch`.

if-else

Управляющая конструкция `if-else` выполняет команду, если верно заданное условие. В качестве опции другая команда может выполняться, если заданное условие окажется неверным. Синтаксис таков:

```
if (cond) statement
if (cond) statement1 else statement2
```

Вот примеры:

```
if (is.character(grade)) grade <- as.factor(grade)
if (!is.factor(grade)) grade <- as.factor(grade) else print("Grade
already is a factor")
```

В первом случае, если переменная `grade` – текстовый вектор, она преобразуется в фактор. Во втором случае выполняется одна из двух команд. Если переменная `grade` – не фактор (обратите внимание на символ `!`), то она преобразуется в него. Если же эта переменная – уже фактор, то на экран выводится сообщение об этом (Grade already is a factor).

ifelse

Конструкция `ifelse` – компактная и векторизованная версия конструкции `if-else`. Синтаксис таков:
`ifelse(cond, statement1, statement2)`
Первая команда выполняется, если условие *cond* истинно. Если условие ложно – выполняется вторая команда. Вот примеры:

```
ifelse(score > 0.5, print("Сдал"), print("Провалился"))
outcome <- ifelse (score > 0.5, "Провалился", "Сдал")
```

Используйте эту конструкцию, если вам нужно произвести бинарную операцию или оперировать векторами.

switch

Конструкция `switch` выбирает команды в зависимости от значения, которое принимает выражение. Синтаксис таков:
`switch(expr, ...)`
где троеточие означает команды, соответствующие возможным значениям *expr*. Проще всего понять, как работает эта конструкция, на примере приведенного ниже программного кода.

Программный код 5.7. Пример применения конструкции `switch`

```
> чувства <- c("печаль", "страх")
> for (i in чувства)
    print(
      switch(i,
        счастье  = "Я рад, что ты счастлив",
        страх = "Тут нечего бояться",
        печаль    = "Приободрись",
        злость  = "Успокойся"
      )
    )
[1] "Приободрись"
[1] "Тут нечего бояться"
```

Это дурацкий пример, зато он демонстрирует основные принципы применения этой конструкции. Вы узнаете, как использовать ее в написанных пользователем функциях, из следующего раздела.

5.5. Написанные пользователем функции

Одно из существенных преимуществ R – это то, что пользователи могут создавать собственные функции. Структура этих функций выглядит примерно так:

```
myfunction <- function(arg1, arg2, ... ){
  statements
  return(object)
}
```

Объекты внутри функции существуют только там. Объект, который появляется в результате действия функции, может быть любого типа – от скаляра до списка. Давайте взглянем на пример.

Допустим, вам хотелось бы иметь функцию, которая бы вычисляла основную тенденцию и разброс данных. Эта функция должна предоставлять выбор между параметрическими (среднее арифметическое и стандартное отклонение) и непараметрическими (медиана и ее абсолютное отклонение) статистиками. Результат должен быть представлен в виде списка с названиями. Кроме того, пользователь должен иметь возможность выбора – будут ли результаты автоматически выведены на экран, или нет. По умолчанию функция должна рассчитывать параметрические статистики и не выводить результаты на экран. Одно из возможных решений представлено в этом программном коде.

Программный код 5.8. `mystats()` – написанная пользователем функция для вычисления общих характеристик данных

```
mystats <- function(x, parametric=TRUE, print=FALSE) {
  if (parametric) {
    center <- mean(x); spread <- sd(x)
  } else {
    center <- median(x); spread <- mad(x)
  }
  if (print & parametric) {
    cat("Mean=", center, "\n", "SD=", spread, "\n")
  } else if (print & !parametric) {
    cat("Median=", center, "\n"MAD=", spread, "\n")
  }
  result <- list(center=center, spread=spread) ",
  return(result)
}
```

Чтобы увидеть эту функцию в действии, сначала создадим некоторые данные (случайная выборка из 500 чисел, принадлежащих нормальному распределению):

```
set.seed(1234)
x <- rnorm(500)
```

После выполнения команды

```
y <- mystats(x)
```

элемент `y$center` будет равен среднему арифметическому (0.00184), а элемент `y$spread` – стандартному отклонению (1.03). На экран ничего не выводится. Если будет выполнена команда

```
y <- mystats(x, parametric=FALSE, print=TRUE),
```

то элемент `y$center` будет равен медиане (–0.0207), а элемент `y$spread` – ее абсолютному отклонению (1.001). Кроме того, результат появится на экране:

```
Median= -0.0207
MAD= 1
```

В качестве следующего шага давайте рассмотрим написанную пользователем функцию с использованием конструкции `switch`. Эта функция позволяет выбрать формат представления сегодняшней даты. Значения, присвоенные параметрам функции при ее написании, используются как значения по умолчанию. Для функции `mydate()` `long` – это формат даты по умолчанию, если параметр `type` не указан:

```
mydate <- function(type="long") {
  switch(type,
    long =  format(Sys.time(), "%A %B %d %Y"),
    short = format(Sys.time(), "%m-%d-%y"),
    cat(type, "это неизвестный формат\n")
   )
}
```

Вот как эта функция работает:

```
> mydate("long")
[1] "Вторник Декабрь 02 2010"
> mydate("short")
[1] "12-02-10"
> mydate()
[1] "Вторник Декабрь 02 2010"
> mydate("medium")
medium это неизвестный формат
```

Обратите внимание на то, что функция `cat()` выполняется только, если введенный тип функции не совпадает с "`long`" или "`short`". Обычно полезно бывает включить в функцию выражение, которое позволяет «отлавливать» вводимые пользователями ошибочные аргументы.

Существует несколько функций, которые помогают исправлять ошибки в написанных вами функциях. Для создания сообщений об ошибке можно использовать функцию `warning()`. Функция `message()` создает диагностические сообщения, а функция `stop()` останавливает выполнение функции и выводит на экран сообщение об ошибке. Прочтите онлайн-справку по этим функциям для получения дальнейшей информации.

Совет. *Как только вы начнете писать функции, вне зависимости от их длины и сложности вам понадобятся хорошие средства для отладки команд. В R есть много встроенных функций для поиска ошибок, а дополнительные пакеты предоставляют еще бо**л**ьшие возможности. Прекрасный источник информации по этой теме – онлайн-пособие Дункан Мардоч (Duncan Murdoch) «Отладка команд в R»*[6] *(http://www.stats.uwo.ca/faculty/murdoch/software/debuggingR).*

[6] *"Debugging in R" на английском языке. – Прим. пер.*

Когда функции созданы, вам может захотеться, чтобы они были доступны в начале каждой сессии. В приложении В описано, как изменить конфигурацию программы так, чтобы пользовательские функции автоматически загружались при запуске программы. Мы рассмотрим дополнительные примеры написанных пользователем функций в главах 6 и 8.

Вы уже можете сделать очень многое, используя основные приемы, описанные в этой главе. Если же вам нужно изучить написание функций во всех деталях или составлять программный код на профессиональном уровне, чтобы распространять его среди других людей, я советую прочитать две прекрасных книги, ссылки на которые вы найдете в списке литературы в конце этой книги: Venableу & Ripley (2000) и Chambers (2008). Взятые вместе, они предоставляют необходимый уровень детализации и разнообразия примеров.

Теперь, когда мы рассмотрели пользовательские функции, закончим эту главу описанием способов агрегирования (объединения) и переформатирования данных.

5.6. Агрегирование и переформатирование данных

В R реализовано множество мощных методов объединения и переформатирования данных. При агрегации данных группы наблюдений заменяются статистиками, основанными на этих наблюдениях. При переформатировании данных вы изменяете структуру (строки и столбцы), определяющую организацию данных. В этом разделе описаны разные методы выполнения этих задач.

В следующих двух подразделах мы будем использовать встроенный в базовую версию программы набор данных `mtcars`. Эти данные, взятые из журнала *Motor Trend* за 1974 год, содержат информацию о дизайне и технических характеристиках (число цилиндров, объем и мощность двигателя, расход топлива и т.д.) для 34 марок автомобилей. Чтобы узнать больше об этом наборе данных, наберите `help(mtcars)`.

5.6.1. Транспонирование

Транспонирование (переворот таблицы на 90°, когда строки и столбцы меняются местами) – наверное, самый простой способ переформатирования данных. Используйте функцию `t()`, чтобы транспонировать матрицу или таблицу данных. В последнем случае названия строк становятся именами переменных (столбцов). Пример представлен в приведенном ниже программном коде.

Программный код 5.9. Транспонирование данных

```
> cars <- mtcars[1:5,1:4]
> cars
                   mpg cyl disp  hp
Mazda RX4         21.0   6  160 110
Mazda RX4 Wag     21.0   6  160 110
Datsun 710        22.8   4  108  93
Hornet 4 Drive    21.4   6  258 110
Hornet Sportabout 18.7   8  360 175
> t(cars)
     Mazda RX4 Mazda RX4 Wag Datsun 710 Hornet 4 Drive Hornet Sportabout
mpg         21            21       22.8           21.4              18.7
cyl          6             6        4.0            6.0               8.0
disp       160           160      108.0          258.0             360.0
hp         110           110       93.0          110.0             175.0
```

В этом программном коде использована лишь часть набора данных `mtcars` для экономии места в книге. Позже в этом разделе вы познакомитесь с более гибким способом транспонирования данных, когда мы рассмотрим пакет `shape`.

5.6.2. Агрегирование данных

В R довольно просто компактизировать данные, используя заданную функцию и одну или более переменных в качестве критерия (by). Формат применения функции таков:
aggregate(*x*, *by*, *FUN*)
где *x* – это исходный объект с данными, *by* – это перечень переменных, которые будут служить критерием для компактизации, а *FUN* – это скалярная функция, которая используется для расчета стастистик, которые послужат значениями нового объекта с данными.

В качестве примера мы агрегируем данные mtcars по числу цилиндров и передач, вычислив средние арифметические всех числовых переменных (см. следующий программный код).

Программный код 5.10. Агрегирование данных

```
> options(digits=3)
> attach(mtcars)
> aggdata <-aggregate(mtcars, by=list(cyl,gear), FUN=mean, na.rm=TRUE)
> aggdata
  Group.1 Group.2  mpg cyl disp  hp drat   wt qsec  vs   am gear carb
1       4       3 21.5   4  120  97 3.70 2.46 20.0 1.0 0.00    3 1.00
2       6       3 19.8   6  242 108 2.92 3.34 19.8 1.0 0.00    3 1.00
3       8       3 15.1   8  358 194 3.12 4.10 17.1 0.0 0.00    3 3.08
4       4       4 26.9   4  103  76 4.11 2.38 19.6 1.0 0.75    4 1.50
5       6       4 19.8   6  164 116 3.91 3.09 17.7 0.5 0.50    4 4.00
6       4       5 28.2   4  108 102 4.10 1.83 16.8 0.5 1.00    5 2.00
7       6       5 19.7   6  145 175 3.62 2.77 15.5 0.0 1.00    5 6.00
8       8       5 15.4   8  326 300 3.88 3.37 14.6 0.0 1.00    5 6.00
```

В представленных результатах Group.1 – это число цилиндров (4, 6 или 8), а Group.2 – это число передач (3, 4 или 5). Например, машинам с четырьмя цилиндрами и тремя передачами в среднем хватает одного галлона топлива, чтобы проехать в среднем 21.5 миль (mpg – miles per gallon).

При использовании функции aggregate() переменные, по которым идет агрегация (by), должны быть представлены в виде списка (даже если туда входит всего одна переменная). Каждой группе из списка можно дать удобное название, например, так: by=list(Group.cyl=cyl, Group.gears=gear). Для агрегирования данных можно использовать любую встроенную или пользовательскую функцию. Это дает команде aggregate() больш**и**е возможности. Однако когда дело касается больш**и**х возможностей, ничто не затмит пакет reshape.

5.6.3. Пакет reshape

Пакет reshape – это потрясающе многогранный подход и к реструктуризации, и к преобразованию данных. Из-за этой многогранности он может оказаться несколько трудным для освоения. Мы будем двигаться медленно, используя маленький объем данных, так, чтобы всегда было ясно, что происходит. Поскольку пакет reshape не включен в базовую версию R, его нужно сначала установить, набрав install.packages("reshape").

Основная идея состоит в том, что вы «расплавляете» ваши данные, так что каждая строка представляет собой уникальную комбинацию идентификационных переменных и собственно переменных, несущих информацию. Затем вы «слепляете» расплавленные данные в любую нужную вам форму. Во время «лепки» можно агрегировать данные при помощи любой функции. Данные, с которыми мы будем работать, представлены в табл. 5.8.

Таблица 5.8. Исходные данные (mydata)

ID	Time	X1	X2
1	1	5	6
1	2	3	5
2	1	6	1
2	2	2	4

В этом наборе данных *измерения* (X1, X2) – это значения в последних двух столбцах (5, 6, 3, 5, 6, 1, 2 и 4). Каждое измерение обозначено уникальной комбинацией идентификационных переменных (в этом случае ID и Time). Например, значение измерения 5 в первом ряду можно обозначить уникальной комбинацией значения ID равного 1 и значения Time равного 1, зная, что оно принадлежит переменной X1.

«Расплавление»

Когда вы «расплавляете» данные, вы преобразуете их в такой формат, что каждая измеренная переменная расположена в собственной строке вместе с переменными, необходимыми для того, чтобы уникально обозначить ее. Если вы расплавите данные из табл. 5.8, используя такой программный код:

```
library(reshape)
md <- melt(mydata, id=(c("id", "time")))
```

у вас получится набор данных с такой структурой, которая показана в табл. 5.9.

Таблица 5.9. «Расплавленный» набор данных

ID	Time	Variable	Value
1	1	X1	5
1	2	X1	3
2	1	X1	6
2	2	X1	2
1	1	X2	6
1	2	X2	5
2	1	X2	1
2	2	X2	4

Обратите внимание на то, что вы должны указать переменные, которые необходимы для уникального обозначения каждого измерения (ID и Time) и на то, что переменная, в которой записаны названия столбцов с измерениями (X1 и X2), создается автоматически.

Теперь, когда ваши данные «расплавлены», им можно придать любую форму, используя функцию `cast()`.

Придание данным формы

Функция `cast()` применяется к «расплавленным» данным и придает им заданную форму, используя введенную формулу и еще функцию (это не обязательно) для агрегирования данных. Формат таков:

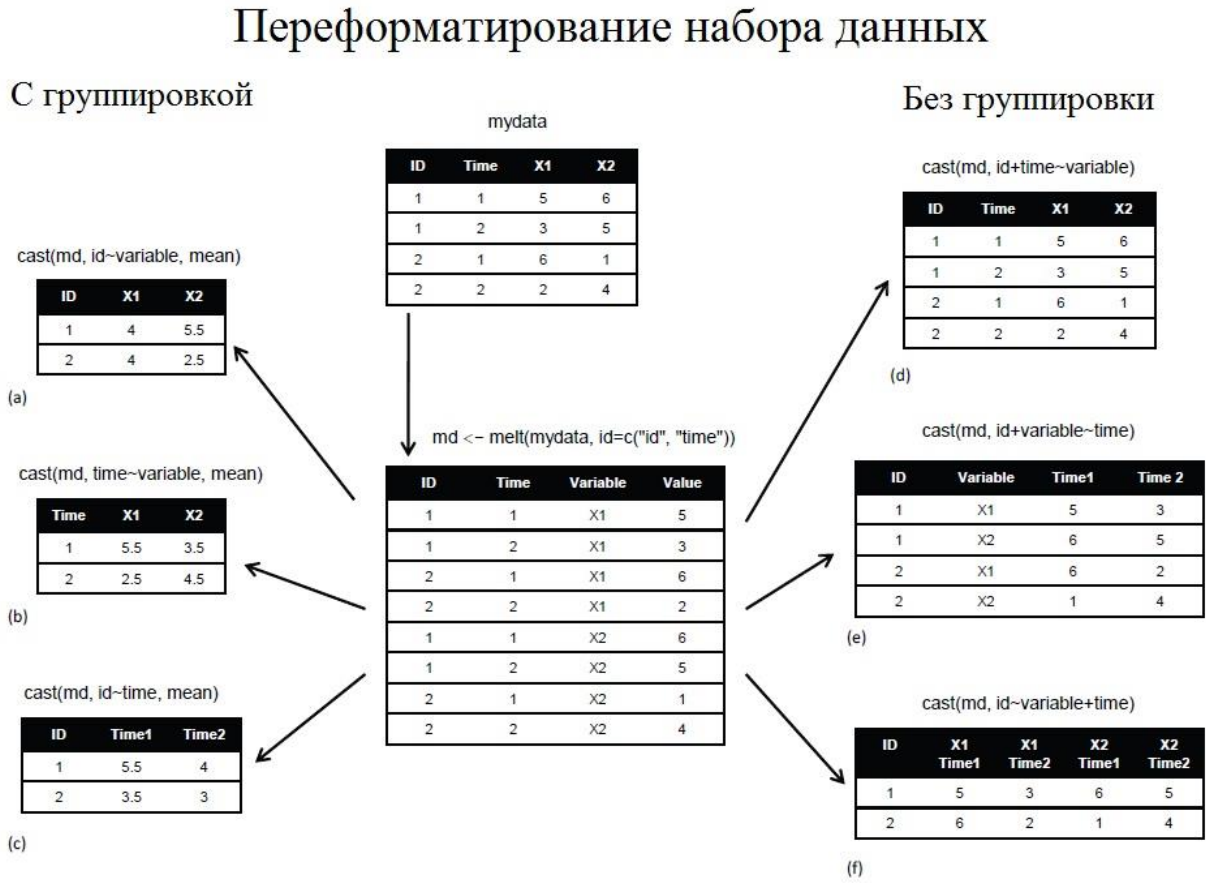
```
newdata <- cast(md, formula, FUN)
```

где *md* – это «расплавленные» данные, *formula* описывает желаемый результат, *FUN* – необязательная функция для агрегирования. Формула имеет вид

```
rowvar1 + rowvar2 + … ~ colvar1 + colvar2 + …
```

В такой формуле *rowvar1 + rowvar2 + …* задает набор переменных, которые определяют строки, а *colvar1 + colvar2 + …* – набор переменных, определяющих столбцы. Примеры представлены на рис. 5.1.

Рис. 5.1. Переформатирование данных при помощи функций `melt()` и `cast()`



Поскольку формулы, представленные в правой части рисунка (d, e и f), не содержат функции, данные просто переформатируются. Напротив, в примерах, расположенных слева (a, b и c) в качестве агрегирующей функции задано вычисление среднего арифметического. Так что данные не только переформатируются, но и агрегируются. Например, в случае (a) вычисляются средние значения переменных X1 и X2 для каждого значения переменной ID. В случае (b) приведены средние значения этих переменных для каждого значения переменной Time. В случае (c) усреднены значения переменных X1 и X2 для каждой комбинации значений переменных ID и Time.

Вы можете видеть, что гибкость применения функций `melt()` и `cast()` просто изумительна. Часто бывает так, что данные перед началом анализа приходится переформатировать или агрегировать. Например, часто бывает нужно преобразовать данные в так называемый «длинный формат», как в табл. 5.9, когда анализируются повторные измерения (данные, в которых каждое наблюдение представлено множественными измерениями). Пример анализа таких данных приведен в разделе 9.6.

5.7. Резюме

В этой главе рассмотрены десятки математических, статистических и вероятностных функций, которые используются для работы с данными. Мы узнали, как применять эти функции к широкому спектру объектов с данными, включая вектора, матрицы и таблицы данных. Мы познакомились с конструкциями, управляющими выполнением функций, для повторного выполнения одной и той же функции или выполнении команд только при определенных условиях. У вас была возможность создать свои собственные команды и применить их к данным. Наконец, мы узнали способы компактизации, агрегирования и переформатирования данных.

Теперь вы вооружены всеми средствами, чтобы преобразовать ваши данные в нужный вид. Мы готовы распрощаться с первой частью книги и попасть в волнующий мир анализа данных! В следующих главах мы начнем исследовать разнообразные статистические и графические методы для извлечения информации из данных.

Часть 2. Базовые методы

В первой части мы изучали программную среду R и узнавали, как получить данные из самых разных источников, комбинировать и изменять эти данные, подготавливая их к дальнейшему анализу. Как только ваши данные импортированы в программу и подготовлены к обработке, следующий этап, как правило, – это изучить каждую переменную отдельно. В результате можно узнать, как распределены данные. Это помогает узнать характеристики выборки, обнаружить неожиданные или проблемные значения и выбрать подходящие статистические методы. Затем, как правило, переменные изучаются попарно. При этом обнаруживаются основные связи между переменными и начинается построение более сложных моделей.

Вторая часть посвящена графическим и статистическим методам получения основных сведений о данных. В шестой главе описаны методы визуализации распределения отдельных переменных. Для категориальных данных это столбчатые и круговые диаграммы, а также появившаяся недавно веерная диаграмма. Для числовых переменных используются гистограммы, диаграммы распределения плотности, «ящики с усами», точечные диаграммы и менее известные «скрипичные диаграммы». Все эти диаграммы нужны для того, чтобы понять, как распределены значения одной переменной.

В седьмой главе описаны статистические методы для вычисления общих характеристик отдельных переменных и взаимосвязей между парами переменных. Глава начинается с описания того, как находить общие характеристики числовых данных (как для всего набора данных, так и для его части). Затем описано построение сводных и частотных таблиц для характеристики категориальных данных. Глава заканчивается рассмотрением основных способов изучения взаимосвязи между двумя переменными, включая корреляцию, тест хи-квадрат, тест Стьюдента и непараметрические методы.

После прочтения второй части книги вы сможете использовать основные графические и статистические методы в R для описания своих данных, исследования различий между группами и обнаружения значимых взаимосвязей между переменными.

Глава 6. Базовые диаграммы

В этой главе:

- столбчатые, точечные диаграммы и «ящики с усами»;
- круговые и веерные диаграммы;
- гистограммы и диаграммы распределения плотности.

Когда бы мы не анализировали данные, первое что мы должны сделать – это *посмотреть* на них. Какие самые частые значения у каждой переменной? Каков разброс данных? Есть ли какие-нибудь необычные наблюдения? В R реализовано множество функций для визуализации данных. В этой главе мы рассмотрим диаграммы, которые построены на основании одной категориальной или непрерывной переменной. Такие диаграммы могут отображать распределение данных или помогать сравнивать группы данных. В обоих случаях переменная может быть непрерывной (например, расход топлива машиной в милях на галлон) или категориальной (например, результат лечения: отсутствует, небольшой, заметный). В следующих главах мы познакомимся с диаграммами, которые отображают взаимосвязи между двумя или многими переменными.

Разделы этой главы посвящены столбчатым, круговым и веерным диаграммам, гистограммам, диаграммам распределения плотности, «ящикам с усами», скрипичным и точечным диаграммам. Какие-то из них могут быть уже знакомы вам, а какие-то (такие как скрипичные или веерные диаграммы) окажутся новыми. Нашей задачей будет, как всегда, лучше понять ваши данные и поделиться своими открытиями с другими людьми.

Давайте начнем со столбчатых диаграмм.

6.1. Столбчатые диаграммы

Столбчатые диаграммы (bar plot) отражают распределение (частоту разных значений) категориальной переменной в виде вертикальных или горизонтальных столбиков. Самый простой способ применить функцию `barplot()` таков:

`barplot(`*`height`*`)`

где *`height`* – это вектор или матрица.

В приведенных ниже примерах мы будем демонстрировать результаты исследования нового метода лечения ревматоидного артрита. Соответствующие данные содержатся в таблице данных `Arthritis`, распространяемой с пакетом `vcd`. Поскольку этот пакет не входит в базовую версию R, убедитесь, что вы скачали и установили его перед тем, как использовать его в первый раз (`install.packages("vcd")`).

Учтите, что пакет `vcd` не нужен для построения столбчатых диаграмм. Мы загружаем его для того, чтобы получить доступ к таблице данных `Arthritis`. Зато этот пакет понадобится нам для создания спинограмм, которые описаны в разделе 6.1.5.

6.1.1. Простые столбчатые диаграммы

Если переменная *`height`* – вектор, его значения определяют высоту столбцов, и создается вертикальная столбчатая диаграмма. Добавление параметра `horiz=TRUE` позволяет получить горизонтальную столбчатую диаграмму. Также можно добавить создающие надписи параметры. Параметр `main` определяет название диаграммы, тогда как параметры `xlab` и `ylab` добавляют подписи по оси x и оси y соответственно.

В таблице данных по артриту в переменной `Improved` указана реакция пациентов на плацебо или лекарство.

```
> library(vcd)
> counts <- table(Arthritis$Improved)
> counts
  None   Some Marked
    42     14     28
```

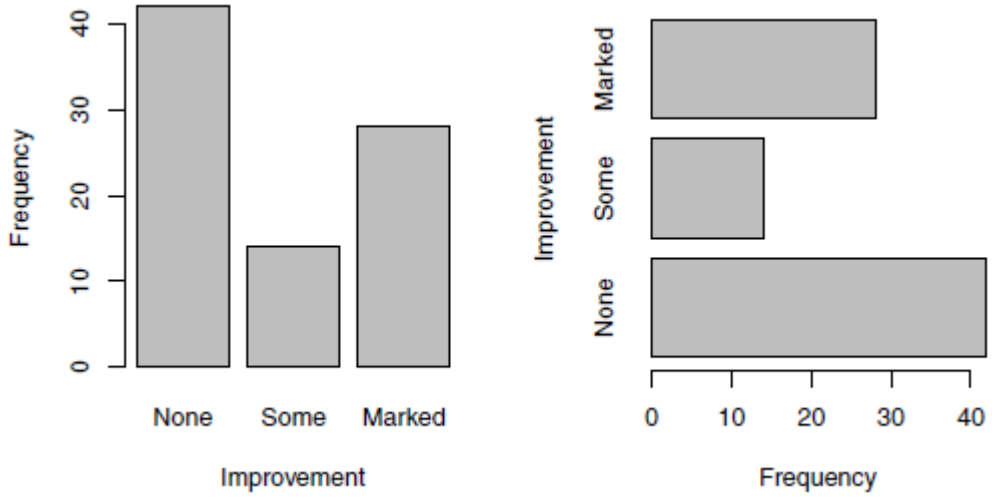
Здесь видно, что здоровье 28 пациентов заметно улучшилось (Marked), 14 пациентам стало немного лучше (Some) и 42 человека так и не поправились (None). В седьмой главе мы более подробно обсудим применение функции `table()` для подсчета значений в ячейках.

Значения переменной `counts` можно графически отобразить в виде вертикальной или горизонтальной столбчатой диаграммы. Программный код для выполнения этой задачи приведен ниже, а результат изображен на рис. 6.1.

Программный код 6.1. Простые столбчатые диаграммы

```
barplot(counts,      ←Простая столбчатая диаграмма
        main="Simple Bar Plot",
        xlab="Improvement", ylab="Frequency")
barplot(counts,      ←Горизонтальная столбчатая диаграмма
        main="Horizontal Bar Plot",
        xlab="Frequency", ylab="Improvement",
        horiz=TRUE)
```

Рис. 6.1. Простые вертикальная и горизонтальная столбчатые диаграммы



Совет. *Если категориальная переменная, которую вы хотите изобразить графически, представляет собой фактор или упорядоченный фактор, можно быстро создать вертикальную диаграмму при помощи функции* `plot()`. *Поскольку переменная* `Arthritis$Improved` *– фактор, программный код*

```
plot(Arthritis$Improved, main="Simple Bar Plot",
     xlab="Improved", ylab="Frequency")
plot(Arthritis$Improved, horiz=TRUE, main="Horizontal Bar Plot",
     xlab="Frequency", ylab="Improved")
```

позволит получить такие же диаграммы, как код 6.1. При этом вам не придется сводить значения в таблицу при помощи функции `table()`.

Что произойдет, если значения фактора слишком длинные? В разделе 6.1.4 вы увидите, как оптимизировать значения, чтобы они не перекрывались на диаграмме.

6.1.2. Столбчатые диаграммы с вертикальным и горизонтальным разбиением на подгруппы

Если `height` – матрица, а не вектор, для нее будет построена столбчатая диаграмма с вертикальным (stacked bar plot) или горизонтальным (grouped bar plot) разбиением на подгруппы. Если `beside=FALSE` (по умолчанию), то каждой переменной матрицы соответствует столбец диаграммы, а значения переменной задают высоту «наслоенных» друг на друга частей этого столбца. Если `beside=TRUE`, то каждой переменной соответствует группа столбцов диаграммы, а значения переменной отражены в расположенных рядом столбцах этой группы.

Рассмотрим сведенные в таблицу данные о реакции пациента на разные способы лечения:

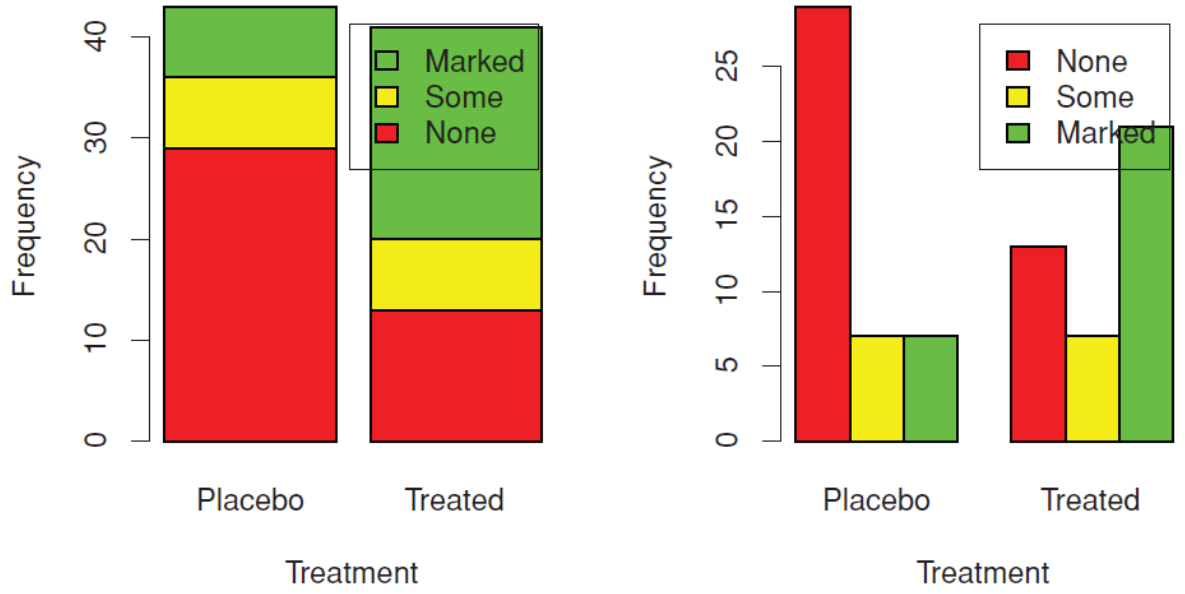
```
> library(vcd)
> counts <- table(Arthritis$Improved, Arthritis$Treatment)
> counts
        Treatment
Improved Placebo Treated
  None        29      13
  Some         7       7
  Marked       7      21
```

Эти результаты можно представить в виде столбчатой диаграммы с вертикальным или горизонтальным разбиением на подгруппы (см. приведенный ниже программный код). Получившиеся диаграммы представлены на рис. 6.2.

Программный код 6.2. Столбчатые диаграммы с вертикальным и горизонтальным разбиением на подгруппы

```
barplot(counts, ←Столбчатая диаграмма с вертикальным разбиением на подгруппы
        main="Stacked Bar Plot",
        xlab="Treatment", ylab="Frequency",
        col=c("red", "yellow","green"),
        legend=rownames(counts))
barplot(counts, ←Столбчатая диаграмма с горизонтальным разбиением на подгруппы
        main="Grouped Bar Plot",
        xlab="Treatment", ylab="Frequency",
        col=c("red", "yellow", "green"),
        legend=rownames(counts), beside=TRUE)
```

Рис. 6.2. Столбчатые диаграммы с вертикальным и горизонтальным разбиением на подгруппы



Первая команда позволяет получить столбчатую диаграмму с вертикальным разбиением на подгруппы («слоистую» диаграмму), а вторая – с горизонтальным. Мы также добавили опцию `col`, чтобы раскрасить столбцы в разные цвета. Параметр `legend` выводит подписи к столбцам в условные обозначения (это полезно только, если `height` – матрица).

В третьей главе мы обсуждали, как располагать условные обозначения на диаграмме наилучшим образом. Посмотрим, сможете ли вы сделать так, чтобы условные обозначения на этих диаграммах не перекрывались со столбцами.

6.1.3. Столбчатые диаграммы для средних значений

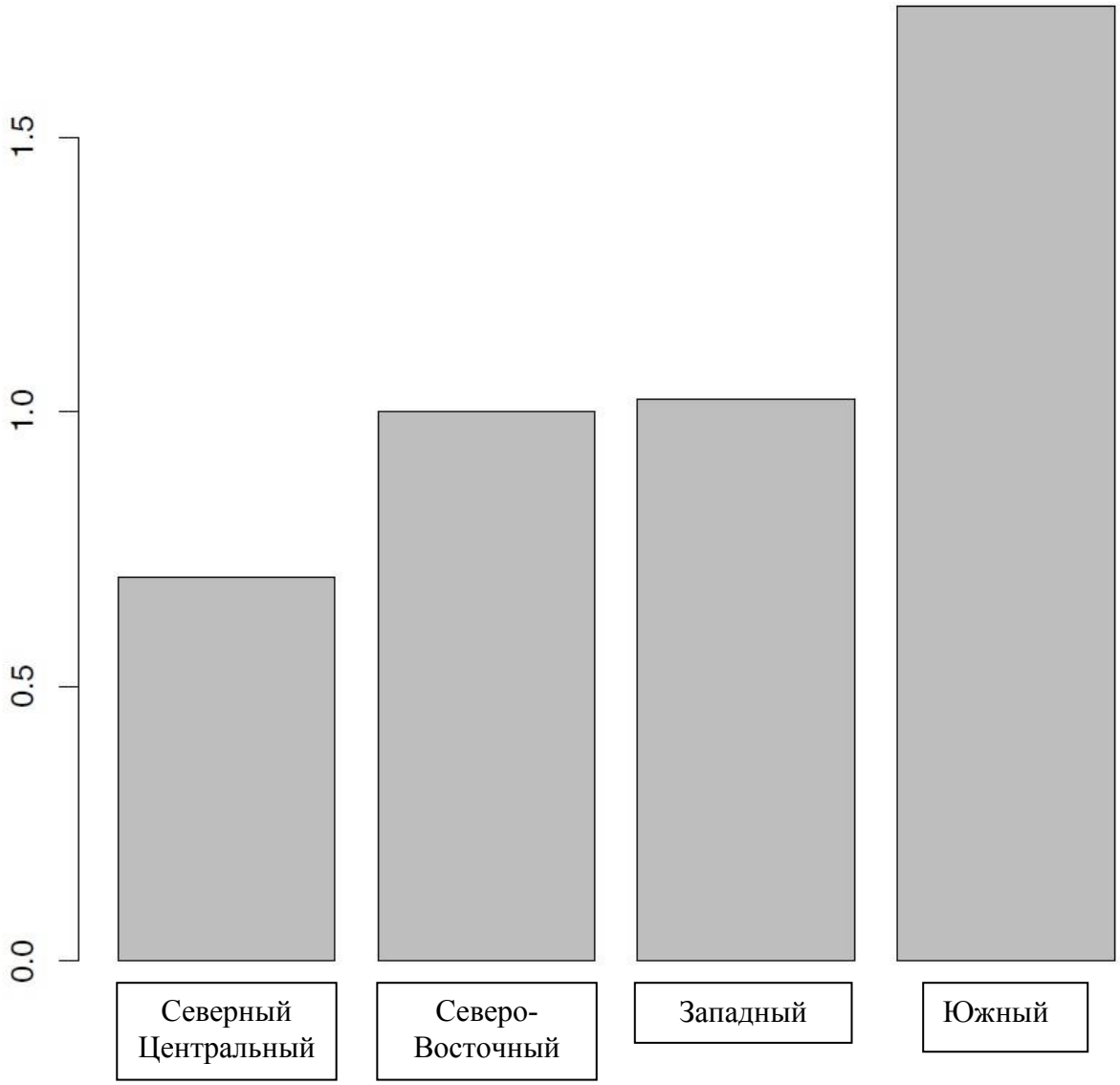
Столбчатая диаграмма не обязательно отражает число или частоту значений. Можно строить столбчатые диаграммы для средних значений, медиан, стандартных отклонений и так далее, используя результаты действия функции `aggregate()` как данные для функции `barplot()`. Представленный программный код содержит пример такой диаграммы, представленной на рис. 6.3.

Программный код 6.3. Столбчатая диаграмма с отсортированными средними значениями

```
> states <- data.frame(state.region, state.x77)
> means <- aggregate(states$Illiteracy, by=list(state.region), FUN=mean)
> means
        Group.1    x
1     Northeast 1.00
2         South 1.74
3 North Central 0.70
4          West 1.02
> means <- means[order(means$x),] ←❶ Сортируем средние значения по возрастанию
> means
        Group.1    x
3 North Central 0.70
1     Northeast 1.00
4          West 1.02
2         South 1.74
> barplot(means$x, names.arg=means$Group.1) ←❷Добавляем название
> title("Mean Illiteracy Rate")
```

Программный код 6.3 позволяет отсортировать средние значения по возрастанию ❶. Отметим, что команда `title()` ❷ аналогична опции `main` в команде `plot`. Объект `means$x` – вектор, содержащий информацию о высоте столбцов, опция `names.arg=means$Group.1` создает подписи для столбцов.

Рис. 6.3. Столбчатая диаграмма для отсортированных в порядке возрастания средних значений неграмотности в разных частях США



Этот пример можно расширить. Столбцы можно соединить отрезками, используя команду `lines()`. Также можно построить столбчатую диаграмму для средних значений с наложенными на нее доверительными интервалами при помощи функции `barplot2()` из пакета `gplots`. Найдите примеры применения этой функции в руководстве «barplot2: усовершенствованные столбчатые диаграммы» (“barplot2: Enhanced Bar Plots”) на сайте графической галереи R (R Graph Gallery website: http://addictedtor.free.fr/graphiques).

6.1.4. Улучшение столбчатых диаграмм

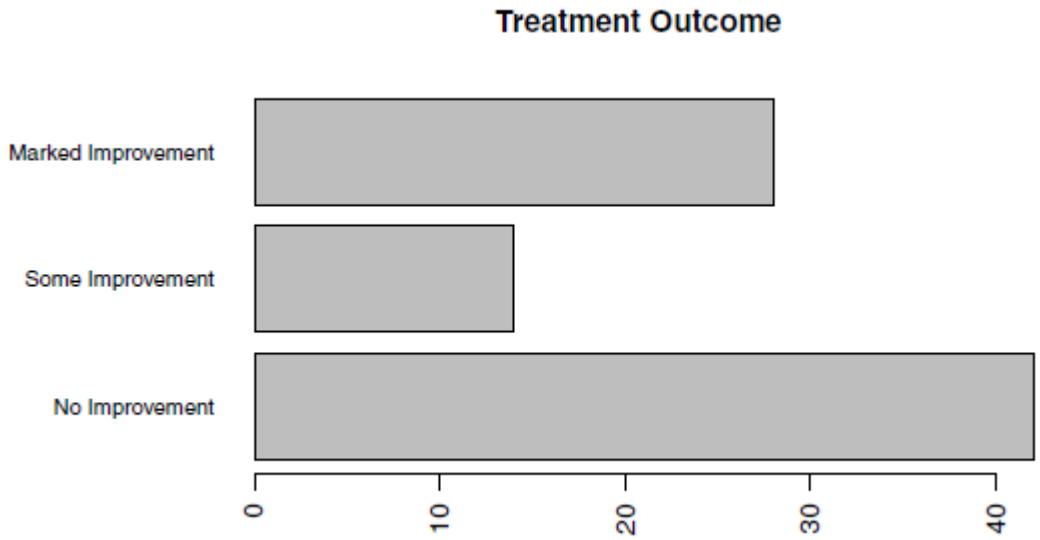
Есть несколько способов улучшения внешнего вида столбчатых диаграмм. К примеру, если у вас есть много столбцов, их подписи могут начать перекрываться. Можно уменьшить размер шрифта при помощи параметра `cex.names` (для достижения нужного эффекта его значения должны быть меньше единицы). В качестве альтернативы можно использовать параметр `names.arg`, который позволяет задать текстовый вектор с подписями для столбцов. Также для оптимизации расположения текста можно использовать графические параметры. Пример приведен в расположенном ниже программном коде, результат его действия представлен на рис. 6.4.

Программный код 6.4. Оптимизация расположения подписей на столбчатой диаграмме

```
par(mar=c(5,8,4,2))
par(las=2)
```

```
counts <- table(Arthritis$Improved)
barplot(counts,
           main="Treatment Outcome",
           horiz=TRUE, cex.names=0.8,
         names.arg=c("No Improvement", "Some Improvement",
                     "Marked Improvement"))
```

Рис. 6.4. Горизонтальная столбчатая диаграмма с оптимизированными подписями



В этом примере мы повернули столбцы диаграммы (`las=2`), изменили текст подписей, а также одновременно увеличили левого поля рисунка (при помощи `mar`) и уменьшили размер шрифта (`cex.names=0.8`), чтобы как следует разместить подписи. Функция `par()` позволяет существенно изменять создаваемые по умолчанию диаграммы. Более подробная информация по этой теме содержится в главе 3.

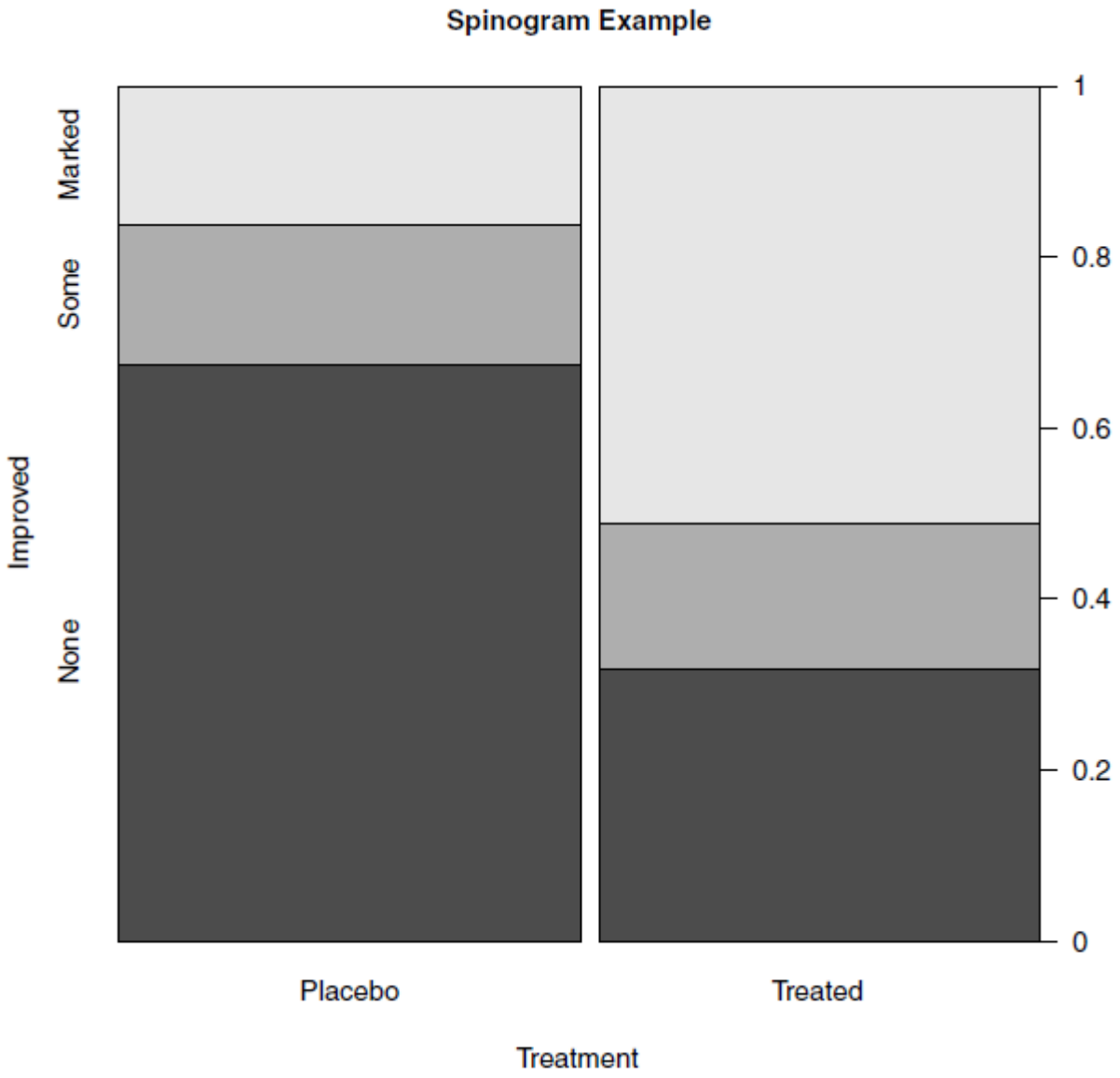
6.1.5. Спинограммы

Прежде чем завершить обсуждение столбчатых диаграмм давайте рассмотрим их специализированный вариант, который называется *спинограмма*. В этом случае масштаб столбчатой диаграммы с вертикальным расположением столбцов меняется так, что высота каждого результирующего столбца становится равной 1, а высоты его составляющих отражают пропорции. Спинограммы создаются при помощи команды `spine()` из пакета `vcd`. Такой программный код позволяет получить простую спинограмму:

```
library(vcd)
attach(Arthritis)
counts <- table(Treatment, Improved)
spine(counts, main="Spinogram Example")
detach(Arthritis)
```

Результат показан на рис. 6.5. Хорошо видно, что пациентов, состояние которых заметно улучшилось, заметно больше среди тех, кто получал настоящее лекарство, а не плацебо.

Рис. 6.5. Спинограмма, на которой видны результаты лечения артрита



Наряду со столбчатыми диаграммами, круговые диаграммы – это еще один распространенный способ визуализации категориальных данных. Мы рассмотрим их следом.

6.2. Круговые диаграммы

Несмотря на то, что круговые диаграммы широко используются в деловом мире, они критикуются большинством статистиков, включая тех, кто пишет помощь для R. Они рекомендуют применять столбчатые или точечные диаграммы вместо круговых диаграмм, поскольку людям легче сравнивать длины, чем площади и объемы. Возможно, именно из-за таких соображений возможности построения круговых диаграмм в R довольно сильно ограничены по сравнению с другими статистическими программами.

Круговые диаграммы создаются при помощи функции

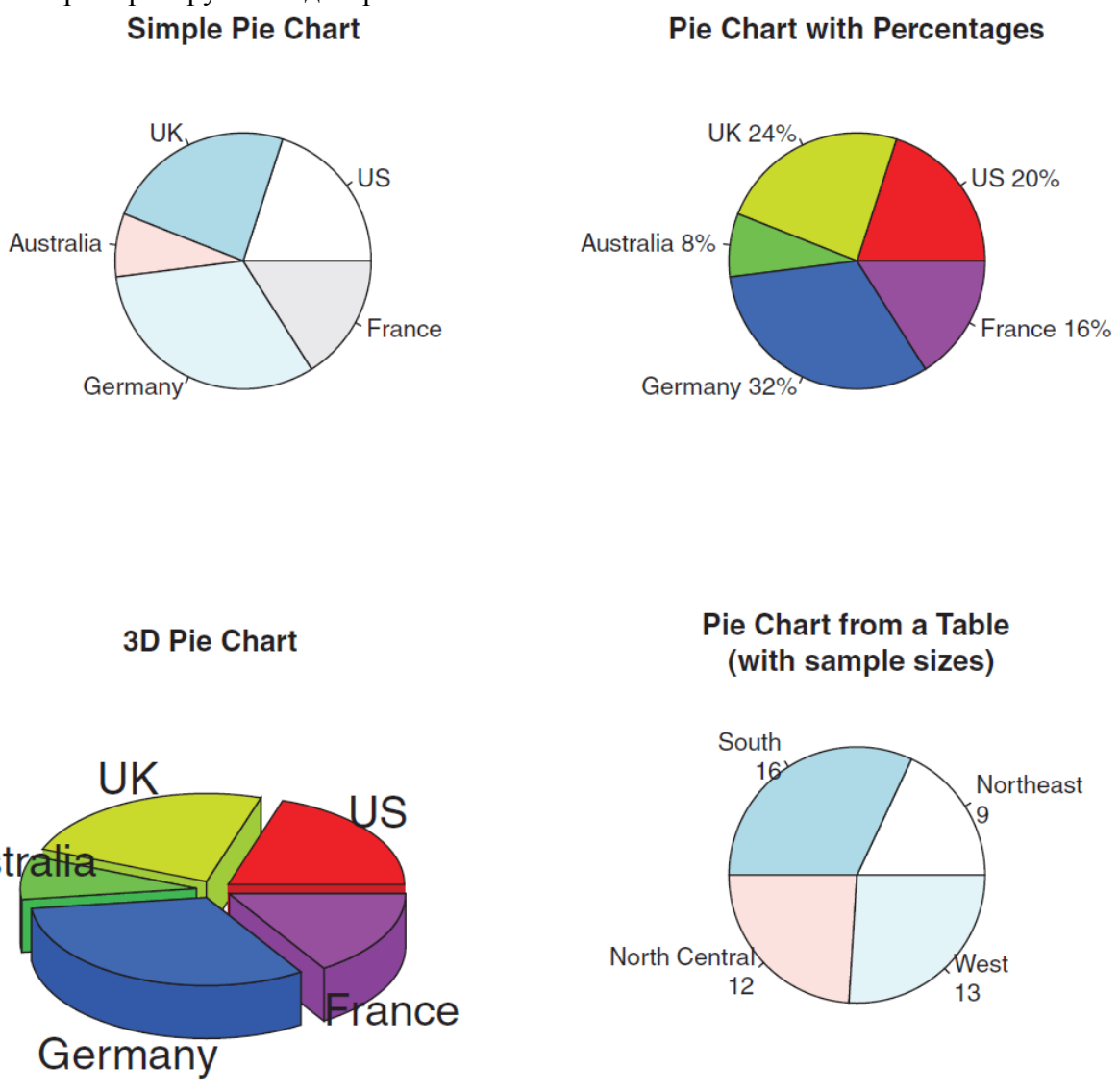
```
pie(x, labels)
```

где *x* – это лишенный отрицательных значений числовой вектор, определяющий площадь каждого сегмента диаграммы, а *labels* – текстовый вектор, содержащий подписи для сегментов. В представленном ниже программном коде содержатся четыре примера, результат представлен на рис. 6.6.

Программный код 6.5. Круговые диаграммы

```
par(mfrow=c(2, 2)) ←❶Объединяем четыре графика
slices <- c(10, 12,4, 16, 8)
lbls <- c("US", "UK", "Australia", "Germany", "France")

pie( slices, labels = lbls,
main="Simple Pie Chart")

pct <- round(slices/sum(slices)*100) ←❷Добавляем проценты к круговой
lbls2 <- paste(lbls, " ", pct, "%", sep="")                диаграмме
pie(slices, labels=lbls2, col=rainbow(length(lbls2)),
      main="Pie Chart with Percentages")

library(plotrix)
pie3D(slices, labels=lbls,explode=0.1,
      main="3D Pie Chart ")

mytable <- table(state.region)  ←❸ Делаем диаграмму из таблицы
lbls3 <- paste(names(mytable), "\n", mytable, sep="")
pie(mytable, labels = lbls3,
    main="Pie Chart from a Table\n (with sample sizes)")
```

Рис. 6.6. Примеры круговых диаграмм



Сначала вы задаете параметры диаграммы, так что все четыре элемента объединены на одной панели ❶ (объединение нескольких диаграмм в одну комбиированную рассмотрено в главе 3). Затем вы вводите данные, которые будут использованы в первых трех диаграммах.

Для второй круговой диаграммы ❷ вы преобразовываете размер выборок в проценты и добавляете эту информацию к подписям сегментов. Кроме того, сегменты этой диаграммы раскрашены с использованием функции `rainbow()`, описанной в главе 3. В нашем случае `rainbow(length(lbls2))` означает `rainbow(5)`, что задает пять цветов для диаграммы.

Третья диаграмма – это трехмерная круговая диаграмма, созданная при помощи команды `pie3D()` из пакета `plotrix`. Проверьте, что вы скачали и установили этот пакет перед первым использованием. Статистики, которые не любят круговые диаграммы, прямо-таки ненавидят их трехмерную версию (хотя, возможно, втайне считают ее симпатичной). Это происходит потому, что 3D-эффект не дает никакой дополнительной информации о данных, а только отвлекает внимание, как приятная глазу вещь.

На примере четвертой круговой диаграммы вы увидите, как построить ее на основе таблицы ❸. В этом случае вы подсчитываете число штатов в каждом регионе США и добавляете эту информацию к подписям сегментов перед созданием диаграммы.

На круговых диаграммах сложно сравнивать значения сегментов (если только они не приведены в подписях). К примеру, сможете ли вы сравнить США и Германию, глядя на простую круговую диаграмму? Если можете, то вы проницательней меня. В качестве попытки улучшить ситуацию, был разработан вариант круговой диаграммы под названием веерная диаграмма. Эта диаграмма (Lemon & Tyagi, 2009) предоставляет пользователю возможность представить графически и сами значения, и различия между ними. В R эту диаграмму можно построить при помощи функции `fan.plot()` из пакета `plotrix`.

Рассмотрим следующий программный код и полученную диаграмму (рис. 6.7):

```
library(plotrix)
slices <- c(10, 12,4, 16, 8)
lbls <- c("US", "UK", "Australia", "Germany", "France")
fan.plot(slices, labels = lbls, main="Fan Plot")
```

Рис. 6.7. Веерная диаграмма для данных по странам

На веерной диаграмме сегменты расположены так, что они налегают друг на друга, а их радиусы имеют такой размер, чтобы каждый сегмент был виден. Здесь можно видеть, что Германии соответствует самый большой сегмент и что размер сегмента США составляет примерно 60% от Германии. Франция составляет около половины от Германии и вдвое превосходит Австралию. Помните, что здесь важна *ширина* сегментов, а не их радиус.

Как вы можете увидеть, на веерной диаграмме гораздо легче сравнивать размер сегментов по сравнению с круговой диаграммой. Веерные диаграммы пока еще не успели войти в моду. Теперь, когда мы обсудили круговые и веерные диаграммы, давайте перейдем к гистограммам. В отличие от столбчатых и круговых диаграмм гистограммы предназначены для характеристики распределения значений непрерывных переменных.

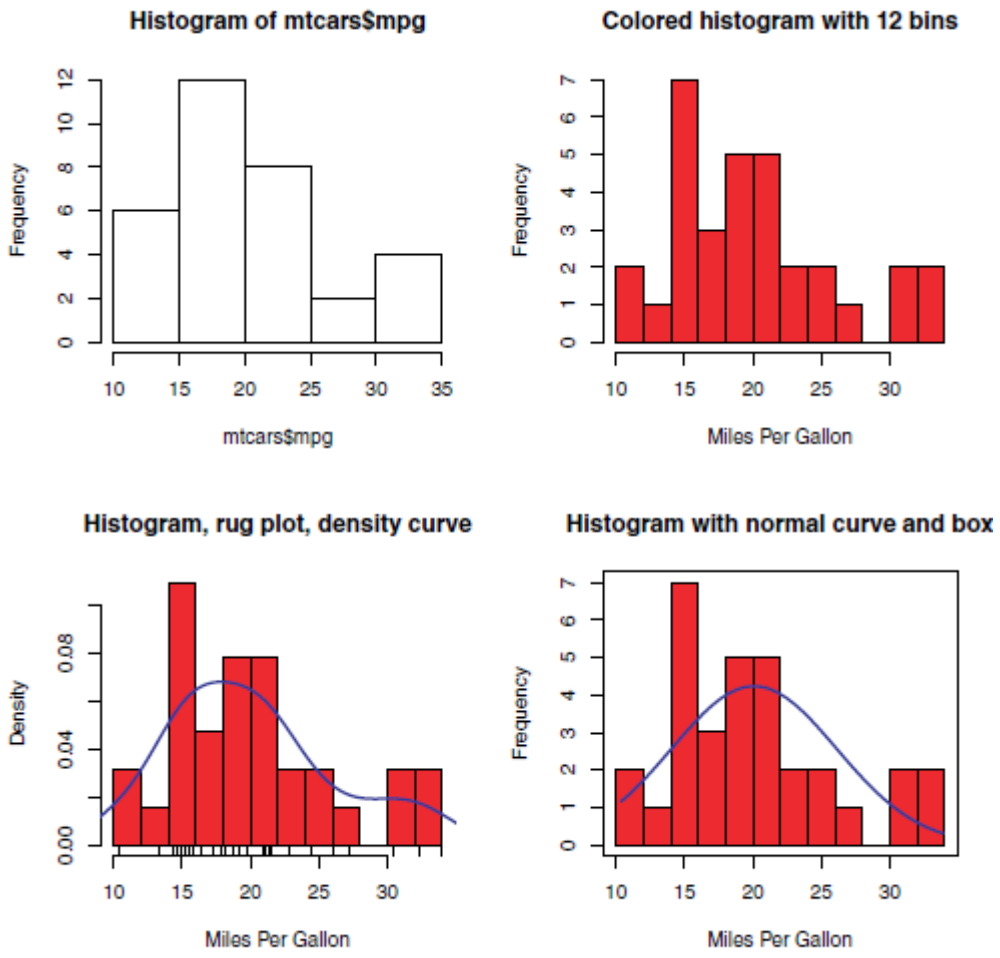
6.3. Гистограммы

Гистограммы графически отображают распределение значений непрерывных переменных, разделяя диапазон значений на заданное число отрезков по оси x и отображая частоту значений внутри каждого отрезка на оси y. Гистограмму можно создать при помощи команды
`hist(x)`
где *x* – это числовой вектор. Опция `freq=FALSE` позволяет построить гистограмму на основе плотности вероятности, а не частот значений. Параметр `breaks` определяет число отрезков. По умолчанию все отрезки имеют одинаковую длину. В программном коде 6.6 приведены примеры создания четырех вариантов гистограмм; результаты отражены на рис. 6.8.

Программный код 6.6. Гистограммы

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

hist(mtcars$mpg) ←❶ Простая гистограмма

hist(mtcars$mpg, ←❷ Гистограмма с заданным числом интервалов и
     breaks=12,                          раскрашенными столбцами
     col="red",
     xlab="Miles Per Gallon",
     main="Colored histogram with 12 bins")

hist(mtcars$mpg, ←❸ С кривой плотности распределения точек
     freq=FALSE,
     breaks=12,
     col="red",
     xlab="Miles Per Gallon",
     main="Histogram, rug plot, density curve")
rug(jitter(mtcars$mpg))
lines(density(mtcars$mpg), col="blue", lwd=2)

x <- mtcars$mpg  ←❹ С кривой нормального распределения и в рамочке
h<-hist(x,
        breaks=12,
        col="red",
        xlab="Miles Per Gallon",
        main="Histogram with normal curve and box")
xfit<-seq(min(x), max(x), length=40)
yfit<-dnorm(xfit, mean=mean(x), sd=sd(x))
yfit <- yfit*diff(h$mids[1:2])*length(x)
lines(xfit, yfit, col="blue", lwd=2)
box()
```

Рис. 6.8. Примеры построения гистограмм



Первая гистограмма ❶ – это то, что получается по умолчанию, когда не указаны никакие параметры функции. В этом случае диапазон значений разбивается на пять отрезков, заголовок и подписи по осям создаются автоматически. Для второй гистограммы ❷ даны более информативные и привлекательные подписи, а также указано, что диапазон значений нужно разбить на 12 отрезков и покрасить столбики в красный цвет.

Третья гистограмма ❸ повторяет вторую во всем, что касается цветов, отрезков диапазона, подписей и надписи, но на нее еще наложена кривая плотности и график-щетка. Кривая плотности – это оценка плотности ядра, которая описана в следующем разделе. Она позволяет изобразить распределение значений в сглаженном виде. Можно использовать команду `lines()`, чтобы покрасить эту кривую в синий цвет и сделать ее в два раза толще, чем она нарисована по умолчанию. Наконец, график-щетка позволяет увидеть реальные значения переменной, положенные на прямую. Если в выборке много близких значений, можно немного сместить их друг относительно друга при помощи команды вроде этой:

```
rug(jitter(mtcars$mpag, amount=0.01))
```

Эта команда добавляет малое случайное число (постоянную случайную величину±`amount`) к каждому значению данных, чтобы избежать наложения точек на диаграмме.

Четвертая диаграмма ❹ похожа на вторую, только на нее еще наложена кривая нормального распределения и вокруг нарисована рамочка. Программный код, который позволяет добавить кривую нормального распределения, предложен Петером Дальгаардом (Peter Dalgaard) в справочной системе рассылки R. Рамочку можно нарисовать при помощи команды `box()`.

6.4. Диаграммы плотности ядра

В предыдущем разделе вы видели диаграмму плотности ядра, добавленную к гистограмме. С технической точки зрения оценка плотности ядра – это непараметрический метод оценки значений

функции вероятности плотности случайной переменной. Хотя математические выкладки выходят за рамки этого текста, в целом диаграммы плотности ядра – хороший способ изобразить распределение значений непрерывной переменной. Способ построения диаграммы плотности (которая не будет наложена на другую диаграмму) таков:

```
plot(density(x))
```

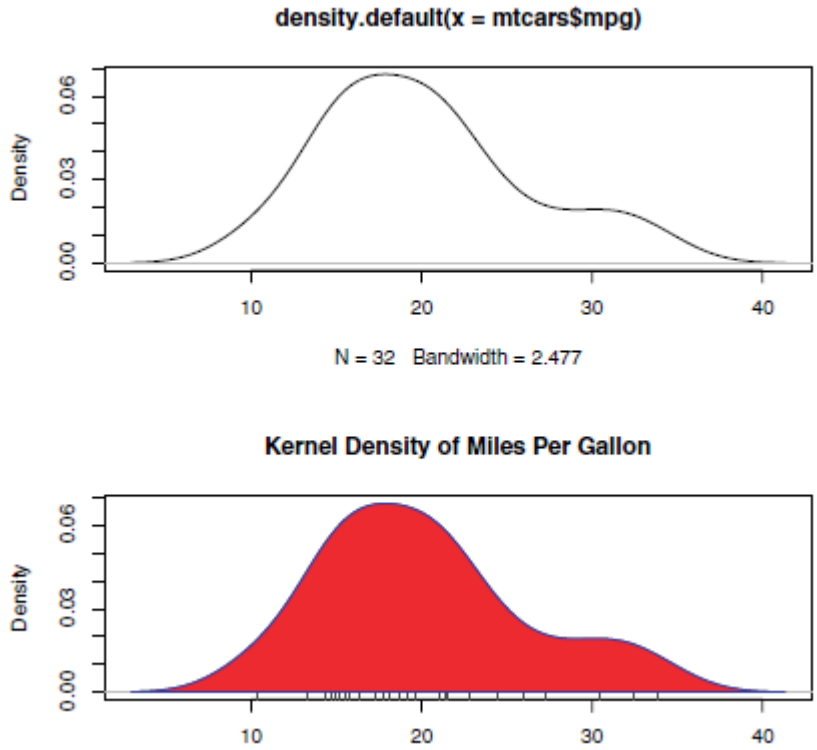
где *x* – это числовой вектор. Поскольку команда `plot()` начинает строить новую диаграмму, используйте команду `lines()` (программный код 6.6), когда вы хотите добавить кривую плотности к уже существующей диаграмме.

Два примера построения диаграмм плотности ядра даны в следующем программном коде, а результат представлен на рис. 6.9.

Программный код 6.7. Диаграммы плотности ядра

```
par(mfrow=c(2,1))
d <- density(mtcars$mpg)

plot(d)

d <- density(mtcars$mpg)
plot(d, main="Kernel Density of Miles Per Gallon")
polygon(d, col="red", border="blue")
rug(mtcars$mpg, col="brown")
```

Рис. 6.9. Диаграммы плотности ядра



На первой диаграмме вы видите минималистский вариант, то, что получается по умолчанию. На второй диаграмме вы добавляете название, красите кривую в синий цвет, пространство под ней заполняете красным и добавляете график-щетку с коричневыми делениями. Команда `polygon()` позволяет изобразить многоугольник, вершины которого задаются x и y координатами (в данном случае они определяются функцией `density()`).

Диаграммы плотности ядра можно использовать для сравнения групп. Этот подход применяется гораздо реже, чем он заслуживает, возможно, из-за отсутствия легко доступных программ. К счастью, пакет `sm` замечательно восполняет этот недостаток.

Функция `sm.density.compare()` пакета `sm` позволяет наложить друг на друга диаграммы плотности ядра для двух и более групп. Формат применения функции таков:
`sm.density.compare(`*`x, factor`*`)`
где *`x`* – это числовой вектор и *`factor`* – переменная, определяющая принадлежность к группе. Не забудьте установить пакет `sm` перед его первым использованием. Пример, в котором сравнивается расход топлива для машин с 4, 6 и 8 цилиндрами, приведен в программном коде 6.8.

Программный код 6.8. Сравнение диаграмм плотности ядра

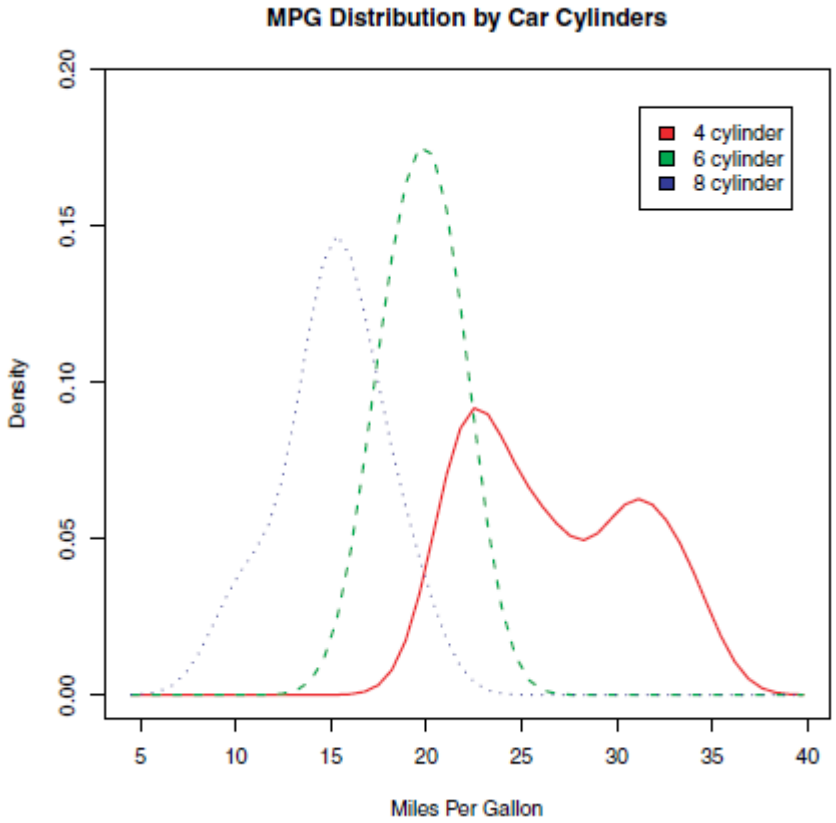
```
par(lwd=2) ←❶ Удваиваем ширину линий
library(sm)
attach(mtcars)

cyl.f <- factor(cyl, levels= c(4,6,8), ←❷ Создаем фактор с номером
              labels = c("4 cylinder", "6 cylinder",     группы
                         "8 cylinder"))

sm.density.compare(mpg, cyl, xlab="Miles Per Gallon") title(main="MPG
Distribution by Car Cylinders")←❸ Отображаем
colfill<-c(2:(1+length(levels(cyl.f))))   плотности распределения
                                                              точек
legend(locator(1), levels(cyl.f), fill=colfill)    ←❹
                                                  Добавляем легенду по щелчку мыши
detach(mtcars)
```

Функция `par()` используется, чтобы удвоить толщину линий на диаграмме (`lwd=2`), так их легче воспринимать читателям ❶. Пакет `sm` загружен и таблица данных `mtcars` активирована.

В таблице данных mtscars ❷ `cyl` – это числовая переменная, принимающая значения 4, 6, 8. Она преобразована в фактор, названный `cyl.f`, в котором содержатся подписи к диаграмме. Функция `sm.density.compare` создает диаграмму ❸, а функция `title()` добавляет ее название.

Наконец, вы добавляете легенду, чтобы сделать диаграмму понятнее ❹ (легенды обсуждались в главе 3). Сначала мы создаем вектор цветов. Здесь `colfill` равно `c (2, 3, 4)`. Затем добавляем легенду на диаграмму при помощи функции `legend()`. Опция `locator(1)` означает то, что вы разместите легенду на диаграмме в интерактивном режиме, указав курсором на то место, где она должна появиться. Вторая опция представляет собой текстовый вектор, содержащий подписи. Третья опция присваивает цвет из вектора `colfill` каждому уровню фактора `cyl.f`. Полученная диаграмма представлена на рис. 6.10.

Рис. 6.10. Диаграммы плотности ядра для расхода топлива у машин с разным числом цилиндров



Видно, что при помощи наложенных друг на друга диаграмм плотности ядра можно эффективно сравнить группы данных. Здесь видны и форма распределения значений в каждой группе, и степень перекрытия между группами. Мораль этой истории заключается в том, что у моей следующей машины будет восемь цилиндров – или аккумулятор.

Ящики с усами также замечательный (и гораздо более широко используемый) подход для визуализации распределения значений и различий для групп наблюдений. Следующими мы обсудим их.

6.5. Ящики с усами

Диаграммы типа «ящик-с-усами» (box plot) иллюстрируют распределение значений непрерывной переменной, отображая пять параметров: минимум, нижнюю квартиль (25% перцентиль), медиану (50% перцентиль), верхнюю квартиль (75% перцентиль) и максимум. На этой диаграмме также могут быть отображены вероятные выбросы (значения, выходящие за диапазон в ±1.5 межквартильного размаха, разности верхней и нижней квартилей). Например, команда:

```
boxplot(mtcars$mpg, main="Box plot", ylab="Miles per Gallon")
```

позволяет построить диаграмму, показанную на рис. 6.11. Я добавил подписи к элементам диаграммы вручную.

Рис. 6.11. Ящик с усами с добавленными вручную подписями

По умолчанию, каждый «ус» продолжается до самой минимального или максимального значения, которое не выходит за пределы 1.5 межквартильных размахов. Выходящие за эти пределы значения отмечаются точками (не показано).

Например, в нашей выборке данных о машинах медиана для расхода топлива составляет 19.2 миль на галлон, 50% значений попадают в диапазон между 15.3 и 22.8, наименьшее значение равно 10.4, а наибольшее – 33.9. Как это я смог считать значения с диаграммы с такой точностью? Команда `boxplot.stats(mtcars$mpg)` выводит значения статистик, которые использовались для построения диаграммы (иначе говоря, я схитрил). Здесь не наблюдается выбросов и есть небольшой сдвиг распределения в положительную сторону (верхний ус длиннее нижнего).

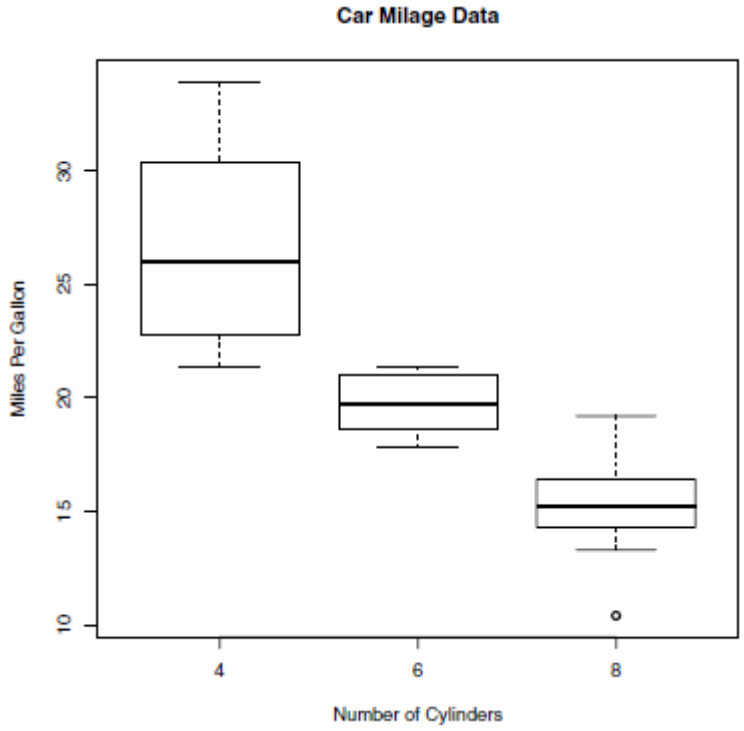
6.5.1. Использование расположенных рядом ящиков с усами для сравнения групп между собой

Ящики с усами можно построить для отдельных переменных или для групп переменных. Общий вид команды таков:
`boxplot(`*`formula`*`, data=`*`dataframe`*`)`
где *`formula`* – это формула, а *`dataframe`* обозначает таблицу данных (или список), где содержатся данные. Примером формулы может служить выражение `y~A`, в этом случае для каждого значения категориальной переменной `A` будет построен отдельный ящик с усами для числовой переменной `y`. Формула `y~A*B` позволит получить отдельный ящик с усами для числовой переменной `y` для каждой комбинации значений категориальных переменных `A` и `B`.

Параметр `varwidth=TRUE` позволяет получить диаграмму, на которой ширина «ящиков» будет пропорциональна квадратному корню из размера выборки. Используйте параметр `horizontal=TRUE`, чтобы поменять оси местами. В размещенном ниже программном коде мы возвращается к сравнению расхода топлива автомобилями с четырьмя, шестью и восемью цилиндрами, теперь уже при помощи расположенных рядом ящиков с усами. Диаграмма представлена на рис. 6.12.

```
boxplot(mpg ~ cyl, data=mtcars,
        main="Car Mileage Data",
        xlab="Number of Cylinders",
        ylab="Miles Per Gallon")
```

Рис. 6.12. Ящики с усами для расхода топлива автомобилями в зависимости от числа цилиндров



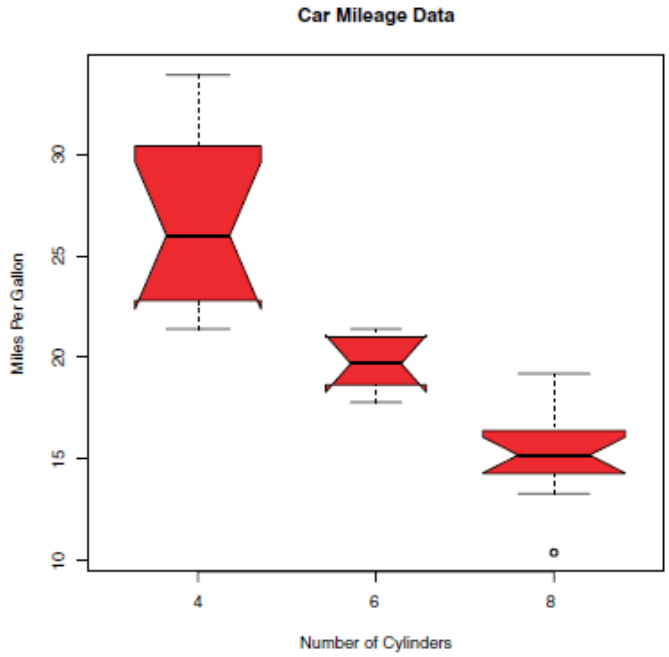
На рис. 6.12 можно увидеть, что автомобили с разным числом цилиндров хорошо различаются по расходу топлива. Также ясно, что распределение значений расхода топлива у машин с шестью цилиндрами более симметрично по сравнению с четырьмя- и восьмицилиндровыми машинами. Для автомобилей с четырьмя цилиндрами характерен б**о**льший разброс (и положительный сдвиг распределения) значений расхода топлива. В группе с восьмицилиндровыми двигателями есть выброс.

Ящики с усами – очень многообразны. Если добавить параметр `notch=TRUE`, то получатся выемчатые ящики с усами. Если выемки двух ящиков не перекрываются, высока вероятность того, что медианы соответствующих выборок действительно различаются (Chambers et al., 1983, стр. 62). Этот программный код позволяет получить выемчатые ящики с усами для нашего примера с расходом топлива:

```
boxplot(mpg ~ cyl, data=mtcars,
        notch=TRUE,
        varwidth=TRUE,
        col="red",
        main="Car Mileage Data",
        xlab="Number of Cylinders",
        ylab="Miles Per Gallon")
```

Опция `col` позволяет окрасить ящики в красный цвет, а параметр `varwidth=TRUE` делает ширину ящиков пропорциональной размеру выборок. На рис. 6.13 видно, что медианные значения расхода топлива у четырех-, шести- и восьмицилиндровых машин различаются. Расход топлива заметно уменьшается с увеличением числа цилиндров.

Рис. 6.13. Выемчатые ящики с усами для расхода топлива автомобилями с разным числом цилиндров



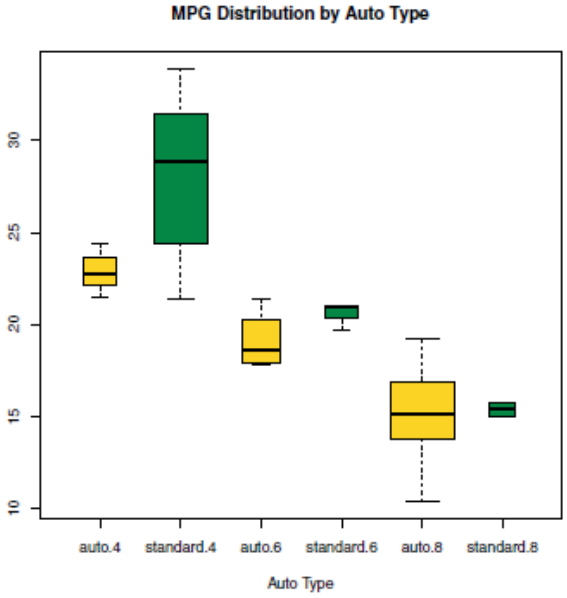
Наконец, можно создавать ящики с усами для нескольких группирующих переменных. В программном коде 6.9 показано, как построить ящики с усами для машин с разным числом цилиндров и разными коробками передач. Мы вновь используем параметр `col` для окраски «ящиков». Обратите внимание, цвета чередуются. В данном случае у нас есть шесть ящиков и только два заданных цвета, так что каждый цвет повторяется три раза.

Программный код 6.9. Ящики с усами для всех сочетаний значений двух факторов

```
mtcars$cyl.f <- factor(mtcars$cyl,←Представляем число цилиндров в виде
levels=c(4,6,8),                                                фактора
                       labels=c("4","6","8"))
mtcars$am.f <- factor(mtcars$am,← Представляем тип передачи в виде
                      levels=c(0,1),                    фактора
                      labels=c("auto", "standard"))
boxplot(mpg ~ am.f *cyl.f,       ← Создаем «ящик с усами»
        data=mtcars,
        varwidth=TRUE,
        col=c("gold","darkgreen"),
        main="MPG Distribution by Auto Type",
        xlab="Auto Type")
```

Полученная диаграмма представлена на рис. 6.14.

Рис.6.14. Ящики с усами, показывающие расход топлива для всех комбинаций числа цилиндров и типа коробки передач



На этой диаграмме опять же прекрасно видно, что медианное значение расхода топлива уменьшается с возрастанием числа цилиндров. Для четырех- и шестицилиндровых автомобилей расход топлива выше с ручной (standart) коробкой передач, а для восьмицилиндровых машин такой разницы не наблюдается. Ширина «ящиков» говорит о том, что четырехцилиндровые машины с ручной коробкой передач и восьмицилиндровые автомобили с автоматической коробкой передач преобладают в данной выборке.

6.5.2. Скрипичные диаграммы

Прежде чем мы закончим обсуждать ящики с усами, нам стоит рассмотреть их модификацию, названную скрипичной диаграммой (violin plot). Это сочетание ящика с усами и диаграммы плотности ядра. Эту диаграмму можно создать при помощи функции `vioplot()` из пакета `vioplot`. Не забудьте установить этот пакет перед первым использованием.

Формат применения функции `vioplot()` таков:

`vioplot(`*`x1`*`, `*`x2`*`, … , names=, col=)`

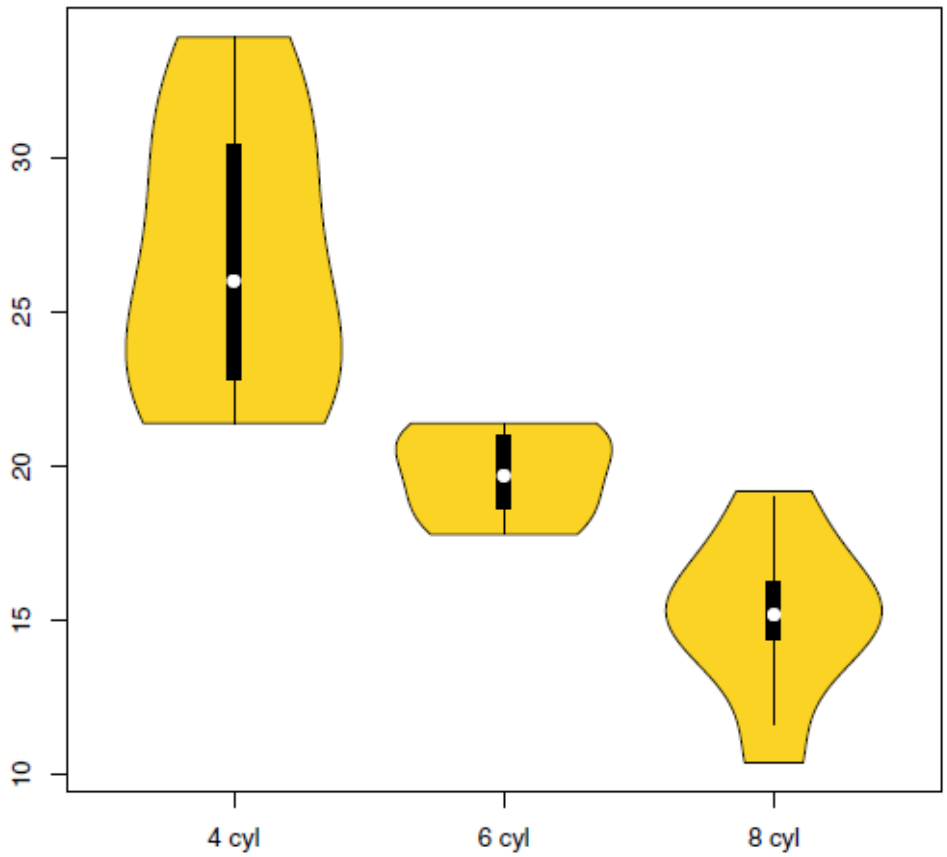
где *`x1`*`, x2, …` – это один или более числовых векторов, которые нужно изобразить графически (для каждого вектора будет построена своя скрипичная диаграмма). Параметр `names` задает текстовый вектор с подписями для диаграмм, `col` – вектор, содержащий названия цветов каждой диаграммы. Пример использования функции приведен в следующем программном коде.

Программный код 6.15. Скрипичные диаграммы

```
library(vioplot)
x1 <- mtcars$mpg[mtcars$cyl==4]
x2 <- mtcars$mpg[mtcars$cyl==6]
x3 <- mtcars$mpg[mtcars$cyl==8]
vioplot(x1, x2, x3,
        names=c("4 cyl", "6 cyl", "8 cyl"),
        col="gold")
title("Violin Plots of Miles Per Gallon")
```

Учтите, что для использования функции `violin()` нужно разделить группы наблюдений на отдельные переменные. Результат представлен на рис. 6.15.

Рис. 6.15. Скрипичные диаграммы для расхода топлива автомобилями с разным числом цилиндров



Скрипичные диаграммы представляют собой зеркально отраженные диаграммы плотности ядра, наложенные на ящики с усами. Здесь белая точка – медиана, черный прямоугольник – межквартильный размах, а тонкие черные линии – «усы». Внешний контур фигуры – это диаграмма плотности ядра. Скрипичные диаграммы еще не вошли в моду. Опять же, это может быть обусловлено отсутствием доступного программного обеспечения. Время покажет.

Мы закончим эту главу рассмотрением точечных диаграмм. На них, в отличие от всех рассмотренных нами до этого диаграмм, отображается каждое отдельное значение переменной.

6.6. Точечные диаграммы

Точечные диаграммы (dot plot) позволяют изобразить множество подписанных значений на простой горизонтальной шкале. Эта диаграмма создается при помощи функции `dotchart()`, примененной в таком формате:

```
dotchart(x, labels=)
```

где *x* – это числовой вектор, а параметр `labels` задает вектор, в котором содержатся подписи к каждой точке. Можно добавить параметр `groups`, чтобы назначить фактор, определяющий, как будут группироваться элементы вектора *x*. В этом случае параметр `gcolor` определяет, какого цвета будут подписи для разных групп, а параметр `cex` определяет размер подписей. Вот пример для набора данных `mtcars`:

```
dotchart(mtcars$mpg, labels=row.names(mtcars), cex=.7,
         main="Gas Mileage for Car Models",
         xlab="Miles Per Gallon")
```

Получившаяся диаграмма представлена на рис. 6.16.

Рис. 6.16. Точечная диаграмма для расхода топлива автомобилями разных марок

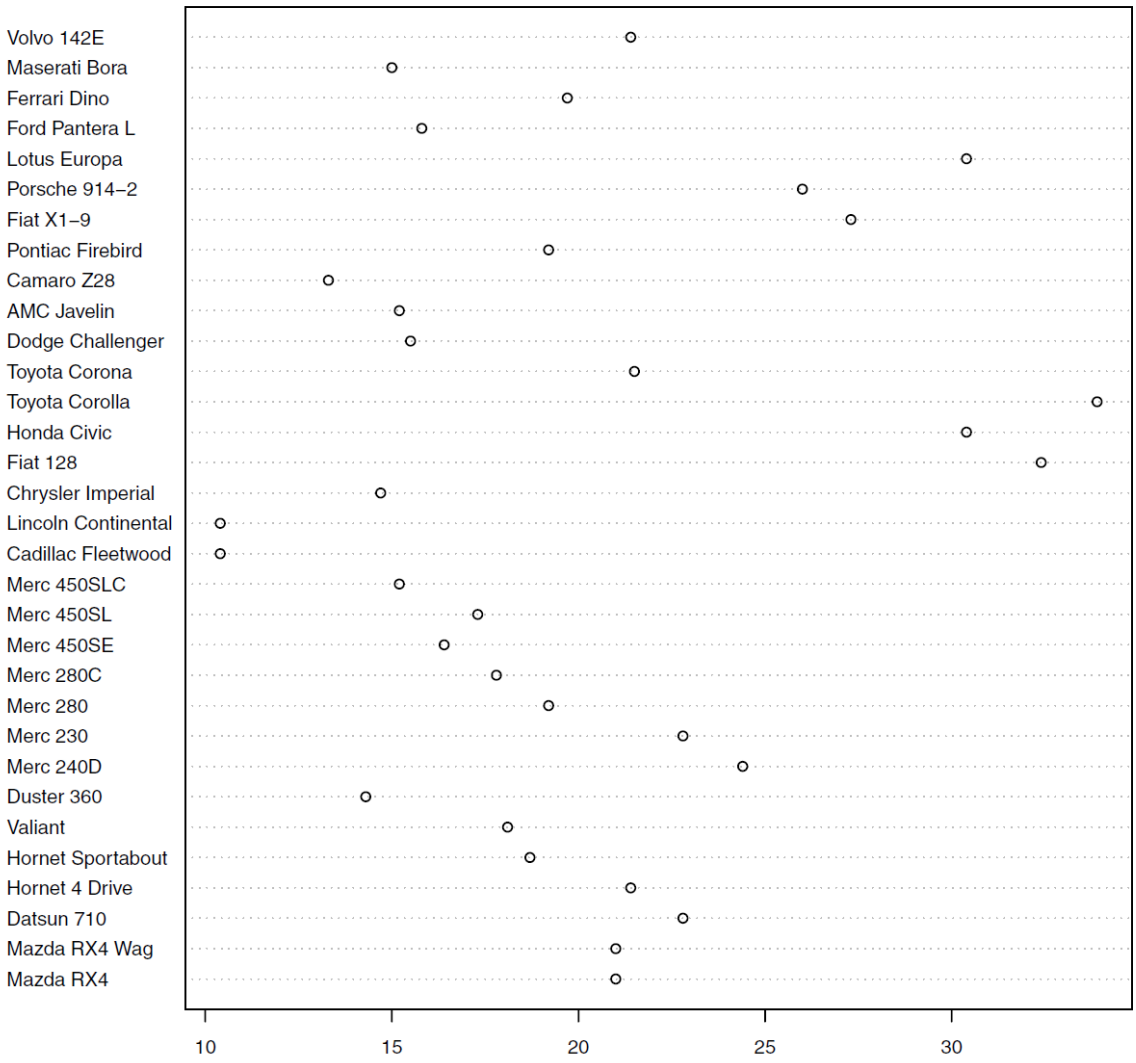


Диаграмма на рис. 6.16 позволяет увидеть расход топлива для каждой марки автомобилей на одной и той же горизонтальной оси. Обычно точечные диаграммы становятся более интересными, когда значения отсортированы, а каждой группе соответствуют свои символ и цвет. Пример приведен в следующем программном коде.

Программный код 6.11. Точечная диаграмма с отсортированными, сгруппированными и раскрашенными значениями

```
x <- mtcars[order(mtcars$mpg),]
x$cyl <- factor(x$cyl)
x$color[x$cyl==4] <- "red"
x$color[x$cyl==6] <- "blue"
x$color[x$cyl==8] <- "darkgreen"
dotchart(x$mpg,
         labels = row.names(x),
         cex=.7,
         groups = x$cyl,
         gcolor = "black",
         color = x$color,
         pch=19,
         main = "Gas Mileage for Car Models\ngrouped by cylinder",
         xlab = "Miles Per Gallon")
```

В этом примере строки таблицы данных `mtcars` отсортированы по значениям расхода топлива (по возрастанию) и сохранены как таблица данных `x`. Числовой вектор `cyl` преобразован в фактор.

Текстовый вектор (`color`), принимающий значения "`red`" (красный), "`blue`" (синий) или "`dark green`" (темно-зеленый) в зависимости от значения фактора `cyl`, добавлен к таблице данных `x`. Для подписей точек взяты названия строк исходной таблицы данных (марки машин). Точки сгруппированы по числу цилиндров. Числа 4, 6 и 8 напечатаны черным. Цвета точек и подписей определяются вектором `color`, точки обозначены заполненными кружками. В результате получается диаграмма, представленная на рис. 6.17.

При рассмотрении этого рисунка многие вещи впервые становятся очевидными. Вновь мы видим увеличение расхода топлива с уменьшением числа цилиндров. Однако теперь заметны исключения. Например, восьмицилиндровый Pontiac Firebird имеет больший расход топлива, чем шестицилиндровые Mercury 280C и Valiant. Шестицилиндровый Hornet 4 Drive расходует столько же топлива, сколько четырехцилиндровый Volvo 142E. Также ясно, что Toyota Corolla – самая экономная машина, а Lincoln Continental и Cadillac Fleetwood – самые «прожорливые».

В этом примере из точечной диаграммы можно получить массу информации, поскольку все точки подписаны и расположены так, чтобы упростить их сопоставление. Однако с увеличением числа наблюдений точечные диаграммы становятся менее полезными.

Замечание. *Существует много разновидностей точечных диаграмм. Jacoby (2006) написал очень полезный обзор точечных диаграмм и приводит программные коды для реализации новых модификаций этих диаграмм в R. Кроме того, в пакете* `Hmisc` *реализована функция (находчиво названная* `dotchart2`*), которая позволяет строить точечные диаграммы со множеством дополнительных возможностей.*

6.7. Резюме

В этой главе мы узнали, как описывать непрерывные и категориальные переменные. Мы поняли, как столбчатые и (в меньшей степени) круговые диаграммы можно использовать для характеристики распределения значений категориальных переменных, и как столбчатые диаграммы с вертикальным и горизонтальным расположением столбцов помогают увидеть различия между разными группами категориальных значений. Мы также узнали, как при помощи гистограмм, диаграмм плотности ядра, ящиков с усами, графиков-щеток и точечных диаграмм можно визуализировать распределение значений непрерывных переменных. Наконец, мы исследовали, как использовать наложенные друг на друга диаграммы плотности ядра, расположенные рядом ящики с усами и точечные диаграммы с группированными значениями для демонстрации различий между группами значений непрерывной переменной.

В следующих главах мы расширим наши знания, познакомившись со способами графического представления сразу двух или многих переменных. Вы увидите, как одновременно изобразить взаимосвязи между многими переменными, используя такие методы как диаграммы рассеяния, многогрупповые графики, мозаичные диаграммы, кореллограммы, панельные диаграммы и т.д.

В следующей главе мы рассмотрим основные статистические методы, которые используются для численной характеристики распределения значений и взаимосвязей между двумя переменными. Мы также познакомимся с методами, которые позволяют понять, реальна ли взаимосвязь между переменными или она обнаружена из-за ошибки составления выборки.

Глава 7. Основные методы статистической обработки данных

В этой главе:

- описательные статистики;
- таблицы частот и таблицы сопряженности;
- корреляция и ковариация;
- тесты Стьюдента;
- непараметрические методы.

В предыдущих главах вы узнали, как импортировать данные в R и как использовать разнообразные функции для упорядочения данных и преобразования их в нужный формат. Затем мы рассмотрели базовые методы визуализации данных.

Обычно следующий шаг после упорядочения данных и их визуального исследования – это описание распределения значений каждой переменной при помощи числовых показателей. Затем, как правило, исследуют взаимосвязь между парами переменных. Цель этих процедур – ответить на вопросы вроде таких:

- Каков расход топлива у современных автомобилей? А именно, каково распределение значений расхода топлива (среднее, стандартное отклонение, медиана, размах значений и т.д.) для имеющихся у нас данных по разным маркам машин?
- Каков результат испытаний нового лекарства (нет улучшения, некоторое улучшение, заметное улучшение) по сравнению с плацебо? Зависит ли результат испытаний от пола пациентов?
- Какова корреляция между доходом и средней продолжительностью жизни? Отличается ли достоверно коэффициент корреляции от нуля?
- Правда ли, что вероятность сесть в тюрьму за преступление неодинакова в разных штатах США? Достоверны ли различия между штатами?

В этой главе мы рассмотрим команды R, которые позволяют вычислять описательные и предсказательные статистики. Для начала мы познакомимся с методами оценки центральной тенденции и разброса значений количественных переменных. Затем мы узнаем, как создавать таблицы частот и таблицы сопряженности (и проводить связанный с ними тест хи-квадрат) для категориальных переменных. Затем мы исследуем разные коэффициенты корреляции, применимые к непрерывным и порядковым переменным. Наконец, мы обратимся к исследованию различий между группами при помощи параметрических (тесты Стьюдента) и непараметрических (тесты Манна-Уитни и Краскела-Уоллиса) методов. Хотя мы сосредоточимся на числовых показателях, мы будем постоянно упоминать графические методы, которые уместно использовать для визуализации этих показателей.

С обсуждаемыми в этой главе статистическими методами люди обычно знакомятся на первых курсах ВУЗов. Если эти методы не знакомы вам, я рекомендую два замечательных пособия MacCall (2000) и Snedecor & Cochran (1989).[7] По каждой теме существует также множество источников в Интернете (таких как Wikipedia).

7.1. Описательные статистики

В этом разделе мы обсудим числовые характеристики центральной тенденции, разброса и типа распределения значений непрерывных переменных. Мы рассмотрим эту тему на примере нескольких переменных из набора данных о характеристиках разных марок автомобилей (Motor Trend Car Road Test, `mtcars`), с которыми вы познакомились еще в первой главе. Мы сосредоточимся на расходе топлива (в милях на галлон – miles per gallon, `mpg`), мощности (лошадиных сил – horsepower, `hp`) и весе (weight, `wt`).

[7] *На русском языке основные статистические методы кратко описаны в пособии П.А. Волковой и А.Б. Шипунова «Статистическая обработка данных в учебно-исследовательских работах» (Форум, 2012) – Прим. пер.*

```
> vars <- c("mpg", "hp", "wt")
> head(mtcars[vars])
                   mpg  hp   wt
Mazda RX4         21.0 110 2.62
Mazda RX4 Wag     21.0 110 2.88
Datsun 710        22.8  93 2.32
Hornet 4 Drive    21.4 110 3.21
Hornet Sportabout 18.7 175 3.44
Valiant           18.1 105 3.46
```

Для начала мы рассмотрим описательные статистики для всех 32 марок автомобилей в целом. Затем мы вычислим описательные статистики отдельно для каждого типа коробки передач (`am`) и числа цилиндров (`cyl`). Тип коробки передач – это дихотомическая переменная, которая принимает значения 0 (автоматическая коробка) и 1 (механическая коробка), число цилиндров может быть равным 4, 5 и 6.

7.1.1. Калейдоскоп методов

Когда дело доходит до вычисления описательных статистик, R потрясает воображение. Давайте начнем с функций, которые включены в базовую версию. Затем мы познакомимся с расширенными возможностями, которые становятся доступными при установке дополнительных пакетов.

В базовой версии реализована функция `summary()`, которая позволяет вычислить описательные статистики. Пример содержится в следующем программном коде.

Программный код 7.1. Вычисление описательных статистик при помощи команды `summary()`

```
> summary(mtcars[vars])
      mpg              hp              wt
 Min.   :10.4   Min.   : 52.0   Min.   :1.51
 1st Qu.:15.4   1st Qu.: 96.5   1st Qu.:2.58
 Median :19.2   Median :123.0   Median :3.33
 Mean   :20.1   Mean   :146.7   Mean   :3.22
 3rd Qu.:22.8   3rd Qu.:180.0   3rd Qu.:3.61
 Max.   :33.9   Max.   :335.0   Max.   :5.42
```

Функция `summary()` позволяет вычислить минимум, максимум, квартили и среднее для числовых переменных или частоты значений для факторов и логических векторов. Можно такжн использовать функции `apply()` или `sapply()`, описанные в главе 5, чтобы вычислить любую описательную статистику. Формат применения функции `sapply()` таков:

`sapply(`*`x`*`, `*`FUN`*`, `*`options`*`)`

где *`x`* – это таблица данных (или матрица), а *`FUN`* – произвольная функция. Если опции (*`options`*) указаны, они относятся к функции *`FUN`*. Обычно в этом случае используются такие функции как `mean`, `sd`, `var`, `min`, `max`, `median`, `length`, `range` и `quantile`. Функция `fivenum()` вычисляет пять описательных статистик Тьюки (Tukey) (минимум, нижняя квартиль, медиана, верхняя квартиль, максимум).

Удивительно, но в базовой версии программы нет функций для характеристики ассиметрии и эксцесса распределения значений, но вы можете создать такие функции самостоятельно. Пример в следующем программном коде демонстрирует, как вычислить несколько описательных статистик, включая ассиметрию и эксцесс.

Программный код 7.2. Вычисление описательных статистик при помощи функции `sapply()`

```
 > mystats <- function(x, na.omit=FALSE){
               if (na.omit)
                   x <- x[!is.na(x)]
               m <- mean(x)
               n <- length(x)
               s <- sd(x)
               skew <- sum((x-m)^3/s^3)/n
               kurt <- sum((x-m)^4/s^4)/n - 3
               return(c(n=n, mean=m, stdev=s, skew=skew, kurtosis=kurt))
             }
> sapply(mtcars[vars], mystats)
            mpg      hp      wt
n        32.000  32.000 32.0000
mean     20.091 146.688  3.2172
stdev     6.027  68.563  0.9785
skew      0.611   0.726  0.4231
kurtosis -0.373  -0.136 -0.0227
```

Для автомобилей из этого примера средний расход топлива составляет 20.1 миль на галлон со стандартным отклонением 6.0. Максимум кривой распределения значений смещен вправо (+0.61) и находится немного ниже, чем это бывает в случае нормального распределения (-0.37). Это будет лучше всего заметно при графическом изображении данных. Учтите, что если бы вы хотели бы удалить строки с пропущенными значениями, то нужно было написать `sapply(mtcars[vars], mystats, na.omit=TRUE)`.

Дополнительные возможности

Функции для вычисления описательных статистик реализованы в нескольких дополнительных пакетах, включая пакеты `Hmisc`, `pastecs` и `psych`. Не забудьте установить эти пакеты перед первым использованием (см. раздел 1.4 в первой главе), поскольку они не входят в базовую версию программы.

Функция `describe()` из пакета `Hmisc` выводит на экран число переменных и наблюдений, число пропущенных и неповторяющихся значений, среднее, квартили, а также пять самых маленьких и пять самых больших значений. Пример представлен в следующем программном коде.

Программный код 7.3. Вычисление описательных статистик при помощи функции `describe()` из пакета `Hmisc`

```
> library(Hmisc)
> describe(mtcars[vars])
 3  Variables      32  Observations
---------------------------------------------------------------------------
mpg
n missing  unique  Mean    .05   .10     .25   .50    .75    .90    .95
32      0    25   20.09 12.00  14.34  15.43  19.20  22.80  30.09  31.30
lowest : 10.4 13.3 14.3 14.7 15.0, highest: 26.0 27.3 30.4 32.4 33.9
---------------------------------------------------------------------------
hp
n missing  unique    Mean    .05     .10   .2     .50   .75   .90     .95
32       0     22   146.7  63.65  66.00 96.50 123.00 180.00 243.50 253.55
lowest :  52  62  65  66  91, highest: 215 230 245 264 335
---------------------------------------------------------------------------
wt
n missing  unique    Mean    .05    .10    .25    .50    .75    .90   .95
32      0      29   3.217  1.736  1.956  2.581  3.325  3.610  4.048 5.293
lowest : 1.513 1.615 1.835 1.935 2.140, highest: 3.845 4.070 5.250 5.345
5.424
---------------------------------------------------------------------------
```

В пакете pastecs есть функция stat.desc(), которая позволяет вычислить множество описательных статистик. Формат ее применения таков:
stat.desc(*x*, basic=TRUE, desc=TRUE, norm=FALSE, p=0.95)
где *x* – это таблица данных или временной ряд. Если basic=TRUE (по умолчанию), то вычисляется число значений, число нулей и пропущенных значений, минимум, максимум, размах и сумма. Если desc=TRUE (тоже по умолчанию), то вычисляются медиана, среднее арифметическое, стандартная ошибка среднего, 95% доверительный интервал для среднего, дисперсия, стандартное отклонение и коэффициент вариации. Наконец, если norm=TRUE (не по умолчанию), вычисляются статистики нормального распределения, включая ассиметрию и эксцесс (и их достоверность) и проводится тест Шапиро-Уилка (Shapiro-Wilk test) на нормальность распределения. Опция p используется при вычислении доверительного интервала для среднего арифметического (по умолчанию она равна 0.95). Пример дан в программном коде 7.4.

Программный код 7.4. Вычисление описательных статистик при помощи функции stat.desc() пакета pastecs

```
> library(pastecs)
> stat.desc(mtcars[vars])
                mpg       hp      wt
nbr.val       32.00   32.000  32.000
nbr.null       0.00    0.000   0.000
nbr.na         0.00    0.000   0.000
min           10.40   52.000   1.513
max           33.90  335.000   5.424
range         23.50  283.000   3.911
sum          642.90 4694.000 102.952
median        19.20  123.000   3.325
mean          20.09  146.688   3.217
SE.mean        1.07   12.120   0.173
CI.mean.0.95   2.17   24.720   0.353
var           36.32 4700.867   0.957
std.dev        6.03   68.563   0.978
coef.var       0.30    0.467   0.304
```

Как будто этого недостаточно, в пакете psych тоже есть функция под названием describe(), которая выводит на экран число непропущенных значений, среднее арифметическое, стандартное отклонение, усеченное среднее, минимум, максимум, размах, ассиметрию, эксцесс и стандартную ошибку среднего. Пример приведен в следующем программном коде.

Программный код 7.5. Вычисление описательных статистик при помощи функции describe() пакета psych

```
> library(psych)
Attaching package: 'psych'
        The following object(s) are masked from package:Hmisc :
         describe
> describe(mtcars[vars])
    var  n   mean    sd median trimmed   mad   min    max
mpg   1 32  20.09  6.03  19.20   19.70  5.41 10.40  33.90
hp    2 32 146.69 68.56 123.00  141.19 77.10 52.00 335.00
wt    3 32   3.22  0.98   3.33    3.15  0.77  1.51   5.42
     range skew kurtosis    se
mpg  23.50 0.61    -0.37  1.07
hp  283.00 0.73    -0.14 12.12
wt    3.91 0.42    -0.02  0.17
```

Я же говорил вам, что это потрясает воображение!

Замечание. *В рассмотренных примерах функция с названием* `describe()` *была и в пакете* `psych`*, и в пакете* `Hmisc`*. Откуда R знает, какую выбрать? Очень просто, преимущество имеет пакет, загруженный последним, как это видно из программного кода 7.5. В данном случае пакет* `psych` *загружен после* `Hmisc`*, и на экран выводится сообщение о том, что функция* `describe()` *из пакета* `Hmisc` *«замаскирована» функцией с таким же названием из пакета* `psych`*. Когда вы пишете название этой функции, R начинает поиск с пакета* `psych` *и выполняет ее. Если вы хотите вместо этого воспользоваться функцией из пакета* `Hmisc`*, можете набрать* `Hmisc::describe(mt)`*. Эта функция никуда не делась. Просто нужно дать программе больше информации для ее поиска.*

Теперь, когда вы узнали, как вычислять описательные статистики для данных в целом, давайте рассмотрим, как получить их для групп данных.

7.1.2. Вычисление описательных статистик для групп данных

При сравнении групп объектов или наблюдений обычно обращают внимание на значения описательных статистик для каждой группы, а не для всей выборки в целом. Опять же, в R реализовано несколько способов достижения цели. Мы начнем с вычисления описательных статистик для каждого типа коробки передач.

В главе 5 мы обсудили способы объединения данных в группы. При вычислении описательных статистик для отдельных групп данных можно использовать функцию `aggregate()`, как это показано в следующем программном коде.

Программный код 7.6. Вычисление описательных статистик для отдельных групп с использованием функции `aggregate()`

```
> aggregate(mtcars[vars], by=list(am=mtcars$am), mean)
  am  mpg  hp   wt
1  0 17.1 160 3.77
2  1 24.4 127 2.41
> aggregate(mtcars[vars], by=list(am=mtcars$am), sd)
  am  mpg   hp    wt
1  0 3.83 53.9 0.777
2  1 6.17 84.1 0.617
```

Обратите внимание на использование выражения `list(am=mtcars$am)`. Если бы вы написали `list(mtcars$am)`, то столбец `am` был бы назван не `am`, а `Group.1`. Здесь мы использовали знак присвоения, чтобы подписать столбец более содержательно. Если у вас есть больше одной группирующей переменной, можно использовать код вроде этого: `by=list(name1=groupvar1, name2=groupvar2, … , groupvarN)`.

К сожалению, функцию `aggregate()` можно использовать только для вычисления одной статистики, такой как среднее, стандартное отклонение и подобных им, за один раз. Для одновременного вычисления нескольких статистик используйте функцию `by()`. Формат ее применения таков:

`by(`*`data`*`, `*`INDICES`*`, `*`FUN`*`)`

где *`data`* – это таблица данных или матрица, *`INDICES`* – фактор или список факторов, который определяет группы и *`FUN`* – произвольная функция. Приведенный ниже программный код содержит пример.

Программный код 7.7. Вычисление описательных статистик для групп данных при помощи функции `by()`

```
> dstats <- function(x)(c(mean=mean(x), sd=sd(x)))
> by(mtcars[vars], mtcars$am, dstats)
mtcars$am: 0
mean.mpg  mean.hp  mean.wt   sd.mpg    sd.hp    sd.wt
  17.147  160.263    3.769    3.834   53.908    0.777
------------------------------------------------
mtcars$am: 1
mean.mpg  mean.hp  mean.wt   sd.mpg    sd.hp    sd.wt
  24.392  126.846    2.411    6.167   84.062    0.617
```

Дополнительные возможности

В пакетах `doBy` и `psych` реализованы команды, вычисляющие описательные статистики для групп данных. Опять же, эти пакеты не поставляются с базовой версией и должны быть установлены перед первым использованием. Функция `summaryBy` из пакета `doBy` имеет такой формат применения:
`summaryBy(`*`formula`*`, data=`*`dataframe`*`, FUN=`*`function`*`)`
где *`formula`* принимает вид
`var1 + var2 + var3 + ... + varN ~ groupvar1 + groupvar2 + … + groupvarN`
Переменные слева от знака ~ – это числовые переменные, которые нужно проанализировать, а переменные справа – это категориальные группирующие переменные. В качестве *`function`* можно использовать любую имеющуюся или заново созданную функцию. Пример с функцией `mystats`, которую мы создали в разделе 7.2.1, дан в приведенном ниже программном коде.

Программный код 7.8. Вычисление описательных статистик для групп при помощи функции `summaryBy()` из пакета `doBy`

```
> library(doBy)
> summaryBy(mpg+hp+wt~am, data=mtcars, FUN=mystats)
  am mpg.n mpg.mean mpg.stdev mpg.skew mpg.kurtosis hp.n hp.mean hp.stdev
1  0    19     17.1      3.83   0.0140       -0.803   19     160     53.9
2  1    13     24.4      6.17   0.0526       -1.455   13     127     84.1
  hp.skew hp.kurtosis wt.n wt.mean wt.stdev wt.skew wt.kurtosis
1 -0.0142      -1.210   19    3.77    0.777   0.976       0.142
2  1.3599       0.563   13    2.41    0.617   0.210      -1.174
```

Как вы увидите из приведенного ниже программного кода, функция `describe.by()` из пакета `phych()` позволяет вычислить те же описательные статистики, что и функция `describe()`, только отдельно для каждой группы.

Программный код 7.9. Вычисление описательных статистик для групп при помощи функции `describe.by()` из пакета `psych`

```
> library(psych)
> describe.by(mtcars[vars], mtcars$am)
group: 0
    var  n   mean    sd median trimmed   mad   min    max
mpg   1 19  17.15  3.83  17.30   17.12  3.11 10.40  24.40
hp    2 19 160.26 53.91 175.00  161.06 77.10 62.00 245.00
wt    3 19   3.77  0.78   3.52    3.75  0.45  2.46   5.42
     range  skew kurtosis    se
mpg  14.00  0.01    -0.80  0.88
hp  183.00 -0.01    -1.21 12.37
wt    2.96  0.98     0.14  0.18
------------------------------------------------
group: 1
    var  n   mean    sd median trimmed   mad   min    max
mpg   1 13  24.39  6.17  22.80   24.38  6.67 15.00  33.90
hp    2 13 126.85 84.06 109.00  114.73 63.75 52.00 335.00
wt    3 13   2.41  0.62   2.32    2.39  0.68  1.51   3.57
     range skew kurtosis    se
mpg  18.90 0.05    -1.46  1.71
hp  283.00 1.36     0.56 23.31
wt    2.06 0.21    -1.17  0.17
```

В отличие от предыдущего примера, при использовании команды `describe.by()` нельзя указать произвольную функцию, так что эта команда менее широко применима. Если у вас есть более одной группирующей переменной, то их можно записать в виде `list(groupvar1, groupvar2, … , groupvarN)`. Однако это будет работать только в том случае, если всем сочетаниям группирующих переменных соответствуют какие-то значения исследуемой переменной.

Наконец, можно использовать описанный в разделе 5.6.3 пакет `reshape`, чтобы вычислить описательные статистики для групп. Если вы не читали этот раздел, я советую вам сделать это, прежде чем продолжать. Сначала мы «растворяем» данные, используя выражение

`dfm <- melt(`*`dataframe`*`, measure.vars=`*`y`*`, id.vars=`*`g`*`)`

где *`dataframe`* содержит данные, *`y`* – вектор, который определяет, какие данные анализировать (по умолчанию все), а *`g`* – вектор, содержащий одну или более группирующих переменных. Затем вы «собираете» данные при помощи выражения

`cast(dfm, `*`groupvar1 + groupvar2`*` + … + variable ~ ., `*`FUN`*`)`

где группирующие переменные разделены знаками +, слово *`variable`* так и должно быть написано, а *`FUN`* – произвольная функция.

В последнем примере этого раздела мы применим подход, реализованный в пакете `reshape`, чтобы вычислить описательные статистики для каждой подгруппы, определенной сочетанием типа коробки передач и числа цилиндров. В качестве описательных статистик мы выбрали размер выборки, среднее и стандартное отклонение. Функции и результат представлены в приведенном ниже программном коде.

Программный код 7.10. Вычисление описательных статистик для групп с использованием пакета `reshape`

```
> library(reshape)
> dstats <- function(x)(c(n=length(x), mean=mean(x), sd=sd(x)))
> dfm <- melt(mtcars, measure.vars=c("mpg", "hp", "wt"),
              id.vars=c("am", "cyl"))
> cast(dfm, am + cyl + variable ~ ., dstats)
   am cyl variable  n   mean     sd
1   0   4      mpg  3  22.90  1.453
2   0   4       hp  3  84.67 19.655
3   0   4       wt  3   2.94  0.408
4   0   6      mpg  4  19.12  1.632
5   0   6       hp  4 115.25  9.179
6   0   6       wt  4   3.39  0.116
7   0   8      mpg 12  15.05  2.774
8   0   8       hp 12 194.17 33.360
9   0   8       wt 12   4.10  0.768
10  1   4      mpg  8  28.07  4.484
11  1   4       hp  8  81.88 22.655
12  1   4       wt  8   2.04  0.409
13  1   6      mpg  3  20.57  0.751
14  1   6       hp  3 131.67 37.528
15  1   6       wt  3   2.75  0.128
16  1   8      mpg  2  15.40  0.566
17  1   8       hp  2 299.50 50.205
18  1   8       wt  2   3.37  0.283
```

Лично мне этот подход кажется наиболее лаконичным и привлекательным. Люди, которые анализируют данные, имеют свои предпочтения относительно того, какие описательные статистики вычислять и в каком виде выводить результаты на экран. Наверное, поэтому существует столько способов вычисления описательных статистик. Выберите тот, который больше вам подходит, или разработайте свой собственный!

7.1.3. Визуализация результатов

Представление характеристик распределения данных в числовом виде полезно, но оно не заменяет собой его графическое изображение. Для количественных данных подходят гистограммы (раздел 6.3), диаграммы плотности рассеяния (раздел 6.4), ящики с усами (раздел 6.5) и точечные диаграммы (раздел 6.6). Они способны дать понимание вещей, которые могут быть легко упущены, если полагаться на небольшой набор описательных статистик.

Функции, которые мы рассматривали до сих пор, позволяют охарактеризовать количественные данные. Функции, описанные в следующем разделе, предназначены для анализа распределения категориальных переменных.

7.2. Таблицы частот и таблицы сопряженности

В этом разделе мы рассмотрим таблицы частот и таблицы сопряженности для категориальных переменных, а также познакомимся с тестами на независимость, мерами связи и способами графического представления результатов. Мы будем использовать команды, реализованные в базовой версии, наряду с командами из пакетов `vcd` и `gmodels`. В представленных ниже примерах `A`, `B` и `C` обозначают категориальные переменные.

В примерах этого раздела использован набор данных `Arthritis` из пакета `vcd`. Эти данные опубликованы Kock & Edward (1988) и представляют собой результаты клинического исследования нового способа лечения ревматоидного артрита двойным слепым методом. Вот первые несколько наблюдений:

```
> library(vcd)
> head(Arthritis)
  ID Treatment  Sex Age Improved
1 57   Treated Male  27     Some
2 46   Treated Male  29     None
3 77   Treated Male  30     None
4 17   Treated Male  32   Marked
5 36   Treated Male  46   Marked
6 23   Treated Male  58   Marked
```

Способ лечения (`Treatment`: плацебо – Placebo и лекарство – Treated), пол (`Sex`: мужской – Male и женский – Female) и степень улучшения состояния больных (`Improved`: None – нет, Some – некоторое, Marked – заметное) – это категориальные факторы. В следующем подразделе мы создадим таблицы частот и сопряженности для этих данных.

7.2.1. Создание таблиц частот

В R реализовано несколько методов создания таблиц частот и сопряженности. Самые важные функции перечислены в табл. 7.1.

Таблица 7.1. Функции для создания и преобразования таблиц сопряженности

Функция	Описание
`table(`*`var1, var2, …, varN`*`)`	Создает N-мерную таблицу сопряженности для N категориальных переменных (факторов)
`xtabs(`*`formula, data`*`)`	Создает N-мерную таблицу сопряженности на основании формулы и матрицы или таблицы данных
`prop.table(`*`table, margins`*`)`	Представляет значения таблицы в виде долей от значений крайнего поля таблицы, задаваемого параметром *`margins`*
`margin.table(`*`table, margins`*`)`	Суммирует значения таблицы по строкам или столбцам (определяется параметром *`margins`*)
`addmargins(`*`table, margins`*`)`	Вычисляет описательные статистики *`margins`* (суммы по умолчанию) для таблицы
`ftable(`*`table`*`)`	Создает компактную «плоскую» таблицу сопряженности

В следующих подразделах мы будем использовать все эти функции для исследования категориальных переменных. Мы начнем с вычисления простых частот, потом построим таблицы сопряженности для двух переменных и закончим созданием таблиц сопряженности для нескольких переменных. В качестве первого шага мы создадим таблицу при помощи функций `table()` и `xtabs()`, а затем будем изменять эту таблицу при помощи других функций.

Таблицы для одной переменной

Частоты значений одной переменной можно рассчитать при помощи функции `table()`. Вот пример:

```
> mytable <- with(Arthritis, table(Improved))
> mytable
Improved
  None   Some Marked
    42     14     28
```

Эти частоты можно преобразовать в доли от целого при помощи команды `prop.table()`:

```
> prop.table(mytable)
Improved
  None   Some Marked
 0.500  0.167  0.333
```

Или в проценты, используя `prop.table()*100`:

```
> prop.table(mytable)*100
Improved
  None   Some Marked
  50.0   16.7   33.3
```

Отсюда видно, что у половины пациентов после лечения произошло хоть какое-то улучшение состояния (16.7 + 33.3).

Таблицы для двух переменных

В случае двух переменных формат применения функции `table()` таков:

`mytable <- table(`*`A`*`, `*`B`*`)`

где *A* – это переменная, значениям которой соответствуют строки таблицы сопряженности, а *B* – столбцы. В качестве альтернативы можно воспользоваться функцией `xtabs()`, которая позволяет при создании таблицы сопряженности, вводить данные в виде формулы. Вот формат ее применения:

`mytable <- xtabs(~ A + B, data=`*`mydata`*`)`

где *mydata* – это матрицы или таблица данных. В общем случае переменные, для которых строится таблица сопряженности, находятся в правой части формулы (то есть справа от значка ~) и разделяются знаками +. Если переменная находится в левой части формулы, предполагается, что это вектор с частотами значений (удобно, если вы их уже вычисляли).

Для набора данных `Arthritis` у вас получится такая таблица:

```
> mytable <- xtabs(~ Treatment+Improved, data=Arthritis)
> mytable
         Improved
Treatment None Some Marked
  Placebo   29    7      7
  Treated   13    7     21
```

При помощи команд `margin.table()` и `prop.table()` можно рассчитать соответственно частоты и доли от целого по столбцам или строкам. При суммировании и вычислении долей по строкам получится следующее:

```
> margin.table(mytable, 1)
Treatment
Placebo Treated
     43      41
> prop.table(mytable, 1)
         Improved
Treatment  None  Some Marked
  Placebo 0.674 0.163  0.163
  Treated 0.317 0.171  0.512
```

Индекс 1 относится к первой переменной в формате применения функции `table()`. Рассматривая полученную таблицу, вы можете заметить, что у 51% пациентов, получавших настоящее лекарство, наступило заметное улучшение, для сравнения, такой эффект наступил только у 16% получавших плацебо пациентов.

Вычисление сумм и долей от целого для столбцов выглядит так:

```
> margin.table(mytable, 2)
Improved
  None   Some Marked
    42     14     28
> prop.table(mytable, 2)
         Improved
Treatment  None  Some Marked
  Placebo 0.690 0.500  0.250
  Treated 0.310 0.500  0.750
```

В этом случае индекс 2 относится ко второй переменной в формате применения функции `table()`.

Доли от целого для ячеек таблицы рассчитываются так:

```
> prop.table(mytable)
         Improved
Treatment   None   Some Marked
  Placebo 0.3452 0.0833 0.0833
  Treated 0.1548 0.0833 0.2500
```

Можно использовать функцию `addmargins()` для добавления сумм по столбцам или строкам:

```
> addmargins(mytable)
         Improved
Treatment None Some Marked Sum
  Placebo   29    7      7  43
  Treated   13    7     21  41
  Sum       42   14     28  84
> addmargins(prop.table(mytable))
         Improved
Treatment   None   Some Marked    Sum
  Placebo 0.3452 0.0833 0.0833 0.5119
  Treated 0.1548 0.0833 0.2500 0.4881
  Sum     0.5000 0.1667 0.3333 1.0000
```

При использовании функции `addmargins()` по умолчанию вычисляются суммы и для столбцов, и для строк таблицы. В противоположность этому команда

```
> addmargins(prop.table(mytable, 1),
         Improved
Treatment  None  Some Marked   Sum
  Placebo 0.674 0.163  0.163 1.000
  Treated 0.317 0.171  0.512 1.000
```

позволяет вычислить сумму только для строк. Аналогично, команда

```
> addmargins(prop.table(mytable, 2), 1)
         Improved
Treatment  None  Some Marked
  Placebo 0.690 0.500  0.250
  Treated 0.310 0.500  0.750
  Sum     1.000 1.000  1.000
```

вычисляет суммы только для столбцов. Из этой таблицы видно, что 25% пациентов, чье состояние заметно улучшилось после лечения, получали плацебо.

Замечание. *Функция* `table()` *по умолчанию игнорирует пропущенные значения. Для того чтобы добавить пропущенные значения в таблицу сопряженности в качестве одного из значений используйте опцию* `useNA="ifany"`.

Третий способ создания таблиц сопряженности для двух переменных заключается в использовании функции `CrossTable()` в пакете `gmodels`. Эта функция создает такую же таблицу, как функции `PROC FREQ` в SAS или `CROSSTABS` в SPSS. Пример представлен в программном коде 7.11.

Программный код 7.11. Построение таблицы сопряженности для двух переменных при помощи функции `CrossTable()`

```
> library(gmodels)
> CrossTable(Arthritis$Treatment, Arthritis$Improved)
   Cell Contents
|-------------------------|
|                       N |
| Chi-square contribution |
|           N / Row Total |
|           N / Col Total |
|         N / Table Total |
|-------------------------|
Total Observations in Table:  84
                  | Arthritis$Improved
Arthritis$Treatment |      None |      Some |    Marked | Row Total |
--------------------|-----------|-----------|-----------|-----------|
            Placebo |        29 |         7 |         7 |        43 |
                    |     2.616 |     0.004 |     3.752 |           |
                    |     0.674 |     0.163 |     0.163 |     0.512 |
                    |     0.690 |     0.500 |     0.250 |           |
                    |     0.345 |     0.083 |     0.083 |           |
--------------------|-----------|-----------|-----------|-----------|
            Treated |        13 |         7 |        21 |        41 |
                    |     2.744 |     0.004 |     3.935 |           |
                    |     0.317 |     0.171 |     0.512 |     0.488 |
                    |     0.310 |     0.500 |     0.750 |           |
                    |     0.155 |     0.083 |     0.250 |           |
--------------------|-----------|-----------|-----------|-----------|
       Column Total |        42 |        14 |        28 |        84 |
                    |     0.500 |     0.167 |     0.333 |           |
--------------------|-----------|-----------|-----------|-----------|
```

У функции `CrossTable()` есть опции вычислять проценты (по строкам, по столбцам, по ячейкам); округлять числа до заданного числа знаков после запятой; проводить тесты хи-квадрат, Фишера и МакНемара на независимость; вычислять ожидаемые значения и остатки (Пирсон, стандартизованные, скорректированные стандартизованные); учитывать пропущенные значения; добавлять подписи в виде названий строк и столбцов; форматировать результат в стиле SAS и SPSS. Более подробную информацию можно получить, набрав `help(CrossTable)`.

Если у нас есть больше двух категориальных переменных, нужно строить многомерные таблицы сопряженности. Теперь мы рассмотрим их.

Многомерные таблицы

Функции `table()` и `xtabs()` можно использовать для создания многомерных таблиц сопряженности для трех и более категориальных переменных. Функции `margin.table()`, `prop.table()` и `addmargins()` тоже можно применять к многомерным таблицам. Кроме того, функция `ftable()` позволяет выводить многомерные таблицы на экран в компактном и привлекательном виде. Пример приведен в программном коде 7.12.

Программный код 7.12. Трехмерная таблица сопряженности

```
> mytable <- xtabs(~ Treatment+Sex+Improved, data=Arthritis)
> mytable
```

←❶ Групповые частоты

```
, , Improved = None
         Sex
Treatment Female Male
  Placebo     19   10
  Treated      6    7
, , Improved = Some
         Sex
Treatment Female Male
  Placebo      7    0
  Treated      5    2
, , Improved = Marked
         Sex
Treatment Female Male
  Placebo      6    1
  Treated     16    5
> ftable(mytable)
                  Sex Female Male
Treatment Improved
Placebo   None             19   10
          Some              7    0
          Marked            6    1
Treated   None              6    7
          Some              5    2
          Marked           16    5
> margin.table(mytable, 1)
```

←❷Сумма частот по строкам и столбцам

```
Treatment
Placebo Treated
     43      41
> margin.table(mytable, 2)
Sex
Female   Male
    59     25
> margin.table(mytable, 3)
Improved
  None   Some Marked
    42     14     28
> margin.table(mytable, c(1, 3))
```

←❸Сумма частот всех сочетаний значений столбцов Treatment и Improved

```
         Improved
Treatment None Some Marked
  Placebo   29    7      7
  Treated   13    7     21
> ftable(prop.table(mytable, c(1, 2)))
```

←❹Сумма частот всех сочетаний значений столбцов Treatment и Sex

```
                 Improved  None  Some Marked
Treatment Sex
Placebo   Female          0.594 0.219  0.188
          Male            0.909 0.000  0.091
Treated   Female          0.222 0.185  0.593
          Male            0.500 0.143  0.357
> ftable(addmargins(prop.table(mytable, c(1, 2)), 3))
                 Improved  None  Some Marked   Sum
```

```
Treatment Sex
Placebo   Female          0.594 0.219  0.188 1.000
          Male            0.909 0.000  0.091 1.000
Treated   Female          0.222 0.185  0.593 1.000
          Male            0.500 0.143  0.357 1.000
```

Выражение ❶ позволяет вычислить частоты для всех сочетаний трех факторов. Также видно, что команда `ftable()` позволяет представить результаты в более компактном и привлекательном виде.

Выражение ❷ рассчитывает частоты для отдельных факторов. Поскольку исходная таблица была создана при помощи формулы `~ Treatment+Sex+Improved`, то индекс 1 соответствует переменной `Treatment`, индекс 2 – переменной `Sex`, а индекс 3 – переменной `Improved`.

Выражение ❸ позволяет вычислить частоты встречаемости всех сочетаний значений `Treatment` и `Improved`, объединив данные для мужчин и женщин. Доля пациентов с разной степенью улучшения для каждого сочетания типа лечения и пола вычислена при помощи команды ❹. Здесь видно, что заметное улучшение наступило у 36% мужчин и 69% женщин, которые получали настоящее лекарство. В общем случае частоты вычисляются так, чтобы их сумма для всех значений не указанного внутри команды `prop.table()` фактора (в данном случае `Improved`) была равна единице. Это видно в последнем примере, где мы суммируем частоты по строкам.

Если вы хотите представить результат в процентах, а не долях единицы, можете умножить все значения полученной таблицы на 100. Например, команда

```
ftable(addmargins(prop.table(mytable, c(1, 2)), 3)) * 100
```

создает такую таблицу:

```
                   Sex Female  Male   Sum
Treatment Improved
Placebo   None            65.5  34.5 100.0
          Some           100.0   0.0 100.0
          Marked          85.7  14.3 100.0
Treated   None            46.2  53.8 100.0
          Some            71.4  28.6 100.0
          Marked          76.2  23.8 100.0
```

В то время как таблицы сопряженности содержат информацию о частотах всех сочетаний значений факторов, вам, возможно, также интересно, связаны ли эти факторы между собой или они независимы. Тесты на независимость обсуждаются в следующем разделе.

7.2.2. Тесты на независимость

В R реализовано несколько способов проверки независимости категориальных данных. В этом разделе описано три теста: хи-квадрат, Фишера и Кохран-Мантель-Хейцеля.

Тест хи-квадрат

Применить функцию `chisq.test()` можно к двумерной таблице, чтобы при помощи теста хи-квадрат (chi-square test) проверить независимость переменных, которые составляют строки и столбцы. Пример приведен в следующем программном коде.

Программный код 7.13. Тест хи-квадрат

```
> library(vcd)
> mytable <- xtabs(~Treatment+Improved, data=Arthritis)
> chisq.test(mytable)
        Pearson's Chi-squared test
data:  mytable ←❶Тип лечения и его результат не независимы
X-squared = 13.1, df = 2, p-value = 0.001463
> mytable <- xtabs(~Improved+Sex, data=Arthritis)
> chisq.test(mytable)
        Pearson's Chi-squared test
data:  mytable ←❷Тип лечения и пол пациента независимы
X-squared = 4.84, df = 2, p-value = 0.0889
Warning message:
In chisq.test(mytable) : Chi-squared approximation may be incorrect
```

Результаты теста ❶ свидетельствуют о том, что существует связь между типом лечения и степенью улучшения состояния пациентов ($p<0.01$). А вот между полом пациентов и степенью улучшения их состояния связи нет ($p>0.05$) ❷. Значение статистической ошибки первого рода (p-value) – это вероятность того, что в генеральной совокупности зависимость между данными переменными отсутствует. Поскольку эта вероятность достаточно мала в случае ❶, мы отвергаем гипотезу о независимости результата лечения от его типа. Раз вероятность в случае ❷ не так уж мала, есть основание допускать, что способ лечения и пол пациентов не зависят друг от друга. Предупреждение (warning message) в программном коде 7.13 появляется потому, что в одной из шести ячеек таблицы сопряженности (некоторое улучшение состояния у мужчин) ожидаемое значение меньше пяти, что может сделать недействительным аппроксимацию хи-квадрат.

Точный критерий Фишера

Тест Фишера (Fisher's exact test) можно провести при помощи команды `fisher.test()`. Этот тест проверяет нулевую гипотезу о независимости столбцов и строк в таблице сопряженности. Формат применения этой функции таков:

`fisher.test(`*`mytable`*`)`

где *`mytable`* – это двухмерная таблица. Вот пример:

```
> mytable <- xtabs(~Treatment+Improved, data=Arthritis)
> fisher.test(mytable)
        Fisher's Exact Test for Count Data
data:  mytable
p-value = 0.001393
alternative hypothesis: two.sided
```

В отличие от многих статистических программ, в R функцию `fisher.test()` можно применять к любой двухмерной таблице, с двумя и более столбцами и строками, а не только к таблице размерности 2×2.

Тест Кохран-Мантель-Хейцеля

Функция `mantelhaen.test()` позволяет провести хи-квадрат тест Кохран-Мантель-Хейцеля (Cochran-Mantel-Haenszel) нулевой гипотезы о том, что две номинальные переменные условно независимы при каждом значении третьей переменной. Приведенный ниже программный код позволяет проверить гипотезу о том, что переменные `Treatment` и `Improved` независимы при каждом значении переменной `Sex`. При проведении теста предполагается, что трехстороннее взаимодействие (`Treatment` × `Improved` × `Sex`) отсутствует.

```
> mytable <- xtabs(~Treatment+Improved+Sex, data=Arthritis)
> mantelhaen.test(mytable)
        Cochran-Mantel-Haenszel test
data:  mytable
Cochran-Mantel-Haenszel M^2 = 14.6, df = 2, p-value = 0.0006647
```

Полученный результат позволяет считать, что способ лечения и его результат не независимы и для мужчин, и для женщин, если их рассматривать по отдельности.

7.2.3. Показатели взаимосвязи

Статистические тесты, описанные в предыдущем разделе, оценивали, есть ли достаточные основания, чтобы отвергнуть нулевую гипотезу о независимости двух переменных. Если вы смогли отвергнуть нулевую гипотезу, ваш интерес естественным образом смещается в сторону показателей взаимосвязи, чтобы измерить силу обнаруженных связей. Функция `assocstats()` из пакета `vcd` используется для вычисления фита-коэффициента (phi coefficient), коэффициент сопряженности и V Крамера (Cramer's V) для двухмерной таблицы. Пример приведен в следующем программном коде.

Программный код 7.14. Показатели взаимосвязи для двухмерной таблицы

```
> library(vcd)
> mytable <- xtabs(~Treatment+Improved, data=Arthritis)
> assocstats(mytable)
                    X^2 df  P(> X^2)
Likelihood Ratio 13.530  2 0.0011536
Pearson          13.055  2 0.0014626
Phi-Coefficient   : 0.394
Contingency Coeff.: 0.367
Cramer's V        : 0.394
```

В целом, б**о**льшие значения коэффициентов свидетельствуют о более сильной связи. В пакете `vcd` также реализована функция `kappa()`, которая вычисляет каппу Коэна (Cohen's kappa) и взвешенную каппу для матрицы неточностей (например, степень согласованности между двумя экспертами, классифицирующими набор объектов по категориям).

7.2.4. Визуализация результатов

В R реализованы способы визуализации взаимосвязей между категориальными переменными, которые выходят далеко за пределы возможностей большинства других статистических программ. Обычно для визуализации частот значений одной переменной используются столбчатые диаграммы (см. раздел 6.1). Пакет `vcd` содержит замечательные функции для визуализации взаимоотношений между категориальными переменными в многомерных наборах данным с использованием мозаичных диаграмм и диаграмм связей (см. раздел 11.4). Наконец, функции анализа соответствий в пакете `ca` позволяют визуально исследовать связи между строками и столбцами таблиц сопряженности при помощи различных геометрических образов (Nenadic, Greenacre, 2007).

7.2.5. Преобразование таблиц в неструктурированные файлы

Мы закончим этот раздел темой, которая редко обсуждается в книгах по R, но которая может быть очень актуальной. Что если у вас есть таблица, а вам нужны исходные «сырые» данные? Например, у вас есть такая таблица:

```
                    Sex Female Male
Treatment Improved
Placebo   None              19   10
          Some               7    0
          Marked             6    1
Treated   None               6    7
          Some               5    2
          Marked            16    5
```

а вам нужен такой формат:

```
  ID Treatment  Sex Age Improved
1 57   Treated Male  27     Some
2 46   Treated Male  29     None
3 77   Treated Male  30     None
4 17   Treated Male  32   Marked
5 36   Treated Male  46   Marked
6 23   Treated Male  58   Marked
[дальше идут еще 78 строк]
```

В R есть много статистических функций, которые работают со вторым форматом, а не с первым. Для преобразования таблицы R обратно в неструктурированный («плоский») файл можно использовать функцию, приведенную в программном коде 7.15.

Программный код 7.15. Преобразование таблицы в неструктурированный файл при помощи функции `table2flat()`

```
table2flat <- function(mytable) {
  df <- as.data.frame(mytable)
  rows  <- dim(df)[1]
  cols  <- dim(df)[2]
  x <- NULL
  for (i in 1:rows){
    for (j in 1:df$Freq[i]){
      row <- df[i,c(1:(cols-1))]
      x <- rbind(x,row)
    }
  }
  row.names(x)<-c(1:dim(x)[1])
  return(x)
}
```

Эта функция использует таблицу R (с любым числом строк и столбцов) в качестве аргумента и возвращает таблицу данных в виде неструктурированной таблицы. Эту функцию также можно использовать для импорта опубликованных данных. Скажем, вы наткнулись в журнале на табл. 7.2 и хотите сохранить ее в виде плоского файла R.

Таблица 7.2. Таблица сопряженности для метода лечения и степени улучшения состояния больных из набора данных `Arthritis`

Метод лечения	Улучшение состояния больных		
	Нет	Некоторое	Заметное
Плацебо	29	7	7
Лекарство	13	17	21

Приведенный ниже программный код демонстрирует метод, который даст нужный результат.

Программный код 7.16. Применение функции `table2flat()` к опубликованным данным

```
> treatment <- rep(c("Placebo", "Treated"), times=3)
> improved <- rep(c("None", "Some", "Marked"), each=2)
> Freq <- c(29,13,7,17,7,21)
> mytable <- as.data.frame(cbind(treatment, improved, Freq))
> mydata <- table2flat(mytable)
> head(mydata)
  treatment improved
1   Placebo     None
2   Placebo     None
3   Placebo     None
4   Treated     None
5   Placebo     Some
6   Placebo     Some
[дальше идут еще 12 строк]
```

На этом мы завершаем обсуждение таблиц сопряженности, до того момента, пока мы не начнем рассматривать их более углубленно в главах 11 и 15. Теперь давайте познакомимся с разными типами коэффициентов корреляции.

7.3. Корреляции

Коэффициенты корреляции используются для описания связей между количественными переменными. Знак коэффициента (+ или –) свидетельствует о направлении связи (положительная или отрицательная), а величина коэффициента показывает силу связи (варьирует от 0 – нет связи, до 1 – абсолютно предсказуемая взаимосвязь).

В этом разделе мы рассмотрим разные коэффициенты корреляции наряду с тестами на достоверность. Мы будем использовать набор данных `state.x77`, входящий в базовую версию R. В нем содержаться данные о численности, доходе, проценте неграмотного населения, средней продолжительности жизни, уровне преступности и доле людей со средним образованием для 50 штатов США в 1977 году. Там также есть данные о температурном режиме и о площади штатов, но мы опустим их, чтобы сэкономить место. Наберите `help(state.x77)`, чтобы узнать больше об этих данных. В дополнение к базовой версии программы нам понадобятся пакеты `psych` и `ggm`.

7.3.1. Типы корреляций

В R можно рассчитывать разные коэффициенты корреляции, включая коэффициенты Пирсона, Спирмена, Кэнделла, частные, полихорические и многорядные. Давайте рассмотрим их по порядку.

Коэффициенты Пирсона, Спирмена и Кэнделла

Коээфициент корреляции Пирсона по смешанным моментам (Pearson product moment correlation) отражает степень линейной связи между двумя количественными переменными. Коэффициент ранговой корреляции Спирмана (Spearman's Rank Order correlation) – мера связи между двумя ранжированными переменными. Тау Кэнделла (Kendall's Tau) – также непараметрический показатель ранговой корреляции.

Функция `cor()` позволяет вычислить все три коэффициента, а функция `cov()` рассчитывает ковариации. У этих функций есть много опций, но упрощенный формат вычисления корреляций таков:

```
cor(x, use= , method= )
```

Эти опции описаны в табл. 7.3.

Таблица 7.3. Опции функций `cor()` и `cov()`

Опция	Описание
x	Матрица или таблица данных
`use=`	Упрощает работу с пропущенными данными. Может принимать следующие значения: `all.obs` (предполагается, что пропущенные значения отсутствуют, их наличие вызовет сообщение об ошибке), `everything` (любая корреляция, включающая строку с пропущенным значением, не будет вычисляться – обозначается как `missing`), `complete.obs` (учитываются только строки без пропущенных значений) и `pairwise.complete.obs` (учитываются все полные наблюдения для каждой пары переменных в отдельности)
`method=`	Определяет тип коэффициента корреляции. Возможные значения – `pearson`, `spearman` или `kendall`.

Значения опций по умолчанию: `use="everything"` и `method="pearson"`. Пример представлен в приведенном ниже программном коде.

Программный код 7.17. Ковариации и корреляции

```
> states<- state.x77[,1:6]
> cov(states)
           Population Income Illiteracy Life Exp  Murder  HS Grad
Population   19931684 571230    292.868 -407.842 5663.52 -3551.51
Income         571230 377573   -163.702  280.663 -521.89  3076.77
Illiteracy        293   -164      0.372   -0.482    1.58    -3.24
Life Exp         -408    281     -0.482    1.802   -3.87     6.31
Murder           5664   -522      1.582   -3.869   13.63   -14.55
HS Grad         -3552   3077     -3.235    6.313  -14.55    65.24
> cor(states)
           Population Income Illiteracy Life Exp Murder HS Grad
Population     1.0000  0.208      0.108   -0.068  0.344 -0.0985
Income         0.2082  1.000     -0.437    0.340 -0.230  0.6199
Illiteracy     0.1076 -0.437      1.000   -0.588  0.703 -0.6572
Life Exp      -0.0681  0.340     -0.588    1.000 -0.781  0.5822
Murder         0.3436 -0.230      0.703   -0.781  1.000 -0.4880
HS Grad       -0.0985  0.620     -0.657    0.582 -0.488  1.0000
> cor(states, method="spearman")
           Population Income Illiteracy Life Exp Murder HS Grad
Population      1.000  0.125      0.313   -0.104  0.346  -0.383
Income          0.125  1.000     -0.315    0.324 -0.217   0.510
Illiteracy      0.313 -0.315      1.000   -0.555  0.672  -0.655
Life Exp       -0.104  0.324     -0.555    1.000 -0.780   0.524
Murder          0.346 -0.217      0.672   -0.780  1.000  -0.437
HS Grad        -0.383  0.510     -0.655    0.524 -0.437   1.000
```

Первая команда выводит на экран таблицу дисперсий и ковариаций. Вторая – таблицу корреляционных коэффициентов Пирсона, а третья – корреляционных коэффициентов Спирмена. Можно увидеть, например, что между средним доходом и долей людей со средним образованием существует сильная положительная корреляция, а между средней продолжительностью жизни и долей неграмотного населения – сильная отрицательная корреляция.

Обратите внимание на то, что по умолчанию получается квадратная таблица (приведены сочетания всех переменных со всеми). Вы также можете вывести на экран прямоугольную таблицу, как в приведенном примере.

```
> x <- states[,c("Population", "Income", "Illiteracy", "HS Grad")]
> y <- states[,c("Life Exp", "Murder")]
> cor(x,y)
           Life Exp Murder
Population   -0.068  0.344
Income        0.340 -0.230
Illiteracy   -0.588  0.703
HS Grad       0.582 -0.488
```

Этот способ применения функции особенно удобен, когда вы интересуетесь связью между двумя наборами переменных. Учтите, что эти результаты не позволяют узнать, отличаются ли достоверно коэффициенты корреляции от нуля (иными словами, можно ли на основании имеющейся выборки уверенно утверждать, что коэффициент корреляции для генеральной совокупности отличается от нуля). Для этого нужно применять тесты на достоверность (описаны в разделе 7.3.2).

Частные корреляции

Частная корреляция – это корреляция между двумя количественными переменными, зависящими в свою очередь от одной или более других количественных переменных. Для вычисления коэффициентов частной корреляции можно использовать функцию `pcor()` из пакета `ggm`. Этот пакет не поставляется с базовой версией программы, так что не забудьте установить его перед первым использованием. Формат применения этой функции таков:

`pcor(`*`u`*`, `*`S`*`)`

где *u* – это числовой вектор, в котором первые два числа – это номера переменных, для которых нужно вычислить коэффициент, а остальные числа – номера «влияющих» переменных (воздействие которых должно быть отделено). *S* – это ковариационная матрица для всех этих переменных. Проиллюстрируем это на примере.

```
> library(ggm)
> # частная корреляция между численностью населения и уровнем преступности,
> # освобожденная от влияния дохода, доли неграмотного населения и долей
> # людей со средним образованием
> pcor(c(1,5,2,3,6), cov(states))
[1] 0.346
```

В данном случае 0.346 – это коэффициент корреляции между численностью населения и уровнем преступности без влияния дохода, доли неграмотного населения и долей людей со средним образованием. Частные корреляции обычно используются в социологии.

Другие типы корреляций

Функция `hetcor()` из пакета `polycor` позволяет вычислять комбинированную корреляционную матрицу, содержащую коэффициенты корреляции Пирсона для числовых переменных, многорядные корреляции между числовыми и порядковыми переменными, полихорические корреляции между порядковыми переменными и тетрахорические корреляции между двумя дихотомическими переменными. Многорядные, полихорические и тетрахорические корреляции могут быть вычислены для порядковых и дихотомических переменных, которые происходят из нормального распределения. Дополнительную информацию об этих типах корреляций можно получить из справочного материала для данного пакета.

7.3.2. Проверка корреляций на достоверность

Как проверить на достоверность вычисленные коэффициенты корреляции? Стандартная нулевая гипотеза – это отсутствие связи (то есть коэффициент корреляции для генеральной совокупности равен нулю). Для проверки достоверности отдельных корреляционных коэффициентов Пирсона, Спирмена и Кэнделла можно использовать функцию `cor.test()`. Упрощенный формат ее применения таков:

```
cor.test(x, y, alternative = , method = )
```

где *x* и *y* – это переменные, корреляция между которыми исследуется, опция `alternative` определяет тип теста (`"two.side"`, `"less"` или `"greater"`), опция `method` задает тип корреляции (`"pearson"`, `"kendall"` или `"spearman"`). Используйте опцию `alternative="less"` для проверки гипотезы о том, что в генеральной совокупности коэффициент корреляции меньше нуля, а опцию `alternative="greater"` – для проверки того, что он больше нуля. По умолчанию `alternative="two.side"` (проверяется гипотеза о том, что коэффициент корреляции в генеральной совокупности не равен нулю). Пример приведен в следующем программном коде.

Программный код 7.18. Проверка достоверности коэффициента корреляции

```
> cor.test(states[,3], states[,5])
        Pearson's product-moment correlation
data:  states[, 3] and states[, 5]
t = 6.85, df = 48, p-value = 1.258e-08
alternative hypothesis: true correlation is not equal to 0
95 percent confidence interval:
 0.528 0.821
sample estimates:
  cor
0.703
```

Здесь нулевая гипотеза заключается в том, что пирсоновский коэффициент корреляции между средней продолжительностью жизни и уровнем преступности равен нулю. Если этот коэффициент для генеральной совокупности равен нулю, то его значение для случайной выборки будет равно 0.703 реже, чем в одном случае из 10 миллионов (это и означает p-value = 1.258e-08). Учитывая, насколько мала вероятность, мы отвергнем нулевую гипотезу и примем рабочую – о том, что значение этого коэффициента для генеральной совокупности *не* равно нулю.

К сожалению, при помощи функции `cor.test()` одновременно можно проверить достоверность только одного коэффициента корреляции. Зато в пакете `psych` есть функция `corr.test()`, которая позволяет сделать больше. С ее помощью можно вычислить коэффициенты корреляции Пирсона, Спирмена и Кэнделла между несколькими переменными и оценить их достоверность. Пример приведен в следующем программном коде.

Программный код 7.19. Создание матрицы коэффициентов корреляции и проверка их достоверности при помощи функции `corr.test()`

```
> library(psych)
> corr.test(states, use="complete")
Call:corr.test(x = states, use = "complete")
Correlation matrix
           Population Income Illiteracy Life Exp Murder HS Grad
Population       1.00   0.21       0.11    -0.07   0.34   -0.10
Income           0.21   1.00      -0.44     0.34  -0.23    0.62
Illiteracy       0.11  -0.44       1.00    -0.59   0.70   -0.66
Life Exp        -0.07   0.34      -0.59     1.00  -0.78    0.58
Murder           0.34  -0.23       0.70    -0.78   1.00   -0.49
HS Grad         -0.10   0.62      -0.66     0.58  -0.49    1.00
Sample Size
[1] 50
Probability value
           Population Income Illiteracy Life Exp Murder HS Grad
Population       0.00   0.15       0.46     0.64   0.01     0.5
Income           0.15   0.00       0.00     0.02   0.11     0.0
Illiteracy       0.46   0.00       0.00     0.00   0.00     0.0
Life Exp         0.64   0.02       0.00     0.00   0.00     0.0
Murder           0.01   0.11       0.00     0.00   0.00     0.0
HS Grad          0.50   0.00       0.00     0.00   0.00     0.0
```

Опция `use=` может принимать значения `"pairwise"` или `"complete"` (для попарного или построчного удаления пропущенных значений соответственно). Значения опции `method=` бывают следующими: `"pearson"` (по умолчанию), `"spearman"` или `"kendall"`. Из приведенного примера видно, что коэффициент корреляции между численностью населения и долей людей со средним образованием (–0.10) не отличается достоверно от нуля (p=0.5).

Другие тесты на достоверность

В разделе 7.4.1 мы обсуждали частные корреляции. Отсутствие зависимости между двумя переменными, без учета влияния нескольких других переменных можно проверить при помощи функции `pcor.test()` из пакета `psych` при условии, что значения всех этих переменных распределены нормально. Формат применения этой функции таков:
`pcor.test(`*`r`*`, `*`q`*`, `*`n`*`)`
где *r* – это частная корреляция, вычисленная при помощи функции `pcor()`, *q* – число переменных, влияние которых исключается, *n* – объем выборки.

Прежде чем закончить обсуждение этой темы, нужно упомянуть о функции `r.test()` из пакета `psych`, которая позволяет проводить ряд полезных тестов на достоверность. Эту функцию можно использовать, чтобы проверять достоверность:

- коэффициента корреляции;
- различий между двумя независимыми корреляциями;
- различий между двумя зависимыми корреляциями, у которых есть одна общая переменная;
- различий между двумя зависимыми корреляциями между разными парами переменных.

Введите `help(r.test)`, чтобы получить более полную информацию.

7.3.3. Визуализация корреляций

Связи между парами переменных можно визуализировать при помощи диаграмм рассеяния и составленных из них матриц. Коррелограммы – это непревзойденный действенный метод сравнения большого числа коэффициентов корреляции в легко интерпретируемой форме. Все эти графические подходы рассмотрены в главе 11.

7.4. Тесты Стьюдента

Наиболее обычная деятельность исследователей – это сравнение двух групп объектов. Правда ли, что больные, получившие новое лекарство, чувствуют себя лучше пациентов, которых лечат старым способом? Правда ли, что одна технология производства характеризуется меньшим процентом брака, чем другая? Какой из методов преподавания более эффективен по соотношению цены и качества? Если результат выражен в виде категориальной переменной, можно использовать методы, которые были описаны в разделе 7.3. Здесь мы сосредоточимся на сравнении групп, где исследуется непрерывная переменная, значения которой распределены нормально.

В качестве примера мы используем набор данных `UScrime`, входящий в пакет `MASS`. Эти данные содержат информацию о влиянии карательного законодательства на уровень преступности в 47 штатах США в 1960 году. Мы будем исследовать переменные `Prob` (вероятность угодить в тюрьму), `U1` (уровень безработицы для городских жителей мужского пола в возрасте от 14 до 24 лет) и `U2` (этот же показатель для мужчин в возрасте 35–39 лет). Категориальная переменная `So` (указывающая, относится ли штат к группе южных штатов) будет использована в качестве группирующей. Данные были масштабированы авторами исследования. Примечание: я раздумывал, не назвать ли этот раздел «Преступление и наказание на Далеком Юге»[8], но благоразумие взяло верх.

7.4.1. Тест Стьюдента для независимых переменных

Правда ли, что вероятность оказаться в тюрьме после совершения преступления выше в южных штатах? Чтобы ответить на этот вопрос, нужно сравнить вероятность лишения свободы в южных штатах и остальных. Для проверки гипотезы о равенстве двух средних значений можно использовать тест Стьюдента для независимых переменных. В этом случае подразумевается, что эти две группы не зависят друг от друга и данные происходят из нормальных распределений. Функцию можно применять в двух форматах. Или в таком:

`t.test(`*`y ~ x`*`, data)`

где *y* – это числовая переменная, а *x* – дихотомическая, или в таком:

`t.test(`*`y1, y2`*`)`

где *y1* и *y2* – это числовые векторы (исследуемые переменные для каждой из групп). Необязательный аргумент `data` назначает матрицу или таблицу данных, в которой содержатся переменные. В отличие от большинства статистических программ, в R по умолчанию не подразумевается равенство дисперсий и используется трансформация степеней свободы Велша (Welsh degrees of freedom modification). Указать на равенство дисперсий можно при помощи параметра `var.equal=TRUE`. По умолчанию используется двусторонняя альтернативная гипотеза (о том, что средние значения различаются, но неважно, как). Можно использовать опции `alternative="less"` или `alternative="greater"`, чтобы проверить наличие различий в определенном направлении.

При помощи приведенного программного кода вы сравниваете вероятность попасть в тюрьму в южных (группа 1) и остальных (группа 0) штатах при помощи двустороннего теста, не предполагая равенство дисперсий.

```
> library(MASS)
> t.test(Prob ~ So, data=UScrime)
        Welch Two Sample t-test
data:  Prob by So
t = -3.8954, df = 24.925, p-value = 0.0006506
```

[8] *Deep South – традиционное название южных штатов, связанное с особым укладом жизни и традициями их населения. – Прим. пер.*

```
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.03852569 -0.01187439
sample estimates:
mean in group 0 mean in group 1
     0.03851265      0.06371269
```

На основании этого теста можно отвергнуть гипотезу о равенстве шансов попасть в тюрьму в южных и остальных штатах (p<0.001).

Примечание. *Поскольку исследуемая переменная выражена в долях единицы, перед выполнением теста Стьюдента можно попробовать привести ее к нормальному распределению. В данном случае все разумные преобразования этой переменной* `(Y/1-Y, log(Y/1-Y), arcsin(Y), arcsin(sqrt(Y))` *приведут к одному и тому же результату. Преобразования переменных подробно обсуждаются в главе 8.*

7.4.1. Тест Стьюдента для зависимых переменных

В качестве второго примера, можно узнать, правда ли уровень безработицы у юношей (14–24 года) выше, чем у мужчин (35–39 лет). В данном случае эти две группы не независимы. Вы ведь не станете ожидать, что уровень безработицы у юношей и у мужчин в Алабаме – независимые величины? Когда данные для двух групп связаны между собой, следует проводить тест для зависимых переменных. Это же относится и к результатам повторных измерений или к измерениям одного и того же объекта, проведенным до и после какого-либо воздействия.

Подразумевается, что разность значений параметра для двух групп имеет нормальное распределение. В данном случае формат применения функции таков:

`t.test(`*`y1`*`,` *`y2`*`, paired=TRUE)`

где *y1* и *y2* – это числовые векторы для двух зависимых групп. Результаты применения теста выглядят следующим образом:

```
> library(MASS)
> sapply(UScrime[c("U1","U2")], function(x)(c(mean=mean(x),sd=sd(x))))
       U1    U2
mean 95.5 33.98
sd   18.0  8.45
> with(UScrime, t.test(U1, U2, paired=TRUE))
        Paired t-test
data:  U1 and U2
t = 32.4066, df = 46, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 57.67003 65.30870
sample estimates:
mean of the differences
               61.48936
```

Разность средних (61.5) достаточно велика, чтобы обосновать отклонение гипотезы о равенстве уровня безработицы для юношей и мужчин (у юношей она выше). В самом деле, вероятность получить такое значение разности средних для выборки в то время, как они будут равны для генеральной совокупности, меньше 0.00000000000000022 (то есть, 2.2e–16).

7.4.2. Когда существует больше двух групп

Что вы будете делать, если вам понадобится одновременно сравнить больше двух групп? Если предположить, что эти группы независимо происходят из нормального распределения, то можно использовать дисперсионный анализ (ANOVA). Это сложный подход, который можно применять для

анализа разных типов экспериментов и квази-экспериментов, поэтому он заслуживает отдельной главы. Вы можете спокойно прервать чтение этого раздела в любом месте на и перейти к главе 9.

7.5. Непараметрические тесты межгрупповых различий

Если у вас есть непараметрические данные, можно обратиться к непараметрическим методам. Описанные в этом разделе методы можно использовать, например, в том случае, если исследуемые переменные порядковые, или их распределение сильно отличается от нормального.

7.5.1. Сравнение двух групп

Если две группы независимы, можно использовать тест ранговых сумм Вилкоксона (Wilcoxon rank sum test), более широко известный как тест Манна-Уитни (Mann–Whitney U test), чтобы узнать, происходят ли наблюдения из одного и того же распределения вероятностей (иными словами, правда ли вероятность получить более высокие значения выше в одной выборке, чем в другой). Эту функцию можно применять в таком формате:

```
wilcox.test(y ~ x, data)
```

где *y* – это числовая переменная, а *x* – дихотомическая, или в таком:

```
wilcox.test(y1, y2)
```

где *y1* и *y2* – это исследуемые переменные для каждой группы. Необязательный аргумент `data` назначает матрицу или таблицу данных, в которой содержатся исследуемые переменные. По умолчанию это двухсторонний тест. Можно добавить параметр `exact`, чтобы вычислить точный критерий (exact test) и параметр `alternative="less"` или `alternative="greater"`, чтобы проверить соответствующую одностороннюю гипотезу.

Если применить тест Вилкоксона[9] для решения вопроса о вероятности попадания в тюрьму, обсуждаемого в предыдущем разделе, вы получите следующий результат:

```
> with(UScrime, by(Prob, So, median))
So: 0
[1] 0.0382
--------------------
So: 1
[1] 0.0556
> wilcox.test(Prob ~ So, data=UScrime)
        Wilcoxon rank sum test
data:  Prob by So
W = 81, p-value = 8.488e-05
alternative hypothesis: true location shift is not equal to 0
```

Вновь можно отвегнуть гипотезу о равенстве вероятностей оказаться в тюрьме в южных и прочих штатах ($p<0.01$).

Тест Вилкоксона[10] – это непараметрическая альтернатива тесту Стьюдента для зависимых переменных. Его следует применять в тех случаях, когда группы зависимы друг от друга и не наблюдается нормальное распределение значений. Формат применения функции такой же, как и в предыдущем примере, только нужно добавить параметр `paired=TRUE`. Давайте применим этот тест для решения вопроса о безработице из предыдущего раздела.

```
> sapply(UScrime[c("U1","U2")], median)
U1 U2
92 34
> with(UScrime, wilcox.test(U1, U2, paired=TRUE))
        Wilcoxon signed rank test with continuity correction
```

[9] *По умолчанию для независимых переменных. – Прим. пер.*
[10] *Для зависимых переменных. – Прим. пер.*

```
data:  U1 and U2
V = 1128, p-value = 2.464e-09
alternative hypothesis: true location shift is not equal to 0
```

Вы вновь пришли к тому же выводу, что и в случае теста Стьюдента для зависимых переменных.

В данном случае, параметрические тесты Стьюдента и их непараметрические аналоги дали одинаковые результаты. В том случае, если условия применения тестов Стьюдента выполняются, параметрические тесты имеют б**о**льшую мощность (вероятность обнаружить существующие различия выше) по сравнению с непараметрическими. Непараметрические тесты разумно применять, когда условия применения параметрических тестов сильно нарушаются (например, для порядковых переменных).

7.5.2. Сравнение более чем двух групп

Когда нужно сравнить больше двух групп, приходится использовать другие методы. Рассмотрим набор данных `state.x77` из раздела 7.4. В нем содержатся данные о численности населения, доходе, уровне неграмотности, среднем продолжительности жизни, уровне преступности и доле людей со средним образовании в разных штатах Америки. Что, если вы хотите сравнить уровень неграмотности в четырех регионах страны (Северо-Восток – Northeast, Юг – South, Север – North Central и Запад – West)? Это называется односторонний дизайн эксперимента, для решения поставленного вопроса существуют и параметрические, и непараметрические методы.

Если наши данные не удовлетворяют требованиям ANOVA для оценки межгрупповых различий средних значений, можно использовать непараметрические методы. Если группы независимы, лучше всего подойдет тест Краскела-Уоллиса (Kruskal–Wallis test). Если группы зависимы (например, это результаты повторных измерений или данные эксперимента с блочным дизайном), то нужно использовать тест Фридмана (Friedman test).

Формат применения теста Краскела-Уоллиса таков:

```
kruskal.test(y ~ A, data)
```

где *y* – это числовая переменная отклика, *A* – группирующая переменная, принимающая два значения и более (в случае с двумя значениями этот тест идентичен тесту Вилкоксона). Формат применения тест Фридмана следующий:

```
friedman.test(y ~ A | B, data)
```

где *y* – это числовая переменная отклика, *A* – группирующая переменная, *B* – блокирующая переменная, в которой содержится информация о парных наблюдениях. В обоих случаях `data` – это необязательный аргумент, который назначает матрицу или таблицу данных, где содержатся исследуемые переменные.

Давайте используем тест Краскела-Уоллиса для решения вопроса об уровне неграмотности. Сначала нужно добавить в наш набор данных информацию о региональной принадлежности штатов. Эта информация содержится в наборе данных `state.region`, входящем в базовую версию программы.

```
states <- as.data.frame(cbind(state.region, state.x77))
```

Теперь можно провести тест:

```
> kruskal.test(Illiteracy ~ state.region, data=states)
        Kruskal-Wallis rank sum test
data:  states$Illiteracy by states$state.region
Kruskal-Wallis chi-squared = 22.7, df = 3, p-value = 4.726e-05
```

Его результаты свидетельствуют о том, что уровень неграмотности не одинаков во всех четырех регионах ($p<0.001$).

Хотя мы опровергли нулевую гипотезу об отсутствии различий, из результатов теста не ясно, *какие* регионы достоверно отличаются от других. Чтобы ответить на этот вопрос, можно сравнить попарно все группы при помощи теста Вилкоксона. Более изящное решение проблемы состоит в одновременном множественном сравнении всех пар регионов с контролем значений статистической ошибки первого рода (вероятность найти несуществующие различия). Этот подход реализован в пакете `npmc`.

Честно говоря, здесь я немного вышел за пределы *основных* методов, заявленных в названии этой главы, но поскольку рассматрение этого подхода вполне здесь уместено, я надеюсь, что вы меня простите. Для начала убедитесь, что пакет `npmc` установлен. Для выполнения одноименной команды нужно, чтобы ваши данные были представлены в виде двух столбцов, один из которых назван `var` (зависимая переменная), а другой – `class` (группирующая переменная). Приведенный ниже программный код содержит пример использования этой команды.

Программный код 7.20. Непараметрические множественные сравнения

```
> class <- state.region
> var <- state.x77[,c("Illiteracy")]
> mydata <- as.data.frame(cbind(class, var))
> rm(class, var)
> library(npmc)
> summary(npmc(mydata), type="BF")
$'Data-structure'
```
←❶Попарные сравнения групп
```
              group.index   class.level nobs
Northeast               1     Northeast    9
South                   2         South   16
North Central           3 North Central   12
West                    4          West   13
$'Results of the multiple Behrens-Fisher-Test'
  cmp effect lower.cl upper.cl p.value.1s p.value.2s
1 1-2 0.8750  0.66149   1.0885   0.000665    0.00135
2 1-3 0.1898 -0.13797   0.5176   0.999999    0.06547
3 1-4 0.3974 -0.00554   0.8004   0.998030    0.92004
4 2-3 0.0104 -0.02060   0.0414   1.000000    0.00000
5 2-4 0.1875 -0.07923   0.4542   1.000000    0.02113
6 3-4 0.5641  0.18740   0.9408   0.797198    0.98430
> aggregate(mydata, by=list(mydata$class), median)
  Group.1 class  var
```
←❷Медианные значения неграмотности в каждом регионе
```
1       1     1 1.10
2       2     2 1.75
3       3     3 0.70
4       4     4 0.60
```

Команда `npmc()` осуществляет шесть попарных сравнений (Северо-Восток с Югом, Северо-Восток с Севером, Северо-Восток с Западом, Юг с Севером, Юг с Западом и Север с Западом) ❶. Из результатов двустороннего теста на достоверность (`p.value.2s`) можно видеть, что Юг достоверно отличается от трех других регионов, а они между собой не различаются. Видно ❷, что для Юга в среднем характерны более высокие значения уровня неграмотности. Учтите, что при этом подходе для промежуточных вычислений используются случайные значения, так что результаты будут слегка варьировать раз от раза.

7.6. Визуализация групповых различий

В разделах 7.4 и 7.5 мы рассматривали статистические методы сравнения групп. Визуальный анализ межгрупповых различий – это тоже очень важный этап комплексного подхода к анализу данных. Он

позволяет получить информацию о величине различий, выявить особенности распределения значений (ассиметрию, бимодальность, выбросы), которые влияют на результаты сравнения, и оценить, насколько данные удовлетворяют требованиям тестов. В R реализовано множество графических методов для сравнения групп, включая ящики с усами (простые, выемчатые и скрипичные), которые описаны в разделе 6.5; наложенные друг на друга диаграммы плотности ядра, рассмотренные в разделе 6.4.1; и графические методы оценки соответствия данных требованиям тестов, которые обсуждаются в главе 9.

7.7. Резюме

В этой главе мы рассмотрели функции R, которые позволяют вычислить основные характеристики данных и провести базовые тесты. Мы рассмотрели выборочные статистики и таблицы частот, тесты на независимость и меры связи для категориальных переменных, корреляции между количественными переменными (и соответствующие статистические тесты), а также сравнения двух и более групп по количественным переменным.

В следующей главе мы исследуем регрессионные методы, которые позволяют понять характер взаимосвязи между одной (простая регрессия) или несколькими (множественная регрессия) независимыми переменными и зависимой переменной. Графические методы помогут обнаружить возможные проблемы, оценить и улучшить соответствие созданных моделей реальным данным, а также выявить неожиданные, но важные эффекты.

Список литературы:

Allison, P. 2001. Missing Data. Thousand Oaks, CA: Sage.

Allison, T., and D. Chichetti. 1976. “Sleep in Mammals: Ecological and Constitutional Correlates.” Science 194 (4266): 732–734.

Anderson, M. J. 2006. “Distance-Based Tests for Homogeneity of Multivariate Dispersions.” Biometrics 62:245–253.

Baade, R., and R. Dye. 1990. “The Impact of Stadiums and Professional Sports on Metropolitan Area Development.” Growth and Change 21:1–14.

Bandalos, D. L., and M. R. Boehm-Kaufman. 2009. “Four Common Misconceptions in Exploratory Factor Analysis.” In Statistical and Methodological Myths and Urban Legends, edited by C. E. Lance and R. J. Vandenberg, 61–87. New York: Routledge.

Bates, D. 2005. “Fitting Linear Mixed Models in R.” R News 5 (1). www.r-project.org/doc/Rnews/Rnews_2005-1.pdf.

Breslow, N., and D. Clayton. 1993. “Approximate Inference in Generalized Linear Mixed Models.” Journal of the American Statistical Association 88:9–25.

Bretz, F., T. Hothorn, and P. Westfall. 2010. Multiple Comparisons Using R. Boca Raton, FL: Chapman & Hall.

Canty, A. J. 2002. “Resampling Methods in R: The boot Package.” http://cran.r-project.org/doc/Rnews/Rnews_2002-3.pdf.

Chambers, J. M. 2008. Software for Data Analysis: Programming with R. New York: Springer.

Cleveland, W. 1981. “LOWESS: A Program for Smoothing Scatter Plots by Robust Locally Weighted Regression.” The American Statistician 35:54.

———. 1985. The Elements of Graphing Data. Monterey, CA: Wadsworth.

———. 1993. Visualizing Data. Summit, NJ: Hobart Press.

Cohen, J. 1988. Statistical Power Analysis for the Behavioral Sciences, 2nd ed. Hillsdale, NJ: Lawrence Erlbaum.

Cook, D., and D. Swayne. 2008. Interactive and Dynamic Graphics for Data Analysis with R and GGobi. New York: Springer.2 REFERENCES

Coxe, S., S. West, and L. Aiken. 2009. “The Analysis of Count Data: A Gentle Introduction to Poisson Regression and Its Alternatives.” Journal of Personality Assessment 91:121–136.

Culbertson, W., and D. Bradford. 1991. “The Price of Beer: Some Evidence for Interstate Comparisons.” International Journal of Industrial Organization 9:275–289.
DiStefano, C., M. Zhu, and D. Mîndrilă. 2009. “Understanding and Using Factor Scores: Considerations for the Applied Researcher.” Practical Assessment, Research & Evaluation 14 (20). http://pareonline.net/pdf/v14n20.pdf.
Dobson, A., and A. Barnett. 2008. An Introduction to Generalized Linear Models, 3rd ed. Boca Raton, FL: Chapman & Hall.
Dunteman, G., and M-H Ho. 2006. An Introduction to Generalized Linear Models. Thousand Oaks, CA: Sage.
Efron, B., and R. Tibshirani. 1998. An Introduction to the Bootstrap. New York: Chapman & Hall.
Fair, R. C. 1978. “A Theory of Extramarital Affairs.” Journal of Political Economy 86:45–61.
Faraway, J. 2006. Extending the Linear Model with R: Generalized Linear, Mixed Effects and Nonparametric
Regression Models. Boca Raton, FL: Chapman & Hall.
Fox, J. 2002. An R and S-Plus Companion to Applied Regression. Thousand Oaks, CA: Sage.
———. 2002. “Bootstrapping Regression Models.” http://mng.bz/pY9m.
———. 2008. Applied Regression Analysis and Generalized Linear Models. Thousand Oaks, CA: Sage.
Fwa, T., ed. 2006. The Handbook of Highway Engineering, 2nd ed. Boca Raton, FL: CRC Press.
Good, P. 2006. Resampling Methods: A Practical Guide to Data Analysis, 3rd ed. Boston: Birkhäuser.
Gorsuch, R. L. 1983. Factor Analysis, 2nd ed. Hillsdale, NJ: Lawrence Erlbaum.
Greene, W. H. 2003. Econometric Analysis, 5th ed. Upper Saddle River, NJ: Prentice Hall.
Grissom, R., and J. Kim. 2005. Effect Sizes for Research: A Broad Practical Approach. Mahwah, NJ: Lawrence Erlbaum.
Groemping, U. 2009. “CRAN Task View: Design of Experiments (DoE) and Analysis of Experimental Data.” http://mng.bz/r45q.
Hand, D. J. and C. C. Taylor. 1987. Multivariate Analysis of Variance and Repeated Measures. London: Chapman & Hall.
Harlow, L., S. Mulaik, and J. Steiger. 1997. What If There Were No significance Test? Mahwah, NJ: Lawrence Erlbaum.
Hayton, J. C., D. G. Allen, and V. Scarpello. 2004. “Factor Retention Decisions in Exploratory Factor Analysis: A Tutorial on Parallel Analysis.” Organizational Research Methods 7:191–204.
Hsu, S., M. Wen, and M. Wu. 2009. “Exploring User Experiences as Predictors of MMORPG Addiction.” Computers and Education 53:990–999.
Jacoby, W. G. 2006. “The Dot Plot: A Graphical Display for Labeled Quantitative Values.” Political Methodologist 14:6–14.
Johnson, J. 2004. “Factors Affecting Relative Weights: The Influence of Sample and Measurement Error.” Organizational Research Methods 7:283–299.
Johnson, J., and J. Lebreton. 2004. “History and Use of Relative Importance Indices in Organizational Research.” Organizational Research Methods 7:238–257.
Koch, G., and S. Edwards. 1988. “Clinical Efficiency Trials with Categorical Data.” In Statistical Analysis with Missing Data, 2nd ed., by R. J. A. Little and D. Rubin. Hoboken, NJ: John Wiley & Sons, 2002.REFERENCES 3
LeBreton, J. M, and S. Tonidandel. 2008. “Multivariate Relative Importance: Extending Relative Weight Analysis to Multivariate Criterion Spaces.” Journal of Applied Psychology 93:329–345.
Lemon, J., and A. Tyagi. 2009. “The Fan Plot: A Technique for Displaying Relative Quantities and Differences.” Statistical Computing and Graphics Newsletter 20:8–10.
Licht, M. 1995. “Multiple Regression and Correlation.” In Reading and Understanding Multivariate Statistics, edited by L. Grimm and P. Yarnold. Washington, DC: American Psychological Association, 19–64.
McCall, R. B. 2000. Fundamental Statistics for the Behavioral Sciences, 8th ed. New York: Wadsworth.
McCullagh, P., and J. Nelder. 1989. Generalized Linear Models, 2nd ed. Boca Raton, FL: Chapman & Hall.
Meyer, D., A. Zeileis, and K. Hornick. 2006. “The Strucplot Framework: Visualizing Multi-way Contingency Tables with vcd.” Journal of Statistical Software 17:1–48. www.jstatsoft.org/v17/i03/paper.

Montgomery, D. C. 2007. Engineering Statistics. Hoboken, NJ: John Wiley & Sons.
Mooney, C., and R. Duval. 1993. Bootstrapping: A Nonparametric Approach to Statistical Inference. Monterey, CA: Sage.
Mulaik, S. 2009. Foundations of Factor Analysis, 2nd ed. Boca Raton, FL: Chapman & Hall.
Murphy, K., and B. Myors. 1998. Statistical Power Analysis: A Simple and General Model for Traditional and Modern Hypothesis Tests. Mahwah, NJ: Lawrence Erlbaum.
Murrell, P. 2006. R Graphics. Boca Raton, FL: Chapman & Hall/CRC.
Nenadic, O., and M. Greenacre. 2007. "Correspondence Analysis in R, with Two- and ThreeDimensional Graphics: The ca Package." Journal of Statistical Software 20 (3). www.jstatsoft.org/v20/i03/.
Peace, K. E., ed. 1987. Biopharmaceutical Statistics for Drug Development. New York: Marcel Dekker, 403–451.
Pena, E., and E. Slate. 2006. "Global Validation of Linear Model Assumptions." Journal of the American Statistical Association 101:341–354.
Pinheiro, J. C., and D. M. Bates. 2000. Mixed-Effects Models in S and S-PLUS. New York: Springer.
Potvin, C., M. J. Lechowicz, and S. Tardif. 1990. "The Statistical Analysis of Ecophysiological Response Curves Obtained from Experiments Involving Repeated Measures." Ecology 71:1389–1400.
Rosenthal, R., R. Rosnow, and D. Rubin. 2000. Contrasts and Effect Sizes in Behavioral Research: A Correlational Approach. Cambridge, UK: Cambridge University Press.
Sarkar, D. 2008. Lattice: Multivariate Data Visualization with R. New York: Springer.
Schafer, J., and J. Graham. 2002. "Missing Data: Our View of the State of the Art." Psychological Methods 7:147–177.
Schlomer, G., S. Bauman, and N. Card. 2010. "Best Practices for Missing Data Management in Counseling Psychology." Journal of Counseling Psychology 57:1–10.
Shah, A. 2005. "Getting Started with the boot Package." www.mayin.org/ajayshah/KB/R/documents/boot.html.
Silva, R. B., D. F. Ferreirra, and D. A. Nogueira. 2008. "Robustness of Asymptotic and Bootstrap Tests for Multivariate Homogeneity of Covariance Matrices." Ciênc. agrotec. 32:157–166.
Simon, J. 1997. "Resampling: The New Statistics." www.resample.com/content/text/index.shtml.4 REFERENCES
Snedecor, G. W., and W. G. Cochran. 1988. Statistical Methods, 8th ed. Ames, IA: Iowa State University Press.
UCLA: Academic Technology Services, Statistical Consulting Group. 2009. http://mng.bz/a9c7.
van Buuren, S., and K. Groothuis-Oudshoorn. 2010. "MICE: Multivariate Imputation by Chained Equations in R." Journal of Statistical Software, forthcoming. http://mng.bz/3EH5.
Venables, W. N., and B. D. Ripley. 1999. Modern Applied Statistics with S-PLUS, 3rd ed. New York: Springer.
———. 2000. S Programming. New York: Springer.
Westfall, P. H., et al. 1999. Multiple Comparisons and Multiple Tests Using the SAS System. Cary, NC: SAS Institute.
Wickham, H. 2009a. ggplot2: Elegant Graphics for Data Analysis. New York: Springer.
———. 2009b. "A Layered Grammar of Graphics." Journal of Computational and Graphical Statistics 19:3–28.
Wilkinson, L. 2005. The Grammar of Graphics. New York: Springer-Verlag.
Yu, C. H. 2003. "Resampling Methods: Concepts, Applications, and Justification." Practical Assessment, Research & Evaluation, 8 (19). http://pareonline.net/getvn.asp?v=8&n=19.
Yu-Sung, S., et al. 2010. "Multiple Imputation with Diagnostics (mi) in R: Opening Windows into the Black Box." Journal of Statistical Software. www.jstatsoft.org.
Zuur, A. F., et al. 2009. Mixed Effects Models and Extensions in Ecology with R. New York: Springer.