



R

В ДЕЙСТВИИ

Анализ и визуализация
данных на языке R

Роберт И.Кабаков



Роберт И. Кабаков

R в действии

Анализ и визуализация данных в программе R

R in Action

Data analysis and graphics with R

ROBERT I. KABACOFF



MANNING
SHELTER ISLAND

R в действии

Анализ и визуализация данных в программе R

РОБЕРТ И. КАБАКОВ



Москва, 2014

УДК 311:004.9R

ББК 60.6с515

K12

K12 Роберт И. Кабаков

R в действии. Анализ и визуализация данных в программе R / пер. с англ. Полины А. Волковой. – М.: ДМК Пресс, 2014. – 588 с.: ил.

ISBN 978-5-947060-077-1

R – это мощный язык для статистических вычислений и графики, который может справиться поистине с любой задачей в области обработки данных. Он работает во всех важных операционных системах и поддерживает тысячи специализированных модулей и утилит. Все это делает R замечательным средством для извлечения полезной информации из гор сырых данных.

«R в действии» – это руководство по обучению этому языку с особым вниманием к практическим задачам. В данной книге представлены полезные примеры статистической обработки данных и описаны изящные методы работы с запутанными и неполными данными, а также с данными, распределение которых отлично от нормального и с которыми трудно справиться обычными методами. Статистический анализ – это только одна сторона дела. Вы также овладеете обширными графическими возможностями для визуального исследования и представления данных.

УДК 311:004.9R

ББК 60.6с515

Original English language edition published by Manning Publications Co., Rights and Contracts Special Sales Department, 20 Baldwin Road, PO Box 261, Shelter Island, NY 11964. ©2012 by Manning Publications Co.. Russian-language edition copyright © 2013 by DMK Press. All rights reserved.

Все права защищены. Любая часть этой книги не может быть воспроизведена в какой бы то ни было форме и какими бы то ни было средствами без письменного разрешения владельцев авторских прав.

Материал, изложенный в данной книге, многократно проверен. Но, поскольку вероятность технических ошибок все равно существует, издательство не может гарантировать абсолютную точность и правильность приводимых сведений. В связи с этим издательство не несет ответственности за возможные ошибки, связанные с использованием книги.

ISBN 978-1-93518-239-9 (англ.)

ISBN 978-5-97060-077-1 (рус.)

©2012 by Manning Publications Co.

© Оформление, перевод на русский язык
ДМК Пресс, 2014



ОТ ПЕРЕВОДЧИКА

По моему глубокому убеждению, на сегодняшний день это лучшая книга, посвященная обработке данных в статистической среде R, для неспециалистов. Я рада, что теперь она стала доступной русскоязычным читателям. Надеюсь, мой перевод не сильно испортил эту книгу. По крайней мере, в некоторых местах она точно стала лучше, потому что я исправила довольно многочисленные и не всегда безобидные опечатки, обнаруженные в исходном издании мною и другими читателями, которые оставили свои отзывы на форуме издательства.

Я благодарю Александра Лободу, который рассказал мне о существовании данной книги, Дмитрия Мовчана, с энтузиазмом воспринявшего мое предложение опубликовать ее перевод, Бориса Демешева за консультации по переводу некоторых статистических терминов и Алексея Шипунова за техническую поддержку. Я особенно признательна Сергею Петрову и Сергею Мастицкому за внимательное прочтение рукописи перевода и конструктивные замечания.



ОГЛАВЛЕНИЕ

От переводчика	5
Предисловие	15
Благодарности.....	18
Об этой книге	20
Об иллюстрации на обложке.....	26
 ЧАСТЬ I.	
Начало работы	27
Глава 1. Знакомство с R	30
1.1. Зачем использовать R?	32
1.2. Получение и установка R	35
1.3. Работа в R	35
1.3.1. Начало работы	36
1.3.2. Как получить помошь	39
1.3.3. Рабочее пространство.....	40
1.3.4. Ввод и вывод.....	43
1.4. Пакеты	44
1.4.1. Что такое пакеты?	44
1.4.2. Установка пакета.....	46
1.4.3. Загрузка пакета.....	46
1.4.4. Получение информации о пакете	46
1.5. Пакетная обработка	47
1.6. Использование вывода в качестве ввода – повторное использование результатов	48
1.7. Работа с большими массивами данных	49
1.8. Учимся на примере	49
1.9. Резюме	51
Глава 2. Создание набора данных.....	52
2.1. Что такое набор данных?	53
2.2. Структуры данных	54
2.2.1. Векторы	55
2.2.2. Матрицы	56
2.2.3. Массивы данных	58



2.2.4. Таблицы данных	59
2.2.5. Факторы.....	63
2.2.6. Списки	65
2.3. Ввод данных.....	67
2.3.1. Ввод данных с клавиатуры.....	68
2.3.2. Импорт данных из текстового файла с разделителями	69
2.3.3. Импорт данных из Excel.....	71
2.3.4. Импорт данных из XML-файлов	72
2.3.5. Извлечение данных из веб-страниц.....	72
2.3.6. Импорт данных из SPSS	72
2.3.7. Импорт данных из SAS	73
2.3.8. Импорт данных из Stata	73
2.3.9. Импорт данных из netCDF	74
2.3.10. Импорт данных из HDF5	74
2.3.11. Импорт данных из систем управления базами данных.....	75
2.3.12. Импорт данных при помощи Stat/Transfer	77
2.4. Аннотирование наборов данных.....	77
2.4.1. Подписи для переменных.....	78
2.4.2. Пояснение значений переменных	78
2.5. Полезные функции для работы с объектами.....	79
2.6. Резюме	80
Глава 3. Начало работы с диаграммами	81
3.1. Работа с диаграммами.....	82
3.2. Простой пример.....	84
3.3. Графические параметры	86
3.3.1. Символы и линии	87
3.3.2. Цвета	88
3.3.3. Характеристики текста	90
3.3.4. Размеры диаграммы и полей	93
3.4. Добавление текста, настройка параметров осей и условных обозначений.....	95
3.4.1. Заголовки	95
3.4.2. Оси	96
3.4.3. Опорные линии	99
3.4.4. Легенда.....	100
3.4.5. Аннотации	102
3.5. Объединение диаграмм	105
3.5.1. Полный контроль над расположением диаграмм.....	110
3.9. Резюме	112
Глава 4. Основы управления данными	113
4.1. Рабочий пример.....	113
4.2. Создание новых переменных	116
4.3. Перекодировка переменных	117
4.4. Переименование переменных.....	119

4.5. Пропущенные значения	121
4.5.1. Перекодировка значений в отсутствующие	122
4.5.2. Исключение пропущенных значений из анализа.....	122
4.6. Календарные даты как данные	124
4.6.1. Преобразование дат в текстовые переменные	126
4.6.2. Получение дальнейшей информации	126
4.7. Преобразования данных из одного типа в другой.....	127
4.8. Сортировка данных	128
4.9. Объединение наборов данных	129
4.9.1. Добавление столбцов	129
4.9.2. Добавление строк	130
4.10. Разделение наборов данных на составляющие	130
4.10.1. Выбор переменных	130
4.10.2. Исключение переменных.....	131
4.10.3. Выбор наблюдений	132
4.10.4. Функция subset()	133
4.10.5. Случайные выборки.....	134
4.11. Использование команд SQL для преобразования таблиц данных	135
4.12. Резюме	136

Глава 5. Более сложные способы управления данными

137

5.1. Задача по управлению данными, которую нужно решить	138
5.2. Числовые и текстовые функции.....	139
5.2.1. Математические функции	139
5.2.2. Статистические функции.....	140
5.2.3. Функции распределения	143
5.2.4. Текстовые функции	148
5.2.5. Другие полезные функции.....	149
5.2.6. Применение функций к матрицам и таблицам данных.....	151
5.3. Решение нашей задачи по управлению данными	152
5.4. Управление выполнением команд.....	157
5.4.1. Повторение и циклы	158
5.4.2. Выполнение при условии	159
5.5. Функции, написанные пользователем	160
5.6. Агрегирование и изменение структуры данных	163
5.6.1. Транспонирование	163
5.6.2. Агрегирование данных	164
5.6.3. Пакет reshape	165
5.7. Резюме	167

ЧАСТЬ II. Базовые методы

169



Глава 6. Базовые диаграммы	171
6.1. Столбчатые диаграммы	172
6.1.1. Простые столбчатые диаграммы	172
6.1.2. Столбчатые диаграммы: составные и с группировкой	174
6.1.3. Столбчатые диаграммы для средних значений.....	175
6.1.4. Оптимизация столбчатых диаграмм	177
6.1.5. Спинограммы.....	178
6.2. Круговые диаграммы	179
6.3. Гистограммы.....	182
6.4. Диаграммы ядерной оценки функции плотности	185
6.5. Диаграммы размахов.....	188
6.5.1. Использование диаграмм размахов для сравнения групп между собой	189
6.5.2. Скрипичные диаграммы	193
6.6. Точечные диаграммы	194
6.7. Резюме	197
Глава 7. Основные методы статистической обработки данных.....	198
7.1. Описательные статистики	199
7.1.1. Калейдоскоп методов	200
7.1.2. Вычисление описательных статистик для групп данных	204
7.1.3. Визуализация результатов	208
7.2. Таблицы частот и таблицы сопряженности	208
7.2.1. Создание таблиц частот	209
7.2.2. Тесты на независимость.....	216
7.2.3. Показатели взаимосвязи.....	218
7.2.4. Визуализация результатов	219
7.2.5. Преобразование таблиц в неструктурированные файлы	219
7.3. Корреляции	221
7.3.1. Типы корреляций.....	222
7.3.2. Проверка статистической значимости корреляций	225
7.3.3. Визуализация корреляций	228
7.4. Тесты Стьюдента.....	228
7.4.1. Тест Стьюдента для независимых выборок.....	229
7.4.2. Тест Стьюдента для зависимых выборок	230
7.4.3. Когда имеется больше двух групп.....	231
7.5. Непараметрические тесты межгрупповых различий.....	231
7.5.1. Сравнение двух групп.....	231
7.5.2. Сравнение более двух групп.....	233
7.6. Визуализация групповых различий	236
7.7. Резюме	236
ЧАСТЬ III.	
Методы обработки данных средней сложности ...	237

Глава 8. Регрессия	239
8.1. Многоликая регрессия	241
8.1.1. Ситуации, в которых используется МНК-регрессия	242
8.1.2. Что вам нужно знать	244
8.2. МНК-регрессия	244
8.2.1. Подгонка регрессионных моделей при помощи команды lm()	245
8.2.2. Простая линейная регрессия	247
8.2.3. Полиномиальная регрессия	250
8.2.4. Множественная линейная регрессия	253
8.2.5. Множественная линейная регрессия со взаимодействиями	257
8.3. Диагностика регрессионных моделей	259
8.3.1. Стандартный подход	260
8.3.2. Усовершенствованный подход	264
8.3.3. Общая проверка выполнения требований, предъявляемых к линейным моделям	272
8.3.4. Мультиколлинеарность	273
8.4. Необычные наблюдения	274
8.4.1. Выбросы	275
8.4.2. Точки высокой напряженности	275
8.4.3. Влиятельные наблюдения	277
8.5. Способы корректировки	281
8.5.1. Удаление наблюдений	281
8.5.2. Преобразование переменных	281
8.5.3. Добавление или удаление переменных	284
8.5.4. Попытка применить другой подход	284
8.6. Выбор «лучшей» регрессионной модели	285
8.6.1. Сравнение моделей	285
8.6.2. Выбор переменных	286
8.7. Продолжение анализа	291
8.7.1. Кросс-валидация	292
8.7.2. Относительная важность	294
8.8. Резюме	298
Глава 9. Дисперсионный анализ	299
9.1. Ускоренный курс терминологии	300
9.2. Подгонка ANOVA-моделей	304
9.2.1. Функция aov()	304
9.2.2. Порядок членов в формуле	305
9.3. Однофакторный дисперсионный анализ	307
9.3.1. Множественные сравнения	308
9.3.2. Проверка справедливости допущений, лежащих в основе теста	312
9.4. Однофакторный ковариационный анализ	314
9.4.1. Проверка допущений, лежащих в основе теста	316



9.4.2. Визуализация результатов	317
9.5. Двухфакторный дисперсионный анализ	318
9.6. Дисперсионный анализ для повторных измерений	323
9.7. Многомерный дисперсионный анализ	326
9.7.1. Проверка предположений, лежащих в основе теста	328
9.7.2. Устойчивый многомерный дисперсионный анализ	330
9.8. Дисперсионный анализ как регрессия	331
9.9. Резюме	333
Глава 10. Анализ мощности	335
10.1. Краткий обзор процедуры проверки гипотез.....	336
10.2. Проведение анализа мощности при помощи	
пакета pwr	339
10.2.1. Тесты Стьюдента	340
10.2.2. Дисперсионный анализ	342
10.2.3. Корреляции.....	343
10.2.4. Линейные модели	344
10.2.5. Сравнение пропорций.....	345
10.2.6. Тесты хи-квадрат	346
10.2.7. Выбор подходящего размера эффекта в незнакомых	
ситуациях	348
10.3. Графический анализ мощности	350
10.4. Другие пакеты.....	352
10.5. Резюме	354
Глава 11. Диаграммы средней сложности.....	356
11.1. Диаграммы рассеяния	357
11.1.1. Матрицы диаграмм рассеяния	361
11.1.2. Диаграммы рассеяния высокой плотности.....	367
11.1.3. Трехмерные диаграммы рассеяния	370
11.1.4. Пузырьковые диаграммы	375
11.2. Линейные графики.....	377
11.3. Кореллогramмы	382
11.4. Мозаичные диаграммы	388
11.5. Резюме	391
Глава 12. Статистика повторных выборок	
и бутстреп-анализ	392
12.1. Перестановочные тесты	393
12.2. Перестановочные тесты в пакете coin	395
12.2.1. Тесты на независимость для двух и k выборок	397
12.2.2. Независимость в таблицах сопряженности	399
12.2.3. Независимость между числовыми переменными	400
12.2.4. Тесты для двух и k зависимых выборок	400
12.2.5. Дополнительная информация	401

12.3. Перестановочные тесты, реализованные в пакете lmPerm.....	401
12.3.1. Простая и полиномиальная регрессия	402
12.3.2. Множественная регрессия	403
12.3.3. Однофакторные дисперсионный и ковариационный анализы.....	404
12.3.4. Двухфакторный дисперсионный анализ	405
12.4. Дополнительные замечания о перестановочных тестах	407
12.5. Бутстреп-анализ	408
12.6. Бутстреп-анализ при помощи пакета boot.....	409
12.6.1. Бутстреп-анализ для одной статистики	411
12.6.2. Бутстреп-анализ для нескольких статистик	413
12.7. Резюме	416
ЧАСТЬ IV.	
Продвинутые методы	417
Глава 13. Обобщенные линейные модели	419
13.1. Обобщенные линейные модели и функция glm()	420
13.1.1. Функция glm().....	421
13.1.2. Вспомогательные функции.....	423
13.1.3. Соответствие модели данным и регрессионная диагностика.....	424
13.2. Логистическая регрессия.....	425
13.2.1. Интерпретация параметров модели	428
13.2.2. Оценка влияния независимых переменных на вероятность исхода.....	430
13.2.3. Избыточная дисперсия.....	431
13.2.4. Дополнительные методы.....	432
13.3. Пуассоновская регрессия	433
13.3.1. Интерпретация параметров модели	436
13.3.2. Избыточная дисперсия.....	437
13.3.3. Дополнительные методы.....	439
13.4. Резюме	442
Глава 14. Главные компоненты и факторный анализ	443
14.1. Выполнение анализа главных компонент и факторного анализа в R	446
14.2. Главные компоненты	447
14.2.1. Выбор необходимого числа компонент	449
14.2.2. Выделение главных компонент	451
14.2.3. Вращение главных компонент	455
14.2.4. Вычисление значений главных компонент	456
14.3. Разведочный факторный анализ	459



14.3.1. Определение числа извлекаемых факторов	460
14.3.2. Выделение общих факторов	462
14.3.3. Вращение факторов	463
14.3.4. Значения факторов	467
14.3.5. Другие пакеты для проведения факторного анализа	468
14.4. Другие модели для латентных переменных	468
14.5. Резюме	470
Глава 15. Продвинутые методы работы с пропущенными данными	472
15.1. Этапы работы с пропущенными данными	474
15.2. Обнаружение пропущенных значений	476
15.3. Исследование структуры пропущенных данных	477
15.3.1. Представление пропущенных значений в виде таблицы	478
15.3.2. Визуальное исследование структуры пропущенных данных.....	479
15.3.3. Использование корреляции для исследования пропущенных значений	482
15.4. Выявление источников пропущенных данных и эффекта от них	484
15.5. Рациональный подход	486
15.6. Анализ полных строк (постстрочное удаление)	487
15.7. Метод множественного восстановления пропущенных данных.....	489
15.8. Другие подходы к пропущенным данным	495
15.8.1. Попарное удаление	496
15.8.2. Простое (нестохастическое) восстановление данных.....	496
15.9. Резюме	497
Глава 16. Продвинутые графические методы	499
16.1. Четыре графические системы R	500
16.2. Пакет lattice	501
16.2.1. Условные переменные	507
16.2.2. Функции для изменения формата ячеек	509
16.2.3. Группировка переменных	512
16.2.4. Графические параметры	518
16.2.5. Расположение диаграмм на странице	519
16.3. Пакет ggplot2	520
16.4. Интерактивная графика	526
16.4.1. Взаимодействие с диаграммами: идентификация точек	527
16.4.2. Пакет playwith	527
16.4.3. Пакет latticist	529
16.4.4. Создание интерактивной графики при помощи пакета ipplots	530



16.4.5. Пакет rggobi	532
16.5. Резюме	533
Послесловие: В погоне за кроликом	535
Приложение А.	
Графические пользовательские интерфейсы	539
Приложение В.	
Настройка начальной конфигурации программы ...	543
Приложение С.	
Экспорт данных из R	545
С.1. Текстовый файл с разделителями.....	545
С.2. Таблица Excel.....	545
С.3. Другие статистические программы	546
Приложение D.	
Сохранение результатов в пригодном	
для публикации качестве	547
D.1. Подготовка отчета типографского качества	
при помощи пакета Sweave (R + LaTeX)	548
D.2. Объединение сил с OpenOffice при помощи	
пакета odfWeave	554
D.3. Комментарии.....	557
Приложение Е.	
Матричная алгебра в R	558
Приложение F.	
Пакеты, упомянутые в этой книге	561
Приложение G.	
Работа с большими наборами данных	570
G.1. Эффективное программирование	571
G.2. Хранение данных вне оперативной памяти	572
G.3. Аналитические пакеты для больших объемов данных	573
Приложение H.	
Обновление версии R	574
Список литературы	576
Указатель пакетов и функций.....	581



ПРЕДИСЛОВИЕ

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

Алиса. «Алиса в Стране чудес»¹

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

Кью. Сериал «Звездный путь: следующее поколение»

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

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

Я впервые познакомился с R несколько лет назад, когда хотел получить новую должность консультанта по статистике. Предполагаемый работодатель перед интервью спросил меня, владею ли я R. Следуя обычным советам специалистов по подбору персонала, я немедленно сказал «да» и стал учиться работать в этой программе. Я был опытным статистиком и исследователем с 25 годами опыта

¹ Перевод Н. Демуровой.

программирования в SAS и SPSS, свободно владевшим несколькими языками программирования. Чего же тут может быть сложного? Знаменитые последние слова.

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

При освоении R я подошел к процессу с точки зрения исследователя, которому нужно обрабатывать данные. Я пытался понять, что нужно сделать, чтобы успешно обработать, проанализировать и понять данные, включая:

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

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

Затем, около года назад, Марьян Бейс (Marjan Bace), издатель, позвонила и спросила, не хочу ли я написать книгу про R. К этому времени я уже написал 50 статей в научных журналах, четыре технических руководства, многочисленные главы в книгах и целую книгу по методологии исследования, так чего же тут могло быть сложного? Рискуя повториться – знаменитые последние слова.

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

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



БЛАГОДАРНОСТИ

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

- в первую очередь это Марьян Бейс (Marjan Bace), глава издательства Маннинг (Manning), которая предложила мне написать эту книгу;
- Себастьян Стирлинг (Sebastian Stirling), редактор-консультант по аудитории (development editor), который провел многие часы в телефонных беседах со мной, помогая выстроить материал, прояснить основные идеи и в целом сделать текст более интересным. Он также помог мне на многих этапах подготовки книги к изданию;
- Карен Тегмейер (Karen Tegtmeyer), редактор-рецензент (review editor), которая помогла найти рецензентов и координировала процесс рецензирования;
- Мэри Пиргис (Mary Piergies), которая помогала следить за процессом подготовки книги к печати, и ее команду: Лиз Велч (Liz Welch), Сьюзан Харкинс (Susan Harkins) и Рахель Шредер (Rachel Schroeder);
- Пабло Доминик Васелли (Pablo Dominguez Vaselli), корректор, который помог обнаружить ошибки и свежим опытным взглядом проверил программный код;
- рецензенты, которые потратили много времени на внимательное чтение текста, находили опечатки и делали ценные замечания: Крис Вилльямс (Chris Williams), Чарльз Мальпас (Charles Malpas), Анжела Стейплз (Angela Staples), Даниэль Рейс Перейра (Daniel Reis Pereira), Д.Х. Ван Райн (D. H. van Rijn), Кристиан Маркуардт (Christian Marquardt), Амос Фоларин (Amos Folarin), Стюарт Джейфрис (Stuart Jefferys), Дрор Берел (Dror Berel), Патрик Брин (Patrick Breen), Элизабет Островски (Elizabeth Ostrowski), Атеф Оуни (Atef Ouni), Карл Феноллоза (Carles Fenollosa), Рикардо Питробон (Ricardo Pietrobon), Самуэль МакКвиллин (Samuel McQuillin),

Ландон Кокс (Landon Cox), Августин Циглер (Austin Ziegler), Рик Вагнер (Rick Wagner), Райн Кокс (Ryan Cox), Сумит Пал (Sumit Pal), Филипп К. Джантер (Philipp K. Janert), Дипак Ворхара (Deepak Vohra) и Софи Мормеде (Sophie Mormede);

- многие участники программы раннего доступа издательства Маннинг (Manning Early Access Program, MEAP).

Каждый из перечисленных людей сделал эту книгу лучше и полнее.

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

На самом деле мне следовало бы начать эту книгу с благодарности моей жене и другу Кэрол Линн (Carol Lynn). Хотя она особенно не интересуется статистикой или программированием, она неоднократно прочитала каждую главу и сделала бесчисленные исправления и предложения. Никаким другим способом нельзя выразить свою любовь к другому человеку лучше, чем прочесть ради него текст по многомерной статистике. Столь же важно, что она переживала многие вечера и выходные, которые я проводил в работе над этой книгой, с тиктом, поддержкой и сочувствием. И за что это мне так повезло?

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



ОБ ЭТОЙ КНИГЕ

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

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

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

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

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

зации данных и обретете достаточно умений, чтобы справиться как с простыми, так и со сложными задачами анализа данных.

Кому следует прочесть эту книгу

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

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

Структура книги

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

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

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

Многие пользователи изначально выбирают R потому, что они хотят создавать диаграммы, так что мы сразу переходим к этой теме в

главе 3. Вам не понадобится долго ждать. Мы обсуждаем, как создавать диаграммы, изменять их и сохранять в разных форматах.

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

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

В главе 6 рассказано, как создавать наиболее распространенные одномерные диаграммы, такие как столбчатая и круговая диаграммы, диаграмма распределения плотности, диаграмма размахов («ящик с усами») и точечная диаграмма. Все эти диаграммы полезны для изучения характера распределения значений одной переменной.

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

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

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

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

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

рассеяния, матриц диаграмм рассеяния, графиков, коррелограмм и мозаичных диаграмм.

В главе 12 представлены аналитические методы, которые хорошо работают, когда данные происходят из неизвестных или смешанных типов распределения, когда размеры выборок малы, когда выбросы представляют собой проблему или когда разработка статистического теста на основании наблюдаемого распределения слишком сложна. Это метод повторной выборки (*resampling*) и бутстреп-анализ (*bootstrapping*) – подходы, требующие большого объема вычислений и легко реализуемые в R.

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

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

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

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

В послесловии перечислены многие из лучших сайтов, которые следует посетить, чтобы научиться работать в R, влиться в сообщество пользователей R, получить ответы на возникшие вопросы и отсле-

живать изменения в этом стремительно развивающемся программном продукте.

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

Примеры

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

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

Наборы данных либо поставляются с базовой версией R, либо доступны в составе дополнительных пакетов, которые можно скачать из Интернета. Программный код для каждого примера размещен на сайте <http://www.manning.com/RinAction>. Для получения максимальной отдачи от этой книги я рекомендую выполнять примеры по ходу их прочтения.

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

Принятые обозначения

В книге использованы следующие типографские обозначения:

- моноширинный шрифт использован для программного кода, который нужно вводить именно так, как указано в книге;

- моноширинный шрифт также использован внутри основного текста для обозначения фрагментов кода или ранее упомянутых объектов;
- *курсив* внутри программного кода – это указатель места заполнения. Его следует заменять подходящим текстом или значениями, соответствующими вашей задаче. Например, *путь_к_моему_файлу* должен быть заменен указанием пути к реальному файлу на вашем компьютере;
- R – это интерактивный язык, который информирует пользователя о готовности принять команду приглашением (> по умолчанию). Многие фрагменты программного кода в книге скопированы из интерактивных сессий. Если вы видите строчки кода, которые начинаются с >, не набирайте этот символ приглашения к вводу команды;
- пояснения к программному коду приведены в виде внутритекстовых комментариев. В дополнение к этому некоторые пояснения обозначены нумерованными кружками, такими как ①, которые отсылают к объяснению ниже по тексту;
- для того чтобы сэкономить место или сделать текст более понятным, мы иногда добавляли в вывод результатов интерактивных сессий дополнительные пробелы или удаляли текст, который напрямую не относился к обсуждаемой теме.

Об авторе

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



ОБ ИЛЛЮСТРАЦИИ НА ОБЛОЖКЕ

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

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

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

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



Часть I.

НАЧАЛО РАБОТЫ

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

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

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

Как только вы познакомились с интерфейсом R, возникает следующая задача – загрузить ваши данные в программу. В современном богатом информацией мире данные могут поступать из разных источников и в разных форматах. В главе 2 описано множество методов, которые можно использовать для импорта данных в R. Первая половина главы посвящена описанию форматов, в которых R хранит данные, и

рассказу о том, как вводить данные вручную. Во второй части обсуждаются методы импорта данных из текстовых файлов, веб-страниц, электронных таблиц, других статистических программ и баз данных.

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

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

Глава 5 основана на материале, изложенном в главе 4. Здесь рассказано, как использовать числовые (арифметические, тригонометрические и статистические) и текстовые функции (разбиение строк, объединение и замену) в управлении данными. Для иллюстрации многих из описанных функций в этом разделе использованы многочисленные примеры. Далее разобраны управляющие конструкции (циклы, исполняемые при определенных условиях команды). После прочтения этого раздела вы научитесь создавать собственные функции в R. Это позволит расширить возможности R, объединив многие команды в одну легко настраиваемую функцию. В заключение обсуждаются мощные методы реорганизации и группировки данных, которые часто бывают полезными при подготовке данных к дальнейшему анализу.

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



ГЛАВА 1.

Знакомство с R

В этой главе:

- Устанавливаем R.
- Знакомимся с языком программирования R.
- Запускаем программу.

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

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

Появление персональных компьютеров изменило подход к анализу данных еще одним образом. Когда анализ данных проводился в

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

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

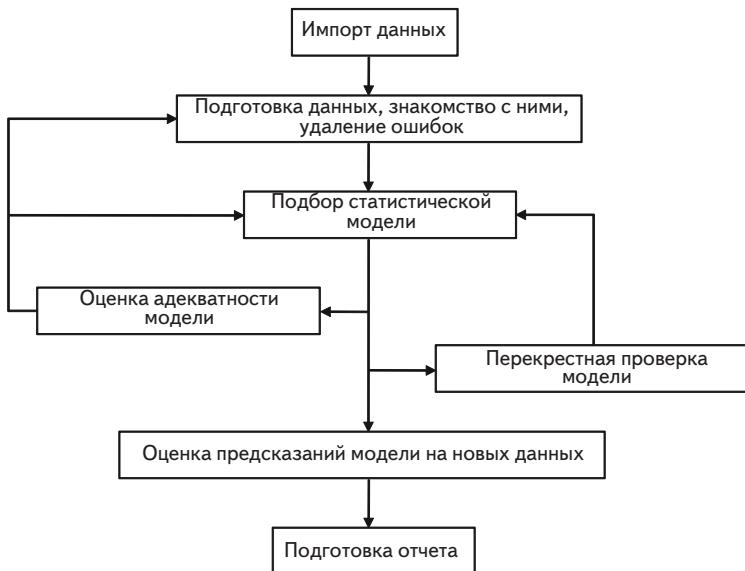


Рис. 1.1. Этапы типичного анализа данных

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

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

1.1. Зачем использовать R?

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

У R есть много особенностей, которые позволяют рекомендовать именно эту программу:

- большинство коммерческих статистических программ стоят тысячи, если не десятки тысяч долларов. R – это бесплатная программа! Если вы учитель или студент, выгода очевидна;
- R – это мощная статистическая программа, в которой реализованы все способы анализа данных;
- R имеет современные графические возможности. Если вам нужно визуализировать сложные данные, то учтите, что в R реализованы самые разнообразные и мощные методы анализа данных из доступных;
- получение данных из разных источников в пригодном для использования виде может быть сложной задачей. R может

импортировать данные из самых разных источников, включая текстовые файлы, системы управления базами данных, другие статистические программы и специализированные хранилища данных. R может также записывать данные в форматах всех этих систем;

- R представляет собой не имеющую аналогов платформу для простого написания программ, реализующих новые статистические методы;
- в R реализованы сложные статистические процедуры, еще недоступные в других программах. На самом деле новые функции становятся доступными для скачивания еженедельно. Если вы работаете в SAS, представьте, насколько невероятно было бы получать новую SAS-процедуру каждые несколько дней;
- если вы не хотите учить новый язык, существует множество графических пользовательских интерфейсов, в которых мощь R реализована в форме меню и диалогов;
- R работает на разных операционных системах, включая Windows, Unix и Mac OS X. Она, скорее всего, запустится на любом компьютере, который у вас есть (я даже видел руководства по установке R на iPhone, что впечатляет, но вряд ли является хорошей идеей).

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

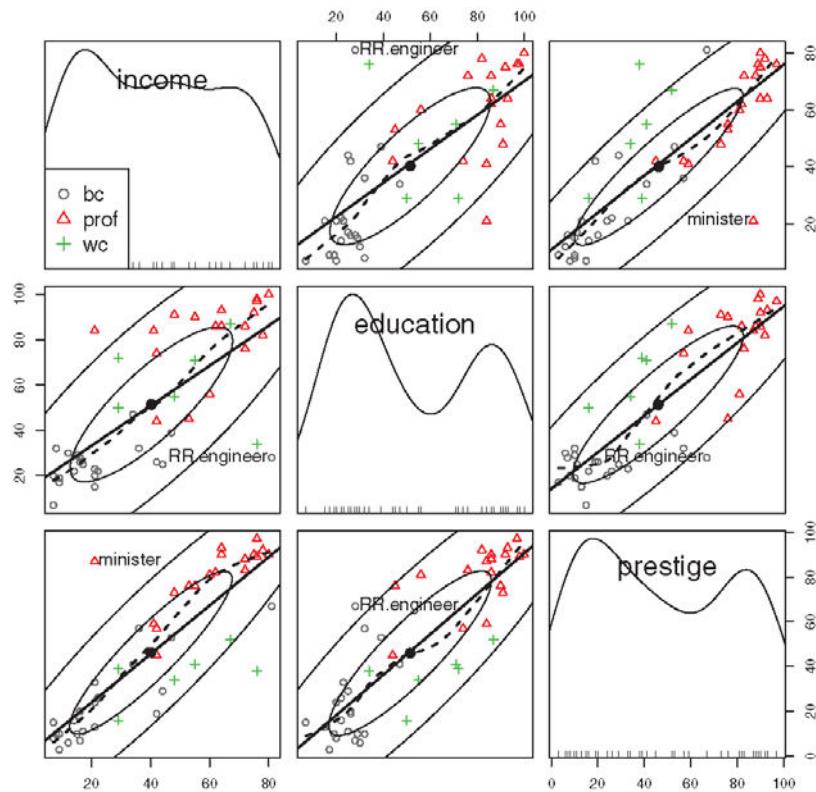


Рис. 1.2. Взаимосвязь между уровнем дохода (income), образования (education) и престижа (prestige) для представителей рабочих (blue-collar: bc), конторских (white-collar: wc) и специальных (prof) профессий. Источник: дополнительный пакет `car` (команда `scatterplotMatrix`, созданная Джоном Фоксом (John Fox)).

Диаграммы, подобные этой, сложно создавать в других статистических программах. В R их можно сделать при помощи одной-двух строк программного кода.

Вот главное, что показывает этот график:

- между уровнями образования, дохода и престижа работы существует линейная связь;
- в среднем рабочие профессии характеризуются более низким уровнем образования и престижа, а специальные профессии – более высоким. Конторские профессии занимают промежуточное положение;

- существует несколько интересных исключений. Инженеры на железных дорогах (RR.engineer) зарабатывают много, но имеют низкое образование. Быть министром (minister) престижно, но не прибыльно;
- уровни образования и (в меньшей степени) престижа профессии распределены бимодально, то есть низкие и высокие значения этих показателей встречаются чаще, чем средние.

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

К сожалению, для языка R свойственна пологая кривая обучения. Поскольку он может делать так много всего, объем документации и файлов помощи очень велик. Кроме того, т. к. многие из этих опций реализованы в дополнительных модулях, созданных независимыми исследователями, справочная документация бывает разобщенной и труднодоступной. Научиться выполнять все виды анализа, реализованные в R, действительно очень непросто.

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

1. 2. Получение и установка R

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

1.3. Работа в R

R – это чувствительный к регистру клавиатуры интерпретируемый язык программирования. Вы можете либо вводить команды по одной

в ответ на приглашение на ввод команды (>), либо запускать набор команд из исходного файла. Типы данных очень разнообразны: векторы, матрицы, таблицы данных и списки (совокупность нескольких объектов). Мы обсудим все эти типы данных в главе 2.

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

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

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

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

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

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

1.3.1. Начало работы

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

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

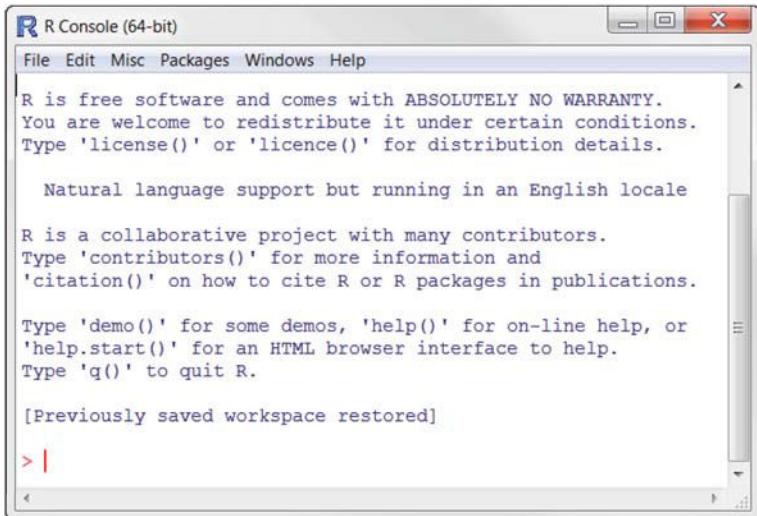


Рис. 1.3. Пример интерфейса R в операционной системе Windows

Таблица 1.1. Возраст и вес 10 младенцев

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

Примечание. Это выдуманные данные.

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

Программный код 1.1. Пример сессии в R

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

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

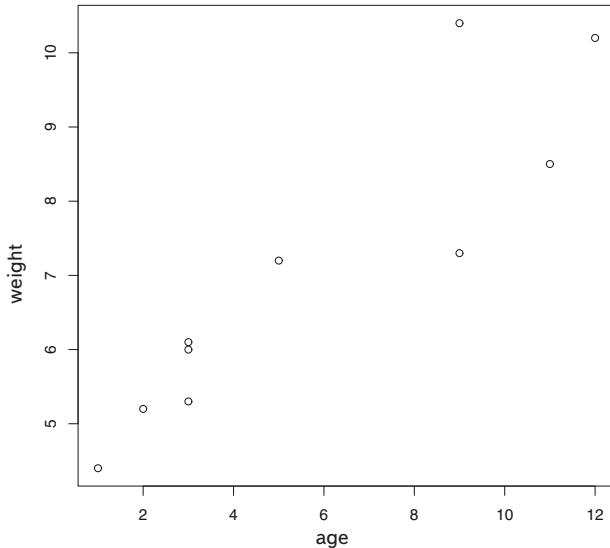


Рис. 1.4. Диаграмма рассеяния веса младенцев (weight, в килограммах) в зависимости от их возраста (age, в месяцах)

Диаграмма рассеяния на рис. 1.4 информативна, но выглядит не очень привлекательно. Из последующих глав вы узнаете, как модифицировать графики в соответствии с вашими потребностями.

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

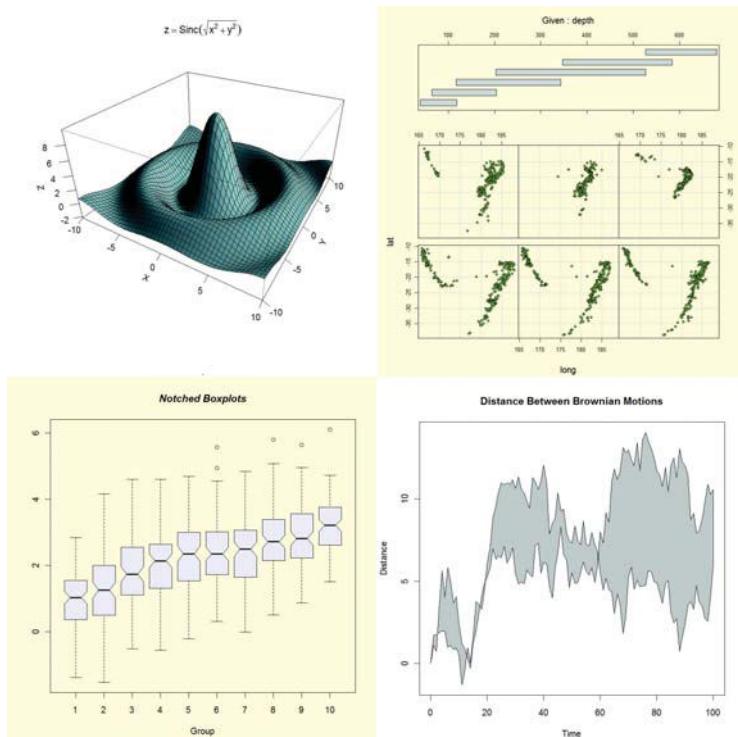


Рис. 1.5. Некоторые графики, которые появляются при вводе функции `demo()`

1.3.2. Как получить помощь

R обладает обширными справочными материалами. Научитесь ориентироваться в них, и это поможет вам при работе в программе. Встроенная система помощи содержит подробные разъяснения, ссылки на литературу и примеры для каждой функции из установленных пакетов¹. Справку можно вызвать при помощи функций, перечисленных в табл. 1.2.

1 На английском языке. – Прим. пер.

Таблица 1.2. Функции вызова справки в R

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

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

1.3.3. Рабочее пространство

Рабочее пространство – это текущая рабочая среда R в памяти вашего компьютера, которая включает в себя любые созданные пользователем объекты (векторы, матрицы, функции, таблицы данных или списки). В конце каждой сессии вы можете сохранить рабочее пространство, и оно автоматически загрузится при следующем запуске программы. Команды интерактивно вводятся в ответ на приглашение к их вводу. Можно использовать стрелки вверх и вниз для перемеще-

ния между введенными ранее командами. Это позволяет вызвать предыдущую команду, отредактировать ее и вновь выполнить ее, нажав клавишу **Enter**.

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

Некоторые стандартные команды для управления рабочим пространством приведены в табл. 1.3.

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

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

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

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

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

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

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

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

1.3.4. Ввод и вывод

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

Ввод

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

Текстовый вывод

Функция `sink ("имя_файла")` выводит все результаты в файл с названием `имя_файла`. По умолчанию, если этот файл уже существует, новая версия записывается поверх старой. Параметр `append=TRUE` позволяет добавлять новый текст в файл, а не записывать его вместо старого текста. Параметр `split=TRUE` позволяет выводить результаты и на экран, и в текстовый файл. Выполнение команды `sink ()` без аргументов восстановит вывод результатов только на экран.

Графический вывод

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

Таблица 1.4. Функции для сохранения графиков

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

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

Если вы наберете следующие команды:

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

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

Наконец, если вы введете такие команды:

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

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

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

1.4. Пакеты

R в базовой установке уже обладает обширными возможностями. Однако некоторые наиболее впечатляющие опции программы реализованы в дополнительных модулях, которые можно скачать и установить. Существует более 2500 созданных пользователями модулей, называемых *пакетами* (packages), которые вы можете скачать с <http://cran.r-project.org/web/packages>. В них заключены почти безграничные возможности – от анализа геостатистических данных до масс-спектроскопии белков и анализа психологических тестов! Мы используем многие из этих дополнительных пакетов по ходу повествования.

1.4.1. Что такое пакеты?

Пакеты – это собрания функций R, данных и скомпилированного программного кода в определенном формате. Директория, в которой

пакеты хранятся на вашем компьютере, называется библиотекой. Функция `.libPath()` показывает, где расположена ваша библиотека, а функция `library()` выводит на экран названия всех имеющихся в библиотеке пакетов.

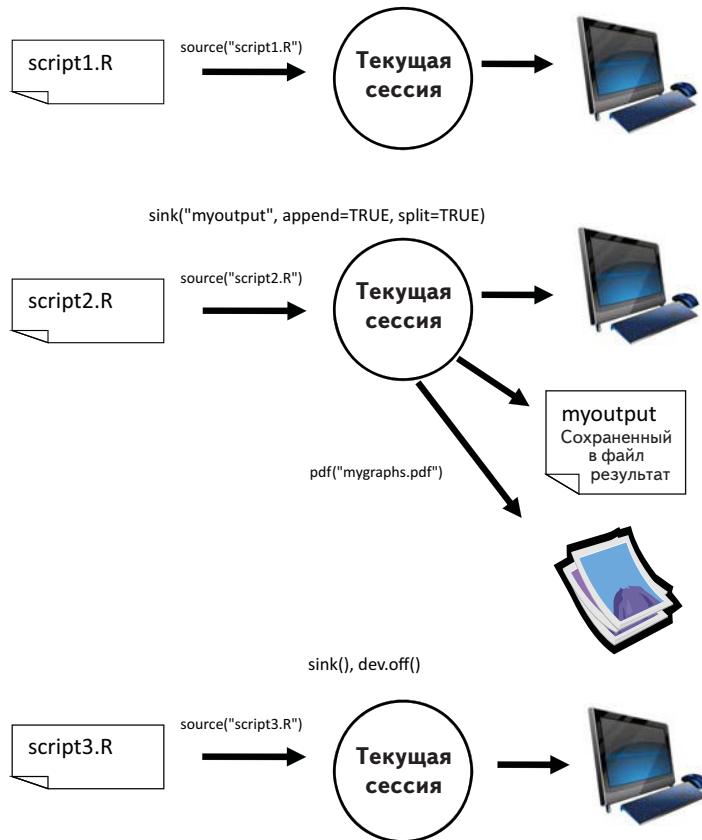


Рис. 1.6. Ввод при помощи функции `source()` и вывод посредством функции `sink()`

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

1.4.2. Установка пакета

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

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

1.4.3. Загрузка пакета

Когда вы устанавливаете пакет, он скачивается с сайта CRAN и загружается в вашу библиотеку. Для использования этого пакета в текущей сессии программы вам нужно загрузить его при помощи команды `library()`. Например, для того чтобы использовать пакет `gclus`, введите команду `library(gclus)`. Разумеется, прежде чем загрузить пакет, необходимо его установить. В течение сессии достаточно загрузить пакет один раз. При необходимости можно настроить рабочее пространство так, чтобы часто используемые пакеты автоматически загружались в начале каждой сессии. Настройка начала работы подробно описана в приложении B.

1.4.4. Получение информации о пакете

После загрузки пакета становятся доступны новые функции и наборы данных. Небольшие наборы данных поставляются вместе с демонстрационным программным кодом, что позволяет протестировать новые возможности. Справочная система содержит описание каждой функции (с примерами) и информацию о каждом встроенном наборе

данных. Функция `help(package="название_пакета")` выводит краткое описание этого пакета и алфавитный указатель всех входящих в него функций и наборов данных. Использование команды `help()` для названия каждой из этих функций или наборов данных позволит выяснить новые детали. Этую же информацию можно скачать с сайта CRAN в виде руководства в формате PDF.

Распространенные ошибки при программировании в R

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

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

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

1.5. Пакетная обработка

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

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

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

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

В Windows используйте команду

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

в которой прописаны пути к файлу *R.exe* и к файлу с программным кодом. Дополнительная информация по этой теме, в том числе использование опций в командной строке, содержится в пособии «Введение в R» («Introduction to R»)² на сайте CRAN (<http://cran.r-project.org>).

1.6. Использование вывода в качестве ввода – повторное использование результатов

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

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

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

Результаты выводятся на экран, но не сохраняются.

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

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

Это присвоение создало объект типа список с названием *lmfit*, в котором записаны подробные результаты анализа (включая предсказанные значения, остатки, коэффициенты регрессии и т. д.). Хотя при

2 На английском языке. – Прим. пер.

этом на экране ничего не появилось, результаты можно просматривать и подвергать дальнейшей обработке.

Команда `summary(lmfit)` выводит на экран сводную информацию (`summary`) о результатах, при помощи команды `plot(lmfit)` можно построить диагностические графики. Команда `cooks.distance(lmfit)` рассчитывает статистики воздействия (`influence statistics`), а команда `plot(cook)` отображает их графически. Для того чтобы предсказать расход топлива, зная вес машины, по новым данным, нужно использовать `predict(lmfit, mynewdata)`.

Для того чтобы узнать, какие значения возвращает функция, загляните в раздел «значение» (`Value`) онлайн-справки для этой функции³. В командной строке можно набрать `help(lm)` или `?lm`. Так вы узнаете, что было сохранено, когда вы присвоили результаты функции объекту.

1.7. Работа с большими массивами данных

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

1.8. Учимся на примере

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

1. Откройте общий файл справки и загляните в раздел «Введение в R».
2. Установите пакет `vcd` (пакет для визуализации категориальных данных, который мы будем более подробно рассматривать в главе 11).

3 На английском языке, как и все прочие упомянутые справочные материалы. – *Прим. нер.*

3. Просмотрите список всех функций и наборов данных, содержащихся в пакете.
4. Загрузите пакет и прочтите описание набора данных *Arthritis*.
5. Выведите этот набор данных на экран (набрав его название в командной строке).
6. Запустите пример, который прилагается к набору данных *Arthritis*. Не беспокойтесь, если вы не понимаете результаты. В общих чертах они показывают, что страдающие артритом пациенты, которые получали лекарство, выздоравливали гораздо быстрее по сравнению с теми, которые получали плацебо.
7. Выйдите из программы.

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

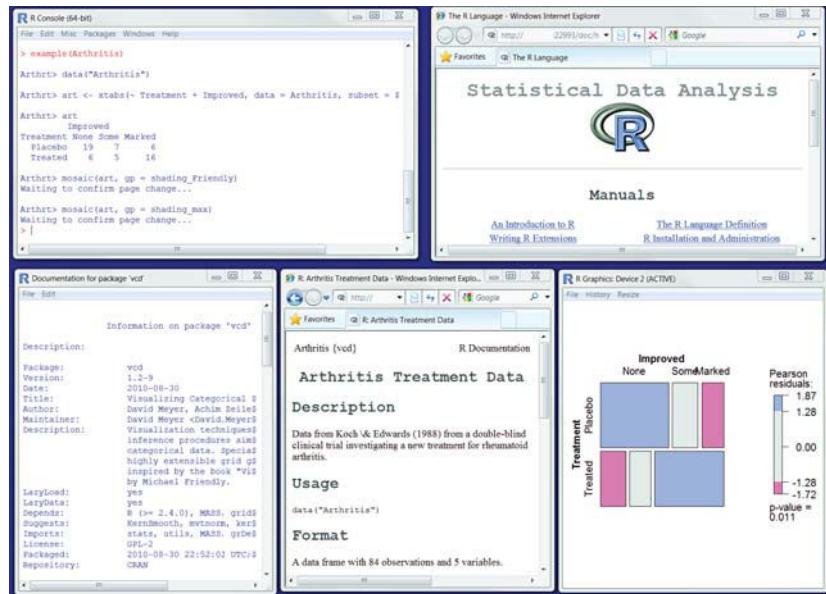


Рис. 1.7. Результат действия программного кода 1.3, включающий (слева направо) примеры обработки набора данных *Arthritis*, общую помощь, сведения о пакете *vcd*, сведения о наборе данных *Arthritis* и диаграмму, иллюстрирующую зависимость результата лечения артрита от способа лечения

Программный код 1.3. Работа с новым пакетом

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

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

1.9. Резюме

В этой главе мы рассмотрели некоторые сильные стороны R, которые делают эту программу привлекательной для студентов, статистиков и аналитиков, которые хотят понять, что значат их данные. Мы обсудили, как установить программу и как расширить ее возможности при помощи дополнительных пакетов. Мы исследовали основные характеристики интерфейса, запустили программу интерактивно и в пакетном режиме, а также создали несколько графиков для примера. Мы также узнали, как сохранять результаты работы в текстовых и графических файлах. Чтобы разобраться с возможными затруднениями, мы потратили некоторое время на знакомство со способами получения доступа к обширной справочной информации. Надеюсь, вы уже почувствовали, какой мощной может быть эта бесплатная программа.

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



ГЛАВА 2.

Создание набора данных

В этой главе:

- Изучаем структуры данных в R.
- Вводим данные.
- Импортируем данные.
- Аннотируем наборы данных.

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

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

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

Вторая часть главы (раздел 2.3) посвящена разным способам импорта данных в R. Данные можно вводить вручную или импортировать из внешнего источника. Таким источником могут быть текстовые файлы, электронные таблицы, статистические программы и системы управления базами данных. Например, я обычно работаю с данными, которые изначально содержатся в базах данных языка структурированных запросов (Structured Query Language, SQL). Хотя иногда я получаю данные из старомодных дисковых операционных систем

(Disk Operating System, DOS) и из современных баз данных SAS и SPSS. Вероятно, вам достаточно будет использовать один или два метода из описанных в этом разделе, так что просто выберите тот, что вам подходит.

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

2.1. Что такое набор данных?

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

Таблица 2.1. Набор данных о пациентах

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

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

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

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

Diabetes – номинальная (nominal) переменная и Status – это порядковая (или шкальная – ordered) переменная.

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

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

2.2. Структуры данных

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

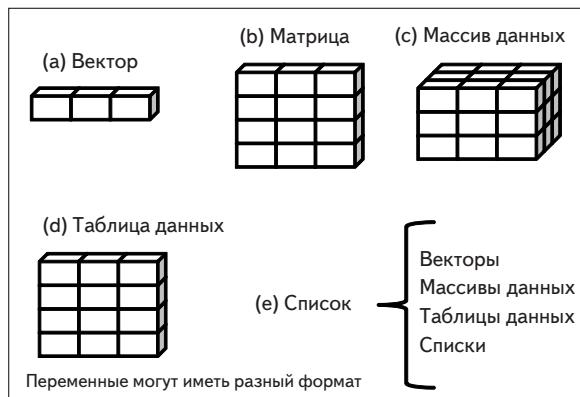


Рис. 2.1. Типы структуры данных в R

Некоторые определения

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

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

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

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

Большинство остальных терминов должны быть вам уже знакомы, они широко используются в статистике и в вычислениях.

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

2.2.1. Векторы

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

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

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

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

Отдельные элементы вектора можно вызывать при помощи числового вектора, состоящего из номеров элементов и заключенного в

квадратные скобки. Например, `a[c(2, 4)]` обозначает второй и четвертый элементы вектора `a`. Вот еще примеры:

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

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

2.2.2. Матрицы

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

```
mymatrix <- matrix(вектор, nrow=число_строк, ncol=число_столбцов,
                     byrow=логическое_значение, dimnames=list(
    текст_вектор_названия_строк, текст_вектор_названия_столбцов))
```

где `vector` содержит элементы матрицы, `nrow` и `ncol` определяют число строк и столбцов в матрице, а `dimnames` содержит названия строк и столбцов (их указывать не обязательно), которые хранятся в виде текстовых векторов. Параметр `byrow` определяет, как должна заполняться матрица – по строкам (`byrow=TRUE`) или по столбцам (`byrow=False`). По умолчанию матрица заполняется по столбцам. Приведенный ниже программный код иллюстрирует применение функции `matrix`.

Программный код 2.1. Создание матриц

```
> y <- matrix(1:20, nrow=5, ncol=4)
> y
 [,1] [,2] [,3] [,4]
[1,]    1     6    11    16
[2,]    2     7    12    17
[3,]    3     8    13    18
[4,]    4     9    14    19
[5,]    5    10    15    20
> cells   <- c(1,26,24,68)
> rnames  <- c(«R1», «R2»)
> cnames  <- c(«C1», «C2»)
```

← ❶ Создаем таблицу 5×4

← ❷ Таблица 2×2
заполнена по рядам

```
> mymatrix <- matrix(cells, nrow=2, ncol=2, byrow=TRUE,
                      dimnames=list(rnames, cnames))
> mymatrix
   C1 C2
R1  1 26
R2 24 68
> mymatrix <- matrix(cells, nrow=2, ncol=2, byrow=FALSE, ↗
                      dimnames=list(rnames, cnames))
> mymatrix
   C1 C2
R1  1 24
R2 26 68
```

**Таблица 2×2
заполнена
по столбцам**

Сначала вы создаете матрицу 5×4 ①. Затем вы делаете матрицу 2×2 с названиями строк и столбцов и заполняете ее по строкам ②. Наконец, вы создаете матрицу 2×2 и заполняете ее по столбцам ③.

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

Программный код 2.2. Использование индексов при работе с матрицами

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

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

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

2.2.3. Массивы данных

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

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

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

Программный код 2.3. Создание массива данных

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

Очевидно, что массивы данных – это просто расширенные матрицы. Они могут быть полезны при написании программ для реализации новых статистических методов. Как и в матрицах, все элементы массива данных должны иметь одинаковый тип. Система обозначе-

ний элементов здесь такая же, как для матриц. В приведенном выше примере элемент $z[1, 2, 3]$ – это 15.

2.2.4. Таблицы данных

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

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

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

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

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

Программный код 2.4. Создание таблицы данных

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

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

Существует несколько способов обозначить элементы таблицы данных. Можно использовать индексы, как мы делали это раньше (например, для матриц), или можно указывать номера столбцов.

Приведенный ниже программный код на примере созданной раньше таблицы данных patientdata демонстрирует оба способа.

Программный код 2.5. Обозначение элементов таблицы данных

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

← ① Обозначаем переменную `age` (возраст)
в таблице данных `patientdata`

Знак \$ из третьего примера раньше нам не встречался ①. Он используется, чтобы обозначить определенную переменную в таблице данных. Например, если вы захотите создать сводную таблицу типов диабета в зависимости от состояния больного, вы можете использовать следующий программный код:

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

	Excellent	Improved	Poor
Type1	1	0	2
Type2	0	1	0

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

Attach, detach и with

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

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

Это можно также записать в виде

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

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

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

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

Когда мы добавили к траектории поиска таблицу данных `mtcars`, в рабочем пространстве уже имелся объект с названием `mpg`. В подобных случаях преимущество получает объект, который был создан первым, а это не то, чего мы хотели. Команда `plot()` не выполняется, потому что `mpg` теперь состоит из трех элементов, а `disp` – из 32 элементов. Функции `attach()` и `detach()` лучше всего использовать, когда вы работаете с одной таблицей данных, и вероятность того, что у вас будет несколько объектов с одинаковыми именами, мала. В любом случае обращайте внимание на предупреждения о том, что объекты маскируются одноименными объектами (`objects are masked`)¹.

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

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

¹ То есть становятся недоступными для непосредственного вызова. – *Прим. пер.*

```
plot(mpg, disp)
plot(mpg, wt)
})
```

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

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

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

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

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

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

Названия строк

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

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

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

2.2.5. Факторы

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

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

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

К примеру, представьте, что у вас есть вектор
`diabetes <- c("Type1", "Type2", "Type1", "Type1")`.

Команда `diabetes <- factor(diabetes)` преобразует этот вектор в (1, 2, 1, 1) и устанавливает внутреннее соответствие 1=Type1 и 2=Type2 (присвоение числовых значений происходит в алфавитном порядке). Любой анализ, который вы будете проводить с вектором `diabetes`, будет воспринимать эту переменную как номинальную и

выбирать статистические методы, подходящие для этого типа данных.

При работе с векторами, которые представлены порядковыми данными, для функции `factor()` нужно добавлять параметр `ordered=TRUE`. Примененная к вектору

```
status <- c("Poor", "Improved", "Excellent", "Poor")
```

команда `status <- factor(status, ordered=TRUE)` преобразует этот вектор в вид $(3, 2, 1, 3)$ и установит внутреннее соответствие как $1=\text{Excellent}$, $2=\text{Improved}$, $3=\text{Poor}$. Во время любой обработки этого вектора он будет воспринят как порядковая переменная с применением соответствующих статистических методов.

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

Установку по умолчанию можно изменить при помощи параметра `levels`. Например,

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

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

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

Программный код 2.6. Использование факторов

```
> patientID <- c(1, 2, 3, 4)                                     ← ❶ Вводим данные  
> age <- c(25, 34, 28, 52)  
> diabetes <- c("Type1", "Type2", "Type1", "Type1")  
> status <- c("Poor", "Improved", "Excellent", "Poor")  
> diabetes <- factor(diabetes)  
> status <- factor(status, order=TRUE)  
> patientdata <- data.frame(patientID, age, diabetes, status)  
> str(patientdata)  
'data.frame': 4 obs. of 4 variables:                                     ← ❷ Смотрим  
  ..$ patientID: num 1 2 3 4  
  ..$ age       : num 25 34 28 52  
  ..$ diabetes  : Factor w/ 2 levels "Type1", "Type2": 1 2 1 2  
  ..$ status    : Factor w/ 4 levels "Poor", "Improved", "Excellent", "Poor": 3 4 2 1
```

```
$ patientID: num  1 2 3 4
$ age      : num  25 34 28 52
$ diabetes : Factor w/ 2 levels "Type1","Type2": 1 2 1 1
$ status   : Ord.factor w/ 3 levels "Excellent"<"Improved"<...: 3 2 1 3
> summary(patientdata)
  patientID       age      diabetes    status
Min.    :1.00     Min.   :25.00  Type1:3  Excellent:1
1st Qu.:1.75    1st Qu.:27.25  Type2:1  Improved :1
Median  :2.50    Median  :31.00          Poor    :2
Mean    :2.50    Mean    :34.75
3rd Qu.:3.25    3rd Qu.:38.50
Max.    :4.00    Max.   :52.00
```

3
Смотрим
сводную
статистику
для объекта

Сначала вы вводите данные как векторы ①. Затем вы указываете, что `diabetes` – это фактор, а `status` – это упорядоченный фактор. Наконец, вы объединяете данные в таблицу. Функция `str(object)` выводит информацию об объекте (в нашем случае это таблица данных) ②. Ясно видно, что `diabetes` – это фактор, а `status` – это упорядоченный фактор; также указано, как он закодирован внутри программы. Обратите внимание, что функция `summary()` обрабатывает переменные по-разному ③. Для непрерывной переменной `age` вычислены минимум (`minimum, Min.`), максимум (`maximum, Max.`), среднее (`Mean`) и квартили (`first and third quartiles: 1st Qu., 3rd Qu.`), а для категориальных переменных `diabetes` и `status` подсчитана частота встречаемости каждого значения.

2.2.6. Списки

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

```
mylist <- list(объект 1, объект 2, ...),
```

где объекты – это любые структуры данных, которые мы обсуждали до этого. Объектам в списке можно присваивать имена:

```
mylist <- list(name1=объект 1, name2=объект 2, ...).
```

Пример работы со списками приведен ниже.

Программный код 2.7. Создание списка

```
> g <- "My First List"
```

```

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

◀─ Выводим на экран весь список

◀─ Выводим на экран второй объект списка

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

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

Информация для программистов

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

- точка (.) в названиях объектов не имеет никакого специального значения. Однако знак доллара (\$) имеет примерно такое же значение, как точка в других языках программирования, обозначая часть объекта. Например, `A$x` обозначает переменную `x` в таблице данных `A`;

- в R нет возможности создавать многострочные или блочные комментарии. Каждую строку комментария нужно начинать со знака #. При отладке программных кодов можно заключать участок кода, который должен быть пропущен программным интерпретатором, внутрь конструкции `if (FALSE) { ... }`. Замена FALSE на TRUE сделает возможным выполнение кода;
- присвоение значений несуществующему элементу вектора, матрицы, массива данных или списка расширит существующий объект, чтобы вместить новое значение. Рассмотрим следующий пример:

```
> x <- c(8, 6, 4)
> x[7] <- 10
> x
[1]  8  6  4 NA NA NA 10
```

В результате присвоения размер вектора x увеличился с трех до семи элементов;

- в R нет скаляров. Они представлены в виде вектора, состоящего из одного элемента;
- нумерация в R начинается с 1, а не с 0. В приведенном выше векторе элемент `x[1]` – это 8.

Дальнейшую информацию можно получить в превосходном блоге Джона Кука (John Cook) «Программирование в R для тех, кто раньше работал с другими языками» (www.johndcook.com/R_language_for_programmers.html)².

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

2.3. Ввод данных

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

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

² Это и все последующие руководства, упомянутые в книге, написаны на английском языке. – Прим. пер.

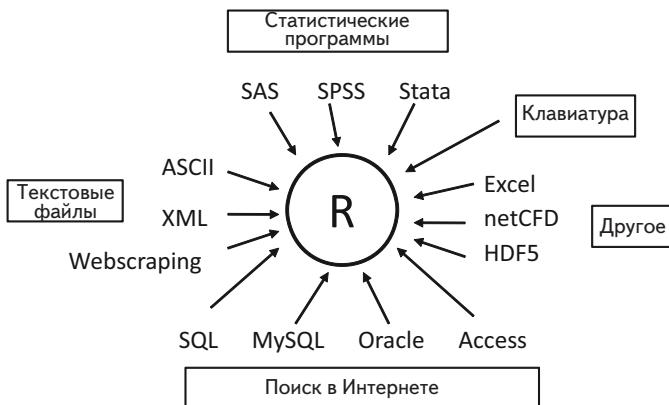


Рис. 2.2. Источники, из которых можно импортировать данные в R

2.3. 1. Ввод данных с клавиатуры

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

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

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

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

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

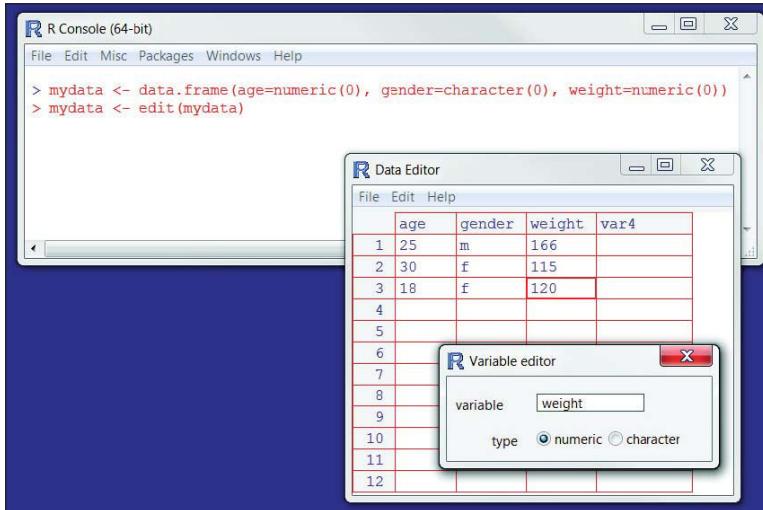


Рис. 2.3. Редактирование данных при помощи встроенного текстового редактора под Windows

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

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

2.3.2. Импорт данных из текстового файла с разделителями

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

```
mydataframe <- read.table(file, header=логическое_значение,  
sep="разделитель", row.names="название")
```

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

Например, программный код

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

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

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

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

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

Примечание. Многие примеры в этой главе основаны на импорте уже имеющихся на компьютере файлов. В R также реализовано несколько алгоритмов получения данных из Интернета. Например, вместо имени файла можно использовать функции *file()*, *gzfile()*, *bzfile()*, *xzfile()*, *unz()* и *url()*. Функция *file()* позволяет получать доступ к файлам, буферу обмена и

к стандартным потокам ввода-вывода. Функции `gzfile()`, `bzfile()`, `xzfile()` и `unz()` дают возможность доступа к сжатым файлам. Функция `url()` предоставляет доступ к файлам в Интернете через полный URL-адрес, который включает `http://`, `ftp://` или `file://`. Для HTTP или FTP можно назначить программы-посредники (*proxy*). Полные URL-адреса (заключенные в прямые кавычки) можно использовать также вместо названий файлов. Более подробную информацию можно получить, введя команду `help(file)`.

2.3.3. Импорт данных из Excel

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

Прежде всего скачайте и установите пакет RODBC.

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

Теперь вы можете использовать следующий программный код для импорта данных:

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

Здесь `myfile.xls` – это файл Excel, `mysheet` – это название нужного листа из рабочей книги Excel, `channel` – это вспомогательный объект RODBC, созданный функцией `odbcConnectExcel()`, и `mydataframe` – это получившаяся таблица данных. Этот пакет можно также использовать для импорта данных из Microsoft Access. Подробности изложены в файле справки: `help(RODBC)`.

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

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

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

2.3.4. Импорт данных из XML-файлов

В последнее время все больше данных появляется в виде файлов формата XML. В R есть несколько пакетов для работы с такими файлами. Например, пакет `xml`, созданный Дунканом Темплом Лангом (Duncan Temple Lang), позволяет пользователям читать, записывать и преобразовывать XML-файлы. Обсуждение этого формата выходит за рамки нашей книги. Заинтересованным в работе с XML-файлами пользователям R рекомендую обратиться к прекрасной справочной документации пакета на сайте www.omegahat.org/RXML.

2.3.5. Извлечение данных из веб-страниц

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

2.3.6. Импорт данных из SPSS

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

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

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

Затем используйте следующий программный код для импорта данных:

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

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

2.3.7. Импорт данных из SAS

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

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

Программа SAS:

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

Программа R:

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

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

2.3.8. Импорт данных из Stata

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

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

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

2.3.9. Импорт данных из netCDF

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

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

Рассмотрим следующий программный код:

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

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

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

2.3.10. Импорт данных из HDF5

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

2.3.11. Импорт данных из систем управления базами данных

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

Интерфейс ODBC

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

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

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

Таблица 2.2. Функции пакета RODBC

Функция	Описание
<code>odbcConnect (dsn, uid="", pwd "")</code>	Открывает соединение с базой данных ODBC
<code>sqlFetch (channel, sqltable)</code>	Сохраняет таблицу базы данных ODBC в виде таблицы данных
<code>sqlQuery (channel, query)</code>	Посыпает запрос в базу данных ODBC и выводит на экран результаты
<code>sqlSave (channel, mydf, tablename = sqtable, append=FALSE)</code>	Сохраняет или обновляет (append=TRUE) таблицу данных в виде таблицы в базе данных ODBC

Функция	Описание
<code>sqlDrop(channel, sqtable)</code>	Удаляет таблицу из базы данных ODBC
<code>close(channel)</code>	Закрывает соединение

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

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

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

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

Пакеты, связанные с DBI

Пакет DBI предоставляет широко применимый и последовательный пользовательский интерфейс для DBMS. Созданный на его основе пакет RJDBC предоставляет доступ к DBMS через драйвер JDBC. Убедитесь, что перед его использованием на вашем компьютере установлены драйверы JDBC для соответствующей операционной системы и базы данных. Другие полезные пакеты, созданные на базе DBI, – это RMySQL, ROracle, RPostgreSQL и RSQLite. Эти пакеты

предоставляют драйверы, предназначенные для конкретных баз данных, однако они могут работать не на всех операционных системах. Ознакомьтесь с документацией CRAN (<http://cran.r-project.org>) для того, чтобы узнать подробности.

2.3.12. Импорт данных при помощи Stat/Transfer

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

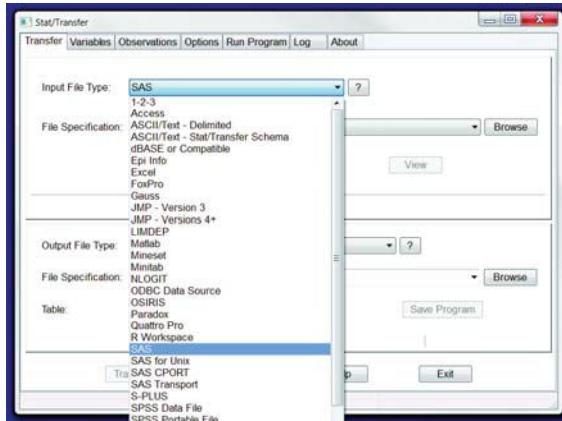


Рис. 2.4. Основное диалоговое окно программы Stat/Transfer под Windows

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

2.4. Аннотирование наборов данных

Исследователи обычно аннотируют наборы данных для облегчения интерпретации результатов. Как правило, аннотация заключается в

добавлении поясняющих подписей к названиям переменных и расшифровке кодировки, использованной для категориальных признаков. Например, для переменной `age` вы можете захотеть добавить более понятное название «*Age at hospitalization (in years)*» («Возраст в момент госпитализации (в годах)»). Для переменной `gender`, со значениями 1 и 2, у вас может появиться желание поставить в соответствие цифрам надписи «*male*» (мужской) и «*female*» (женский).

2.4.1. Подписи для переменных

К сожалению, возможности R по работе с подписями для переменных ограничены. Один подход заключается в том, чтобы использовать подпись для переменной в качестве ее названия, а обозначать эту переменную ее номером. Рассмотрим наш предыдущий пример с данными о пациентах. Во втором столбце с названием `age` был записан возраст пациентов, в котором они впервые попали в больницу. Программный код

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

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

2.4.2. Пояснение значений переменных

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

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

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

2.5. Полезные функции для работы с объектами

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

Таблица 2.3. Полезные функции для работы с объектами

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

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

2.6. Резюме

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

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

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

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



ГЛАВА 3.

Начало работы с диаграммами

В этой главе:

- Создаем и сохраняем диаграммы.
- Изменяем тип символов и линий, цвета и оси.
- Добавляем текст и названия.
- Задаем число измерений диаграммы.
- Объединяем несколько диаграмм в одну.

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

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

В этой главе мы обсудим основные методы работы с диаграммами. Мы начнем с создания и сохранения диаграмм. Затем мы узнаем, как изменять общие для всех типов диаграмм параметры. Сюда входят

названия диаграмм, оси, подписи, цвета, линии, символы и текстовые аннотации. Мы сосредоточимся на общих методах, которые можно применить ко всем типам диаграмм.

3.1. Работа с диаграммами

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

Рассмотрим следующие пять строк:

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

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

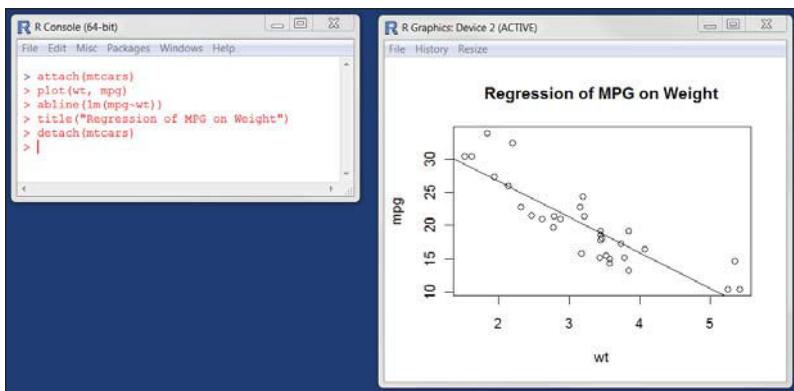


Рис. 3.1. Создание диаграммы

Диаграммы можно сохранять при помощи программного кода или меню графического пользовательского интерфейса. Для сохранения диаграммы при помощи кода разместите создающие диаграмму

команды между командами, которые назначают место вывода и закрывают вывод. Например, следующий программный код позволяет сохранить диаграмму в формате PDF под названием `mygraph.pdf` в текущей рабочей директории:

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

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

Способ сохранения диаграмм при помощи графического пользователяского интерфейса различается в зависимости от оперативной системы. Под Windows при активированном графическом устройстве нужно выбрать в меню **Файл**¹ → **Сохранить как**, а затем в появившемся диалоге выбрать нужный формат графического файла и директорию для сохранения. Под Mac нужно выбрать **File** → **Save as** в строке меню, когда активировано графическое устройство Quartz. В этом случае диаграмму можно сохранить в единственном формате – PDF. Под Unix диаграммы можно сохранять только при помощи кода. В приложении А мы рассмотрим другие графические интерфейсы для каждой операционной системы, которые имеют больше возможностей.

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

Во-первых, можно открыть новое графическое устройство, *перед* тем как создавать новую диаграмму:

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

Каждая новая диаграмма будет появляться в последнем открытом окне.

¹ В русифицированной версии R. – *Прим. пер.*

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

В-третьих и в-последних, можно использовать функции `dev.new()`, `dev.next()`, `dev.prev()`, `dev.set()` и `dev.off()` для одновременного открытия нескольких окон графики и выбора необходимой диаграммы. Эти команды работают под всеми операционными системами. Введите `help(dev.cur)`, чтобы узнать больше об этом подходе.

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

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

3.2. Простой пример

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

Таблица 3.1. Реакция пациента на два лекарства в пяти дозировках

Дозировка	Реакция на лекарство А	Реакция на лекарство В
20	16	15
30	20	18

Дозировка	Реакция на лекарство А	Реакция на лекарство В
40	27	25
45	40	31
60	60	40

Эти данные можно ввести при помощи следующего программного кода:

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

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

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

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

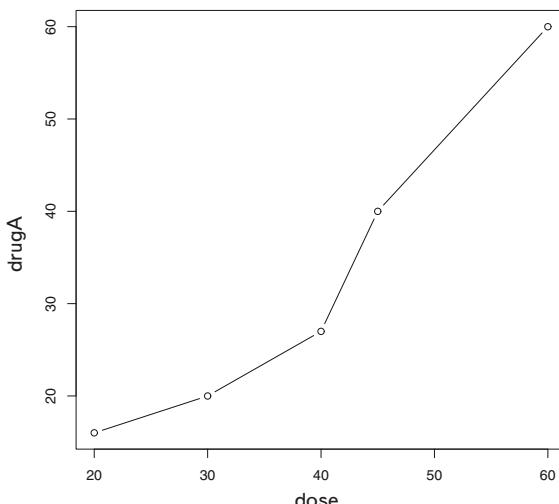


Рис. 3.2. График зависимости реакции пациента от дозы лекарства А

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

3.3. Графические параметры

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

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

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

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

Получившийся график показан на рис. 3.3.

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

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

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

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

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

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

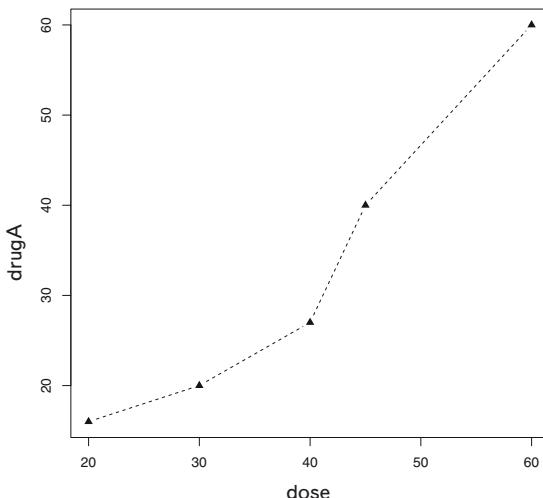


Рис. 3.3. График зависимости реакции пациента от дозы лекарства А с измененными типом линии и символами

3.3.1. Символы и линии

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

Таблица 3.2. Параметры для указания типов символов и линий

Параметр	Описание
<code>pch</code>	Определяет тип символа (см. рис. 3.4)
<code>cex</code>	Определяет размер символа. <code>cex</code> – это число, обозначающее, как символы должны быть масштабированы по отношению к размеру по умолчанию. 1 = размер по умолчанию, 1.5 – на 50% крупнее, 0.5 – на 50% мельче и т. д.

Параметр	Описание
lty	Определяет тип линии (см. рис. 3.5)
lwd	Определяет толщину линии по сравнению с толщиной линии по умолчанию (1). Например, lwd=2 делает линию в два раза толще, чем по умолчанию

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

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

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



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

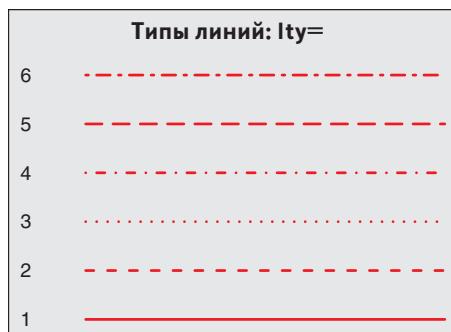


Рис. 3.5. Типы линий, назначаемые при помощи параметра `lty`

Программный код, объединяющий все эти параметры,

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

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

Теперь давайте посмотрим, как назначать цвета.

3.3.2. Цвета

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

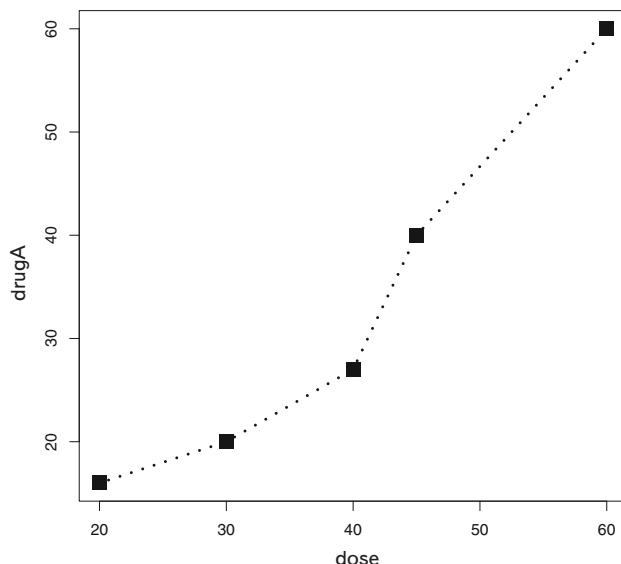


Рис. 3.6. График зависимости реакции пациента от дозы лекарства А с измененными типом и шириной линии, а также типом и размером символов

Таблица 3.3. Параметры для назначения цвета

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

В R цвета можно обозначать номером, названием, в шестнадцатеричной системе, а также в системах RGB или HSV. Например, `col=1`, `col="white"`, `col="#FFFFFF"`, `col=rgb(1,1,1)` и `col= hsv(0,0,1)` –

взаимозаменяемые способы обозначить белый цвет. Функция `rgb()` определяет цвета по значениям красного, зеленого и синего, а `hsv()` основана на значениях оттенка и насыщенности. Более полную информацию об этих функциях можно получить из файла справки.

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

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

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

3.3.3. Характеристики текста

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

Таблица 3.4. Параметры, определяющие размер шрифта

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

Например, на всех диаграммах, созданных после команды
`par(font.lab=3, cex.lab=1.5, font.main=4, cex.main=2)`

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

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

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

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

Например, после выполнения команды

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

вы сможете использовать А, В и С как названия семейств шрифтов. В этом случае `par(family="A")` назначит шрифт **Arial Black** (программный код 3.2 в разделе 3.4.2 содержит пример изменения парамет-

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

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

У русскоязычных пользователей могут возникнуть проблемы с отображением кириллицы на диаграммах R, сохраненных в векторных форматах (например, PDF). Особенно это актуально при работе под Windows. Для предотвращения подобных проблем нужно действовать по определенному алгоритму. Допустим, вы хотите создать упомянутую выше диаграмму `mygraph.pdf`, дав ей русское название.

1. Установите вспомогательную программу Ghostscript (используйте версию 8.61, а не самую последнюю, в которой выявлены проблемы):
<http://sourceforge.net/projects/ghostscript/files/GPL%20Ghostscript/8.61/gs861w32.exe/download>.
2. Пропишите путь к этой программе в системной переменной Path: открываем папку «Мой компьютер», нажимаем правую кнопку мыши и выбираем «свойства» в выпадающем меню, далее выбираем вкладки «дополнительно», затем «переменные среды», далее в поле «Path» записываем что-то вроде `%SystemRoot%\system32;%SystemRoot%;%SystemRoot%\System32\Wbem;C:\PROGRA~1\gs\gs8.61\bin;C:\PROGRA~1\gs\gs8.61\lib`.
3. После этого, открывая графическое устройство, нужно указать не только название будущего графического файла и шрифт, но и кодировку: `pdf("mygraph.pdf", family="NimbusSan", encoding="CP1251.enc")`.
4. Затем делаем все как в упомянутом примере, только используем русское название:

```
attach(mtcars)
plot(wt, mpg)
abline(lm(mpg~wt))
title("Регрессия расхода топлива в зависимости от веса автомобиля")
detach(mtcars)
dev.off()
```

5. Наконец, после закрытия графического устройства нужно встроить шрифты в полученный файл: `embedFonts("mygraph.pdf")`. – Прим. пер.

3.3.4. Размеры диаграммы и полей

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

Таблица 3.6. Параметры для определения размеров диаграммы и полей

Параметр	Описание
pin	Размер диаграммы (ширина, высота) в дюймах
mai	Числовой вектор, задающий размеры полей, где параметры с (низ, лево, верх, право) измеряются в дюймах
mag	Числовой вектор, задающий размеры полей, где параметры с (низ, лево, верх, право) измеряются в числе строк. По умолчанию это с (5, 4, 4, 2) + 0.1

1 дюйм = 2.5 см. – Прим. пер.

Команда

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

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

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

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

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

Сначала вы вводите данные в виде векторов, затем сохраняете текущие графические параметры (чтобы можно было восстановить их позднее). Вы изменяете графические параметры так, чтобы графики были 2 дюйма в ширину и 3 дюйма в высоту. Кроме того, линии будут в два раза шире, а символы – в 1.5 раза крупнее, чем по умолчанию. Значения на осях даны курсивом, их размер составляет 75% от размера по умолчанию. Затем создан первый график с заполненными красными кружками и пунктирными линиями. Второй график содержит зеленые ромбы с синей каймой и синие линии². Наконец, вы восстанавливаиваете исходные графические параметры.

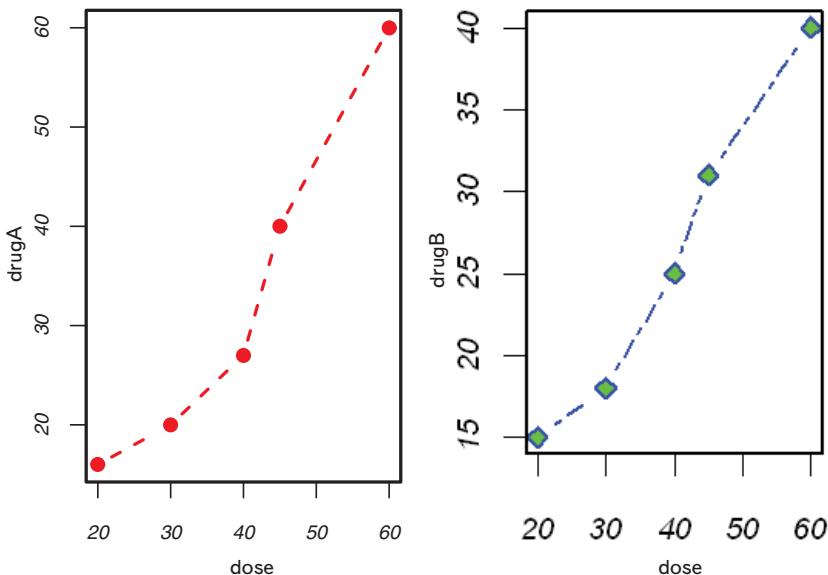


Рис. 3.7. Линейный график зависимости реакции пациента от дозы лекарств А и В

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

2 Цветная версия переведенной книги доступна только в электронном виде. – Прим. ред.

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

3.4. Добавление текста, настройка параметров осей и условных обозначений

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

```
plot(dose, drugA, type="b",
      col="red", lty=2, pch=2, lwd=2,
      main="Клинические испытания препарата А",
      sub="Это вымышленные данные",
      xlab="Доза", ylab="Эффект от препарата",
      xlim=c(0, 60), ylim=c(0, 70))
```

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

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

3.4.1. Заголовки

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

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

Графические параметры (такие как размер и тип шрифта, ориентация и цвет текста) также можно задать при помощи функции

`title()`. К примеру, следующий программный код позволяет получить диаграмму с красным заголовком, синим подзаголовком и зелеными подписями по осям, размер которых на 25% меньше, чем по умолчанию:

```
title(main="Мой заголовок", col.main="red",
      sub="Мой подзаголовок", col.sub="blue",
      xlab="Моя подпись оси X", ylab="Моя подпись оси Y",
      col.lab="green", cex.lab=0.75)
```

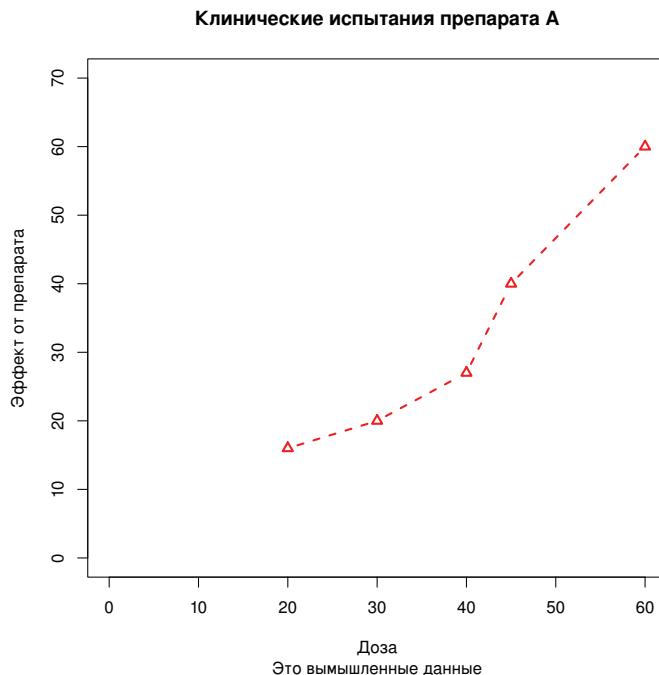


Рис. 3.8. Линейный график зависимости реакции пациента от дозы лекарства А с заголовком, подзаголовком и модифицированными осями

3.4.2. Оси

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

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

Таблица 3.7. Параметры осей

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

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

Программный код 3.2. Пример настройки параметров осей

```
axis(2, at=x, labels=x, col.axis="red", las=2) ← Рисуем оси
axis(4, at=z, labels=round(z, digits=2),
     col.axis="blue", las=2, cex.axis=0.7, tck=-.01)
mtext("y=1/x", side=4, line=3, cex.lab=1, las=2, col="blue") ←
title("Пример креативных осей",
      xlab="Значения переменной X",
      ylab="Y=X")
par(opar)
```

**Добавляем
подписи
и текст**

Пример креативных осей

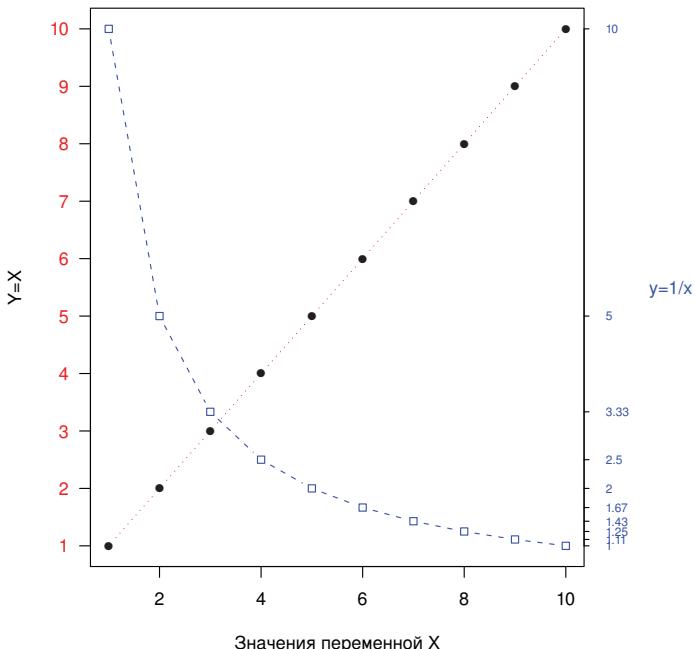


Рис. 3.9. Пример настройки параметров осей

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

Вспомогательные деления на осях

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

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

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

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

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

3.4.3. Опорные линии

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

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

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

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

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

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

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



3.4.4. Легенда

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

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

Распространенные аргументы перечислены в табл. 3.8.

Таблица 3.8. Аргументы функции `legend()`

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

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

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

Программный код 3.3. Сопоставление реакции пациента на разные дозы лекарств А и В

```
dose <- c(20, 30, 40, 45, 60)
drugA <- c(16, 20, 27, 40, 60)
drugB <- c(15, 18, 25, 31, 40)
opar <- par(no.readonly=TRUE)
par(lwd=2, cex=1.5, font.lab=2)           ← Увеличиваем ширину
plot(dose, drugA, type="b",               линии, размер
      pch=15, lty=1, col="red", ylim=c(0, 60), ← символов и подписей
      main="Сравнение препаратов А и В",
      xlab="Дозировка препарата", ylab="Эффект от препарата")
lines(dose, drugB, type="b",
      pch=17, lty=2, col="blue")             ← Создаем график
abline(h=c(30), lwd=1.5, lty=2, col="gray") ← Добавляем
library(Hmisc)                           промежуточные
minor.tick(nx=3, ny=3, tick.ratio=0.5)   деления на осях
legend("topleft", inset=.05, title="Тип препарата", c("А", "В"),
       lty=c(1, 2), pch=c(15, 17), col=c("red", "blue")) ← Добавляем
par(opar)                                условные
                                            обозначения
```

Сравнение препаратов А и В

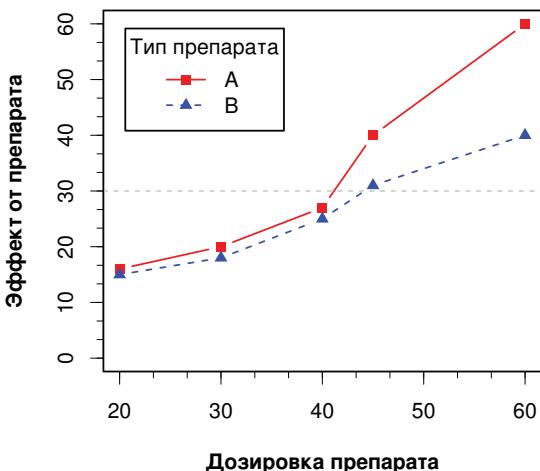


Рис. 3.10. Сопоставление препаратов А и В, снаженное условными обозначениями

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

3.4.5. Аннотации

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

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

Часто используемые аргументы перечислены в табл. 3.9.

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

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

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

Функция `text()` обычно используется для добавления подписей точек на диаграмме и других надписей. Необходимо указать положение надписей в виде набора x, y координат и сами надписи в виде текстового вектора. Векторы x, y координат и надписей должны иметь одинаковую длину. Ниже приведен пример; полученная диаграмма представлена на рис. 3.11.

```
attach(mtcars)
plot(wt, mpg,
      main="Зависимость расхода топлива от веса автомобиля",
      xlab="Вес", ylab="Расход топлива",
      pch=18, col="blue")
text(wt, mpg,
      row.names(mtcars),
      cex=0.6, pos=4, col="red")
detach(mtcars)
```

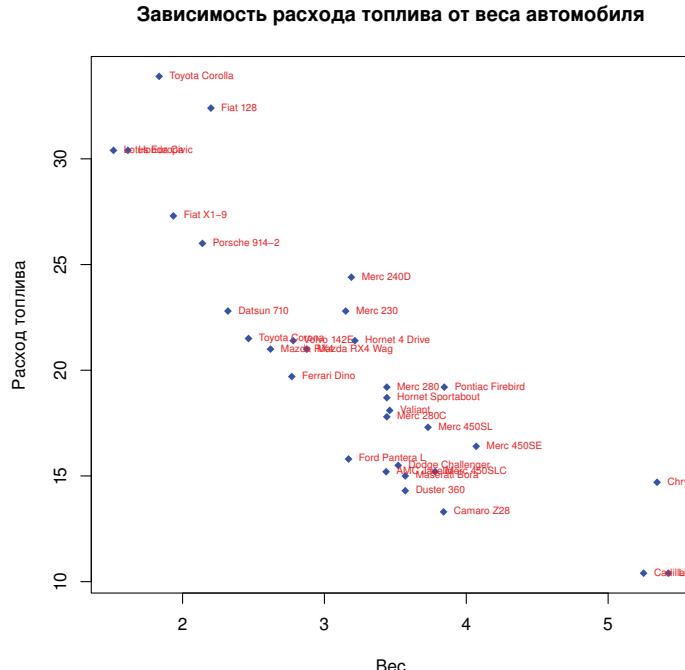


Рис. 3.11. Пример диаграммы рассеяния (зависимость расхода топлива от веса машины) с подписанными точками (марка машины)

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

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



```
opar <- par(no.readonly=TRUE)
par(cex=.9)
plot(1:7,1:7,type="n")
text(3,3,"Пример шрифта, используемого по умолчанию")
text(4,4,family="mono","Пример моноширинного шрифта")
text(5,5,family="serif","Пример шрифта с засечками")
par(opar)
```

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

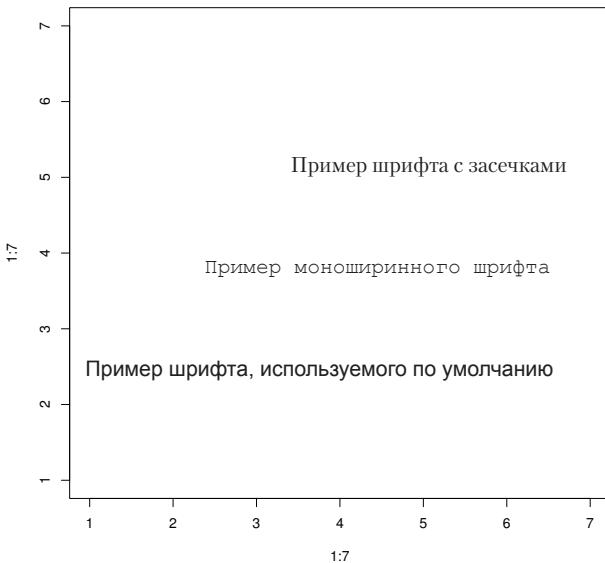


Рис. 3.12. Примеры семейств шрифтов
в операционной системе Windows

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

Подписи с математическими символами

Наконец, вы можете размещать на диаграммах математические символы и формулы, используя правила, сходные с теми, что применяются в TeX. Подробная информация и примеры даны в `help(plotmath)`. Вы можете также попробовать ввести `demo(plotmath)`, чтобы уви-

деть, как это работает. Часть этой демонстрации возможностей функции представлена на рис. 3.13. Функция может быть использована для добавления математических символов в названия, подписи осей или надписи внутри диаграммы или на ее полях.

Арифметические операторы		Корни	
$x + y$	$x + y$	$\text{sqrt}(x)$	\sqrt{x}
$x - y$	$x - y$	$\text{sqrt}(x, y)$	$\sqrt[y]{x}$
Соотношения			
x/y	x/y	$x == y$	$x = y$
$x \%+-\% y$	$x \pm y$	$x != y$	$x \uparrow y$
$x\%/\%y$	$x \sqrt{y}$	$x < y$	$x < y$
$x \%*\% y$	$x \times y$	$x <= y$	$x " y$
$x \%. \% y$	$x \cdot y$	$x > y$	$x > y$
$-x$	$-x$	$x >= y$	$x \geq y$
$+x$	$+x$	$x \%{\sim}\% y$	$x \oplus y$
Под/надстрочные индексы		$x \%{=}\% y$	$x \equiv y$
$x[i]$	x_i	$x \%{==}\% y$	$x \equiv y$
x^2	x^2	$x \%{\text{prop}}\% y$	$x \propto y$
Размещение рядом		Гарнитура шрифта	
$x * y$	$x y$	$\text{plain}(x)$	x
$\text{paste}(x, y, z)$	$x y z$	$\text{italic}(x)$	x
Перечни		$\text{bold}(x)$	x
$\text{list}(x, y, z)$	x, y, z	$\text{bolditalic}(x)$	x
		$\text{underline}(x)$	x

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

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

3.5. Объединение диаграмм

В R очень легко объединить несколько диаграмм в одну общую, используя функции `par()` или `layout()`. На этом этапе не нужно об-

рашать внимания на типы объединяемых диаграмм, сейчас важно понять общие способы, которые используются для их объединения. Создание и интерпретация каждого типа диаграмм подробно рассмотрены в следующих главах.

Вы можете добавить графический параметр `mfrow=c(nrows, ncols)` в функцию `par()` для создания матрицы из диаграмм размером $nrows \times ncols$, которая будет заполнена по рядам. Для заполнения этой матрицы по столбцам нужно использовать параметр `mfcoll=c(nrows, ncols)`.

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

```
attach(mtcars)
opar <- par(no.readonly=TRUE)
par(mfrow=c(2,2))
plot(wt,mpg, main="Диаграмма рассеяния для \nрасхода топлива и
`веса машины")
plot(wt,disp, main="Диаграмма рассеяния для объема \nдвигателя и
`веса машины")
hist(wt, main="Распределение значений \nвеса машин")
boxplot(wt, main="Ящик с усами для веса машин")
par(opar)
detach(mtcars)
```

Результат приведен на рис. 3.14³.

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

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

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

³ Символ `\n` позволяет разбивать длинные русские названия диаграмм на две строки, чтобы они уместились на диаграмме. – Прим. пер.

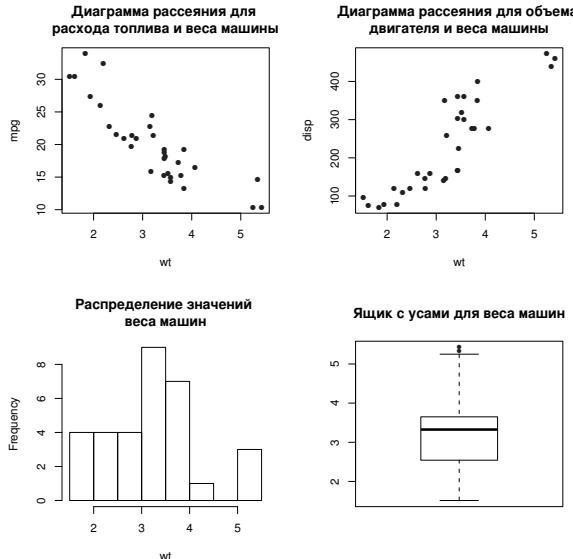


Рис. 3.14. Четыре диаграммы, объединенные в одну, при помощи команды `par(mfrow=c(2, 2))`

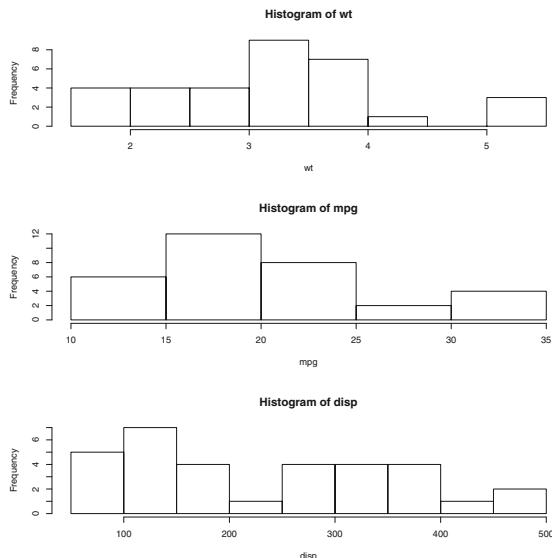


Рис. 3.15. Диаграмма, на которой три гистограммы объединены при помощи команды `par(mfrow=c(3, 1))`

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

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

Получившаяся диаграмма представлена на рис. 3.16.

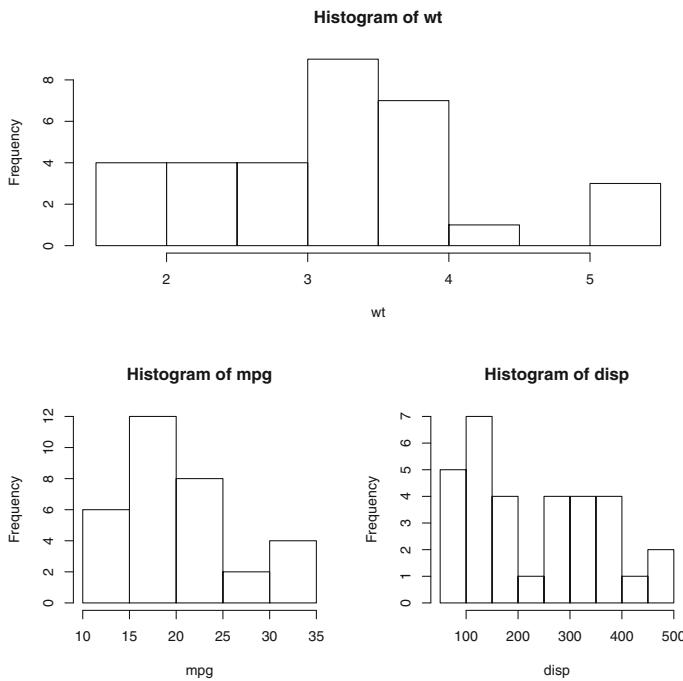


Рис. 3.16. Три диаграммы, объединенные в одну, при помощи функции `layout()` без указания ширины диаграмм

Для указания размера каждой диаграммы можно использовать дополнительные аргументы функции `layout()`: `widths=` и `heights=`. Эти аргументы нужно применять в виде

`widths=` вектор значений ширины каждого столбца;
`heights=` вектор значений высоты каждой строчки.

Относительные размеры назначаются при помощи чисел. Абсолютные размеры (в сантиметрах) назначаются при помощи функции `lcm()`.

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

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

Полученная диаграмма представлена на рис. 3.17.

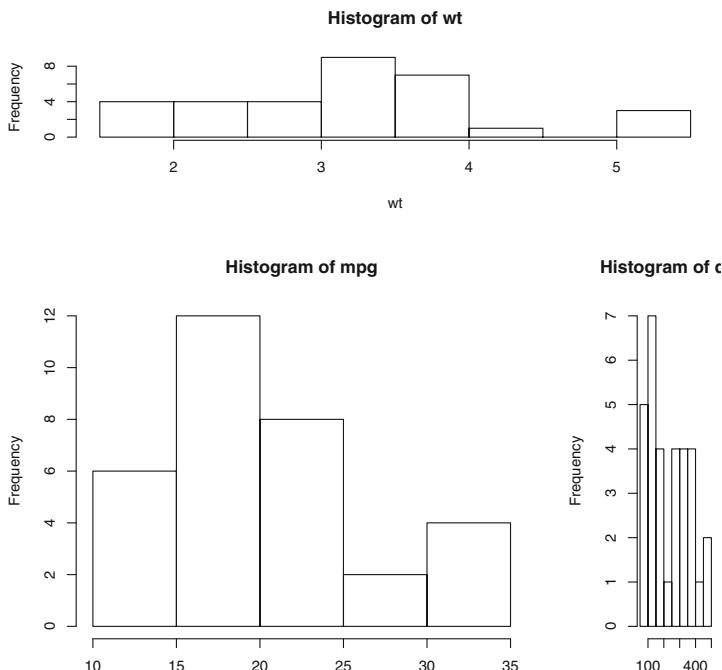


Рис. 3.17. Три диаграммы, объединенные в одну, при помощи функции `layout()` с указанием ширины диаграмм



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

3.5.1. Полный контроль над расположением диаграмм

Иногда для получения одной нужной диаграммы необходимо совместить и наложить несколько диаграмм. Для этого требуется тщательно контролировать их расположение. Этого можно достичь при помощи графического параметра `fig=`. В приведенном ниже программном коде две диаграммы размахов добавлены к диаграмме рассеяния, в результате чего получилась одна усовершенствованная диаграмма (она показана на рис. 3.18).

Усовершенствованная диаграмма рассеяния

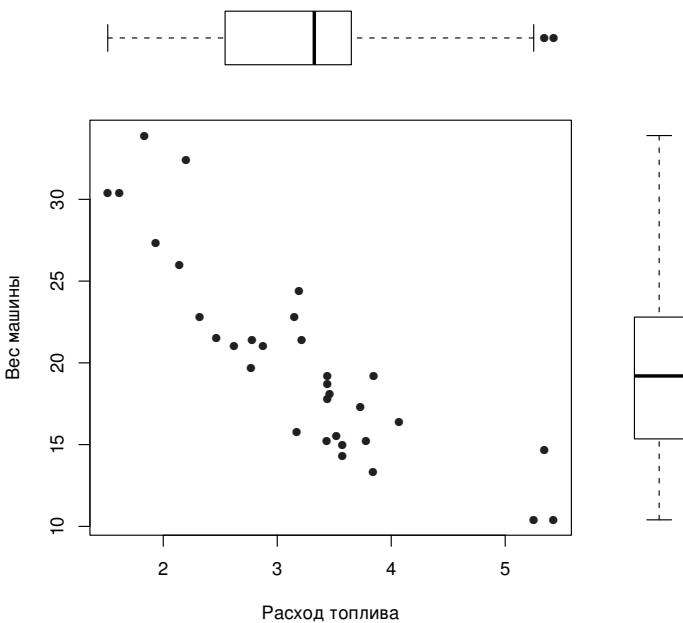


Рис. 3.18. Диаграмма рассеяния с двумя диаграммами размахов на полях

Программный код 3.4. Полный контроль над расположением диаграмм

```

opar <- par(no.readonly=TRUE)
par(fig=c(0, 0.8, 0, 0.8))
plot(mtcars$wt, mtcars$mpg,
      xlab="Расход топлива",
      ylab="Вес машины")
par(fig=c(0, 0.8, 0.55, 1), new=TRUE)    ↘ Построение диаграммы
boxplot(mtcars$wt, horizontal=TRUE, axes=FALSE)   ↘ Добавление «ящика
par(fig=c(0.65, 1, 0, 0.8), new=TRUE)    ↘ с усами» сверху
boxplot(mtcars$mpg, axes=FALSE)           ↘ Добавление «ящика
mtext("Усовершенствованная диаграмма рассеяния", side=3,
      outer=TRUE, line=-3)
par(opar)

```

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

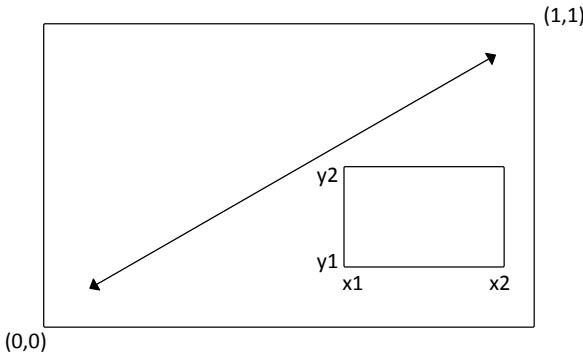


Рис. 3.19. Определение положения диаграммы
при помощи графического параметра `fig=`

Первое применение параметра `fig=` позволяет разместить диаграмму рассеяния на пространстве от 0 до 0.8 по оси x и от 0 до 0.8 по оси y . Верхняя диаграмма размахов занимает пространство от 0 до 0.8 по оси x и от 0.55 до 1 по оси y . Диаграмма размахов справа занимает пространство от 0.65 до 1 по оси x и от 0 до 0.8 по оси y . Параметр `fig=` строит новую диаграмму, поэтому при добавлении диаграммы к уже имеющейся используйте параметр `new=TRUE`.

Я выбрал значение 0.55, а не 0.8, чтобы разместить верхнюю диаграмму размахов поближе к диаграмме рассеяния. По этой же причине я выбрал значение 0.65 для правой диаграммы размахов. Приходится



пробовать разные варианты, чтобы добиться правильного взаимного расположения диаграмм.

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

3.9. Резюме

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

В этой главе мы сконцентрировались на общих подходах, которые можно применить к любым диаграммам (кроме панельных диаграмм, обсуждаемых в главе 16). В следующих главах мы рассмотрим разные типы диаграмм. Например, в главе 7 рассмотрены способы графического отображения одной переменной. Изображение взаимосвязей между переменными будет описано в главе 11. В главе 16 мы поговорим о панельной, или категоризированной, графике (диаграммах, которые позволяют отобразить взаимосвязи между переменными для каждого значения других переменных) и интерактивной графике. Интерактивные диаграммы позволяют при помощи мышки исследовать изображенные взаимосвязи в реальном времени.

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

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



ГЛАВА 4.

Основы управления данными

В этой главе:

- Работаем с датами и пропущенными значениями.
- Учимся преобразовывать один тип данных в другой.
- Создаем и перекодируем переменные.
- Сортируем, объединяем и разделяем наборы данных.
- Выбираем и исключаем переменные из анализа.

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 нему. 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 должностях по степени лояльности к вышестоящему начальству?

- зависит ли это от страны, или выявленные гендерные (половые) различия носят универсальный характер?

Один из способов ответить на эти вопросы – взять начальников из разных стран и ранжировать подчиненных им 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

Manager (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-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 ответа 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. 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`

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

Таблица 4.2. Арифметические операторы

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

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, исходя из значений 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).

Таблица 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 проходить 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"
```

Название таблицы данных входит в `leadership$agecat`, чтобы новые переменные сохранялись именно 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; особенно просто перекодировать числовые и текстовые векторы и факторы при помощи функции `recode()` из пакета `car`. В пакете `dplyr` 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 изменить 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).

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

```
rename(таблица_данных, с(старое_название="новое_название",
  ↪ старое_название="новое_название", ...))
```

Вот пример:

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

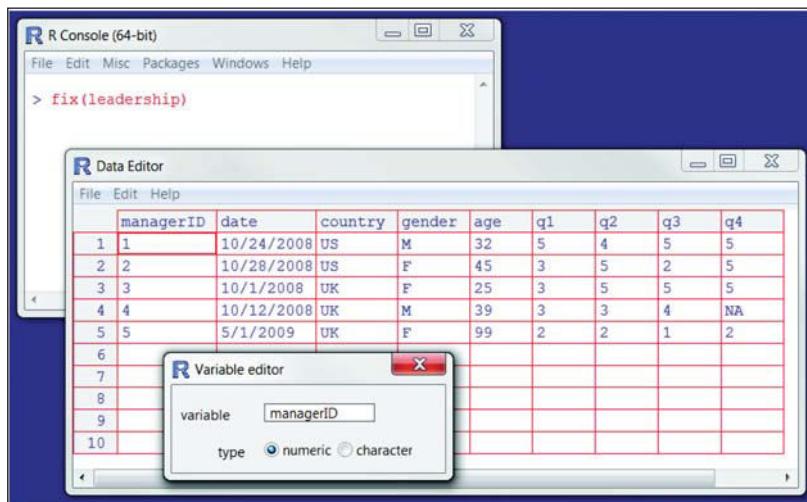


Рис. 4.1. Интерактивное переименование переменных с использованием функции `fix()`

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 с большой вероятностью будут неполными из-за пропущенных вопросов, барахлящего оборудования или ошибок, допущенных при вводе 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

```
> leadership                                ◀— Таблица данных с пропущенными данными
   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 данных, состоящая
> newdata                                     только из полных строк
   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 календарных датах.

4.6. Календарные даты как данные

Даты обычно вводятся в R в виде текстовых строк, а затем переводятся в 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).

Таблица 4.4. Форматы данных

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

¹ Понедельник – Прим. пер.

² Январь – Прим. пер.

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 по 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 календарные 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 временных отмечок 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"
```

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 бы мог подумать? Вопрос на засыпку: в какой день недели я родился?

4.6.1. Преобразование дат в текстовые переменные

Можно перекодировать даты и в текстовые значения, хотя это бывает нужно не так часто. 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 датами. 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 происходят 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.

Таблица 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 убывания¹. Приведенный ниже пример иллюстрирует сортировку на примере данных по лояльности к начальству.

Команда

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

создает новый набор данных, в котором строки отсортированы, начиная с самого молодого менеджера и заканчивая 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.

1 Такой прием не всегда работает. Правильнее использовать аргумент `decreasing=T`. Например, так можно отсортировать менеджеров в порядке убывания возраста: `leadership[order(leadership$age, decreasing=T),].` – Прим. пер.

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 слияния двух таблиц 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")
```

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

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

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

Примечание. Если вы хотите просто 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 объединения двух таблиц данных в вертикальном направлении используйте функцию `rbind()`:

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

Объединяемые таблицы должны содержать одинаковые переменные, однако им не обязательно быть расположеными 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 имеются обширные возможности для извлечения отдельных 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 также 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 подлежащих извлечению переменных.

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 до начала анализа данных. 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, TRUE, TRUE, FALSE)`.
3. Оператор «не» (`!`) изменяет логические значения на противоположные: `c(TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, FALSE, FALSE, TRUE)`.
4. `leadership[c(TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, FALSE, FALSE, TRUE)]` выбирает столбцы, для которых значе-

ния 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 удален².

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)3
```

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.

- 2 В данном случае для удаления столбцов проще использовать такой синтаксис: `leadership[-c(8, 9)].` – Прим. пер.
- 3 В приведенном примере команда `which()` не нужна. Попробуйте убедиться в том, что выбор строк произойдет и без нее. – Прим. пер.

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 отдельно. 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 `q4` (`gender`, `q4` 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 больших 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 данных⁴.

⁴ Существует и более специализированная функция `sample.int()`. – Прим. пер.

Дополнительная информация

В R реализованы обширные возможности создания выборок, включая получение и проверку пробных выборок (пакет *sampling*) и анализ сложных 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, введя `install.packages("sqldf")`, 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 считут пакет `sqldf` полезным приложением для управления данными в R. Посетите домашнюю интернет-страницу проекта (<http://code.google.com/p/sqldf/>), для того чтобы узнать о нем больше.

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 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, кто анализирует данные!)



ГЛАВА 5.

Более сложные способы управления данными

В этой главе:

- Математические и статистические функции.
- Текстовые функции.
- Циклы и выполнение команд при условии.
- Пользовательские функции.
- Способы объединять и преобразовывать данные.

В главе 4 мы рассмотрели основные способы управления данными в R. В этой главе мы сосредоточимся на более сложных подходах. Глава состоит из трех основных разделов. 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 данной 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 выполнение команд при условии. Потом вы познакомитесь со структурой пользовательских функций и узнаете, как их применять.

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 А¹ первым по успеваемости 20% студентов, оценку В² – следующим по успеваемости 20% и т. д. 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

1 Отлично» в американской системе оценивания. – Прим. пер.

2 «Хорошо» в американской системе оценивания. – Прим. пер.

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 подходящее решение для нашей задачи по управлению данными.

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, распределения) 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 табл. 5.2 перечислены наиболее распространенные математические функции вместе с короткими примерами.

Таблица 5.2. Математические функции

Функция	Описание
<code>abs(x)</code>	Модуль <code>abs(-4)</code> равно 4
<code>sqrt(x)</code>	Квадратный корень <code>sqrt(25)</code> равно 5 Это то же, что и $25^{(0.5)}$
<code>ceiling(x)</code>	Наименьшее целочисленное значение, не меньшее, чем x <code>ceiling(3.457)</code> равно 4
<code>floor(x)</code>	Наибольшее целочисленное значение, не большее, чем x <code>floor(3.457)</code> равно 3
<code>trunk(x)</code>	Целое число, полученное при округлении x в сторону нуля <code>trunk(5.99)</code> равно 5
<code>round(x, digits=n)</code>	Округляет x до заданного числа знаков после запятой <code>round(3.475, digits=2)</code> равно 3.48

Функция	Описание
<code>signif(x, digits=n)</code>	Округляет x до заданного числа значащих цифр <code>signif(3.475, digits=2)</code> равно 3.5
<code>cos(x), sin(x), tan(x)</code>	Косинус, синус и тангенс <code>cos(2)</code> равно -0.416
<code>acos(x), asin(x), atan(x)</code>	Аркосинус, арксинус и арктангенс <code>acos(-0.416)</code> равно 2
<code>cosh(x), sinh(x), tanh(x)</code>	Гиперболические косинус, синус и тангенс <code>sinh(2)</code> равно 3.627
<code>acosh(x), asinh(x), atanh(x)</code>	Гиперболические аркосинус, арксинус и арктангенс <code>asinh(3.627)</code> равно 2
<code>log(x, base=n)</code> <code>log(x)</code> <code>log10(x)</code>	Логарифм x по основанию n Для удобства: <code>log(x)</code> – натуральный логарифм <code>log10(x)</code> – десятичный логарифм <code>log(10)</code> равно 2.3026 <code>log10(10)</code> равно 1
<code>exp(x)</code>	Экспоненциальная функция <code>exp(2.3026)</code> равно 10

В основном эти функции применяются для преобразования данных. К примеру, данные с положительно асимметричным 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 табл. 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% наибольших и 5% наименьших значений в выборке, не принимая при этом во внимание пропущенные значения. Используйте команду `help()`, чтобы узнать больше о каждой функции и ее аргументах.

Таблица 5.3. Статистические функции

Функция	Описание
<code>mean(x)</code>	Среднее арифметическое <code>mean(c(1, 2, 3, 4))</code> равно 2.5
<code>median(x)</code>	Медиана <code>median(c(1, 2, 3, 4))</code> равно 2.5
<code>sd(x)</code>	Стандартное отклонение <code>sd(c(1, 2, 3, 4))</code> равно 1.29
<code>var(x)</code>	Дисперсия <code>var(c(1, 2, 3, 4))</code> равно 1.67
<code>mad(x)</code>	Абсолютное отклонение медианы <code>mad(c(1, 2, 3, 4))</code> равно 1.48
<code>quantile(x, probs)</code>	Квантили, где <code>x</code> – числовой вектор, для которого нужно вычислить квантили, а <code>probs</code> – числовой вектор с указанием вероятностей в диапазоне [0; 1] <code># 30-й и 84-й процентили x</code> <code>y <- quantile(x, c(.3, .84))</code>
<code>range(x)</code>	Размах значений <code>x <- c(1, 2, 3, 4)</code> <code>range(x)</code> равно <code>c(1, 4)</code> . <code>diff(range(x))</code> равно 3
<code>sum(x)</code>	Сумма <code>sum(c(1, 2, 3, 4))</code> равно 10
<code>diff(x, lag=n)</code>	Разность значений в выборке, взятых с заданным интервалом (<code>lag</code>). По умолчанию интервал равен 1. <code>x <- c(1, 5, 23, 29)</code> <code>diff(x)</code> равно <code>c(4, 18, 6)</code>
<code>min(x)</code>	Минимум <code>min(c(1, 2, 3, 4))</code> равно 1
<code>max(x)</code>	Максимум <code>max(c(1, 2, 3, 4))</code> равно 4
<code>scale(x, center=TRUE, scale=TRUE)</code>	Значения объекта <code>x</code> , центрованные (<code>center=TRUE</code>) или стандартизованные (<code>center=TRUE, scale=TRUE</code>) по столбцам. Пример дан в программном коде 5.6

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)
> mean(x)                                ← Быстрый способ
[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 сумма квадратов отклонений (corrected sum of squares – `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 (более детально мы рассмотрим решение задач матричной алгебры в приложении Е).

Стандартизация данных

По умолчанию функция `scale()` стандартизирует заданный столбец матрицы или таблицы данных так, чтобы его среднее арифметическое было равно нулю, а стандартное отклонение – единице³:

3 В русскоязычной литературе часто вместо термина «стандартизация» используется термин «нормализация». – Прим. пер.

```
newdata <- scale(mydata)
```

Для преобразования каждого столбца 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 среднего арифметического, а SD – стандартного отклонения.

Применение функции `scale()` к столбцам с нечисловыми данными вызывает сообщение об ошибке. 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. Функции распределения

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

В программе R функции распределения имеют вид

```
[dqr]сокращенное_название_распределения()
```

где первые буквы означают параметры распределения данных:

`d` = плотность;

`p` = функция распределения;

`q` = функция, определяющая квантили;

`r` = генератор случайных отклонений.

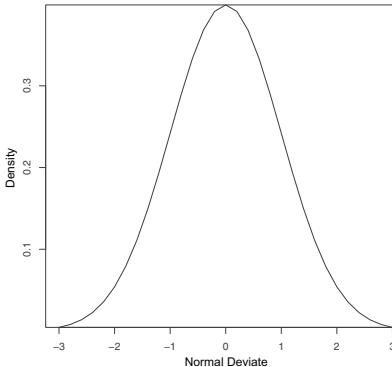
Наиболее распространенные функции распределения перечислены в табл. 5.4.

Таблица 5.4. Типы распределений

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

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)?	<pre>x <- pretty(c(-3, 3), 30) y <- dnorm(x) plot(x, y, type = "l", xlab = "Normal Deviate", ylab = "Density", yaxs = "i")</pre> 
Какова площадь под кривой стандартного нормального распределения слева от $z=1.96$?	<code>pnorm(1.96)</code> равно 0.975
Каково значение 90-го процентиля нормального распределения со средним значением 500 и стандартным отклонением 100?	<code>qnorm(.9, mean=500, sd=100)</code> равно 628.16
Как создать 50 случайных чисел, принадлежащих нормальному распределению со средним значением 50 и стандартным отклонением 10?	<code>rnorm(50, mean=50, sd=10)</code>

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 стартовое 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 чи-

сел, 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 ковариационной матрицей. Функция `mvtnorm()` из пакета MASS позволяет легко справиться с этой задачей. Общий вид применения функции таков:

```
mvtnorm(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 многомерномуциальному 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)          ← ① Определяем случайное
                           начальное число

> mean <- c(230.7, 146.7, 3.6)      ← ② Назначаем вектор
> 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)   ← ③ Генерируем данные
> mydata <- as.data.frame(mydata)
> names(mydata) <- c("y", "x1", "x2")

> dim(mydata)                         ← ④ 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 ①. Вы задаете вектор средних значений и ковариационную матрицу ② и генерируете 500 псевдослучайных чисел ③. 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 переменные, и выводите на экран первые 10 наблюдений ④. Учтите, что, поскольку корреляционная матрица также представляет собой ковариационную матрицу, вы не можете напрямую задать корреляционную структуру.

С помощью функций распределения в R вы можете генерировать синтетические (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 изменяют 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
<code>nchar(x)</code>	Подсчитывает число элементов <code>x</code> <code>x <- c("ab", "cde", "fghij")</code> <code>length(x)</code> равно 3 (см. табл. 5.7). <code>nchar(x[3])</code> равно 5
<code>substr(x, start, stop)</code>	Отделяет или заменяет часть текстового вектора. <code>x <- "abcdef"</code> <code>substr(x, 2, 4)</code> равно "bcd". <code>substr(x, 2, 4) <- "22222"</code> <code>(x</code> теперь представляет собой "a222ef")
<code>grep(pattern, x, ignore.case=FALSE, fixed=FALSE)</code>	Ищет определенные элементы в <code>x</code> . Если <code>fixed=FALSE</code> , то элемент представляет собой регулярное выражение. Если <code>fixed=TRUE</code> , тогда элемент представляет собой текстовую строку. Команда возвращает номера подходящих под условие элементов. <code>grep("A", c("b", "A", "c"), fixed=TRUE)</code> равно 2
<code>sub(pattern, replacement, x, ignore.case=FALSE, fixed=FALSE)</code>	Находит определенный элемент в <code>x</code> и заменяет его заданным текстом. Если <code>fixed=FALSE</code> , то элемент – это регулярное выражение. Если <code>fixed=TRUE</code> , то элемент – это текстовая строка. <code>sub("\s",".", "Hello There")</code> возвращает надпись Hello.There. Обратите внимание на то, что "\s" – это регулярное выражение для поиска пробелов; используйте вместо него \\s, поскольку "\" в R – это знак экранирования символов (см. раздел 1.3.3)

Функция	Описание
<code>strsplit(x, split, fixed=FALSE)</code>	Разбивает элементы текстового вектора <code>x</code> по значениям <code>split</code> . Если <code>fixed=FALSE</code> , то элемент – это регулярное выражение. Если <code>fixed=TRUE</code> , то элемент – это текстовая строка. <code>y <- strsplit("abc", "")</code> возвращает список, состоящий из одного компонента и трех элементов: "a" "b" "c". Как <code>unlist(y) [2]</code> , так и <code>sapply(y, "[", 2)</code> возвращает "b"
<code>paste(..., sep="")</code>	Объединяет элементы, разделяя их указанным символом. <code>paste("x", 1:3, sep="")</code> возвращает <code>c("x1", "x2", "x3")</code> . <code>paste("x", 1:3, sep="M")</code> возвращает <code>c("xM1", "xM2" "xM3")</code> . <code>paste("Today is", date())</code> возвращает <code>Today is Thu Jun 25 14:17:32 2011</code>
<code>toupper(x)</code>	Перевод в верхний регистр <code>toupper("abc")</code> возвращает "ABC"
<code>tolower(x)</code>	Перевод в нижний регистр <code>tolower("ABC")</code> возвращает "abc"

Обратите внимание на то, что функции `grep()`, `sub()` и `strsplit()` способны осуществлять поиск текстовых строк (`fixed=TRUE`) или регулярных выражений (`fixed=FALSE` по умолчанию). Регулярные выражения имеют простой и последовательный синтаксис для обозначения определенного элемента 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`. Чтобы узнать больше, прочтите статью «регулярные выражения» в Wikipedia⁴.

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.

4 См. более полный вариант статьи на английском языке: Regular expressions. – *Прим. нер.*

Таблица 5.7. Другие полезные функции

Функция	Описание
<code>length(x)</code>	Число элементов объекта <code>x</code> . <code>x <- c(2, 5, 6, 9)</code> <code>length(x)</code> равно 4
<code>seq(from, to, by)</code>	Создание последовательности элементов. <code>indices <- seq(1, 10, 2)</code> <code>indices</code> равно <code>c(1, 3, 5, 7, 9)</code>
<code>rep(x, n)</code>	Повторяет <code>x n</code> раз. <code>y <- rep(1:3, 2)</code> <code>y</code> равно <code>c(1, 2, 3, 1, 2, 3)</code>
<code>cut(x, n)</code>	Преобразует непрерывную переменную <code>x</code> в фактор с <code>n</code> уровнями. Для создания упорядоченного фактора добавьте опцию <code>ordered_result = TRUE</code>
<code>pretty(x, n)</code>	Создает «красивые» пограничные значения. Разделяет непрерывную переменную <code>x</code> на <code>n</code> интервалов, выбрав <code>n+1</code> одинаково отстоящих друг от друга округленных значений. Часто используется при построении диаграмм
<code>cat(..., file = "myfile", append = FALSE)</code>	Объединяет объекты в ... и выводит их на экран или в файл (если указано его название). <code>firstname <- c("Jane")</code> <code>cat("Hello", firstname, "\n")</code>

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

```
name <- "Боб"
cat("Привет", name, "\b.\n", "Правда R", "\t", "ЭТО ЗДОРОВО?\n")
дает следующий результат
```

Привет Боб.
Правда R ЭТО ЗДОРОВО?

Обратите внимание на то, что вторая строка смешена 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 «Привет Боб.».

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 с 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 другие 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 значения для столбцов
[1] -0.1699  0.0127 -0.6475 -0.6575  0.2312
```

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% наибольших и наименьших 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 из раздела 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 сту-

дента, 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])          ← Вычисляем объединенный
> score <- apply(z, 1, mean)        ← показатель успеваемости
> roster <- cbind(roster, score)
> y <- quantile(score, c(.8,.6,.4,.2)) ← Оцениваем студентов
> 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), " ") ← Извлекаем фамилии
> lastname <- sapply(name, «[«, 2)
> firstname <- sapply(name, «[«, 1)
> roster <- cbind(firstname, lastname, roster[,-1])

> roster <- roster[order(lastname,firstname),]

> 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 насыщенный, 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 средние и стандартные отклонения 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 стандартизованных 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 (процентилей) показателя успеваемости. Вы видите, что нижняя граница для оценки А равна 0.74, для оценки В – 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 приводит к созданию списка:

```
> 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 векторы 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()`:

	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 условным операторам 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 порядком выполнения 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 составная 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 по единице после каждого цикла, то R никогда бы не закончила 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`, `ifelse` 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) утверждение  
if (cond) утверждение 1 else утверждение 2
```

Вот примеры:

```
if (is.character(grade)) grade <- as.factor(grade)  
if (!is.factor(grade)) grade <- as.factor(grade) else print  
  ("Переменная grade -- уже фактор")
```

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.

ifelse

Конструкция `ifelse` – компактная и векторизованная версия конструкции `if-else`. Синтаксис таков:

```
ifelse(cond, утверждение1, утверждение2)
```

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 разделить данные на две категории или оперировать векторами.

switch

Конструкция `switch` выбирает команды в зависимости от значения, которое принимает выражение. Синтаксис таков:

```
switch(expr, ...)
```

где многоточие 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 надуманный пример, зато он демонстрирует основные принципы применения этой конструкции. Вы узнаете, как использовать ее в созданных пользователем функциях, из следующего раздела.

5.5. Функции, написанные пользователем

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

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

Объекты внутри функции существуют только там. Объект, который возвращается 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 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 следующем 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 большие возможности. Прекрасный источник информации по этой теме – онлайн-пособие Дункан Мардоч (Duncan Murdoch) «Отладка команд в R»⁵ (<http://www.stats.uwo.ca/faculty/murdoch/software/debuggingR>).*

⁵ «Debugging in R» на английском языке. – Прим. пер.

Когда функции созданы, вам может захочеться, 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 прекрасные книги, ссылки на которые вы найдете в списке литературы в конце этой книги: Venabley & 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 изменения структуры данных.

5.6. Агрегирование и изменение структуры данных

В R реализовано множество мощных методов объединения и изменения структуры данных. При агрегации данных подгруппы наблюдений заменяются статистиками, основанными на этих наблюдениях. При реструктуризации 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 реструктуризации 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 агрегировать 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 агрегирования, а `FUN` – это скалярная функция, которая используется для расчета статистик, 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 хватает одного галлона топлива, чтобы проехать в среднем 21.5 мили (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()` большие возможности. Однако когда дело касается больших возможностей, ничто не затмит пакета `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 «расплываете» 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 со структурой, которая показана в табл. 5.9.

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()*.

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

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.

Поскольку формулы, представленные в правой части рисунка (d, e и f), не содержат функций, данные просто реструктуризируются. Напротив, в примерах, расположенных слева (a, b и c), в качестве агрегирующей функции задано вычисление среднего арифметического. Так что здесь не только изменяется структура данных, но они еще и агрегируются. Например, в случае (a) вычисляются средние значения переменных X_1 и X_2 для каждого значения переменной ID. В случае (b) приведены средние значения этих переменных для каждого значения переменной Time. В случае (c) усреднены значения переменных X_1 и X_2 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 реструктуризовать или агрегировать. Например, нередко 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 и статистических функций, а также функций распределения, которые используются для работы с данными. Мы узнали, как применять эти функции к широкому спектру объектов с данными, включая векторы, матрицы и таблицы данных. Мы познакомились с конструкциями, управляющими выполнением функций, для повторного применения одной и той же функции или выполнения команд только при определенных условиях. У вас была возможность создать свои собственные команды и применить их к данным. Наконец, мы узнали способы «свертывания», агрегирования и изменения структуры данных.

Теперь вы вооружены всеми средствами, чтобы преобразовать ваши данные в нужный вид. Мы готовы расщепиться 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.

Переформатирование набора данных

С группировкой

Без группировки

mydata

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

cast(md, id~variable, mean)

ID	X1	X2
1	4	5.5
2	4	2.5

(a)

cast(md, id+time~variable)

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

(d)

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

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

cast(md, id+time~variable)

ID	Variable	Time1	Time2
1	X1	5	3
1	X2	6	5
2	X1	6	2
2	X2	1	4

(e)

cast(md, id-time, mean)

ID	Time1	Time2
1	5.5	4
2	3.5	3

(f)

ID	X1	Time1	X2	Time2
1	5	3	6	5
2	6	2	1	4

Рис. 5.1. Изменение структуры данных при помощи функций melt() и cast()



Часть II.

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, диаграммы размахов, 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 обобщенных характеристик отдельных переменных и взаимосвязей между парами переменных. Глава начинается с описания того, как находить обобщенные характеристики числовых данных (как для всего набора данных, так и для его части). Затем описано построение сводных и частотных таблиц для характеристик 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 диаграммы размахов.
- Круговые и веерные диаграммы.
- Гистограммы и диаграммы распределения плотности.

Когда бы мы ни 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 машины в милях на галлон) или категориальной (например, результат лечения: отсутствует, незначительный, 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, диаграммам размахов, 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 `Arthritis`, распространяемой с пакетом `vcd`. Поскольку этот пакет не входит в базовую версию R, убедитесь, что перед первым использованием вы скачали и установили его командой `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 потом мы его используем для создания списков, 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 параметры, создающие надписи. 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,
        main="Вертикальная столбчатая диаграмма",
        xlab="Улучшение", ylab="Частота")
barplot(counts,
        main="Горизонтальная столбчатая диаграмма",
        xlab="Частота", ylab="Улучшение",
        horiz=TRUE)
```

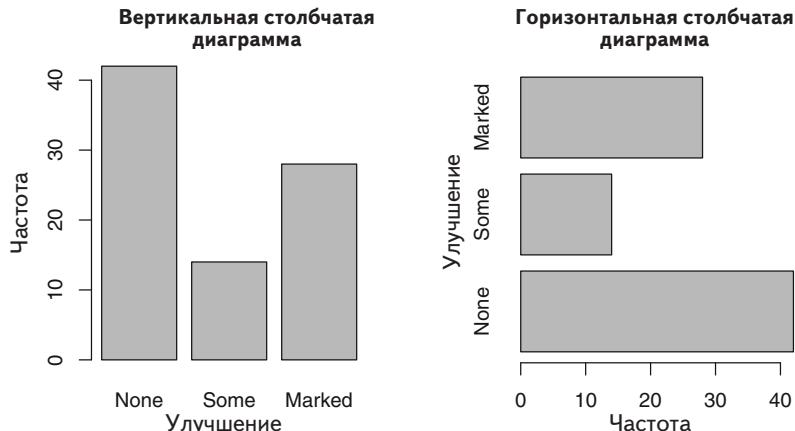


Рис. 6.1. Простые вертикальная и горизонтальная столбчатые диаграммы

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$Improved, main="Вертикальная столбчатая диаграмма",
      xlab="Улучшение", ylab="Частота")
plot(Arthritis$Improved, horiz=TRUE, main="Горизонтальная столбчатая
      диаграмма",
      xlab="Частота", ylab="Улучшение")
```

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 обозначения уровней фактора слишком длинные? В разделе 6.1.4 вы увидите, как оптимизировать эти обозначения, чтобы они не перекрывались на диаграмме.

6.1.2. Столбчатые диаграммы: составные и с группировкой

Если `height` – матрица, а не вектор, для нее будет построена составная столбчатая диаграмма (stacked bar plot) или столбчатая диаграмма с группировкой (grouped bar plot). 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, составной или с группировкой (см. приведенный ниже программный код). Получившиеся диаграммы представлены на рис. 6.2.

Программный код 6.2. Столбчатые диаграммы, составные или с группировкой	◀ Столбчатая диаграмма с вертикальным разбиением на подгруппы
<pre>barplot(counts, main="Составная столбчатая диаграмма", xlab="Лечение", ylab="Частота", col=c("red", "yellow", "green"), legend=rownames(counts))</pre>	◀ Столбчатая диаграмма с горизонтальным разбиением на подгруппы
<pre>barplot(counts, main="Столбчатая диаграмма с группировкой", xlab="Лечение", ylab="Частота", col=c("red", "yellow", "green"), legend=rownames(counts), beside=TRUE)</pre>	

Первая команда позволяет получить составную столбчатую диаграмму («слоистую» диаграмму), а вторая – столбчатую диаграмму

с группировкой. 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).

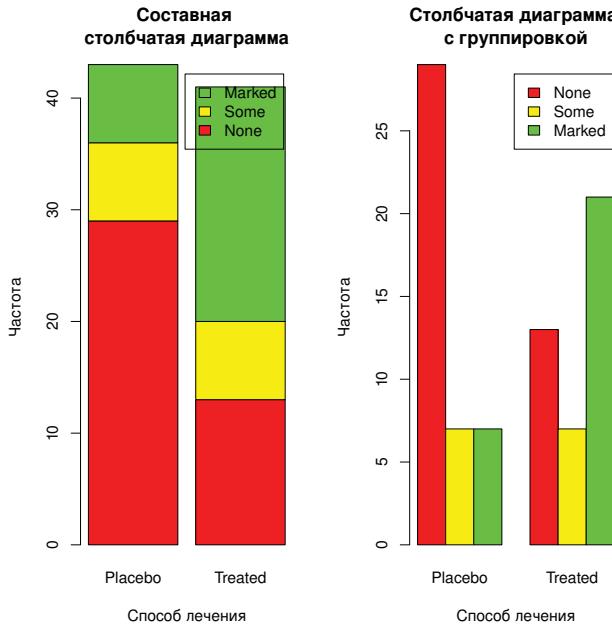


Рис. 6.2. Столбчатые диаграммы: составные и с группировкой

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 т. д., 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)    ↪❷ Добавляем
название
> title("Средний уровень неграмотности")
```

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.

Средний уровень неграмотности

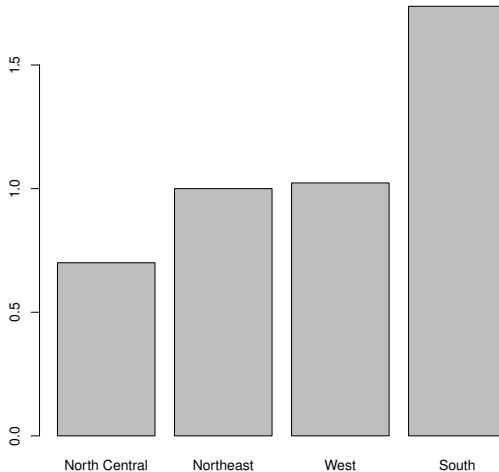


Рис. 6.3. Столбчатая диаграмма для отсортированных в порядке возрастания средних значений неграмотности в разных частях США

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. Оптимизация столбчатых диаграмм

Есть несколько способов оптимизации 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 расположеннном 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="Результат лечения",
        horiz=TRUE, cex.names=0.8,
        names.arg=c("Нет улучшения", "Некоторое улучшение",
                   "Заметное улучшение"))
```

В этом примере мы повернули столбцы диаграммы (`las=2`), изменили текст подписей, а также одновременно увеличили левое поле 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.

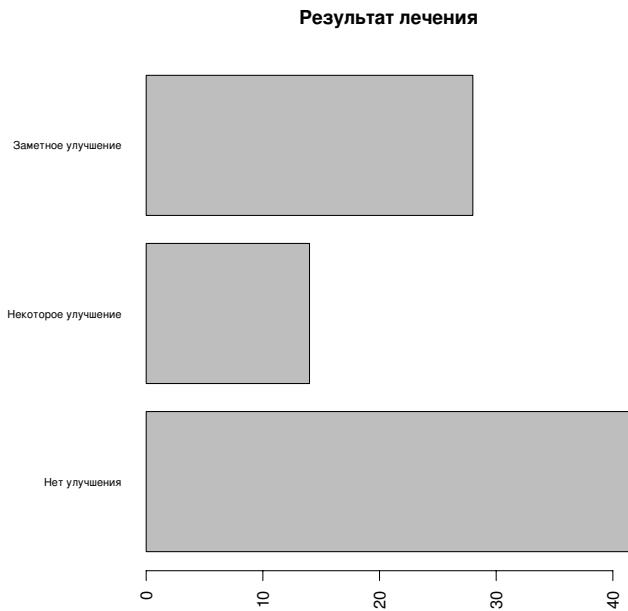


Рис. 6.4. Горизонтальная столбчатая диаграмма с оптимизированными подписями

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 изменяется так, что высота каждого результирующего столбца становится равной 1, а высоты его составляющих отражают пропорции. Спинограммы создаются при помощи команды `spine()` из пакета `vcd`. Следующий программный код позволяет получить простую спинограмму:

```
library(vcd)
attach(Arthritis)
counts <- table(Treatment, Improved)
spine(counts, main="Пример спинограммы")
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.

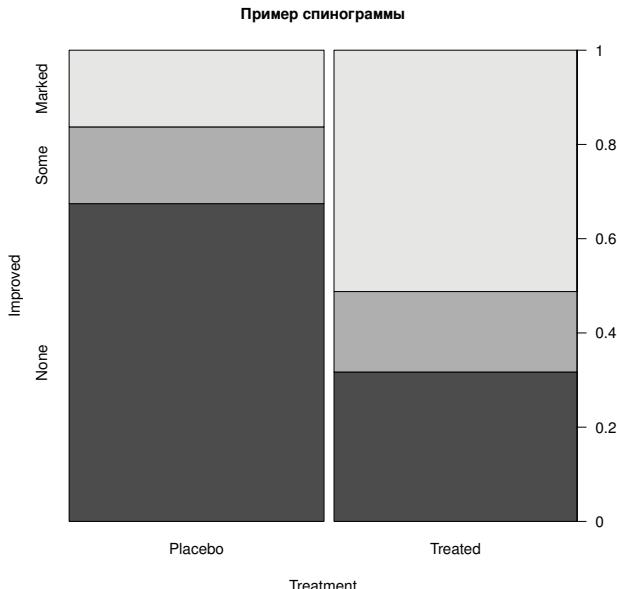


Рис. 6.5. Спинограмма, на которой отражены результаты лечения артрита

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

6.2. Круговые диаграммы

Несмотря на то, что круговые диаграммы широко используются в деловом мире, они критикуются большинством статистиков, включая тех, кто пишет справочные тексты 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` – тек-

стовый вектор, содержащий подписи для сегментов. В представленном ниже программном коде содержатся четыре примера, а результат представлен на рис. 6.6.

Программный код 6.5. Круговые диаграммы

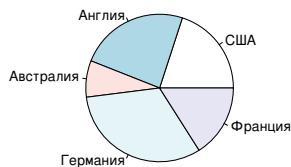
```
par(mfrow=c(2, 2))  
slices <- c(10, 12, 4, 16, 8)           ↪ ① Объединяем четыре  
lbls <- c("США", "Англия", "Австралия", "Германия", "Франция")  
  
pie(slices, labels = lbls,  
main="Простая круговая диаграмма")  
  
pct <- round(slices/sum(slices)*100)    ↪ ② Добавляем проценты  
lbls2 <- paste(lbls, " ", pct, "%", sep="")  
pie(slices, labels=lbls2, col=rainbow(length(lbls2)),  
main="Круговая диаграмма с процентами")  
  
library(plotrix)  
pie3D(slices, labels=lbls, explode=0.1,  
main="»Трехмерная круговая диаграмма")  
  
mytable <- table(state.region)          ↪ ③ Делаем диаграмму  
lbls3 <- paste(names(mytable), "\n", mytable, sep="")  
pie(mytable, labels = lbls3,  
main="Круговая диаграмма для таблицы \n(с размерами выборок)")
```

Сначала вы задаете параметры диаграммы, так что все четыре элемента объединены на одной панели ① (объединение нескольких диаграмм в одну комбинированную рассмотрено в главе 3). Затем вы вводите данные, которые будут использованы в первых трех диаграммах.

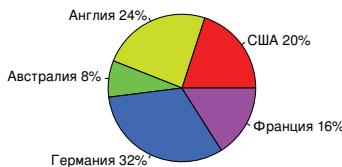
Для второй круговой диаграммы ② вы преобразовываете размер выборок в проценты и добавляете эту информацию к подписям сегментов. Кроме того, сегменты этой диаграммы раскрашены с использованием функции `rainbow()`, описанной в главе 3. В нашем случае `rainbow(length(lbls2))` означает `rainbow(5)`, что 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.

Простая круговая диаграмма



Круговая диаграмма с процентами



Трехмерная круговая диаграмма

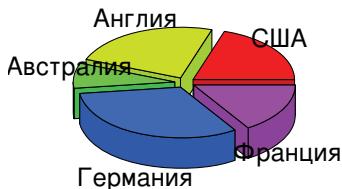
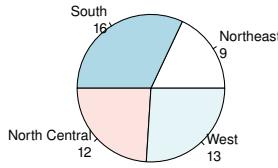
Круговая диаграмма для таблицы
(с размерами выборок)

Рис. 6.6. Примеры круговых диаграмм

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("США", "Англия", "Австралия", "Германия", "Франция")
fan.plot(slices, labels = lbls, main="Веерная диаграмма")
```

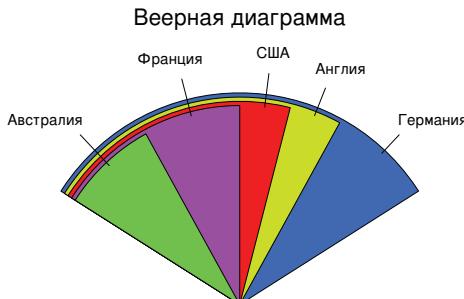


Рис. 6.7. Веерная диаграмма для данных по странам

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 видеть, на веерной диаграмме гораздо легче сравнивать размер сегментов по сравнению с круговой диаграммой. Веерные диаграммы пока 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.

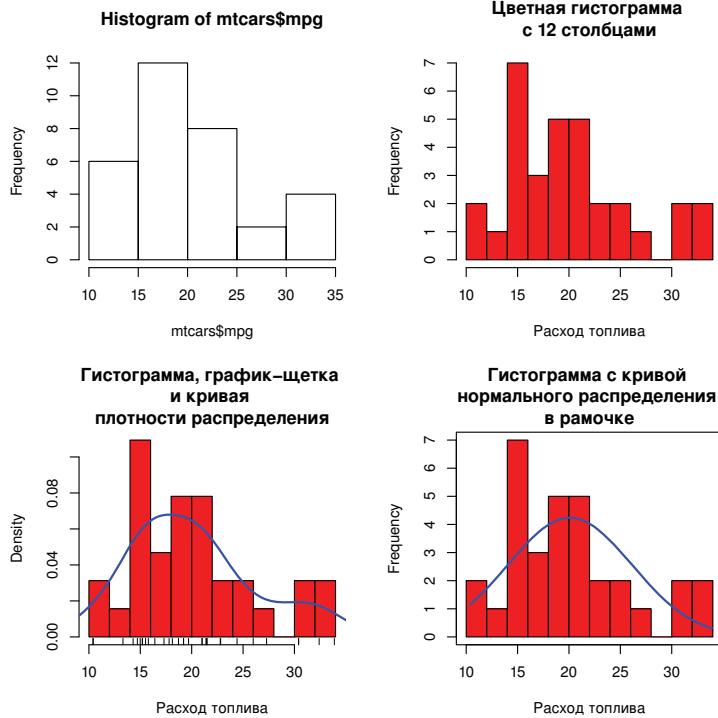


Рис. 6.8. Примеры построения гистограмм

Программный код 6.6. Гистограммы

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

hist(mtcars$mpg)
```

❶ Простая гистограмма

```
hist(mtcars$mpg,
      breaks=12,
      col="red",
      xlab="Расход топлива",
      main="Цветная гистограмма с 12 столбцами")
```

❷ Гистограмма с заданным числом интервалов и раскрашенными столбцами

```
hist(mtcars$mpg,
      freq=FALSE,
      breaks=12,
      col="red",
      xlab="Расход топлива",
      main="Гистограмма, график-щетка и кривая плотности распределения точек")
rug(jitter(mtcars$mpg))
```

❸ С кривой плотности распределения точек

```
lines(density(mtcars$mpg), col="blue", lwd=2)

x <- mtcars$mpg
h<-hist(x,
          breaks=12,
          col="red",
          xlab="Расход топлива",
          main="Гистограмма с кривой нормального распределения в рамочке")
xfit<-seq(min(x), max(x), length=40)
yfit<-dnorm(xfit, mean=mean(x), sd=sd(x))
yfit <- yfit*diff(h$mid[1:2])*length(x)
lines(xfit, yfit, col="blue", lwd=2)
box()
```

4 С кривой нормального распределения и в рамочке

Первая гистограмма ❶ – это то, что получается по умолчанию, когда никаких параметров функции не указано. 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 закрасить столбики красным цветом.

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 ядерная оценка функции плотности, 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 вдвое толще, чем по умолчанию. Наконец, график-щетка позволяет увидеть реальные значения переменной, наложенные на прямую. Если в выборке много близких значений, можно немного сместить их друг относительно друга при помощи команды `rug()`:

```
rug(jitter(mtcars$mpg, 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 рамка. Программный код, который позволяет добавить кривую нормального распределения, предложен Петером Дальгаардом (Peter Dalgaard) в справочной системе рассылки R. Рамку можно нарисовать при помощи команды `box()`.

6.4. Диаграммы ядерной оценки функции плотности

В предыдущем разделе вы видели диаграмму ядерной оценки функции плотности, добавленную к гистограмме. С технической точки зрения ядерная оценка функции плотности – это непараметрический метод оценки значений функции вероятности плотности случайной переменной. Хотя математические выкладки выходят за рамки этого текста, в целом ядерная оценка функции плотности – 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 ядерной оценки функции плотности даны в следующем программном коде, а результат представлен на рис. 6.9.

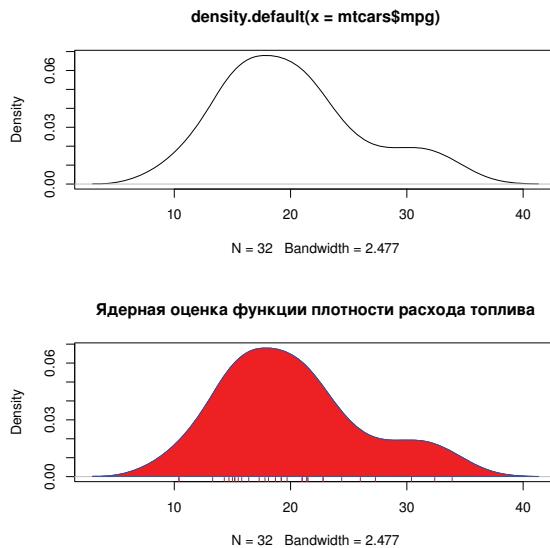


Рис. 6.9. Диаграммы ядерной оценки функции плотности

Программный код 6.7. Диаграммы ядерной оценки функции плотности

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

plot(d)

d <- density(mtcars$mpg)
plot(d, main="Ядерная оценка функции плотности расхода топлива")
polygon(d, col="red", border="blue")
rug(mtcars$mpg, col="brown")
```

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 координатами *x* и *y* (в данном случае они определяются функцией `density()`).

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

Функция `sm.density.compare()` пакета `sm` позволяет наложить друг на друга диаграммы ядерной оценки функции плотности для двух и более групп. Формат применения функции таков:

```
sm.density.compare(x, factor)
```

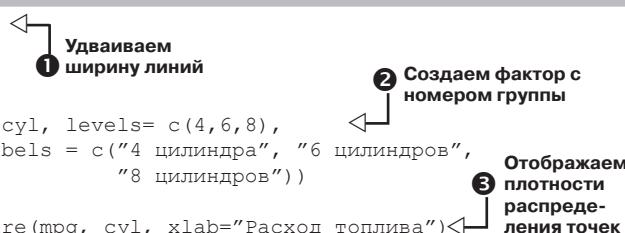
где *x* – это числовой вектор и *factor* – переменная, определяющая принадлежность к группе. Не забудьте установить пакет `sm` перед его первым использованием. Пример, в котором сравнивается расход топлива для машин с 4, 6 и 8 цилиндрами, приведен в программном коде 6.8.

Программный код 6.8. Сравнение диаграмм ядерной оценки функции плотности

```
par(lwd=2)
library(sm)
attach(mtcars)

cyl.f <- factor(cyl, levels= c(4,6,8),
                 labels = c("4 цилиндра", "6 цилинров",
                           "8 цилинров"))

sm.density.compare(mpg, cyl, xlab="Расход топлива")
```



```
title(main="Расход топлива для машин с разным числом цилиндров")
colfill<-c(2:(1+length(levels(cyl.f))))
legend(locator(1), levels(cyl.f), fill=colfill)
detach(mtcars)
```

4 Добавляем легенду по щелчку мыши

Функция `par()` используется для удвоения толщины линий на диаграмме (`lwd=2`) – так их легче воспринимать читателям ①. Пакет `sm` загружен, и таблица данных `mtcars` активирована.

В таблице данных `mtcars` ② `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 ③, а функция `title()` добавляет ее название.

Наконец, вы добавляете легенду, чтобы сделать диаграмму понятнее ④ (легенды обсуждались в главе 3). Сначала мы создаем вектор цветов. Здесь `colfill` равен `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.

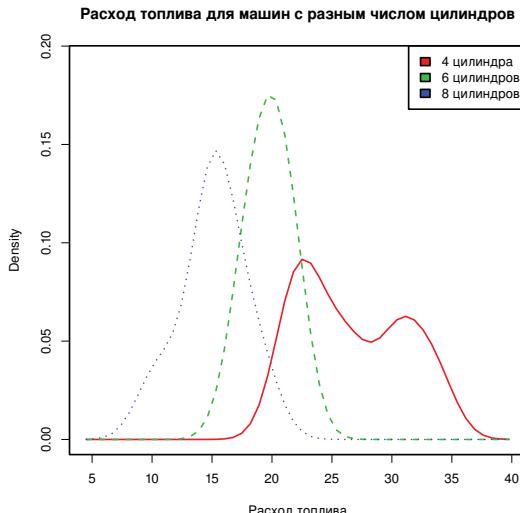


Рис. 6.10. Диаграммы ядерной оценки функции плотности для расхода топлива у машин с разным числом цилиндров



Видно, что при помощи наложенных друг на друга диаграмм ядерной оценки функции плотности 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 вообще электропривод.

Диаграммы размахов – также замечательный (и гораздо более широко используемый) подход для визуализации распределения значений и различий для групп наблюдений. Следующими мы обсудим их.

6.5. Диаграммы размахов

Диаграммы размахов (box plot) иллюстрируют распределение значений непрерывной переменной, отображая пять параметров: минимум, нижний quartиль (25-й процентиль), медиану (50-й процентиль), верхний quartиль (75-й процентиль) и максимум. На этой диаграмме также могут быть отображены вероятные выбросы (значения, выходящие за диапазон в ± 1.5 межквартильного размаха, разности верхней и нижней quartилей). Например, команда

```
boxplot(mtcars$mpg, main="Ящик с усами", ylab="Расход топлива")
```

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



Рис. 6.11. Диаграмма размахов («ящик с усами») с добавленными вручную подписями

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

Например, в нашей выборке данных о машинах медиана для расхода топлива составляет 19.2 мили 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 диаграмм размахов для сравнения групп между собой

Диаграммы размахов 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 $y \sim A$, где для каждого значения категориальной переменной *A* будет построена отдельная диаграмма размахов для числовой переменной *y*. Формула $y \sim A * B$ позволит получить отдельные диаграммы размахов для всех комбинаций значений переменной *y*, заданных категориальными переменными *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 возвращаемся к сравнению расхода топлива у автомобилей с четырьмя, шестью и восемью цилиндрами, но теперь уже при помощи расположенных рядом диаграмм размахов. Диаграмма представлена на рис. 6.12.

```
boxplot(mpg ~ cyl, data=mtcars,
        main="данные о расходе топлива",
        xlab="Число цилиндров",
        ylab="Расход топлива")
```

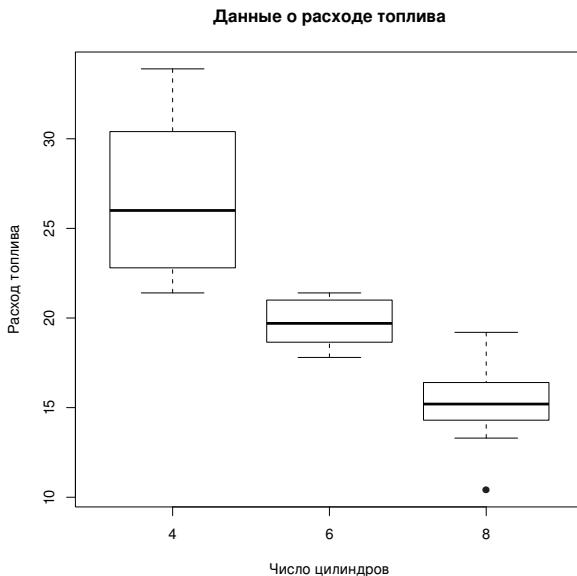


Рис. 6.12. Диаграмма размахов для расхода топлива у автомобилей с разным числом цилиндров

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 больший разброс (и положительный сдвиг распределения) значений расхода топлива. В группе с восьмицилиндровыми двигателями есть выброс.

Диаграммы размахов очень многообразны. Если добавить параметр `notch=TRUE`, то получатся «ящики» с «насечками». Если «насечки» двух ящиков не перекрываются, высока вероятность того, что медианы соответствующих совокупностей различаются (Chambers et al., 1983, стр. 62). Этот программный код позволяет получить «ящики» с «усами» и «насечками» для нашего примера с расходом топлива:

```
boxplot(mpg ~ cyl, data=mtcars,
        notch=TRUE,
        varwidth=TRUE,
        col="red",
        main="Данные о расходе топлива",
        xlab="Число цилиндров",
```

```
ylab="Расход топлива")
```

Опция `col` позволяет залить «ящики» красным цветом, а параметр `varwidth=TRUE` делает ширину «ящиков» пропорциональной разме-
ру выборок. На рис. 6.13 видно, что медианные значения расхода топ-
лива у четырех-, шести- и восьмицилиндровых машин различаются.
Расход топлива заметно уменьшается с увеличением числа цилинд-
ров.

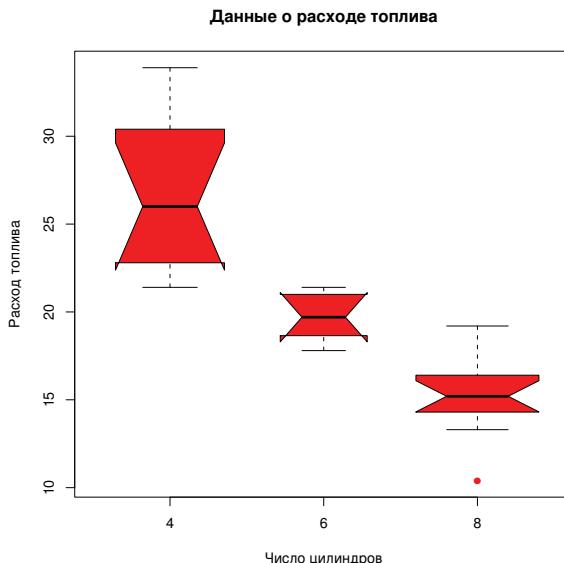


Рис. 6.13. «Ящики с усами» и «насечками», отражающие расход топлива у автомобилей с разным числом цилиндров

Наконец, можно создавать диаграммы размахов для нескольких группирующих переменных. В программном коде 6.9 показано, как построить такие диаграммы для машин с разным числом цилиндров и разными коробками передач. Мы вновь используем параметр `col` для закрашивания «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. Диаграмма размахов для всех сочетаний значений двух факторов

```
mtcars$cyl.f <- factor(mtcars$cyl,
                           levels=c(4,6,8),
                           labels=c("4","6","8"))
```

Представляем
число цилиндров
в виде фактора

```

mtcars$am.f <- factor(mtcars$am,
                       levels=c(0,1),
                       labels=c("авто", "ручная"))
boxplot(mpg ~ am.f * cyl.f,
        data=mtcars,
        varwidth=TRUE,
        col=c("gold","darkgreen"),
        main="Расход топлива у разных типов машин",
        xlab="Тип автомобиля")
    
```

↗ Представляем тип передачи в виде фактора
↗ Создаем «ящик с усами»

Полученная диаграмма представлена на рис. 6.14.

Расход топлива у разных типов машин

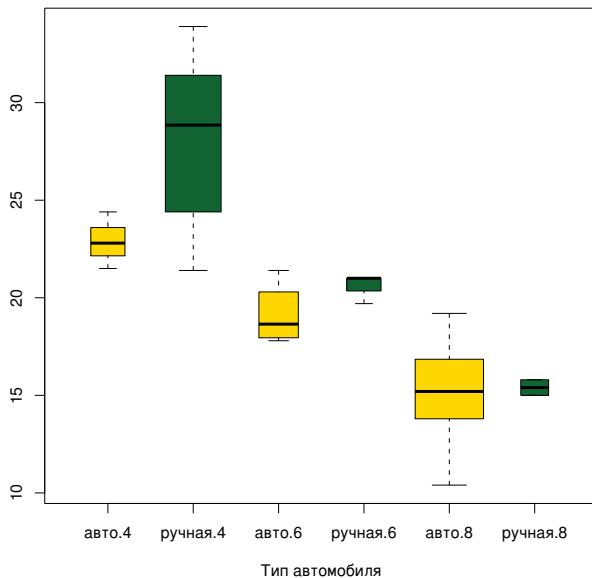


Рис.6.14. Диаграмма размахов, отражающая расход топлива для всех комбинаций числа цилиндров и типа коробки передач

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 при наличии ручной (standart) коробки 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 диаграммы размахов, нам стоит рассмотреть их модификацию, названную «скрипичной диаграммой» (violin plot). Это сочетание диаграммы размахов и диаграммы ядерной оценки функции плотности. Такую 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 6.10. 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 цилиндра", "6 цилиндров", "8 цилиндров"),
        col="gold")
title("Скрипичные диаграммы для расхода топлива")
```

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

Скрипичные диаграммы представляют собой симметричные диаграммы ядерной оценки функции плотности, наложенные на диаграммы размахов. Здесь белая точка – медиана, черный прямоугольник – межквартильный размах, а тонкие черные линии – «усы». Внешний контур фигуры – это диаграмма ядерной оценки функции плотности. 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.

Скрипичные диаграммы для расхода топлива

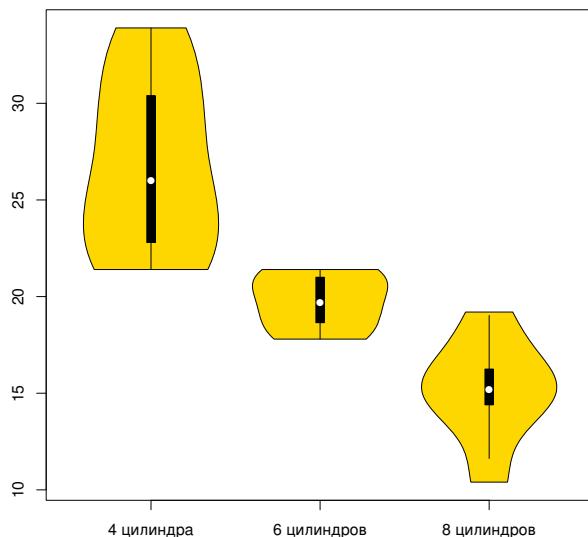


Рис. 6.15. Скрипичные диаграммы, отражающие расход топлива у автомобилей с разным числом цилиндров

6.6. Точечные диаграммы

Точечные диаграммы (dot plot) позволяют изобразить множество поименованных значений на простой горизонтальной шкале. Эта диаграмма создается при помощи функции `dotchart()`, применяемой 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` для назначения фактора, определяющего группирование элементов вектора `x`. В этом случае параметр `gcolor` определяет цвет подписей 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="Расход топлива для разных марок автомобилей",  
        xlab="Мили на галлон")
```

Получившаяся диаграмма представлена на рис. 6.16.

Расход топлива для разных марок автомобилей

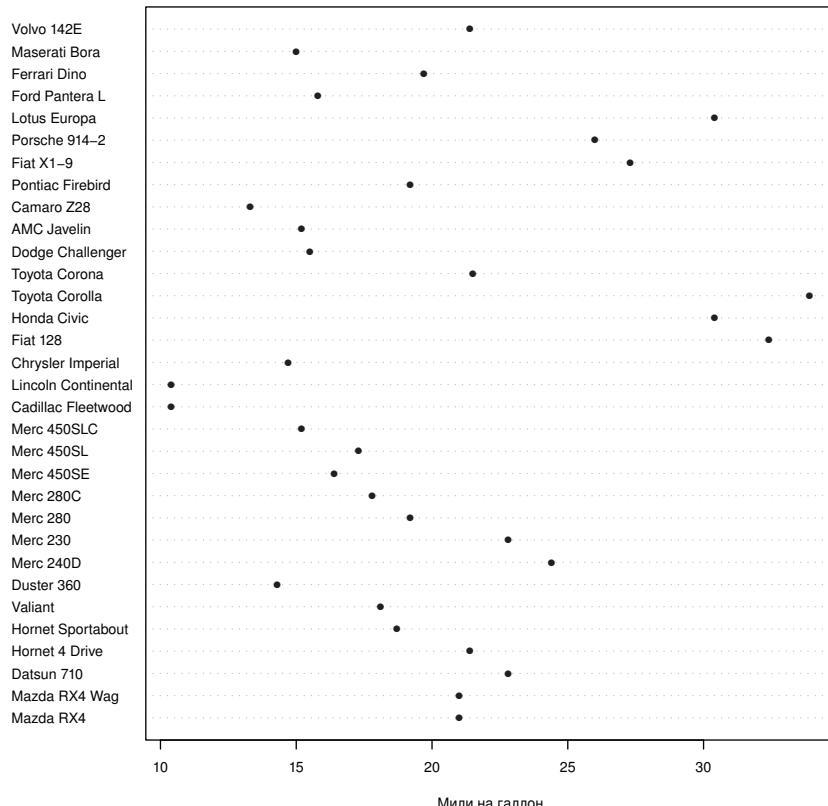


Рис. 6.16. Точечная диаграмма для расхода топлива у автомобилей разных марок

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$cyl[x$cyl==6] <- "blue"
x$cyl[x$cyl==8] <- "darkgreen"
dotchart(x$mpg,
         labels = row.names(x),
         cex=.7,
         groups = x$cyl,
         gcolor = "black",
         color = x$cyl,
         pch=19,
         main = "Расход топлива для разных марок автомобилей,
←сгруппированных по числу цилиндров",
         xlab = "Миль на галлон")
```

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 “`darkgreen`” (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 ядерной оценки функции плотности, диаграмм размахов, графиков-щеток и точечных диаграмм можно визуализировать распределение значений непрерывных переменных. Наконец, мы исследовали, как использовать наложенные друг на друга диаграммы ядерной оценки функции плотности, расположенные рядом диаграммы размахов 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, коррелограммы, 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. Далее 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)¹. 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`).

```
> vars <- c("mpg", "hp", "wt")
```

1 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.

```
> 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 машин 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 по-трепает 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 также 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, нижний квартиль, медиана, верхний квартиль, максимум).

Удивительно, но в базовой версии программы нет функций для характеристики асимметрии и эксцесса распределения значений, но вы можете создать такие функции самостоятельно. Пример в следующем программном коде демонстрирует, как вычислить несколько описательных статистик, включая асимметрию 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 мили 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 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 наименьших и пять наибольших 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 коэф-

фициент вариации. Наконец, если `norm=TRUE` (не по умолчанию), вычисляются статистики нормального распределения, включая асимметрию 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()` из 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, асимметрию, эксцесс и стандартную ошибку среднего. Пример приведен в следующем программном коде.

Программный код 7.5. Вычисление описательных статистик при помощи функции `describe()` из 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 для вычисления только одной статистики за один раз, такой как среднее, стандартное отклонение и подобных им. 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 `dplyr` 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 `dplyr` 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 данных 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 *psych()* 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, для вычисления описательных статистик для групп можно использовать описанный в разделе 5.6.3 пакет `reshape`. Если вы не читали этого раздела, 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, анализирующие 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 вам больше подходит, или разработайте свой собственный!

7.1.3. Визуализация результатов

Представление характеристик распределения данных в числовом виде полезно, но оно не заменяет собой его графического изображения. Для количественных данных подходят гистограммы (раздел 6.3), диаграммы плотности рассеяния (раздел 6.4), диаграммы размахов (раздел 6.5) и точечные диаграммы (раздел 6.6). Они помогают заметить вещи, которые можно легко упустить, 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 R, наряду с командами из пакетов `vcd` и

гмоделс. 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 основе 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)

Функция	Описание
<code>addmargins(table, margins)</code>	Вычисляет описательные статистики <i>margins</i> (суммы по умолчанию) для таблицы
<code>ftable(table)</code>	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 определенное 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 знака `~`) и разделяются знаками `+`. Если переменная находится в левой части формулы, предполагается, что это вектор с частотами значений (удобно, если вы их уже вычислили).

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), 2)
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 добавления пропущенных значений 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()` из пакета `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 Мак-Немара на независимость; вычислять ожидаемые значения и остатки (по Пирсону, 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. Давайте теперь 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
, , 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)
```

Групповые частоты 1

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



2 Сумма частот по строкам и столбцам



3 Сумма частот всех сочетаний значений столбцов Treatment и Improved



4 Сумма частот всех сочетаний значений столбцов Treatment и Sex

Выражение **1** позволяет вычислить частоты для всех сочетаний трех факторов. Также видно, что команда `ftable()` позволяет представить результаты в более компактном и привлекательном виде.

Выражение **2** 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 описаны три теста: хи-квадрат, Фишера и Кохрана-Мантеля-Хензеля.

Тест хи-квадрат

Применить функцию `chisq.test()` можно к двухмерной 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
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
X-squared = 4.84, df = 2, p-value = 0.0889
Warning message:
In chisq.test(mytable) : Chi-squared approximation may be incorrect
```

1 Тип лечения
и его результат
не независимы

2 Тип лечения
и пол пациента
независимы

Результаты теста ① 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 ($p > 0.05$) ②. 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 ①, мы отвергаем гипотезу о независимости результата лечения от его типа. Раз вероятность в случае ② 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 недействительной аппроксимацию хи-квадрат.

Точный критерий Фишера

Точный 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 таблице сопряженности (с двумя и более столбцами и строками), а не только к таблице размерности 2×2 .

Тест Кохрана-Мантеля-Хензеля

Функция `mantelhaen.test()` позволяет провести хи-квадрат тест Кохрана-Мантеля-Хензеля (Cochran-Mantel-Haenszel) в отношении нулевой гипотезы о том, что две номинальные переменные условно независимы при каждом значении третьей переменной. Приведенный ниже программный код дает возможность проверить гипотезу о том, что переменные `Treatment` и `Improved` независимы при каждом значении переменной `Sex`. При проведении теста предполагается, что трехстороннее взаимодействие ($Treatment \times Improved \times Sex$) 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 вне зависимости от пола пациентов между способом лечения и его результатом имеется выраженная связь.

7.2.3. Показатели взаимосвязи

Статистические тесты, описанные в предыдущем разделе, оценивали, есть ли достаточные основания для отклонения нулевой гипотезы о независимости двух переменных. Если вы смогли отвергнуть нулевую гипотезу, ваш интерес естественным образом смещается в сторону показателей взаимосвязи, позволяющих измерить силу обнаруженных связей. Функция `assocstats()` из пакета `vcd` используется для вычисления фи-коэффициента (`phi coefficient`), коэффициента сопряженности и V-коэффициента 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 большие значения коэффициентов свидетельствуют о более сильной связи. В пакете `vcd` также реализована функция `карпа()`, которая вычисляет каппу Коэна (Cohen's карпа) и взвешенную каппу для матрицы соответствий (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 данных 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

[далее 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 в качестве аргумента таблицу R (с любым числом строк и столбцов) и возвращает ее 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 содержатся 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

Линейный коэффициент корреляции Пирсона (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
<code>x</code>	Матрица или таблица данных
<code>use=</code>	Упрощает работу с пропущенными данными. Может принимать следующие значения: <code>all.obs</code> (предполагается, что пропущенные значения отсутствуют; их наличие вызовет сообщение об ошибке), <code>everything</code> (любая корреляция, включающая строку с пропущенным значением, не будет вычисляться – обозначается как <code>missing</code>), <code>complete.obs</code> (учитываются только строки без пропущенных значений) и <code>pairwise.complete.obs</code> (учитываются все полные наблюдения для каждой пары переменных в отдельности)

Опция	Описание
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 статистической значимости корреляций

Как проверить статистическую значимость вычисленных коэффициентов корреляции? Стандартная нулевая гипотеза – это отсутствие связи (то есть коэффициент корреляции для генеральной совокупности равен нулю). Для проверки значимости 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 статистической значимости 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 коэффициент корреляции Пирсона 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 $p\text{-value} = 1.258e - 08$). Учитывая, насколько мала вероятность, мы отвергнем нулевую гипотезу и примем альтернативную – о том, что значение этого коэффициента для генеральной совокупности *не* равно нулю.

К сожалению, при помощи функции `cor.test()` одновременно можно проверить значимость 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 значимости 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	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 от нуля ($p = 0.5$).

Другие тесты на статистическую значимость

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 статистическую значимость. Эту функцию можно использовать, чтобы проверять значимость:

- 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 Юге²», но благородумие взяло верх.

² Deep South – традиционное название южных штатов, связанное с особым укладом жизни и традициями их населения. – *Прим. пер.*

7.4.1. Тест Стьюдента для независимых выборок

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 выборок. 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 y_1 и y_2 – это числовые векторы (анализируемые значения для каждой из групп). Необязательный аргумент `data` назначает матрицу или таблицу данных, в которой содержатся данные. В отличие от большинства статистических программ, в R по умолчанию не подразумевается равенство дисперсий и используется трансформация степеней свободы Уэлча (Welch degrees of freedom modification). Указать на равенство дисперсий можно при помощи параметра `var.equal=TRUE`. По умолчанию используется двухсторонняя альтернативная гипотеза (о том, что средние значения различаются, но не важно, в какую сторону). 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 двухстороннего теста, не предполагая равенства 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
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.

7.4.2. Тест Стьюдента для зависимых выборок

В качестве второго примера можно узнать, правда ли, что 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 выборок, в то время как они будут равны в генеральных совокупностях, меньше 0.00000000000000022 (то есть 2.2e - 16).

7.4.3. Когда имеется 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 и перейти к главе 9.

7.5. Непараметрические тесты межгрупповых различий

Если ваши данные не удовлетворяют условиям применения таких параметрических методов, как t-тест или ANOVA, 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, действительно 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 Вилкоксона³ 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 отвергнуть гипотезу о равенстве вероятностей оказаться в тюрьме в южных по сравнению с штатами ($p < 0.01$).

Тест Вилкоксона⁴ – это непараметрическая альтернатива тесту Стьюдента для зависимых выборок. 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.

³ По умолчанию для независимых переменных. – Прим. пер.

⁴ Для зависимых выборок. – Прим. пер.

```
> 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
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. Когда условия применения тестов Стьюдента выполняются, параметрические тесты имеют большую мощность (вероятность обнаружить существующие различия выше) по сравнению с непараметрическими. Непараметрические тесты разумно применять, когда условия применения параметрических тестов существенно нарушаются (например, для порядковых переменных).

7.5.2. Сравнение более 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, средней продолжительности жизни, уровне преступности и доле людей со средним образованием 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 однофакторный дизайн эксперимента, и для решения поставленного вопроса существуют как параметрические, так 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-отклик, A – группирующая переменная, принимающая два значения и более (в случае с двумя значениями этот тест идентичен тесту Вилкоксона). Формат применения теста Фридмана следующий:

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

где y – это числовая переменная отклика, A – группирующая переменная, B – блочная 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 неодинаков 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 `pmw`.

Честно говоря, здесь я немного вышел за пределы основных методов, заявленных в названии этой главы, но поскольку рассмотрение данного подхода вполне здесь уместно, я надеюсь, что вы меня простите. Для начала убедитесь, что пакет `pmw` установлен. Для выполнения одноименной команды нужно, чтобы ваши данные были представлены в виде двух столбцов, один из которых назван `var` (за-

висимая переменная), а другой – `class` (группирующуюшая 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'
      group.index   class.level nobs
Northeast          1       Northeast    9
South              2           South   16
North Central      3 North Central  12
West               4           West   13
```

1 Попарные
сравнения
групп

```
$'Results of the multiple Behrens-Fisher-Test'
  cmp effect lower.cl upper.cl p.value.ls 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
1        1     1 1.10
2        2     2 1.75
3        3     3 0.70
4        4     4 0.60
```

2 Медианные
значения
неграмотности
в каждом
регионе

Команда `pmc()` 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) ①. Из результатов двустороннего теста на значимость (`p.value.2s`) можно видеть, что юг существенно отличается от трех других регионов, а они между собой не различаются. Видно ②, 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 (асимметрию, бимодальность, выбросы), которые влияют на результаты сравнения, и оценить, насколько данные удовлетворяют требованиям тестов. В R реализовано множество графических методов для сравнения групп, включая диаграммы размахов (простые, с насечками и скрипичные), которые описаны в разделе 6.5; наложенные друг на друга диаграммы ядерной оценки функции плотности, 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 незапланированные, но важные эффекты.



Часть III.

МЕТОДЫ ОБРАБОТКИ ДАННЫХ СРЕДНЕЙ СЛОЖНОСТИ

Вторая часть была посвящена базовым методам обработки данных, а в третьей описаны методы средней сложности. Мы будем двигаться от описания взаимоотношений между двумя переменными к моделированию взаимосвязи между числовой зависимой переменной и набором непрерывных и/или категориальных независимых переменных.

В восьмой главе описаны регрессионные методы для моделирования взаимосвязи между числовой зависимой переменной и набором из одной или более независимых переменных. Моделирование – это, как правило, сложный многоступенчатый интерактивный процесс. В восьмой главе содержится пошаговое описание методов построения линейных моделей, оценки их адекватности и интерпретации их значений.

Девятая глава посвящена обработке результатов основных типов экспериментальных и квази-экспериментальных исследований при помощи дисперсионного анализа и его разновидностей. В данном случае нас будет интересовать, как разные комбинации действующих факторов влияют на числовую зависимую переменную. В этой главе представлены функции, при помощи которых в R осуществляется дисперсионный анализ, ковариационный анализ, дисперсионный анализ для повторных измерений, а также многомерный и многофакторный дисперсионный анализ. Кроме того, обсуждаются

методы оценки соответствия результатов анализа реальной ситуации и способы визуализации результатов.

При разработке схемы проведения экспериментальных и квази-экспериментальных исследований важно определить размер выборки, необходимый для обнаружения интересующих нас закономерностей (проводить анализ мощности). А иначе зачем же проводить исследование? Детальному обсуждению анализа мощности посвящена десятая глава. Она начинается с обсуждения общих вопросов проверки гипотез, но основное внимание уделено тому, как использовать команды R для определения объема выборки, необходимого, чтобы обнаружить эффект заданной силы с заданной степенью уверенности. Это поможет вам планировать исследования, обеспечивающие высокую вероятность получения полезных результатов.

Глава 11 – это логическое продолжение пятой главы. В главе 11 обсуждается создание диаграмм, которые помогут визуализировать взаимосвязи между двумя и более переменными. К таким диаграммам относятся разные типы двух- и трехмерных диаграмм рассеяния, матрицы диаграмм рассеяния, линейные графики и пузырьковые диаграммы. Там также описаны полезные, но менее известные коррелограммы и мозаичные диаграммы.

Линейные модели, описанные в главах 8 и 9, подразумевают, что зависимая переменная не только числовая, но еще и происходит из нормального распределения. Существуют такие ситуации, когда нет никаких оснований предполагать нормальное распределение зависимой переменной. В главе 12 описаны аналитические методы, которые хорошо работают в тех случаях, когда данные происходят из смешанных или неизвестных распределений, когда объем выборок невелик, когда выбросы представляют собой проблему или когда разработка подходящего теста на основании теоретического распределения затруднена с математической точки зрения. Эти методы основаны на использовании повторных выборок и бутстреп-анализа – подходов, требующих интенсивных компьютерных вычислений и широко представленных в R. Описанные в этой главе методы позволят вам разрабатывать тесты для проверки гипотез в отношении данных, которые не могут быть проанализированы при помощи параметрических методов.

После прочтения третьей части вы научитесь решать большинство аналитических задач, встречающихся на практике. И вы сможете создавать некоторые эффектные диаграммы!



ГЛАВА 8.

Регрессия

В этой главе:

- Создание и интерпретация линейных моделей.
- Оценка адекватности допущений, сделанных при построении модели.
- Выбор между альтернативными моделями.

Во многих отношениях регрессионный анализ лежит в основе статистической обработки данных. Это сборное название для набора методов, используемых для предсказания переменной-отклика (также называемой зависимой, результирующей или условной переменной) по значениям одной или более предсказывающих переменных (также называемых независимыми, или объясняющими). В общем, регрессионный анализ можно использовать для *обнаружения* независимых переменных, которые имеют отношение к зависимой, для *описания* типа взаимосвязи и для составления уравнения, позволяющего *предсказать* значения зависимой переменной по значениям независимых.

Например, тренер по фитнесу может использовать регрессионный анализ, чтобы разработать уравнение, которое позволит предсказать ожидаемое число израсходованных калорий при занятиях на беговой дорожки. Зависимая переменная – это число сожженных калорий (вычисленное по объему потребленного кислорода), а к независимым переменным могут быть отнесены длительность упражнений (в минутах), доля времени, когда пульс держался на нужной частоте, средняя скорость (миль в час), возраст (в годах), пол и индекс массы тела.

С точки зрения теоретика, регрессионный анализ поможет найти ответ на вопросы вроде этих:

- какова связь между продолжительностью упражнений и числом сожженных калорий? Она линейная или нелинейная? Например, правда ли, что упражнение оказывает меньшее влияние на число потраченных калорий, если это число превышает некоторое пороговое значение?
- какова роль затраченных усилий (доли времени с нужной частотой пульса, средней скорости ходьбы)?
- одинаковы ли обнаруженные взаимосвязи для молодых и пожилых, мужчин и женщин, толстых и худых?

С точки зрения практика, регрессионный анализ поможет найти ответ на вопросы вроде этих:

- Сколько калорий может надеяться израсходовать 30-летний мужчина с индексом массы тела, равным 28,7, если он будет идти 45 минут со скоростью 4 мили в час, удерживая пульс на нужной частоте в течение 80% времени?
- Какое минимальное число переменных нужно использовать, чтобы точно предсказать число калорий, которое человек израсходует во время ходьбы?
- Насколько точными будут такие прогнозы?

Поскольку регрессионный анализ играет важную роль в современной статистической обработке данных, в этой главе мы рассмотрим его достаточно детально. Сначала мы узнаем, как создавать и интерпретировать регрессионные модели. Затем рассмотрим разные способы обнаружения возможных недостатков этих моделей и узнаем, что делать с ними. В-третьих, мы разберем вопрос о выборе переменных. Как можно решить, какие независимые переменные из доступных стоит включить в окончательную модель? В-четвертых, мы обратимся к проблеме обобщения. Насколько хорошо будет работать ваша модель в реальном мире? Наконец, мы затронем вопрос относительной важности переменных. Какие независимые переменные, включенные в вашу модель, самые важные, находятся на втором месте по важности, наименее важные?

Как вы можете видеть, мы рассматриваем широкий спектр проблем. Эффективный регрессионный анализ – это интерактивный целистный многошаговый процесс, для которого требуется больше, чем немного умения. Вместо того чтобы разбить изложение на несколько глав, я предпочел представить эту тему в одной главе, чтобы вы почувствовали все своеобразие регрессионного анализа. В результате получилась самая длинная и сложная глава во всей книге. Продол-

жайте усердно работать с ней до самого конца, и вы овладеете всеми средствами, необходимыми для того, чтобы справиться с различными исследовательскими задачами. Обещаю!

8.1. Многоликая регрессия

Термин «регрессия» может вызвать путаницу, поскольку существует множество ее разновидностей (табл. 8.1). Кроме того, в R реализован большой арсенал мощных методов подбора регрессионных моделей, и обилие возможностей также может вводить в заблуждение. Например, в 2005 году Вито Риччи (Vito Ricci) составил список из 205 функций, которые используются для регрессионного анализа в R (<http://cran.r-project.org/doc/contrib/Ricci-refcard-regression.pdf>).

Таблица 8.1. Разновидности регрессионного анализа

Тип регрессии	Для чего обычно используется
Простая линейная	Предсказание значений количественной зависимой переменной по значениям одной количественной независимой переменной
Полиномиальная	Предсказание значений количественной зависимой переменной по значениям количественной независимой переменной, когда взаимосвязь моделируется как полином n-ой степени
Множественная линейная	Предсказание значений количественной зависимой переменной по значениям двух и более количественных независимых переменных
Многомерная	Предсказание значений более чем одной зависимой переменной по значениям одной и более независимых переменных
Логистическая	Предсказание значений категориальной зависимой переменной по значениям одной и более независимых переменных
Пуассона	Предсказание значений зависимой счетной переменной по значениям одной или более независимых переменных
Пропорциональных рисков Кокса	Предсказание времени до наступления события (смерти, аварии, рецидива) по значениям одной или более независимых переменных
Временных рядов	Моделирование временных рядов с коррелированными ошибками

Тип регрессии	Для чего обычно используется
Нелинейная	Предсказание значений количественной зависимой переменной по значениям одной и более независимых переменных с использованием нелинейной модели
Непараметрическая	Предсказание значений количественной зависимости переменной по значениям одной и более независимых переменных с использованием полученной из данных и незаданной заранее модели
Устойчивая	Предсказание значений количественной зависимости переменной по значениям одной и более независимых переменных с использованием метода, устойчивого к выбросам

В этой главе мы сосредоточимся на регрессионных методах, которые попадают в группу регрессии, основанной на методе наименьших квадратов (ordinary least squares, МНК), включая простую линейную, полиномиальную и множественную линейную регрессию. МНК-регрессия – это наиболее распространенная разновидность статистического анализа наши дни. Другие типы регрессии (включая логистическую и Пуассона) будут рассмотрены в главе 13.

8.1.1. Ситуации, в которых используется МНК-регрессия

В случае МНК-регрессии значения количественной зависимой переменной предсказываются на основании взвешенной суммы значений независимых переменных, где веса переменных оцениваются, исходя из данных. Давайте взглянем на конкретный пример, взятый из публикации Fwa (2006) с упрощениями.

Инженеру нужно выявить наиболее важные факторы, влияющие на износ мостов (такие как срок службы, интенсивность движения, тип конструкции, использованные материалы и методы, качество постройки и погодные условия), и вывести математическую формулу, которая описывает эти взаимодействия. Инженер собирает все необходимые сведения для презентативной выборки мостов и моделирует данные при помощи МНК-регрессии.

Необходимо использовать в высшей степени интерактивный подход. Инженер подбирает разные модели, проверяет их соответствие статистическим допущениям, на которых основаны эти модели, исследует все неожиданные или отклоняющиеся от нормы факты и, на-

конец, выбирает «лучшую» модель из многих возможных. В случае успеха результаты помогут инженеру:

- сосредоточиться на важных переменных, определив, какие из многих изученных признаков полезны для предсказания износа мостов и какие из них наиболее важны;
- искать мосты, находящиеся, скорее всего, в плохом состоянии при помощи уравнения, которое может использоваться для оценки степени износа мостов в новых ситуациях (когда значения независимых переменных известны, а степень изношенности моста – нет);
- использовать преимущества интуитивной прозорливости, выявляя необычные мосты. Если инженер обнаружит, что некоторые мосты изнашиваются быстрее или медленнее, чем ожидалось, исходя из модели, исследования таких «выбросов» могут привести к важным открытиям, которые помогут понять механизмы износа мостов.

Мосты могут быть вам неинтересны. Я занимаюсь клинической психологией и статистикой, а о строительстве не знаю почти ничего. Однако основные принципы анализа применимы к удивительно широкому спектру задач в психологии, биологии и социологии. На каждый из следующих вопросов также можно найти ответ при помощи МНК-регрессионного анализа:

- Какова связь между засоленностью поверхностного стока и площадью мощенных дорог (Montgomery, 2007)?
- Какие аспекты жизненного опыта игроков обусловливают чрезмерное увлечение массовыми ролевыми сетевыми играми (Hsu, Wen, & Wu, 2009)?
- Какие характеристики образовательной среды наиболее сильно связаны с повышенной успеваемостью студентов?
- Какой вид имеет взаимосвязь между артериальным давлением, количеством потребляемой соли и возрастом?
- Каков вклад стадионов и профессиональных занятий спортом в увеличение площади мегаполисов (Baade & Dye, 1990)?
- Какие факторы влияют на различия в цене пива в разных штатах (Culbertson & Bradford, 1991)? (Этот вопрос определенно привлек ваше внимание!)

Решающие этапы – это наша способность сформулировать интересный вопрос, решить, какую зависимую переменную исследовать, и собрать нужные данные.

8.1.2. Что вам нужно знать

В оставшейся части этой главы я буду описывать функции, которые используются в R для подбора МНК-регрессионной модели, оценки ее соответствия данным, проверки предположений, на которых построена модель, и выбора наилучшей модели из имеющихся. Предполагается, что у читателя есть представления о МНК-регрессии на студенческом уровне. Однако я постарался свести число математических обозначений к минимуму и сосредоточиться на практических, а не теоретических вещах. Существует ряд замечательных пособий, которые посвящены вошедшему в эту главу статистическим методам. Мои любимые – это «Прикладной регрессионный анализ» Джона Фокса (John Fox's Applied Regression Analysis and Generalized Linear Models) – в том, что касается теории, – и «Справочник по прикладному регрессионному анализу в R и S-Plus» (An R and S-Plus Companion to Applied Regression) – в том, что касается практики. Эти два пособия послужили основным источником информации для данной главы. Хороший нетехнический обзор представлен Лихтом (Licht, 1995).

8.2. МНК-регрессия

На протяжении большей части этой главы мы будем предсказывать значения зависимой переменной по набору независимых (это также называют «регрессией» зависимой переменной по независимой – отсюда и название метода) с использованием МНК. МНК-регрессия позволяет подгонять модели вида

$$\widehat{Y}_i = \widehat{\beta}_0 + \widehat{\beta}_1 X_{1i} + \dots + \widehat{\beta}_k X_{ki} \quad i = 1 \dots n,$$

где n – это число наблюдений, а k – это число независимых переменных. (Хотя я старался свести число уравнений к минимуму, это одно из тех мест, где они упрощают изложение.) В этом уравнении:

\widehat{Y}_i	Предсказанное значение зависимой переменной для i -го наблюдения (а именно оценка среднего значения распределения Y по набору независимых переменных)
X_{ki}	Значение k -ой независимой переменной для i -го наблюдения
$\widehat{\beta}_0$	Свободный член уравнения (предсказанное значение Y при нулевом значении всех независимых переменных)
$\widehat{\beta}_k$	Регрессионный коэффициент для k -ой независимой переменной (угол наклона для прямой, которая отражает изменение Y при изменении X на одну единицу измерения)

Наша цель – это выбрать такие параметры модели (свободный член и регрессионные коэффициенты), которые позволят минимизировать различия между реальными и предсказанными значениями зависимой переменной. То есть мы выбираем такие параметры модели, чтобы сумма квадратов остатков была минимальной:

$$\sum_1^n \left(Y_i - \widehat{Y}_i \right)^2 = \sum_1^n \left(Y_i - \widehat{\beta}_0 + \widehat{\beta}_1 X_{1i} + \dots + \widehat{\beta}_k X_{ki} \right)^2 = \sum_1^n \varepsilon^2$$

Для правильной интерпретации коэффициентов МНК-модели нужно, чтобы ваши данные удовлетворяли ряду требований:

- *нормальность* – значения зависимой переменной нормально распределены при фиксированных значениях независимых переменных;
- *независимость* – значения Y_i независимы друг от друга;
- *линейность* – зависимая переменная линейно связана с независимыми;
- *гомоскедастичность* – дисперсия зависимой переменной постоянна при разных значениях независимых переменных. Мы могли бы назвать это свойство «однородность дисперсии», но употребление термина «гомоскедастичность» позволяет мне чувствовать себя умнее.

Если эти требования нарушаются, то тесты значимости и доверительные интервалы могут быть вычислены неточно. Также подразумевается, что независимые переменные определяются и измеряются без ошибок, однако это требование обычно игнорируется на практике.

8.2. 1. Подгонка регрессионных моделей при помощи команды `lm()`

В R основная функция для подгонки регрессионных моделей – это `lm()`. Формат ее применения таков:

```
myfit <- lm(formula, data)
```

где `formula` описывает вид модели, которую нужно подогнать, а `data` – это таблица с данными, которые используются для создания модели. Полученный объект (`myfit` в данном случае) – это список, содержащий обширную информацию о подогнанной модели. Формула обычно записывается в таком виде:

$Y \sim X_1 + X_2 + \dots + X_k$

где \sim отделяет зависимую переменную слева от независимых переменных (разделенных знаками $+$) справа. Для различных изменений этой формулы можно использовать другие символы (табл. 8.2).

Таблица 8.2. Символы, которые часто используются в формулах R

Символ	Назначение
\sim	Отделяет зависимые переменные (слева) от независимых (справа). Например, предсказание значений y по значениям x , z и w будет закодировано так: $y \sim x + z + w$
$+$	Разделяет независимые переменные
$:$	Обозначает взаимодействие между независимыми переменными. Предсказание значений y по значениям x , z и взаимодействия между x и z будет закодировано как $y \sim x + z + x:z$
$*$	Краткое обозначение для всех возможных взаимодействий. Код $y \sim x * z * w$ в полном виде означает $y \sim x + z + w + x:z + x:w + z:w + x:z:w$
*	Обозначает взаимодействия до определенного порядка. Код $y \sim (x + z + w)^2$ в полном виде будет записан как $y \sim x + z + w + x:z + x:w + z:w$
$.$	Символ-заполнитель для всех переменных в таблице данных, кроме зависимой. Например, если таблица данных содержит переменные x , y , z и w , то код $y \sim .$ будет означать $y \sim x + z + w$
$-$	Знак минуса удаляет переменную из уравнения. Например, $y \sim (x + z + w)^2 - x:w$ соответствует $y \sim x + z + w + x:z + z:w$
-1	Подавляет свободный член уравнения. Например, формула $y \sim x - 1$ позволяет подогнать такую регрессионную модель для предсказания значений y по x , чтобы ее график проходил через начало координат
$I()$	Элемент в скобках интерпретируется как арифметическое выражение. Например, $y \sim x + (z + w)^2$ означает $y \sim x + z + w + z:w$. Для сравнения $y \sim x + I((z + w)^2)$ означает $y \sim x + h$, где h – это новая переменная, полученная при возведении в квадрат суммы z и w
<i>function</i>	В формулах можно использовать математические функции. Например, $\log(y) \sim x + z + w$ будет предсказывать значения $\log(y)$ по значениям x , z и w

В дополнение к `lm()` в табл. 8.3 перечислены несколько полезных функций для выполнения простого или множественного регрессионного анализа. Каждая из них применяется к объекту, созданному при помощи функции `lm()`, чтобы получить дополнительную информацию о подогнанной модели.

Таблица 8.3. Другие функции, полезные при подгонке линейных моделей

Функция	Действие
summary ()	Показывает детальную информацию о подогнанной модели
coefficients ()	Перечисляет параметры модели (свободный член и регрессионные коэффициенты)
confint ()	Вычисляет доверительные интервалы для параметров модели (по умолчанию 95%)
fitted ()	Выводит на экран предсказанные значения, согласно подогнанной модели
residuals ()	Показывает остатки для подогнанной модели
anova ()	Создает таблицу ANOVA (дисперсионного анализа) для подогнанной модели или таблицу ANOVA, сравнивающую две или более моделей
vcov ()	Выводит ковариационную матрицу для параметров модели
AIC ()	Вычисляет информационный критерий Акаике (Akaike's Information Criterion)
plot ()	Создает диагностические диаграммы для оценки адекватности модели
predict ()	Использует подогнанную модель для предсказания зависимой переменной для нового набора данных

Когда в регрессионной модели есть одна зависимая и одна независимая переменная, такой подход называется простой линейной регрессией. Когда есть одна зависимая переменная, но в модель входят ее степени (например, X, X^2, X^3), это называется полиномиальной регрессией. Если есть больше одной независимой переменной, мы называем это множественной регрессией. Мы начнем с примера простой линейной регрессии, затем перейдем к примерам полиномиальной и множественной линейной регрессии и закончим примером множественной регрессии с взаимодействием между независимыми переменными.

8.2.2. Простая линейная регрессия

Рассмотрим, как используются функции, перечисленные в табл. 8.3, на простом примере. Набор данных `women`, поставляемый с базовой версией программы, содержит данные о росте и весе 15 женщин в возрасте от 30 до 39 лет. Мы хотим предсказать вес по значениям рос-

та, что поможет выявить женщин с чрезмерным или недостаточным весом. Регрессионный анализ проводится при помощи следующего программного кода (полученная диаграмма приведена на рис. 8.1):

Программный код 8.1. Простая линейная регрессия

```
> fit <- lm(weight ~ height, data=women)
> summary(fit)
Call:
lm(formula=weight ~ height, data=women)
Residuals:
    Min      1Q Median      3Q      Max 
-1.733 -1.133 -0.383  0.742  3.117 
Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) -87.5167    5.9369  -14.7  1.7e-09 ***
height       3.4500    0.0911   37.9  1.1e-14 ***  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Residual standard error: 1.53 on 13 degrees of freedom
Multiple R-squared:  0.991,    Adjusted R-squared:  0.99 
F-statistic: 1.43e+03 on 1 and 13 DF,  p-value: 1.09e-14
> women$weight
[1] 115 117 120 123 126 129 132 135 139 142 146 146 150 150 154 159 164
> fitted(fit)
     1      2      3      4      5      6      7      8      9
112.58 116.03 119.48 122.93 126.38 129.83 133.28 136.73 140.18
     10     11     12     13     14     15
143.63 147.08 150.53 153.98 157.43 160.88
> residuals(fit)
     1      2      3      4      5      6      7      8      9      10     11
2.42  0.97  0.52  0.07 -0.38 -0.83 -1.28 -1.73 -1.18 -1.63 -1.08
     12     13     14     15
-0.53  0.02  1.57  3.12
> plot(women$height,women$weight,
       xlab="Рост (дюймы)",
       ylab="Вес (фунты)")
> abline(fit)
```

Из полученных результатов следует, что уравнение для предсказания веса по росту имеет следующий вид:

$$\widehat{\text{Weight}} = -87.52 + 3.45 \times \text{Height}.$$

Поскольку нулевой рост невозможен, мы не будем пытаться интерпретировать свободный член. В данном случае это просто поправочная константа. Из столбца $\text{Pr}(>|t|)$ ясно, что коэффициент регрессии (3.45) статистически значимо отличается от нуля ($p < 0.001$)

и означает, что на каждый дюйм¹ роста ожидается увеличение веса на 3.45 фунта². Множественный коэффициент детерминации (0.991) означает, что модель объясняет 99.1% дисперсии значений веса. Этот коэффициент представляет собой квадрат коэффициента корреляции между реальными и предсказанными значениями³. Стандартную ошибку остатков (1.53 фунта) можно интерпретировать как усредненную ошибку предсказания веса по росту с использованием данной модели. F-значение позволяет проверить, предсказывают ли независимые переменные значения зависимой лучше, чем по случайности.

В иллюстративных целях мы вывели на экран реальные и предсказанные значения, а также остатки (разность между предсказанными и реальными значениями). Очевидно, что наибольшие остатки характерны для самых низких и высоких женщин, что также можно видеть на диаграмме (рис. 8.1).

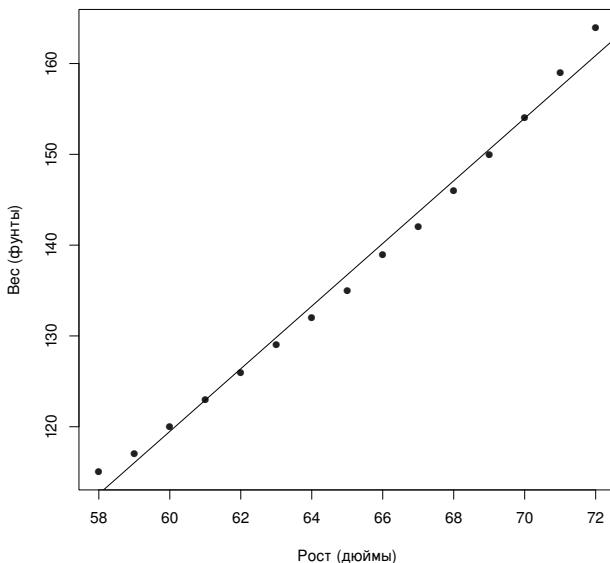


Рис. 8.1. Диаграмма рассеяния с регрессионной прямой для значений веса, предсказанных по значениям роста

На диаграмме заметно, что можно улучшить предсказание, используя линию с одним изгибом. Например, модель вида

1 2.5 см. – Прим. пер.

2 Почти 1.6 кг. – Прим. пер.

3 Такая интерпретация множественного коэффициента детерминации справедлива только для простой линейной регрессии. – Прим. пер.

$$\widehat{Y} = \beta_0 + \beta_1 X + \beta_2 X^2$$

может лучше соответствовать данным. Полиномиальная регрессия позволяет предсказывать зависимую переменную по независимой, если эта связь имеет вид полинома n -ой степени.

8.2.3. Полиномиальная регрессия

Из диаграммы на рис. 8.1 следует, что точность предсказания можно улучшить, если использовать квадратичное регрессионное уравнение (то есть включить в него X^2).

Подогнать модель в виде квадратичного уравнения можно при помощи такого выражения:

```
fit2 <- lm(weight ~ height + I(height^2), data=women)
```

Тут нужно объяснить новое обозначение: `I(height^2)`. `height^2` добавляет в уравнение возведенный в квадрат рост. Функция `I` интерпретирует содержимое скобок как регулярное выражение R. Это нужно, поскольку оператор `^` имеет в формулах особое значение (см. табл. 8.2), которое нужно подавить в данном случае.

В программном коде 8.2 приведены результаты подгонки квадратичного уравнения.

Программный код 8.2. Полиномиальная регрессия

```
> fit2 <- lm(weight ~ height + I(height^2), data=women)
> summary(fit2)

Call:
lm(formula=weight ~ height + I(height^2), data=women)

Residuals:
    Min      1Q  Median      3Q     Max 
-0.5094 -0.2961 -0.0094  0.2862  0.5971 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 261.87818   25.19677   10.39 2.4e-07 ***
height       -7.34832    0.77769   -9.45 6.6e-07 *** 
I(height^2)   0.08306    0.00598   13.89 9.3e-09 *** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.384 on 12 degrees of freedom
Multiple R-squared:  0.999,          Adjusted R-squared:  0.999
```

```
F-statistic: 1.14e+04 on 2 and 12 DF, p-value: <2e-16
```

```
> plot(women$height,women$weight,  
       xlab="Рост (дюймы)",  
       ylab="Вес (фунты)")  
> lines(women$height,fitted(fit2))
```

По результатам этого нового анализа регрессионное уравнение приобретает вид

$$\widehat{\text{Weight}} = 261.88 + 7.35 \times \text{Height} + 0.083 \times \text{Height}^2$$

и оба регрессионных коэффициента оказываются значимыми на уровне $p < 0.0001$. Доля объясненной дисперсии в данном случае повысилась до 99.9%. Значимость квадратного члена ($t = 13.89, p < .001$) указывает на то, что его включение в модель улучшило ее адекватность. Если вы посмотрите на диаграмму для второй модели (рис. 8.2), то увидите, что кривая лучше описывает реальные данные.

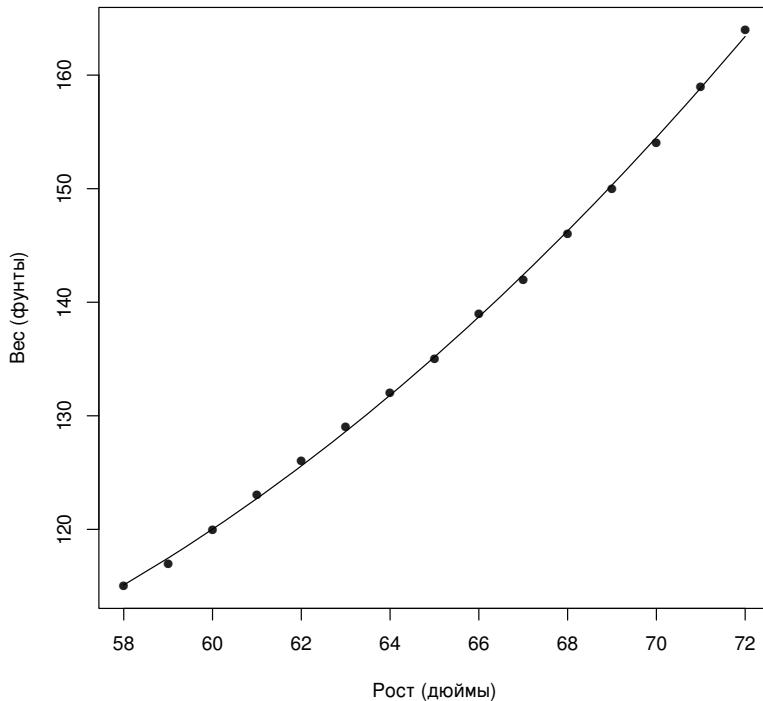


Рис. 8.2. Квадратичная регрессия для предсказаний значений веса по значениям роста

Линейные модели в сравнении с нелинейными

Отметим, что это полиномиальное уравнение по-прежнему попадает в раздел линейной регрессии. Эта модель линейная, поскольку в уравнение входит взвешенная сумма независимых переменных (в данном случае рост и рост в квадрате). Даже модель вроде

$$\widehat{Y}_i = \widehat{\beta}_0 \times \log X_1 + \widehat{\beta}_2 \times \sin X_2$$

считается линейной (исходя из ее параметров) и описывается формулой

$$Y \sim \log(X1) + \sin(X2).$$

А вот пример настоящей нелинейной модели:

$$\widehat{Y}_i = \beta_0 + \widehat{\beta}_1 e^{\frac{X}{\beta^2}}.$$

Нелинейные модели этого вида могут быть подогнаны при помощи функции `nls()`.

В общем случае полиномиальная функция n -ой степени соответствует кривой с $n - 1$ изгибами. Для подгонки модели кубической функции нужно использовать выражение

```
fit3 <- lm(weight ~ height + I(height^2) + I(height^3), data=women)
```

Хотя полиномы более высоких степеней также возможны, я редко сталкивался с тем, чтобы члены уравнения в степени выше третьей оказывались необходимыми.

Прежде чем мы двинемся дальше, я должен упомянуть, что функция `scatterplot()` из пакета `car` – это простой и удобный метод изображения взаимосвязи между двумя переменными. Программный код

```
library(car)
scatterplot(weight ~ height, data=women,
            spread=FALSE, lty.smooth=2, pch=19,
            main="Женщины в возрасте 30-39 лет",
            xlab="Рост (дюймы)",
            ylab="Вес (фунты)")
```

позволяет получить диаграмму, представленную на рис. 8.3.

Эта усовершенствованная диаграмма представляет собой диаграмму рассеяния для веса и роста, диаграммы размахов для обоих переменных на соответствующих полях диаграммы, а также регрессионные прямую и слаженную (`loess`) кривую. Опция `spread=FALSE`

предотвращает вывод информации о разбросе и асимметрии. Опция `lty.smooth=2` задает вид сглаженной кривой (пунктир). Параметр `pch=19` определяет способ изображения точек в виде заполненных кружков (по умолчанию пустые кружки). Сразу можно сказать, что эти две переменные распределены почти симметрично и что регрессионная кривая лучше соответствует данным, чем регрессионная прямая.

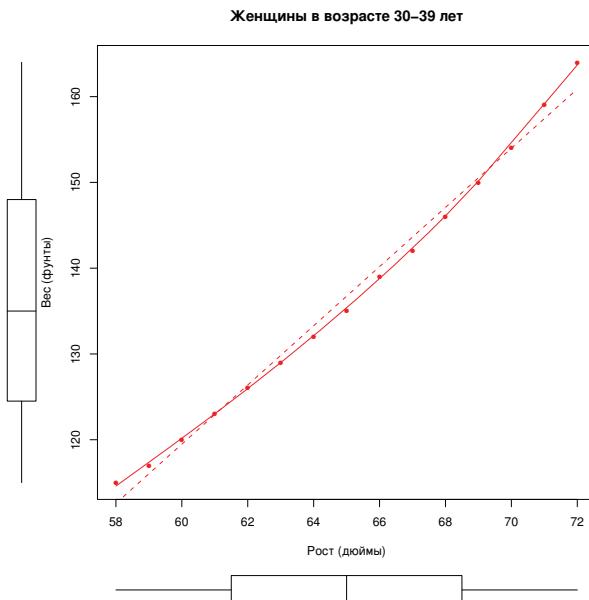


Рис. 8.3. Диаграмма рассеяния для значений роста и веса с регрессионной прямой и диаграммами размахов на полях

8.2.4. Множественная линейная регрессия

Если существует больше одной независимой переменной, простая линейная регрессия превращается во множественную линейную регрессию, а ход вычислений становится более сложным. С технической точки зрения, полиномиальная регрессия – это частный случай множественной регрессии. При квадратичной регрессии есть две независимые переменные (X и X^2), а при кубической регрессии – три независимые переменные (X, X^2, X^3). Рассмотрим более общий пример.

Для этого примера мы используем набор данных `state.x77`, который поставляется с базовой версией программы. Мы хотим исследовать связь между уровнем преступности и другими характеристиками для каждого штата, включая численность населения, уровень неграмотности, средний доход и морозность (среднее число дней с отрицательной температурой).

Поскольку функция `lm()` работает с таблицей данных (а набор данных `state.x77` хранится в виде матрицы), можно упростить свою жизнь при помощи следующего программного кода:

```
states <- as.data.frame(state.x77[,c("Murder", "Population",
    "Illiteracy", "Income", "Frost")])
```

Этот код создает таблицу данных, которая называется `states` и содержит интересующие нас переменные. Мы будем использовать данную таблицу на протяжении всей главы.

Важный первый шаг в множественной регрессии – исследование парных взаимосвязей между переменными. Двухмерные корреляции вычисляются при помощи функции `cor()`, а диаграммы рассеяния создаются при помощи функции `scatterplotMatrix()`⁴ из пакета `car` (см. следующий программный код и рис. 8.4).

Программный код 8.3. Исследование парных взаимосвязей

```
> cor(states)
   Murder Population Illiteracy Income Frost
Murder    1.00      0.34     0.70 -0.23 -0.54
Population 0.34      1.00     0.11  0.21 -0.33
Illiteracy 0.70      0.11     1.00 -0.44 -0.67
Income    -0.23      0.21    -0.44  1.00  0.23
Frost     -0.54     -0.33    -0.67  0.23  1.00

> library(car)
> scatterplotMatrix(states, spread=FALSE, lty.smooth=2,
    main="Матрица диаграмм рассеяния")
```

По умолчанию функция создает диаграммы рассеяния для всех пар переменных с наложенными сглаженной (loess) кривой и регрессионной прямой. На главной диагонали представлены диаграммы плотности и графики-щетки для каждой переменной.

Можно видеть, что уровень преступности имеет бимодальное распределение, а распределение каждой независимой переменной в той или иной степени асимметрично. Уровень преступности растет вмес-

⁴ В первых версиях пакета `car` эта функция называется `scatterplot.matrix()`. – *Прим. пер.*

те с ростом населения и уровнем неграмотности, а падает – с увеличением дохода и морозности. В то же время более холодные штаты характеризуются более низким уровнем неграмотности и численности населения, а также высокими доходами.

Матрица диаграмм рассеяния

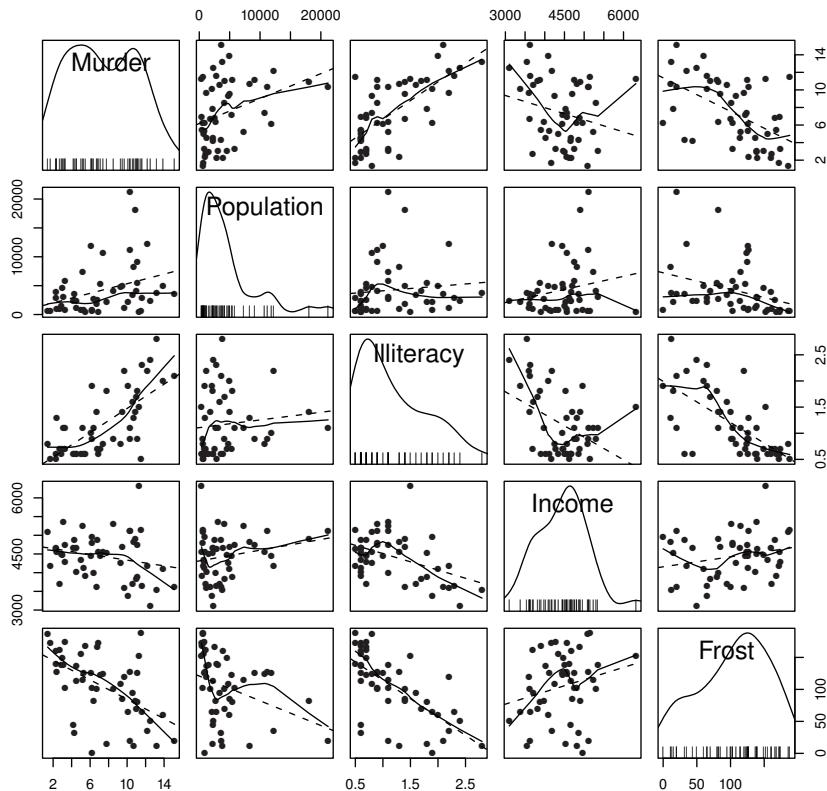


Рис. 8.4. Матрица диаграмм рассеяния значений зависимой и независимых переменных для данных о штатах, включающая регрессионные прямые и сглаживающие линии, а также распределения переменных (диаграмма ядерной оценки функции плотности и график-щетка)

Подгоним теперь множественную регрессионную модель при помощи функции `lm()` (см. следующий программный код).

Программный код 8.4. Множественная линейная регрессия

```
> fit <- lm(Murder ~ Population + Illiteracy + Income + Frost,
+            data=states)
> summary(fit)

Call:
lm(formula=Murder ~ Population + Illiteracy + Income + Frost,
    data=states)

Residuals:
    Min      1Q  Median      3Q     Max 
-4.7960 -1.6495 -0.0811  1.4815  7.6210 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 1.23e+00  3.87e+00   0.32    0.751    
Population  2.24e-04  9.05e-05   2.47    0.017 *  
Illiteracy  4.14e+00  8.74e-01   4.74   2.2e-05 *** 
Income      6.44e-05  6.84e-04   0.09    0.925    
Frost       5.81e-04  1.01e-02   0.06    0.954    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 'v' 1 

Residual standard error: 2.5 on 45 degrees of freedom
Multiple R-squared:  0.567,    Adjusted R-squared:  0.528 
F-statistic: 14.7 on 4 and 45 DF,  p-value: 9.13e-08
```

Если существует больше одной независимой переменной, то регрессионные коэффициенты показывают, на сколько увеличится значение зависимой переменной при изменении данной независимой переменной на единицу при условии, что все остальные независимые переменные останутся неизменными. Например, регрессионный коэффициент для уровня неграмотности равен 4.14. Это свидетельствует о том, что увеличение неграмотности на 1% связано с увеличением уровня преступности на 4.14% при постоянных значениях численности населения, дохода и морозности. Этот коэффициент статистически значимо отличается от нуля при $p < 0.0001$. С другой стороны, регрессионный коэффициент для морозности статистически не отличается от нуля. Это говорит о том, что морозность и уровень преступности не связаны между собой линейно при неизменных значениях всех прочих переменных. Взятые вместе, независимые переменные могут объяснить 57% дисперсии в уровне преступности по штатам.

До этого времени мы предполагали, что независимые переменные не взаимодействуют между собой. В следующем разделе мы рассмотрим пример, когда это не так.

8.2.5. Множественная линейная регрессия со взаимодействиями

Некоторые из наиболее интересных результатов исследований касаются взаимодействий между независимыми переменными. Рассмотрим данные об автомобилях из таблицы `mtcars`. Допустим, вы интересуетесь влиянием веса автомобиля и мощности двигателя на расход топлива. Можно подобрать регрессионную модель, включающую обе независимые переменные, а также взаимодействие между ними, как это сделано в следующем программном коде.

Программный код 8.5. Множественная линейная регрессия со значимым эффектом взаимодействия

```
> fit <- lm(mpg ~ hp + wt + hp:wt, data=mtcars)
> summary(fit)

Call:
lm(formula=mpg ~ hp + wt + hp:wt, data=mtcars)

Residuals:
    Min      1Q  Median      3Q     Max 
-3.063 -1.649 -0.736  1.421  4.551 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 49.80842   3.60516   13.82 5.0e-14 ***
hp          -0.12010   0.02470   -4.86 4.0e-05 ***
wt          -8.21662   1.26971   -6.47 5.2e-07 ***
hp:wt        0.02785   0.00742    3.75  0.00081 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 2.1 on 28 degrees of freedom
Multiple R-squared:  0.885,    Adjusted R-squared:  0.872 
F-statistic: 71.7 on 3 and 28 DF,  p-value: 2.98e-13
```

Из столбца `Pr(>|t|)` видно, что взаимодействие между мощностью двигателя и весом машины значимо. Что это означает? Значимое взаимодействие между двумя независимыми переменными свидетельствует о том, что на взаимосвязь между одной независимой переменной и зависимой влияют значения другой независимой переменной. В данном случае характер зависимости между расходом топлива и мощностью двигателя не одинаков для автомобилей разного веса.

Наша модель для предсказания значений `mpg` такова:

$$\widehat{\text{mpg}} = 49.81 - 0.12 \times \text{hp} - 8.22 \times \text{wt} + 0.03 \times \text{hp} \times \text{wt}.$$

Чтобы интерпретировать взаимодействие, можно подставлять разные значения wt , тем самым упрощая уравнение. Например, можно попробовать взять среднее значение wt (3.2) и значения на одно стандартное отклонение меньше и больше среднего (2.2 и 4.2 соответственно).

Для $\text{wt}=2.2$ уравнение принимает вид:

$$\widehat{\text{mpg}} = 49.81 - 0.12 \times \text{hp} - 8.22 \times (2.2) + 0.03 \times \text{hp} \times (2.2) = 31.41 - 0.06 \times \text{hp}.$$

Для $\text{wt}=3.2$ оно становится таким:

$$\widehat{\text{mpg}} = 23.37 - 0.03 \times \text{hp}.$$

Наконец, для $\text{wt}=4.2$ выражение приобретает вид:

$$\widehat{\text{mpg}} = 15.33 - 0.003 \times \text{hp}.$$

Видно, что с увеличением веса (2.2, 3.2, 4.2) ожидаемое изменение mpg на единицу изменения hp уменьшается (0.06, 0.03, 0.003).

Взаимодействия можно визуализировать при помощи функции `effect()` из одноименного пакета. Формат ее применения таков:

```
plot(effect(term, mod, xlevels), multiline=TRUE)
```

где `term` – это член модели, который нужно отобразить на диаграмме, `mod` – подогнанная модель, выдаваемая функцией `lm()`, а `xlevels` – это список переменных, значения которых будут фиксированы, и самих этих значений. Опция `multiline=TRUE` позволяет наложить на диаграмму линии. Для нашего последнего примера это выглядит так:

```
library(effects)
plot(effect("hp:wt", fit, list(wt=c(2.2,3.2,4.2))), multiline=TRUE)
```

Полученная диаграмма представлена на рис. 8.5.

Из диаграммы видно, что с увеличением веса автомобиля взаимодействие между мощностью мотора и расходом топлива ослабевает. Для $\text{wt}=4.2$ линия почти горизонтальная, что говорит о незначительном изменении значений mpg при увеличении hp .

К сожалению, подгонка модели – это только первый шаг анализа. Прежде чем с уверенностью делать заключения после этого шага, нужно оценить, соответствуют ли ваши данные статистическим ограничениям примененного метода. Это тема следующего раздела.

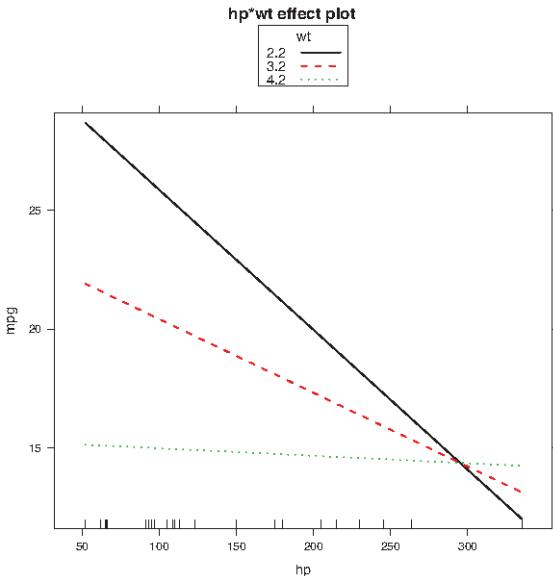


Рис. 8.5. Диаграмма взаимодействий для hp^*wt .
Показана взаимосвязь между mpg и hp для трех значений wt

8.3. Диагностика регрессионных моделей

В предыдущем разделе вы использовали функцию `lm()` для подгонки МНК-модели и функцию `summary()`, чтобы получить параметры этой модели и вычислить характеризующие ее общие статистики. К сожалению, ничего из этого не говорит вам, что вы подобрали подходящую модель. Степень уверенности в параметрах регрессии зависит от того, насколько модель удовлетворяет ограничениям МНК. Хотя функция `summary()`, приведенная в программном коде 8.4, описывает модель, она ничего не сообщает о том, насколько ваша модель соответствует лежащим в ее основе статистическим ограничениям.

Почему это важно? Ошибки в данных или неверно определенные взаимосвязи между зависимыми и независимой переменными могут привести к тому, что вы выберете в значительной степени неточную модель. С одной стороны, вы можете решить, что между зависимой и независимой переменными нет связи, в то время как на самом деле

она существует. С другой стороны, можно прийти к заключению, что зависимая и независимая переменная связаны, а на самом деле это будет не так! Вы также можете получить модель, которая, будучи примененной к реальным данным, делает неудачные предсказания со значительными и ненужными ошибками.

Посмотрим на результат применения функции `confint()` к набору данных `states` для решения задачи по множественной регрессии из раздела 8.2.4.

```
> fit <- lm(Murder ~ Population + Illiteracy + Income + Frost, data=states)
> confint(fit)
              2.5 %    97.5 %
(Intercept) -6.55e+00 9.021318
Population   4.14e-05 0.000406
Illiteracy   2.38e+00 5.903874
Income      -1.31e-03 0.001441
Frost        -1.97e-02 0.020830
```

Полученный результат свидетельствует о том, что вы можете быть на 95% уверенными в том, что интервал [2.38, 5.90] содержит реальное значение, на которое изменяется уровень преступности, при изменении уровня неграмотности на 1%. Кроме того, поскольку доверительный интервал для морозности (`Frost`) содержит ноль, можно заключить, что изменения этого параметра не связаны с уровнем преступности при постоянных значениях прочих переменных. Однако ваша уверенность в этих результатах сильна настолько, насколько выполняются статистические допущения, лежащие в основе модели.

Набор методов под названием «диагностика регрессионных моделей» предоставляет необходимые инструменты для оценки адекватности регрессионной модели и может помочь обнаружить и устраниить проблемы. Мы начнем со стандартного подхода с использованием функций, входящих в базовую версию программы. Затем мы рассмотрим более новые, усовершенствованные методы, реализованные в пакете `car`.

8.3.1. Стандартный подход

В базовой версии программы реализованы многочисленные методы проверки выполнения статистических допущений. Наиболее распространенный подход – применить функцию `plot()` к объекту, представляющему собой результат действия функции `lm()`. В результате появляются четыре диаграммы, полезные для оценки адекватности

модели. Применение этого подхода к примеру простой линейной регрессии

```
fit <- lm(weight ~ height, data=women)
par(mfrow=c(2,2))
plot(fit)
```

позволяет получить диаграммы, представленные на рис. 8.6. Выражение `par(mfrow=c(2, 2))` используется для размещения четырех диаграмм, создаваемых функцией `plot()`, на одной большой диаграмме 2×2 . Функция `par()` описана в главе 3.

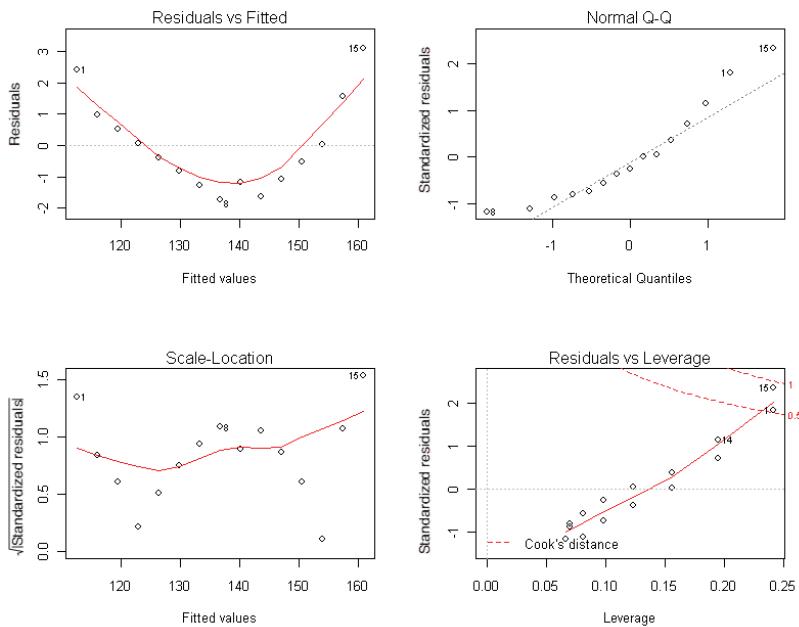


Рис. 8.6. Диагностические диаграммы для регрессии веса по росту

Чтобы понять эти диаграммы, нужно вспомнить допущения МНК-регressии:

- **Нормальность.** Если значения зависимой переменной нормально распределены при постоянных значениях независимых переменных, тогда остатки должны быть нормально распределены со средним значением 0. Графическая проверка данных на нормальность (Normal Q-Q plot – справа сверху) –

это построение графика распределения вероятностей, сопоставляющего стандартизованные остатки и значения, которые ожидаются при нормальном распределении. Если допущение о нормальном распределении выполняется, то точки на этой диаграмме должны ложиться на прямую с углом наклона в 45° . Поскольку здесь это не наблюдается, это допущение не выполняется.

- *Независимость.* Из этих диаграмм нельзя сказать, насколько значения прогнозируемой переменной независимы. Для этого нужно понимать, как были собраны данные. У нас нет никаких априорных оснований полагать, что вес одной женщины зависит от веса другой женщины. Если вы обнаружите, что данные были собраны для нескольких человек из одной семьи, то вам придется изменить свое предположение о независимости.
- *Линейность.* Если зависимая переменная линейно связана с независимой, то связь между остатками и предсказанными (то есть подогнанными) значениями отсутствует. Другими словами, модель должна отражать всю закономерную изменчивость в данных, учитывая все, кроме белого шума. На диаграмме зависимости остатков от предсказанных значений (сверху слева) вы ясно видите нелинейную зависимость, что позволяет задуматься о добавлении квадратного члена в уравнение регрессии.
- *Гомоскедастичность.* Если допущение о постоянной изменчивости выполняется, то точки на нижней левой диаграмме должны располагаться в форме полосы вокруг горизонтальной линии. Похоже, что это допущение выполняется.

Наконец, диаграмма зависимости остатков от «показателя напряженности» (англ. leverage) (слева внизу) содержит информацию о наблюдениях, на которые вам, возможно, следует обратить внимание. Диаграмма выявляет выбросы, точки высокой напряженности и влиятельные наблюдения. Если говорить более конкретно, то:

- *Выброс* – это значение, которое плохо предсказывается подобранной моделью (то есть имеет большой положительный или отрицательный остаток).
- Значение с высоким значением *напряженности* описывается необычной комбинацией независимых переменных. Таким образом, это выброс в пространстве независимых переменных. Значения зависимой переменной не используются при вычислении напряженности.

- *Влияльное наблюдение* – это значение, которое вносит не-пропорциональный вклад в расчет параметров модели. Влияльные наблюдения выявляются при помощи статистики, называемой расстоянием Кука (Cook's distance, Cook's D).

Честно говоря, диаграмма зависимости остатков от напряженности кажется мне сложной для восприятия и не очень полезной. В следующих разделах вы увидите, как можно более удачно представить эту информацию.

Чтобы завершить данный раздел, давайте взглянем на диагностические диаграммы для квадратичной регрессии. Необходимый программный код таков:

```
fit2 <- lm(weight ~ height + I(height^2), data=women)
par(mfrow=c(2,2))
plot(fit2)
```

Полученная диаграмма представлена на рис. 8.7.

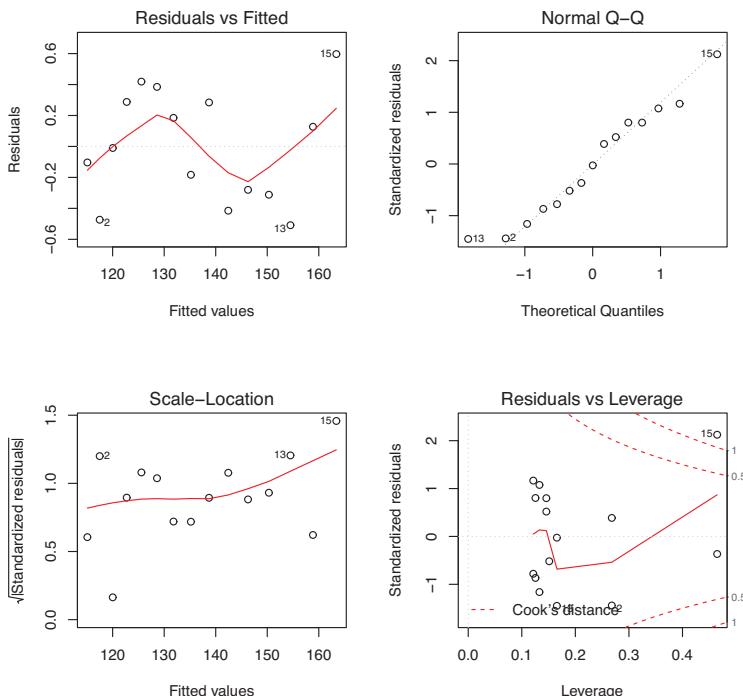


Рис. 8.7. Диагностические диаграммы для регрессии веса по росту и по росту, возведенному в квадрат

Этот второй набор диаграмм свидетельствует о том, что полиномиальная регрессия подходит лучше, поскольку учитывает требования линейности, нормального распределения остатков (за исключением наблюдения 13) и гомоскедастичности (постоянной дисперсии остатков). Наблюдение 15 можно отнести к влиятельным (на основе высокого значения расстояния Кука), его удаление повлияет на оценку параметров модели. Действительно, удаление наблюдений 13 и 15 приведет к лучшему соответствию модели данным. Чтобы убедиться в этом самостоятельно, попробуйте:

```
newfit <- lm(weight ~ height + I(height^2), data=women[-c(13,15),])
```

Однако нужно быть аккуратными с удалением данных. Модели должны соответствовать данным, а не наоборот!

Наконец, применим стандартный подход к решению задачи с множественной регрессией для набора данных `states`:

```
fit <- lm(Murder ~ Population + Illiteracy + Income + Frost, data=states)
par(mfrow=c(2, 2))
plot(fit)
```

Полученные результаты представлены на рис. 8.8. Как вы можете видеть на диаграмме, допущения, лежащие в основе модели, хорошо выполняются, за исключением Невады (выброса).

Хотя стандартные диагностические диаграммы полезны, сейчас в R доступны лучшие средства, и я рекомендую использовать их, а не команду `plot(модель)`.

8.3.2. Усовершенствованный подход

В пакете `car` реализован ряд функций, которые значительно расширяют ваши возможности подгонки и оценки регрессионных моделей (см. табл. 8.4).

Таблица 8.4. Полезные функции для диагностики регрессионных моделей (пакет `car`)

Функция	Назначение
<code>qqPlot()</code>	Диаграмма сравнения квантилей
<code>durbinWatsonTest()</code>	Тест Дарбина-Уотсона (Durbin-Watson test) на автокорреляцию в остатках
<code>crPlots()</code>	Диаграмма компонент и остатков
<code>ncvTest()</code>	Тест на неоднородность дисперсии остатков

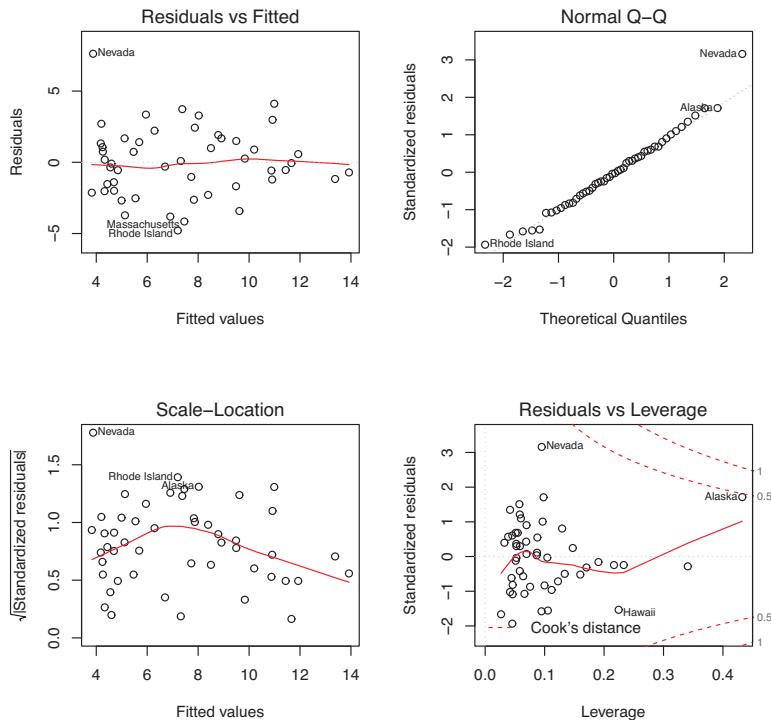


Рис. 8.8. Диагностические диаграммы для регрессии уровня преступности по характеристикам штатов

Функция	Назначение
spreadLevelPlot()	Диаграмма для обнаружения неоднородности дисперсии остатков (spread-level plot)
outlierTest()	Тест Бонферрони на выбросы
avPlots()	Диаграммы добавленных переменных
influencePlot()	Диаграмма влияния наблюдений на регрессию
scatterplot()	Усовершенствованная диаграмма рассеяния
scatterplotMatrix()	Усовершенствованная матрица диаграмм рассеяния
vif()	Фактор инфляции дисперсии

Важно отметить, что между версиями 1.x и 2.x пакета `car` есть много различий, включая изменения названий функций и их работы. В этой главе рассматривается версия 2.

В дополнение к этому в пакете `gvlma` реализован общий тест на соответствие условиям линейной модели. Рассмотрим все эти методы поочереди, применяя их к нашему примеру множественной регрессии.

Нормальность

Функция `qqPlot()` – это более аккуратный метод проверки предположения о нормальности, по сравнению с функцией `plot()` из базовой версии программы. Она изображает связь между *остатками Стьюдента* (также называемыми *исключенными остатками Стьюдента*, или *остатками, вычисленными методом последовательного исключения значений* – *jackknifed residuals*) и квантилями распределения Стьюдента с $n - p - 1$ степенями свободы, где n – это объем выборки, а p – число параметров регрессии (включая свободный член).

Программный код таков:

```
library(car)
fit <- lm(Murder ~ Population + Illiteracy + Income + Frost, data=states)
qqPlot(fit, labels=row.names(states), id.method="identify",
       simulate=TRUE, main="Q-Q Plot")
```

Функция `qqPlot` создает график распределения вероятностей, приведенный на рис. 8.9. Опция `id.method="identify"` делает диаграмму интерактивной – после того как график нарисован, щелчки мышью по точкам графика подписывают их значения, заданные опцией `labels`. Нажатие клавиши **Esc**, выбор команды **Stop** из выпадающего графического меню или щелчок правой кнопкой мыши на графике выключит интерактивный режим. В данном случае я подписан Неваду. Если `simulate=TRUE`, то на графике также изображаются 95%-ные доверительные границы с использованием параметрических бутстреп-методов (эти методы описаны в главе 12).

За исключением Невады, все точки попадают близко к линии и находятся внутри доверительных границ, что свидетельствует о нормальности распределения. Но нам определенно нужно взглянуть на Неваду. У нее есть большой положительный остаток (по сравнению с предсказанным значением), показывающий, что модель недооценивает уровень преступности в этом штате. А именно:

```
> states["Nevada",]

   Murder Population Illiteracy Income Frost
Nevada    11.5        590      0.5    5149    188

> fitted(fit) ["Nevada"]
```

```
Nevada  
3.878958
```

```
> residuals(fit) ["Nevada"]
```

```
Nevada  
7.621042
```

```
> rstudent(fit) ["Nevada"]
```

```
Nevada  
3.542929
```

Здесь вы видите, что уровень преступности равен 11.5%, а модель предсказывает 3.9%.

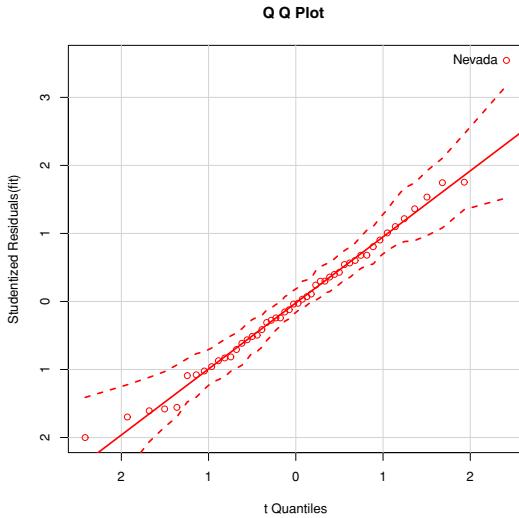


Рис. 8.9. Графическая проверка нормальности распределения (Q-Q-диаграмма)

Вопрос, который вам нужно задать, таков: «Почему в Неваде уровень преступности выше, чем это предсказано по численности населения, уровню дохода, неграмотности и морозности?». Кто-нибудь (кто не смотрел «Славных парней») хочет высказать предположение?

Для полноты изложения приведем еще один способ визуализации остатков. Взгляните на следующий программный код. Функция `residplot()` создает гистограмму остатков Стьюдента с наложенными кривой нормального распределения, кривой ядерной оценки функ-

ции плотности и графиком-щеткой. Для использования этой функции пакет `car` не нужен.

Программный код 8.6. Функция для изображения остатков Стьюдента

```
residplot <- function(fit, nbreaks=10) {
    z <- rstudent(fit)
    hist(z, breaks=nbreaks, freq=FALSE,
        xlab="Остатки Стьюдента",
        main="Распределение остатков")
    rug(jitter(z), col="brown")
    curve(dnorm(x, mean=mean(z), sd=sd(z)),
        add=TRUE, col="blue", lwd=2)
    lines(density(z)$x, density(z)$y,
        col="red", lwd=2, lty=2)
    legend("topright",
        legend = c( "Кривая нормального распределения",
                    "Ядерная оценка плотности"),
        lty=1:2, col=c("blue","red"), cex=.7)
}
residplot(fit)
```

Результаты представлены на рис. 8.10.

Распределение остатков

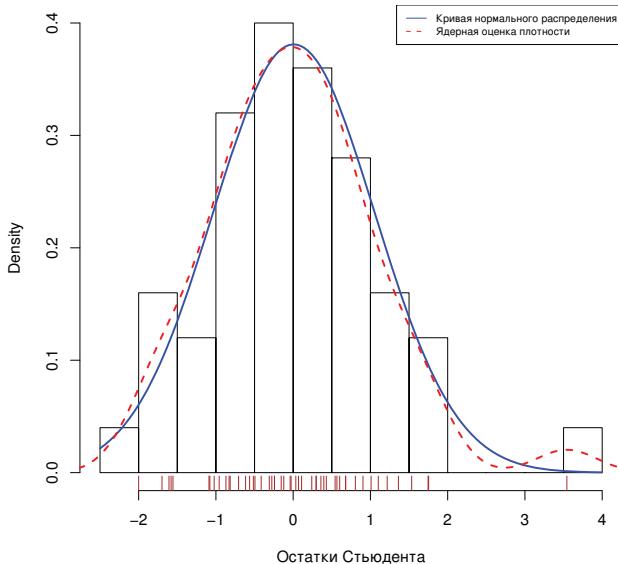


Рис. 8.10. Распределение остатков Стьюдента, изображенное при помощи функции `residplot()`

Как вы можете видеть, остатки достаточно хорошо соответствуют нормальному распределению, за исключением большого выброса. Хотя Q-Q-диаграмма, возможно, более информативна, мне всегда было легче оценить асимметрию распределения по гистограмме или графику функции плотности, чем по вероятностной диаграмме. Почему бы не использовать оба способа?

Независимость остатков

Как указывалось раньше, лучший способ понять, не связаны ли между собой значения зависимой переменной (и, таким образом, остатки), – это знание о способе сбора данных. Например, временным рядам часто свойственна автокорреляция – наблюдения, сделанные в короткие отрезки времени, более сильно коррелируют друг с другом, чем разнесенные во времени наблюдения. В пакете `car` реализована функция для проведения теста Дарбина-Уотсона на наличие автокорреляции в остатках. Этот тест можно применить к нашей задаче по множественной регрессии при помощи следующего программного кода:

```
> durbinWatsonTest (fit)
lag Autocorrelation D-W Statistic p-value
 1           -0.201          2.32   0.282
Alternative hypothesis: rho != 0
```

Высокое значение вероятности статистической ошибки первого рода ($p = 0.282$) свидетельствует об отсутствии автокорреляции и, следовательно, о независимости остатков. Значение интервала (`lag`, в данном случае 1) говорит о том, что каждое значение сравнивается с соседним, следующим за ним значением в наборе данных. Однако, будучи предназначенным для временных рядов, этот тест не так хорошо подходит для других типов данных. Учтите, что функция `durbinWatsonTest (fit)` для вычисления p использует бутстреп-метод (см. главу 12). Если вы не добавите опцию `simulate=FALSE`, результаты теста от раза к разу будут немного различаться.

Линейность

Наличие нелинейной связи между зависимой и независимыми переменными можно проверить при помощи диаграмм компонент и остатков (также известных под названием диаграммы частных остатков). Диаграммы создаются при помощи функции `crPlots()` из пакета `car`. Они помогают найти любые систематические отклонения от заданной линейной модели.

Для создания диаграммы компонент и остатков для переменной X_j нужно отобразить точки зависимости $\varepsilon_i + (\hat{\beta}_j * X_{ji})$ от X_{ji} , где остатки (ε_i) основаны на полной модели, а $i = 1 \dots n$. Прямая линия на каждой диаграмме соответствует зависимости $(\hat{\beta}_j * X_{ji})$ от X_{ji} . Сглаженные линии обсуждаются в главе 11. Эти диаграммы создаются при помощи следующих команд:

```
> library(car)
> crPlots(fit)
```

Полученные диаграммы представлены на рис. 8.11. Нелинейность на любой из них свидетельствует о том, что вы могли некорректно смоделировать функциональную форму этой независимой переменной в уравнении. В этом случае может потребоваться добавить нелинейные составляющие, такие как полиномиальные члены, преобразовать одну или более переменных (например, использовать $\log(X)$ вместо X) или же отказаться от линейной регрессии в пользу какой-нибудь другой ее разновидности. Преобразования обсуждаются ниже в этой главе. Диаграммы компонент и остатков подтверждают, что требование линейности соблюдено. Вид линейной модели подходит для данного набора данных.

Гомоскедастичность

В пакете `car` также реализованы две полезные функции для обнаружения неоднородности дисперсии остатков. Функция `ncvTest()` позволяет проверить гипотезу о постоянстве дисперсии остатков как альтернативу тому, что дисперсия остатков изменяется в зависимости от подобранных значений. Статистически значимый результат⁵ свидетельствует о гетероскедастичности (неоднородности дисперсии остатков).

Функция `spreadLevelPlot()` создает диаграмму рассеяния для абсолютных значений стандартизованных остатков и подобранных значений с наложенной регрессионной прямой. Применение обоих функций продемонстрировано в программном коде 8.7.

Программный код 8.7. Проверка на гомоскедастичность

```
> library(car)
> ncvTest(fit)
```

```
Non-constant Variance Score Test
Variance formula: ~ fitted.values
```

⁵ $p < 0.05$. – Прим. пер.

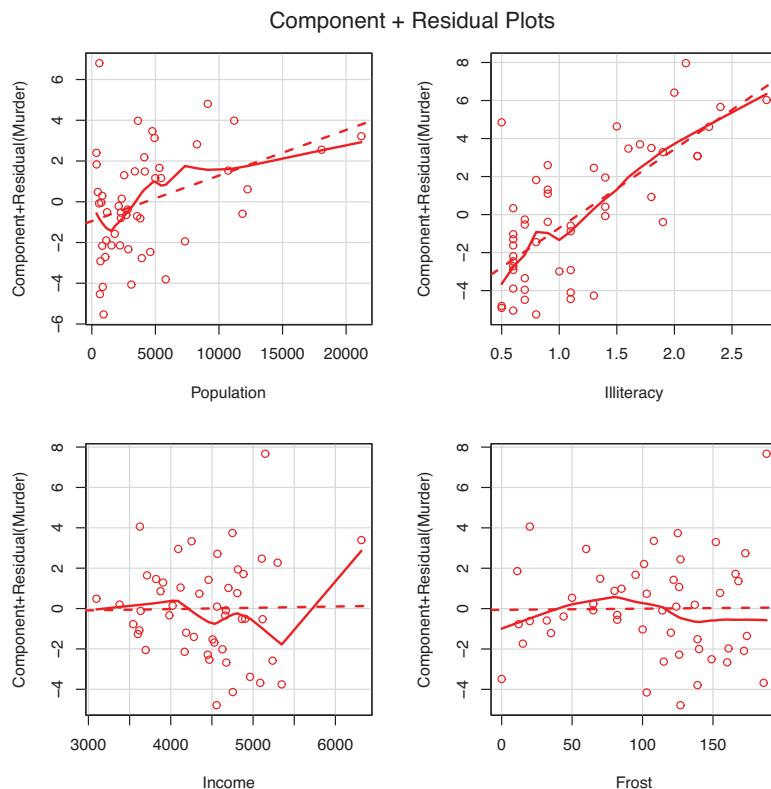


Рис. 8.11. Диаграммы компонент и остатков для регрессии уровня преступности по характеристикам штатов

ChiSquare=1.7 Df=1 p=0.19

> spreadLevelPlot(fit)

Suggested power transformation: 1.2

Результат теста незначим ($p = 0.19$), что свидетельствует о выполнении условия однородности дисперсии. Это также можно увидеть на диаграмме (рис. 8.12). Точки беспорядочно располагаются в виде горизонтальной полосы вдоль горизонтальной регрессионной прямой. Если бы требование гомоскедастичности не выполнялось, мы увидели бы не горизонтальную прямую. Предлагаемое степенное преобразование (Suggested power transformation) в программном

коде 8.7 – это степень p (Y^p), возведение в которую устраниет неоднородность дисперсии остатков. Например, если бы на диаграмме был показан нелинейный тренд и предлагаемое степенное преобразование было 0.5, то использование \sqrt{Y} вместо Y в уравнении регрессии могло бы дать модель, удовлетворяющую требованию гомоскедастичности. Если предлагаемая степень была бы равна 0, то нужно было использовать логарифмическое преобразование. В данном случае признаки гетероскедастичности отсутствуют, и предлагаемая степень близка к 1 (никакого преобразования не требуется).

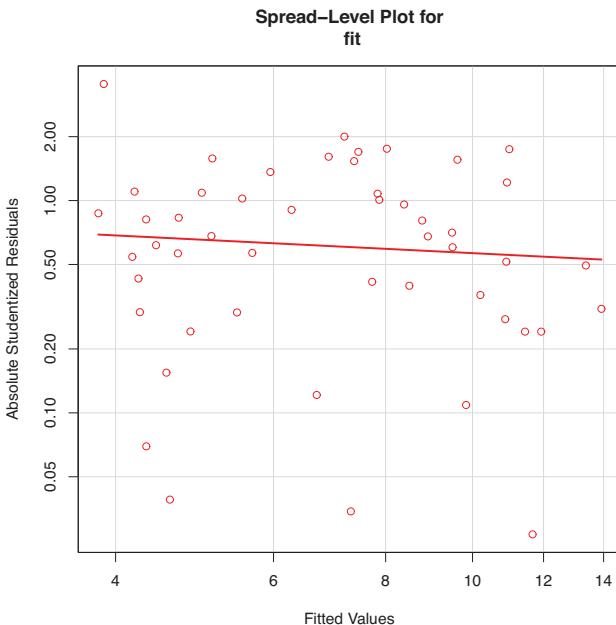


Рис. 8.12. Диаграмма для проверки однородности дисперсии остатков

8.3.3. Общая проверка выполнения требований, предъявляемых к линейным моделям

Наконец, давайте рассмотрим функцию `gvlma()` из пакета `gvlma`. Эта функция (Pena, Slate, 2006) осуществляет общую проверку выполнения требований, предъявляемых к линейным моделям наряду с отдель-

ной оценкой асимметрии, эксцесса и гомоскедастичности. Другими словами, этот тест дает общее заключение (подходит/не подходит) о выполнении требований, лежащих в основе модели. В приведенном ниже программном коде продемонстрировано применение этого теста к набору данных states.

Программный код 8.8. Общая проверка выполнения требований, предъявляемых к линейным моделям

```
> library(gvlma)
> gvmmodel <- gvlma(fit)
> summary(gvmmodel)
ASSESSMENT OF THE LINEAR MODEL ASSUMPTIONS
USING THE GLOBAL TEST ON 4 DEGREES-OF-FREEDOM:
Level of Significance= 0.05
Call:
gvlma(x=fit)

      Value p-value          Decision
Global Stat    2.773   0.597 Assumptions acceptable.
Skewness       1.537   0.215 Assumptions acceptable.
Kurtosis        0.638   0.425 Assumptions acceptable.
Link Function   0.115   0.734 Assumptions acceptable.
Heteroscedasticity 0.482   0.487 Assumptions acceptable.
```

Из результатов теста (строка Global Stat) следует, что данные удовлетворяют всем статистическим допущениям, лежащим в основе МНК-регрессии ($p = 0.597$). Если бы в этой строке было указано, что эти требования нарушены (то есть $p < 0.05$), вам бы пришлось исследовать данные описанными выше методами, чтобы определить, какие именно требования были нарушены.

8.3.4. Мультиколлинеарность

Прежде чем закончить этот раздел, посвященный диагностике регрессионных моделей, обратим внимание на проблему, которая не связана напрямую со статистическими допущениями, но все равно важна, поскольку позволяет вам интерпретировать результаты множественной регрессии.

Представим, что вы выполняете исследование силы рукопожатий. Ваши независимые переменные – это дата рождения (ДР) и возраст. Вы проводите регрессию силы рукопожатий по ДР и возрасту, при этом результат F-теста – $p < 0.001$. Однако рассмотренные по отдельности коэффициенты регрессии для ДР и возраста оказываются незначимыми (то есть нет никаких указаний на то, что любой из них связан с силой рукопожатий). Что же произошло?

Дело в том, что ДР и возраст полностью скоррелированы с точностью до ошибки округления. Коэффициент регрессии измеряет влияние одной независимой переменной на зависимую при постоянных значениях другой переменной. Это означает, что мы анализируем зависимость между силой рукопожатия и возрастом при постоянном значении возраста. Этот феномен носит название *мультиколлинеарности*. Он приводит к большим доверительным интервалам для параметров модели и затрудняет интерпретацию отдельных регрессионных коэффициентов.

Мультиколлинеарность можно выявить при помощи статистики, называемой фактором инфляции дисперсии. Квадратный корень, извлеченный из этой статистики для любой независимой переменной, указывает на степень увеличения доверительного интервала для параметра регрессии данной переменной по сравнению с моделью без скоррелированных независимых переменных (отсюда название статистики). Фактор инфляции дисперсии можно вычислить при помощи функции `vif()` из пакета `car`. Обычно значения квадратного корня этой статистики, превышающие 2, указывают на наличие мультиколлинеарности. Программный код приведен ниже. Полученный результат свидетельствует, что в нашем случае проблема мультиколлинеарности отсутствует.

Программный код 8.9. Оценка мультиколлинеарности

```
>library(car)
> vif(fit)
Population Illiteracy      Income      Frost
          1.2        2.2       1.3        2.1
> sqrt(vif(fit)) > 2 # проблема?
Population Illiteracy      Income      Frost
      FALSE      FALSE      FALSE      FALSE
```

8.4. Необычные наблюдения

Всеобъемлющий регрессионный анализ включает и анализ необычных наблюдений – а именно выбросов, наблюдений с высокой напряженностью и влиятельных наблюдений. Это те данные, которые требуют дальнейшего изучения: либо потому, что они каким-то образом отличаются от прочих, либо потому, что они значительно влияют на общие результаты. Рассмотрим все их по очереди.

8.4.1. Выбросы

Выбросы – это наблюдения, которые плохо предсказываются моделью. Они характеризуются большими положительными или отрицательными остатками ($Y_i - \hat{Y}_i$). Положительные остатки свидетельствуют о том, что модель недооценивает зависимую переменную, отрицательные остатки – признак переоценки.

Вы уже познакомились с одним способом обнаружения выбросов. Точки на рис. 8.9, находившиеся за пределами доверительной области, были интерпретированы как выбросы. Приближенный метод оценки таков: стандартизованные остатки больше 2 или меньше -2 заслуживают внимания.

В пакете `car` также реализован статистический тест на выбросы. Функция `outlierTest()` вычисляет значение вероятности статистической ошибки первого рода с поправкой Бонферрони для наибольшего остатка Стьюдента:

```
> library(car)
> outlierTest(fit)
    rstudent unadjusted p-value Bonferonni p
Nevada      3.5          0.00095      0.048
```

Отсюда видно, что Невада определена как выброс ($p = 0.048$). Обратите внимание, что эта функция проводит тест для одного наибольшего (положительного или отрицательного) остатка на значимость его как выброса. Если тест не дал значимого результата, то в данных нет выбросов. А если результат теста значимый, нужно удалить выброс⁶ и провести тест заново, чтобы узнать, есть ли другие выбросы.

8.4.2. Точки высокой напряженности

Точки с высокой напряженностью – это выбросы в отношении других независимых переменных. Иными словами, они характеризуются необычным сочетанием значений независимых переменных. Значение зависимой переменной не используется при вычислении напряженности.

Наблюдения с высокой напряженностью идентифицируются при помощи показателя влияния (hat statistic). Для определенного набора данных среднее значение этой статистики вычисляется как p/n , где p – это число параметров в модели (включая свободный член), а n – размер выборки. Грубо говоря, наблюдения, для которых значение

⁶ Имеется в виду наибольший по модулю выброс. – Прим. пер.

этой статистики превышает среднее в два или три раза, должны быть проанализированы. Программный код, при помощи которого можно построить диаграмму для значений показателя влияния, таков:

```
hat.plot <- function(fit) {
  p <- length(coefficients(fit))
  n <- length(fitted(fit))
  plot(hatvalues(fit), main="Диаграмма значений
  ↪показателя влияния")
  abline(h=c(2,3)*p/n, col="red", lty=2)
  identify(1:n, hatvalues(fit), names(hatvalues(fit)))
}
hat.plot(fit)
```

Полученная диаграмма приведена на рис. 8.13.

Диаграмма значений показателя влияния

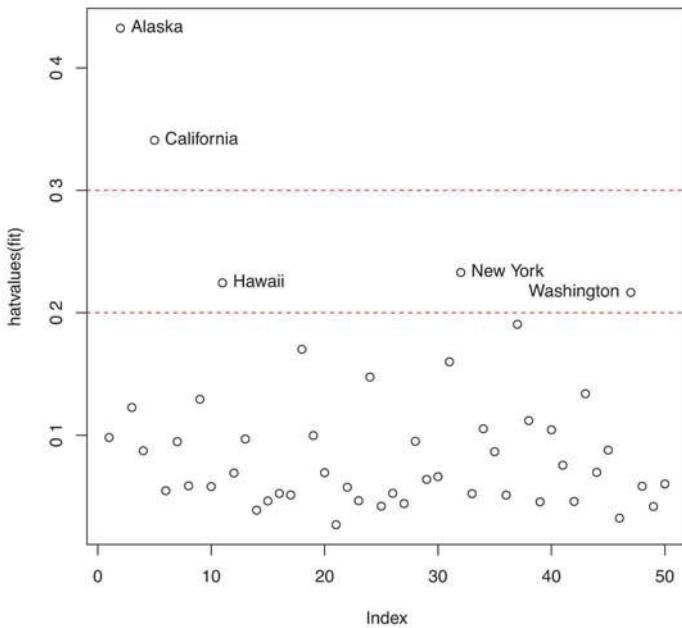


Рис. 8.13. Обнаружение наблюдений с высокой напряженностью: диаграмма значений показателя влияния для каждого наблюдения (их порядковые номера отложены по оси абсцисс)

Горизонтальные линии соответствуют значениям, вдвое и втрое превышающим среднее. Функция ввода координат дает возможность

подписать точки в интерактивном режиме. Щелчки по интересующим вас точкам позволяют называть их, пока не будет нажата клавиша **Esc**, выбран пункт **Stop** в выпадающем графическом меню или нажата правая кнопка мыши. Здесь видно, что Аляска и Калифорния – особенно необычные наблюдения, судя по их значениям независимых переменных. Для Аляски характерны более высокие доходы и более низкие численность населения и температуры. В Калифорнии наряду с более высокими доходами отмечены высокие численность населения и температура. Эти два штата выделяются на фоне остальных 48.

Наблюдения с высокой напряженностью могут быть, а могут и не быть влиятельными. Это зависит от того, являются ли они в то же время выбросами.

8.4.3. Влиятельные наблюдения

Влиятельные наблюдения – это наблюдения, которые оказывают непропорционально большое влияние на значения параметров модели. Представьте, что вы обнаружили значительные изменения модели при удалении единственного наблюдения. Возможность этого заставляет нас проверять данные на наличие влиятельных наблюдений.

Существуют два метода обнаружения влиятельных наблюдений: расстояние Кука (или D-статистика) и диаграммы добавленных переменных. Грубо говоря, значения расстояния Кука, превышающие $4/(n - k - 1)$, где n – объем выборки, а k – число независимых переменных, свидетельствуют о влиятельных наблюдениях. Построить диаграмму расстояний Кука (рис. 8.14) можно при помощи следующего программного кода:

```
cutoff <- 4 / (nrow(states) - length(fit$coefficients) - 2)
plot(fit, which=4, cook.levels=cutoff)
abline(h=cutoff, lty=2, col="red")
```

На диаграмме видно, что Аляска, Гавайи и Невада – это влиятельные наблюдения. Удаление этих штатов заметно влияет на значения свободного члена и угловых коэффициентов в регрессионной модели. Отметим, что хотя полезно применять чувствительные методы для поиска влиятельных значений, мне кажется более разумно использовать пороговое значение расстояния Кука, равное 1, а не $4/(n - k - 1)$. Если использовать критерий $D = 1$, то в данном наборе данных влиятельные значения будут отсутствовать.



Диаграммы дистанции Кука помогают обнаружить влиятельные наблюдения, но они не позволяют понять, как эти наблюдения влияют на модель. В этой ситуации приходят на выручку диаграммы добавленных переменных. Для одной зависимой и k независимых переменных описанным ниже способом создается k диаграмм добавленных переменных.

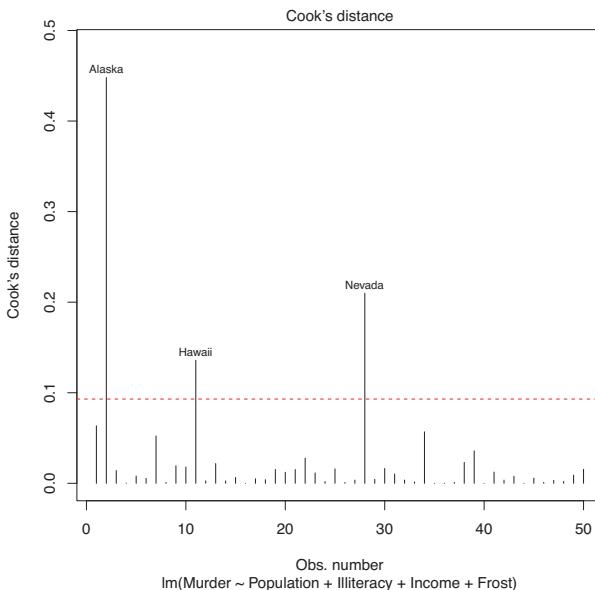


Рис. 8.14. Диаграмма расстояний Кука для обнаружения влиятельных наблюдений

Для каждой независимой переменной X_k отображаются остатки от регрессии зависимой переменной по остальным $k - 1$ независимым переменным. Такие диаграммы добавленных переменных можно построить при помощи функции `avPlots()` из пакета `car`:

```
library(car)
avPlots(fit, ask=FALSE, onepage=TRUE, id.method="identify")
```

Полученные диаграммы представлены на рис. 8.15. Эти диаграммы создаются одновременно, и пользователи могут щелкать на точки, чтобы подписать их. Нажмите **Esc**, выберите **Stop** из графического выпадающего меню или щелкните правой кнопкой мыши, чтобы перейти к следующей диаграмме. Здесь я обозначил Аляску на нижней левой диаграмме.

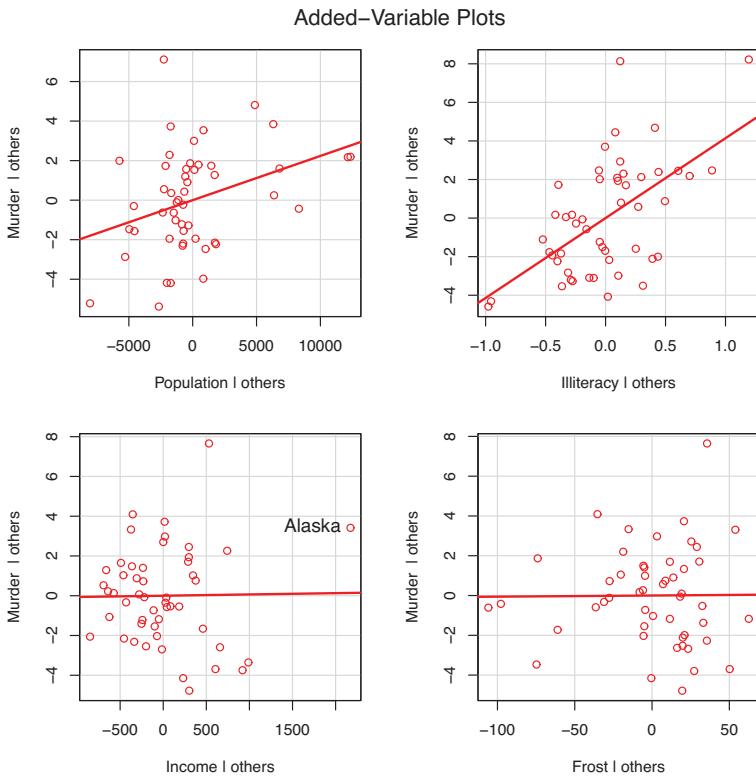


Рис. 8.15. Диаграммы добавленных переменных для исследования вклада влиятельных наблюдений

Прямая на каждой диаграмме – это регрессионный коэффициент для данной независимой переменной. Вклад влиятельных наблюдений можно оценить, если представить, как изменится линия, если удалить точку, соответствующую данному наблюдению. Для примера посмотрите на диаграмму для показателей Убийства | прочие (Murder | others) и Доход | прочие (Income | others) в левом верхнем углу. Можно видеть, что удаление точки, соответствующей Аляске, сместило бы линию в область отрицательных значений. На самом деле удаление Аляски приводит к изменению коэффициента регрессии для дохода с положительного (.00006) на отрицательный (−.00085).

Можно свести информацию о выбросах, точках с высокой напряженностью и влиятельных наблюдениях на одну чрезвычайно информативную диаграмму при помощи функции `influencePlot()` из



пакета car:

```
library(car)
influencePlot(fit, id.method="identify", main="Диаграмма влияния",
              sub="Размер круга пропорционален расстоянию Кука")
```

На полученной диаграмме (рис. 8.16) видно, что Невада и Род-Айленд – это выбросы, Нью-Йорк, Калифорния, Гавайи и Вашингтон характеризуются высокой напряженностью, а Невада, Аляска и Гавайи – влиятельные наблюдения.

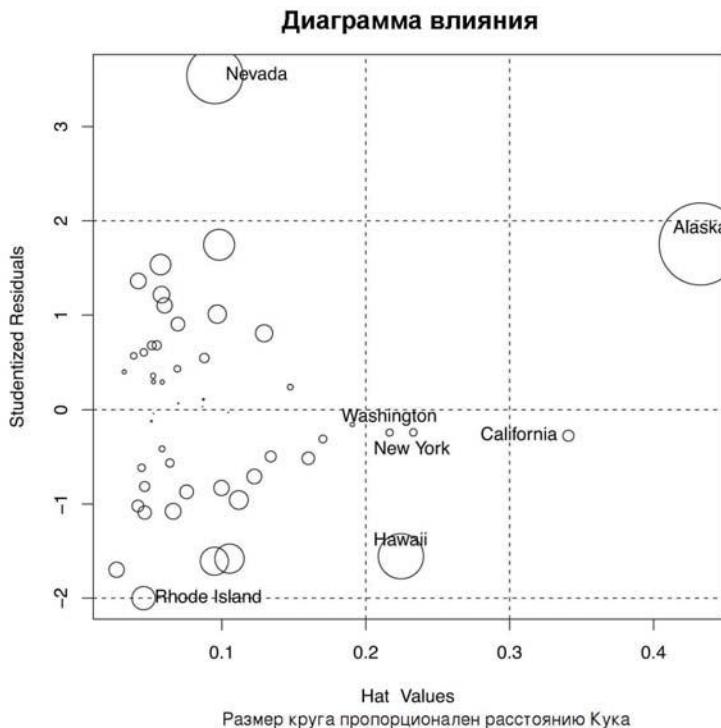


Рис. 8.16. Диаграмма влияния наблюдений на регрессию (influence plot). Штаты выше +2 или ниже -2 – выбросы.

Штаты выше 0.2 или 0.3 характеризуются высокой напряженностью (необычной комбинацией значений независимых переменных).

Размер кругов пропорционален степени влияния наблюдения.

Наблюдения, обозначенные большими кругами, могут оказывать значительное влияние на параметры модели

8.5. Способы корректировки

Читая последние 20 страниц, посвященные регрессионной диагностике, вы могли задаться вопросом «А что же вы делаете, когда обнаруживаете проблему?» Существуют четыре метода работы с отклонениями от допущений, лежащих в основе регрессионных моделей:

- удаление наблюдений;
- преобразование переменных;
- добавление или удаление переменных;
- использование другого регрессионного метода.

Рассмотрим все их по очереди.

8.5.1. Удаление наблюдений

Удаление выбросов часто может улучшить соответствие набора данных требованию нормальности. Влиятельные наблюдения также часто удаляют, поскольку они слишком сильно влияют на результаты. После удаления наибольшего выброса или влиятельного наблюдения модель подбирается заново. Если после этого все равно остаются выбросы или влиятельные наблюдения, процесс повторяется, пока не будет достигнутое допустимое соответствие модели данным.

Опять же, я призываю быть осторожным, когда вы принимаете решение об удалении наблюдений. Иногда можно понять, что наблюдение – это выброс из-за ошибок при сборе данных, или потому, что протокол исследования не был соблюден, или поскольку тестируемый нарушил инструкции. В таких случаях удаление наблюдений, служащих камнем преткновения, является абсолютно обоснованным.

В других случаях необычные наблюдения могут оказаться самыми интересными объектами в полученных данных. Понимание причин, по которым данное наблюдение отличается от остальных, может значительно приблизить вас к разгадке изучаемой проблемы и решению других вопросов, о которых вы могли не задумываться. Некоторые наиболее серьезные достижения могут быть обусловлены интуитивной прозорливостью, когда вы замечаете что-то, не соответствующее вашим ожиданиям (извините за патетику).

8.5.2. Преобразование переменных

Когда модели не отвечают требованию нормальности, линейности или гомоскедастичности, трансформация одной или более пере-

менных может улучшить или исправить ситуацию. Преобразования обычно заключаются в замене переменной Y на переменную Y^λ . Часто используемые значения λ и их интерпретация приведены в табл. 8.5.

Таблица 8.5. Часто используемые трансформации

	-2	-1	-0.5	0	0.5	1	2
Преобразование	$1/Y^2$	$1/Y$	$1/\sqrt{Y}$	$\log(Y)$	\sqrt{Y}	отсутствует	Y^2

Если Y – пропорция, то часто используется логит-преобразование $[\ln(Y/1-Y)]$.

Если модель не соответствует требованиям нормальности, обычно пытаются преобразовать зависимую переменную. При помощи функции `powerTransform()` из пакета `car` можно оценить по методу максимального правдоподобия величину λ , возведение в которую, скорее всего, нормализует переменную X^λ . В приведенном ниже программном коде этот метод применен к набору данных `states`.

Программный код 8.10. Преобразование Бокса-Кокса (Box-Cox) к нормальному виду

```
> library(car)
> summary(powerTransform(states$Murder))

bcPower Transformation to Normality
      Est.Power Std.Err. Wald Lower Bound Wald Upper Bound
states$Murder       0.6        0.26            0.088          1.1

Likelihood ratio tests about transformation parameters
              LRT df  pval
LR test, lambda=(0) 5.7  1 0.017
LR test, lambda=(1) 2.1  1 0.145
```

Из результата применения функции следует, что переменную `Murder` можно нормализовать, заменив ее на $Murder^{0.6}$. Поскольку 0.6 близко к 0.5, можно попробовать приблизить переменную к нормальности преобразованием методом извлечения квадратного корня. Однако в данном случае гипотеза о $\lambda = 1$ не может быть отвергнута ($p = 0.145$), так что необходимость такого преобразования переменной неочевидна. Это согласуется с результатами анализа диагностической диаграммы, представленной на рис. 8.9.

В том случае, когда требования линейности не выполняются, обычно помогает преобразование независимых переменных. Для оценки методом наибольшего правдоподобия той степени, в которую

нужно возвести независимые переменные для большего соответствия модели требованию линейности, можно использовать функцию `boxTidwell()` из пакета `car`. Вот пример преобразований Бокса-Тидвелла (Box-Tidwell) для модели, которая предсказывает уровень преступности в штатах по численности населения и уровню неграмотности:

```
> library(car)
> boxTidwell(Murder~Population+Illiteracy,data=states)
```

	Score	Statistic	p-value	MLE of lambda
Population	-0.32	0.75		0.87
Illiteracy	0.62	0.54		1.36

Из полученного результата следует, что для получения большей линейности стбйт попробовать преобразования `Population`⁸⁷ и `Illiteracy`^{1.36}. Однако результаты теста для переменных `Population` ($p = .75$) и `Illiteracy` ($p = .54$) свидетельствуют, что ни одну из них не нужно преобразовывать. Опять же, эти результаты соответствуют диаграмме компонент и остатков (рис. 8.11).

Наконец, преобразования зависимой переменной могут помочь в случае гетероскедастичности (непостоянной дисперсии остатков). В программном коде 8.7 было показано, что функция `spreadLevelPlot()` из пакета `car` позволяет понять, в какую степень нужно возвести зависимую переменную, чтобы увеличить гомоскедастичность. Опять же, в примере с набором данных `states` требование постоянства дисперсии ошибок выполняется, и никакие преобразования не требуются.

Предостережение, касающееся преобразований

У статистиков есть старая шутка: если вы не можете доказать A , докажите B и сделайте вид, что это было A (статистикам она кажется довольно смешной). В нашем случае важно, что если вы преобразовали переменные, интерпретация модели должна быть основана на преобразованных переменных, а не на исходных. Если преобразования были осмысленными, такими как логарифм дохода или обратные значения расстояния, трактовать полученный результат просто. А как вы собираетесь интерпретировать взаимосвязь между частотой суицидальных настроений и кубическим корнем из депрессии? Избегайте бессмысленных преобразований.



8.5.3. Добавление или удаление переменных

Изменение числа переменных, входящих в модель, будет влиять на степень ее соответствия данным. Иногда добавление важной переменной может исправить многие из тех проблем, которые мы обсуждали. Удаление причиняющих беспокойство переменных может привести к аналогичному эффекту.

Удаление переменных – это особенно важное средство при борьбе с мультиколлинеарностью. Если единственная ваша задача – это прогнозы, тогда мультиколлинеарность – не проблема. Однако если вы хотите интерпретировать отдельные независимые переменные, тогда вы должны как-то справляться с мультиколлинеарностью. Наиболее распространенный подход – удалить одну из переменных, из-за которых наблюдается данный феномен (то есть одну из переменных со значением квадратного корня из фактора инфляции дисперсии больше двух). Альтернативный подход – использовать гребневую регрессию, разновидность множественной регрессии, разработанную для применения в случаях мультиколлинеарности.

8.5.4. Попытка применить другой подход

Как вы только что узнали, один из подходов, применяемых в случае мультиколлинеарности, – это использование другого типа модели (в данном случае гребневой регрессии). При наличии выбросов и/или влиятельных наблюдений можно использовать устойчивую регрессионную модель, а не МНК-регрессию. Если не выполняется требование нормальности, можно подобрать нелинейную регрессионную модель. В случае отклонения от независимости ошибок можно применить модели, которые учитывают структуру остатков, – такие как модели временных рядов или многоуровневые регрессионные модели. Наконец, если требования, лежащие в основе МНК-регрессии, не выполняются, вы можете обратиться к обобщенным линейным моделям.

Мы рассмотрим некоторые из этих альтернативных подходов в главе 13. Сложно решить, когда нужно стараться подобрать МНК-регрессионную модель, а когда – попробовать применить другой подход. Обычно такое решение основано на знании специфики исследуемого явления и оценки того, какой подход даст наилучший результат.

Раз мы заговорили о наилучших результатах, нужно обратиться к проблеме выбора независимых переменных, включаемых в регрессионную модель.

8.6. Выбор «лучшей» регрессионной модели

Составляя уравнение регрессии, вы в неявном виде сталкиваетесь с выбором из большого числа возможных моделей. Следует ли включить все исследуемые переменные или удалить те, которые не вносят значительного вклада в предсказание значений зависимой переменной? Нужно ли добавлять полиномиальные члены и/или эффекты взаимодействия, чтобы улучшить соответствие модели данным? Выбор окончательной регрессионной модели всегда подразумевает компромисс между точностью предсказания (моделью, которая соответствует данным так хорошо, как это возможно) и экономностью (простая и воспроизводимая модель). При прочих равных условиях из двух моделей с одинаковой предсказательной силой вы отдаете предпочтение наиболее простой. В этом разделе описаны методы выбора между конкурирующими моделями. Слово «лучшей» взято в кавычки, поскольку не существует единственного критерия, который можно использовать при совершении выбора. Окончательное решение основывается на мнении исследователя (считайте это гарантией вашей востребованности).

8.6.1. Сравнение моделей

Две вложенные модели можно сравнить по степени соответствия данным при помощи функции `anova()`, входящей в базовую версию программы. Вложенная модель (nested model) – это модель, все члены которой входят в другую модель. Мы обнаружили, что в нашей модели множественной регрессии для набора данных `states` коэффициенты для переменных `Income` и `Frost` были незначимыми. Можно проверить, будет ли модель без этих двух переменных предсказывать значения зависимой переменной так же хорошо, как и модель, в которую они включены (см. следующий программный код).

Программный код 8.11. Сравнение вложенных моделей при помощи функции `anova()`

```
> fit1 <- lm(Murder ~ Population + Illiteracy + Income + Frost,  
+ data=states)  
> fit2 <- lm(Murder ~ Population + Illiteracy, data=states)  
> anova(fit2, fit1)  
Analysis of Variance Table  
Model 1: Murder ~ Population + Illiteracy
```

```
Model 2: Murder ~ Population + Illiteracy + Income + Frost
Res.Df      RSS  Df   Sum of Sq    F Pr(>F)
1        47 289.246
2        45 289.167  2     0.079 0.0061    0.994
```

В данном случае модель 1 вложена в модель 2. Функция `anova()` одновременно проверяет, занижает или завышает модель без переменных `Income` и `Frost` предсказанные значения по сравнению с полным набором переменных. Поскольку результат теста незначим ($p = .994$), мы заключаем, что эти две переменные не увеличивают предсказательную силу модели, так что мы правильно решили исключить их.

Информационный критерий Акаике (Akaike Information Criterion, AIC) – это другой способ сравнения моделей. При расчете этого критерия учитывается статистическое соответствие модели данным и число необходимых для достижения этого соответствия параметров. Предпочтение нужно отдавать моделям с **меньшими** значениями AIC, указывающими на хорошее соответствие данным при использовании меньшего числа параметров. Этот критерий вычисляется при помощи функции `AIC()` (см. следующий программный код).

Программный код 8.12. Сравнение моделей при помощи AIC

```
> fit1 <- lm(Murder ~ Population + Illiteracy + Income + Frost,
+             data=states)
> fit2 <- lm(Murder ~ Population + Illiteracy, data=states)
> AIC(fit1,fit2)
      df      AIC
fit1  6 241.6429
fit2  4 237.6565
```

Значения AIC свидетельствуют, что модель без переменных `Income` и `Frost` лучше. Учтите, что если использование ANOVA требует вложенных моделей, то для применения AIC это необязательно.

Сравнить две модели относительно просто, но что делать, если у вас есть четыре, десять или сто возможных моделей, которые нужно проанализировать? Это тема следующего раздела.

8.6.2. Выбор переменных

Существуют два распространенных способа формировать окончательный набор независимых переменных из большего числа имеющихся переменных – это пошаговый метод и регрессия по всем подмножествам.

Пошаговая регрессия

При пошаговом выборе переменные добавляются в модель или удаляются из нее по одной, пока не будет достигнуто заданное значение критерия для остановки процесса. Например, при методе *пошагового включения* (forward stepwise) переменные по одной добавляются в модель, пока добавление новых переменных не перестанет ее улучшать. При *пошаговом исключении* (backward stepwise) вы начинаете с модели, включающей все независимые переменные, а потом удаляете их по одной до тех пор, пока модель не начнет ухудшаться. При *комбинированном методе* (stepwise stepwise) совмещены оба упомянутых подхода. Переменные добавляются по одной, однако на каждом шаге происходит переоценка модели, и те переменные, которые не вносят значительного вклада, удаляются. Независимая переменная может быть включена в модель и удалена из нее несколько раз, пока не будет достигнуто окончательное решение.

Результат применения метода пошаговой регрессии зависит от критериев включения или удаления переменных. При помощи функции `stepAIC()` из пакета MASS можно провести все три типа пошаговой регрессии с использованием точного критерия AIC. В следующем программном коде метод регрессии с пошаговым исключением применен для решения задачи по множественной регрессии.

Программный код 8.13. Регрессия с пошаговым исключением переменных

```
> library(MASS)
> fit1 <- lm(Murder ~ Population + Illiteracy + Income + Frost,
  data=states)
> stepAIC(fit, direction="backward")
```

```
Start: AIC=97.75
Murder ~ Population + Illiteracy + Income + Frost
```

	Df	Sum of Sq	RSS	AIC
- Frost	1	0.02	289.19	95.75
- Income	1	0.06	289.22	95.76
<none>			289.17	97.75
- Population	1	39.24	328.41	102.11
- Illiteracy	1	144.26	433.43	115.99

```
Step: AIC=95.75
Murder ~ Population + Illiteracy + Income
```

	Df	Sum of Sq	RSS	AIC
- Income	1	0.06	289.25	93.76
<none>			289.19	95.75

```
- Population 1      43.66 332.85 100.78
- Illiteracy 1     236.20 525.38 123.61

Step: AIC=93.76
Murder ~ Population + Illiteracy

          Df Sum of Sq    RSS    AIC
<none>              289.25 93.76
- Population 1      48.52 337.76 99.52
- Illiteracy 1     299.65 588.89 127.31

Call:
lm(formula=Murder ~ Population + Illiteracy, data=states)

Coefficients:
(Intercept)  Population   Illiteracy
1.6515497    0.0002242   4.0807366
```

Мы начинаем с модели, включающей все четыре независимые переменные. В столбце AIC приведено значение одноименного критерия для модели, из которой удалена указанная в соответствующей строке переменная. Значение AIC для строки <none> (никакой) – это значение критерия для модели, из которой не удалено никаких переменных. На первом шаге удалена переменная Frost, что привело к уменьшению AIC с 97.75 до 95.75. На втором шаге удалена переменная Income, при этом значение AIC снизилось до 93.76. Удаление остальных переменных увеличивает значение критерия, поэтому процесс остановлен.

Пошаговая регрессия – спорный подход. Хотя с его помощью можно найти хорошую модель, нет гарантии, что она будет лучшей, поскольку не рассмотрены все возможные модели. Попытка обойти это ограничение делается при использовании *регрессии по всем подмножествам*.

Регрессия по всем подмножествам

В ходе регрессии по всем подмножествам исследуются все возможные модели. Вы можете просмотреть все полученные результаты или вывести на экран определенное число лучших моделей для каждого подмножества (одна независимая переменная, две и т. д.). Например, при значении параметра nbest=2 выводятся на экран две лучшие модели для одной независимой переменной, потом две лучшие модели для двух независимых переменных, затем – для трех, заканчивая двумя лучшими моделями, в которые входят все независимые переменные.

Регрессия по всем подмножествам проводится при помощи функции `regsubsets()` из пакета `leaps`. В качестве критерия «лучшей» модели можно выбрать коэффициент детерминации, скорректированный коэффициент детерминации или Ср-статистику Мэллоуса (Mallows Cp statistic).

Как вы уже знаете, коэффициент детерминации – это доля дисперсии зависимой переменной, объясненная независимыми переменными. Скорректированный коэффициент детерминации учитывает число параметров модели. Дело в том, что коэффициент детерминации всегда увеличивается при добавлении независимых переменных. Когда число независимых переменных достаточно велико (по сравнению с объемом выборки), соответствие модели данным может быть переоценено. Скорректированный коэффициент детерминации создан в попытке дать более устойчивую оценку коэффициента детерминации для генеральной совокупности. Статистика Мэллоуса также используется в качестве критерия «лучшей» модели. Считается, что для хорошей модели эта статистика должна принимать значения, близкие к чистым параметрам модели (включая свободный член).

В программном коде 8.14 мы применим регрессию по всем подмножествам к набору данных `states`. Результаты можно отобразить графически при помощи функции `plot()` из пакета `leaps` (рис. 8.17) или посредством команды `subsets()`, реализованной в пакете `car` (рис. 8.18).

Программный код 8.14. Регрессия по всем подмножествам

```
library(leaps)
leaps <- regsubsets(Murder ~ Population + Illiteracy + Income +
  Frost, data=states, nbest=4)
plot(leaps, scale="adjr2")

library(car)
subsets(leaps, statistic="cp",
  main="Статистика Мэллоуса для регрессии по всем подмножествам")
abline(1,1,lty=2,col="red")
```

В рис. 8.17 может быть нелегко разобраться. Рассматривая первый ряд (считая снизу), можно увидеть, что скорректированный коэффициент детерминации для модели, включающей свободный член и переменную `Income`, равен 0.33. Для модели, включающей свободный член и переменную `Population`, этот коэффициент равен 0.1. Пере скочив на 12-ый ряд, вы увидите, что модель со свободным членом и переменными `Population`, `Illiteracy` и `Income` характеризуется



коэффициентом 0.54, а модель со свободным членом и переменными Population и Illiteracy имеет коэффициент 0.55. Отсюда видно, что значения скорректированного коэффициента детерминации выше для модели с меньшим числом независимых переменных (такого не могло быть в случае нескорректированного коэффициента детерминации). Из приведенной диаграммы следует, что модель с двумя независимыми переменными (Population и Illiteracy) подходит больше всего.

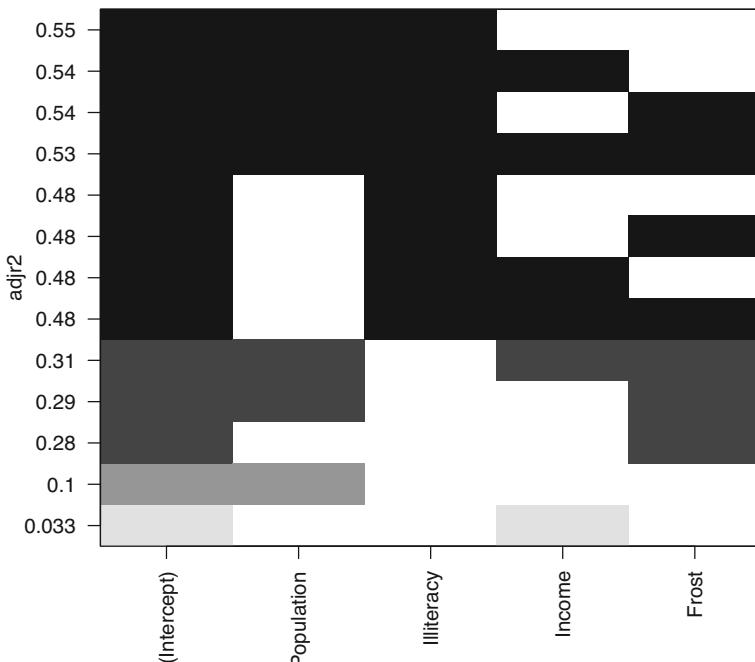


Рис. 8.17. Лучшие четыре модели для подмножеств всех размерностей, определенные на основании скорректированного коэффициента детерминации

На рис. 8.18 показаны четыре лучшие модели для подмножеств всех размерностей, определенные на основе статистики Мэллоуза. Наиболее удачные модели находятся близко к линии со свободным членом и коэффициентом регрессии, равным единице. Из диаграммы видно, что вам нужно выбирать между моделью с двумя независимыми переменными (Population и Illiteracy), двумя моделями с тремя независимыми переменными (Population, Illiteracy и

Frost или Population, Illiteracy и Income – они перекрываются на диаграмме) и моделью со всеми четырьмя независимыми переменными (Population, Illiteracy, Income и Frost). Все остальные модели можно сразу отвергнуть.

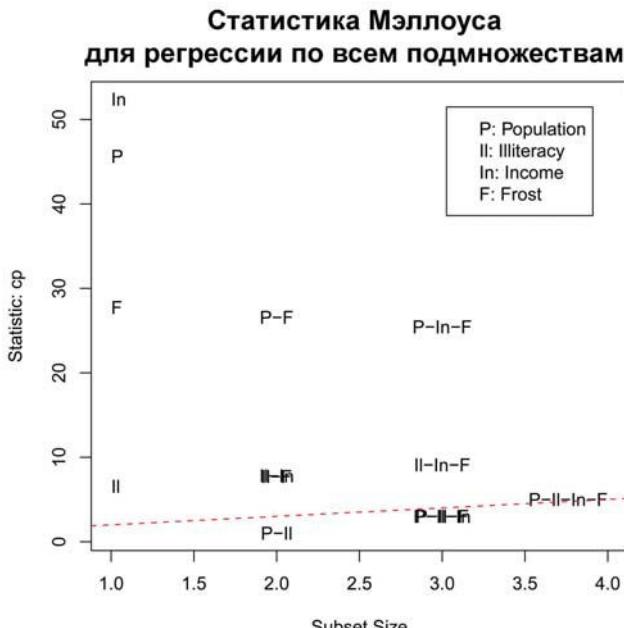


Рис. 8.18. Лучшие четыре модели для подмножеств всех размерностей, определенные на основании Ср-статистики Мэллоуса

В большинстве случаев регрессия по всем подмножествам предпочтительнее пошаговой регрессии, поскольку в первом случае анализируется больше моделей. Однако, если независимых переменных много, вычисления могут занять много машинного времени. В целом автоматизированный выбор переменных нужно рассматривать как помощь, а не определяющий фактор при выборе моделей. В конечном счете вы должны опираться на знание предмета исследования.

8.7. Продолжение анализа

Мы закончим обсуждение регрессии рассмотрением методов оценки применимости модели для генеральной совокупности и анализа относительной важности независимых переменных.



8.7.1. Кросс-валидация

В предыдущем разделе мы изучали методы выбора входящих в уравнение регрессии переменных. Если ваша главная задача – описание, работа заканчивается выбором регрессионной модели и ее интерпретацией. Однако если ваша цель – это предсказание, вы имеете все основания спросить: «Насколько хорошо полученное уравнение работает в реальном мире?»

По определению, регрессионные методы позволяют рассчитать оптимальные для имеющегося набора данных параметры. При использовании МНК-регрессии параметры модели подбираются так, чтобы минимизировать сумму квадратов ошибок предсказаний (остатков) и, напротив, максимизировать долю объясненной дисперсии зависимой переменной (коэффициент детерминации). Поскольку уравнение оптимизировано для имеющегося набора данных, оно не даст таких же хороших результатов для других данных.

Мы начали эту главу с рассказа об ученом-физиологе, который хотел предсказать число затрачиваемых человеком калорий по продолжительности выполнения им упражнений, его возрасту, полу и индексу массы тела. Если вы подберете для этих данных уравнение МНК-регрессии, то получите значения параметров модели, которые максимизируют коэффициент детерминации этого *конкретного* набора данных. Однако наш ученый хочет использовать это уравнение для предсказания числа затраченных людьми калорий в общем, а не только теми, кто участвовал в исходном исследовании. Вы знаете, что уравнение не будет работать так же хорошо для нового набора наблюдений, но сколько именно вы потеряете? Кросс-валидация – это полезный метод для оценки применимости модели к генеральной совокупности.

При кросс-валидации часть данных используется как обучающая выборка, а часть – как тестовая. Регрессионное уравнение подгоняется для обучающей выборки, а затем применяется для проверочной. Поскольку проверочная выборка не использовалась для подбора параметров модели, применимость модели к этой выборке – хорошая оценка применимости полученных параметров модели к новым данным.

При k -кратной кросс-валидации выборка разделяется на k подвыборок. Каждая из них играет роль тестовой выборки, а объединенные данные оставшихся $k - 1$ подвыборок используются как обучающая группа. Применимость k уравнений к k тестовым выборкам фиксируется и затем усредняется. Если $k = n$ (общему числу наблюдений), то такой подход называется оценкой по методу «складного ножа»

(последовательного исключения значений выборки с возвратом – jackknifing).

Выполнить k -кратную кросс-валидацию можно при помощи функции `crossval()` из пакета `bootstrap`. Следующий программный код содержит функцию `shrinkage()` для перекрестной k -кратной проверки коэффициента детерминации.

Программный код 8.15. Функция для k -кратной кросс-валидации

```
shrinkage <- function(fit, k=10) {  
  require(bootstrap)  
  
  theta.fit <- function(x,y){lsfit(x,y)}  
  theta.predict <- function(fit,x){cbind(1,x) %*% fit$coef}  
  
  x <- fit$model[,2:ncol(fit$model)]  
  y <- fit$model[,1]  
  
  results <- crossval(x, y, theta.fit, theta.predict, ngroup=k)  
  r2 <- cor(y, fit$fitted.values)^2  
  r2cv <- cor(y, results$cv.fit)^2  
  cat("Original R-square =", r2, "\n")  
  cat(k, "Fold Cross-Validated R-square =", r2cv, "\n")  
  cat("Change =", r2-r2cv, "\n")  
}
```

При помощи этого программного кода вы создаете нужные функции, формируете матрицу значений независимых и зависимых переменных, вычисляете обычный коэффициент детерминации и проводите расчет этого коэффициента методом кросс-валидации (методы бутстреп-анализа детально разобраны в главе 12).

Далее функция `shrinkage()` использована для 10-кратной кросс-валидации со всеми четырьмя независимыми переменными для набора данных `states`:

```
> fit <- lm(Murder ~ Population + Income + Illiteracy + Frost, data=states)  
> shrinkage(fit)  
  
Original R-square=0.567  
10 Fold Cross-Validated R-square=0.4481  
Change=0.1188
```

Видно, что коэффициент детерминации, рассчитанный для нашей выборки (0.567), чересчур оптимистичен. Лучшая оценка доли изменчивости уровня преступности, объясненной нашей моделью на новых данных, – это коэффициент детерминации, полученный методом перекрестной оценки (0.448). Учтите, что наблюдения разделяются на

к групп случайно, так что у вас будут получаться немного разные результаты при каждом использовании функции `shrinkage()`.

Кросс-валидацию можно использовать при выборе переменных, отдавая предпочтение модели, которая лучше применяется к генеральной совокупности. Например, модель с двумя независимыми переменными (`Population` и `Illiteracy`) характеризуется меньшим снижением⁷ коэффициента детерминации при кросс-валидации (0.03), по сравнению с 0.12 для модели, включающей все переменные:

```
> fit2 <- lm(Murder~Population+Illiteracy,data=states)
> shrinkage(fit2)
```

```
Original R-square=0.5668327
10 Fold Cross-Validated R-square=0.5346871
Change=0.03214554
```

Это обстоятельство может послужить аргументом в пользу модели с двумя переменными.

При прочих равных условиях уравнение регрессии, полученное на основании большей обучающей выборки и лучше применимое для генеральной совокупности, при перекрестной проверке будет оценено выше. В этом случае коэффициент детерминации будет уменьшаться не так сильно, а предсказания получатся более точными.

8.7.2. Относительная важность

До этого момента мы задавались вопросом: «Какие переменные полезны для предсказания результата?» Однако иногда нас по-настоящему интересует такая проблема: «Какие переменные *наиболее важны* для предсказания результата?» На самом деле вы хотите ранжировать переменные согласно их относительной важности. Для постановки второго вопроса могут существовать практические основания. Например, если вы сможете ранжировать техники лидерства по их важности для организационного успеха, то сможете ориентировать менеджеров на более продуктивное поведение.

Если бы независимые переменные не коррелировали друг с другом, эта задача была бы простой. Вы бы упорядочили независимые переменные по степени их корреляции с зависимой переменной. Однако в большинстве случаев независимые переменные коррелируют друг с другом, и это значительно усложняет дело.

7 По-английски `shrinkage`, отсюда и название новой функции. – Прим. пер.

Было предпринято много попыток разработать способы оценки относительной важности независимых переменных. Самое простое – сравнить стандартизованные коэффициенты регрессии. Они характеризуют ожидаемое изменение зависимой переменной (выраженное в числе стандартных отклонений) при изменении отдельной независимой переменной на одно стандартное отклонение при постоянных значениях прочих независимых переменных. В R стандартизованные регрессионные коэффициенты можно получить, преобразовав перед проведением регрессионного анализа все переменные при помощи функции `scale()` так, чтобы их среднее было равно 0, а стандартное отклонение – 1. Учтите, что поскольку функция `scale()` возвращает матрицу, а команда `lm()` работает с таблицей данных, вам нужно изменить формат данных на промежуточном этапе. Программный код и результаты его применения к нашей задаче по множественной регрессии приведены ниже:

```
> zstates <- as.data.frame(scale(states))
> zfit <- lm(Murder~Population + Income + Illiteracy + Frost, data=zstates)
> coef(zfit)
```

(Intercept)	Population	Income	Illiteracy	Frost
-9.406e-17	2.705e-01	1.072e-02	6.840e-01	8.185e-03

Отсюда видно, что увеличение уровня неграмотности (`Illiteracy`) на одно стандартное отклонение влечет за собой увеличение уровня преступности на 0.68 стандартного отклонения при постоянных значениях численности населения (`Population`), дохода (`Income`) и морозности (`Frost`). Исходя из стандартизованных регрессионных коэффициентов, неграмотность – наиболее важный параметр, а морозность – наименее важный.

Было предпринято много других попыток количественного анализа относительной важности независимых переменных. Относительную важность можно выражать в виде вклада каждой переменной в коэффициент детерминации (как по одиночке, так и в сочетании с другими независимыми переменными). Несколько возможных подходов к оценке относительной важности реализованы в пакете `relaimpo`, созданном Ульrike Грёмпингом (Ulrike Grömping) (<http://prof.beuth-hochschule.de/groemping/relaimpo/>).

Новый метод, названный *относительными весами* (relative weights), выглядит довольно многообещающим. Этот метод довольно точно аппроксимирует среднее увеличение коэффициента детерминации при добавлении независимой переменной во все возможные подмодели

(Johnson, 2004; Johnson, Lebreton, 2004; LeBreton, Tonidandel, 2008). Функция для вычисления относительных весов представлена ниже.

Программный код 8.16. Функция `relweights()` для вычисления относительной важности переменных

```
relweights <- function(fit,...){  
  R <- cor(fit$model)  
  nvar <- ncol(R)  
  rxx <- R[2:nvar, 2:nvar]  
  rxy <- R[2:nvar, 1]  
  svd <- eigen(rxx)  
  evec <- svd$vectors  
  ev <- svd$values  
  delta <- diag(sqrt(ev))  
  lambda <- evec %*% delta %*% t(evec)  
  lambdasq <- lambda ^ 2  
  beta <- solve(lambda) %*% rxy  
  rsquare <- colSums(beta ^ 2)  
  rawwgt <- lambdasq %*% beta ^ 2  
  import <- (rawwgt / rsquare) * 100  
  lbls <- names(fit$model[2:nvar])  
  rownames(import) <- lbls  
  colnames(import) <- "Weights"  
  barplot(t(import),names.arg=lbls,  
          ylab=" % предсказанной дисперсии",  
          xlab=" Независимые переменные",  
          main=" Относительные веса независимых переменных",  
          sub=paste("R-Square=", round(rsquare, digits=3)),  
          ...)  
  return(import)  
}
```

Примечание. Программный код 8.16 – адаптированная программа SPSS, любезно предоставленная др. Джонсоном. В его публикации (Johnson, 2000, *Multivariate Behavioral Research*, 35, 1–19) объяснено, как вычисляются относительные веса.

В программном коде 8.17 функция `relweights()` применяется для предсказания уровня преступности по численности населения, уровню неграмотности, дохода и температурам в наборе данных `states`.

Из рис. 8.19 следует, что общая доля объясненной моделью дисперсии (коэффициент детерминации, R-Square = 0.567) распределяется между независимыми переменными. На долю неграмотности приходится 59% коэффициента детерминации, на долю морозности – 20.79% и т. д. Если применить метод относительных весов, то неграмотность имеет наибольшую относительную важность, а дальше следуют (в

порядке убывания важности) морозность, численность популяции и уровень дохода.

Относительные веса независимых переменных

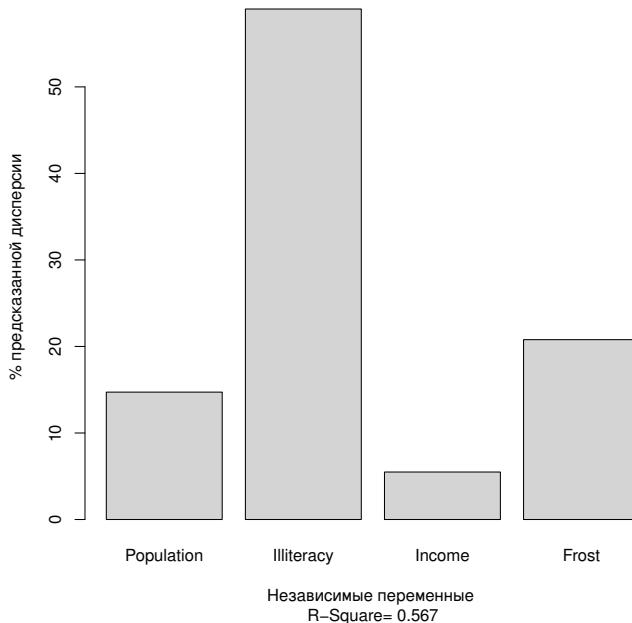


Рис. 8.19. Столбчатая диаграмма для относительных весов независимых переменных из задачи по множественной регрессии для набора данных states

Программный код 8.17. Применение функции relweights()

```
> fit <- lm(Murder ~ Population + Illiteracy + Income + Frost,
  ↴data=states)
> relweights(fit, col="lightgrey")
```

	Weights
Population	14.72
Illiteracy	59.00
Income	5.49
Frost	20.79

Показатели относительной важности переменных (и в особенности метод относительных весов) имеют широкое применение. Они находятся гораздо ближе к нашей интуитивной концепции относительной



важности по сравнению со стандартизованными коэффициентами регрессии. Я уверен, что в ближайшие годы интенсивность использования показателей относительной важности резко возрастет.

8.8. Резюме

Регрессионный анализ – это название, под которым скрываются разнообразные статистические методологии. Вы узнали, что это в значительной степени интерактивный подход, который включает подбор моделей, проверку выполнения допущений, лежащих в их основе, модификацию и данных, и моделей, а также повторный подбор моделей для достижения окончательного результата. Во многих отношениях окончательный результат зависит от искусства и умения – так же, как и наука в целом.

Это была длинная глава, поскольку регрессионный анализ – это процесс, состоящий из многих частей. Мы обсуждали подбор МНК-моделей, использование регрессионной диагностики для оценки соответствия данных требованиям модели и способов модификации данных для более полного соответствия этим требованиям. Мы рассматривали способы выбора окончательной регрессионной модели из многих возможных, и вы научились оценивать применимость модели к новым выборкам. Наконец, мы пытались найти решение сложной проблемы важности переменных: определения, какие независимые переменные наиболее важны для предсказания значений зависимой переменной.

В каждом из примеров этой главы независимые переменные были количественными. Однако никто не запрещает использовать также категориальные независимые переменные. Использование таких категориальных независимых переменных, как пол, способ лечения или тип производственного процесса, позволяет исследовать межгрупповые различия зависимой переменной. Это предмет нашей следующей главы.



ГЛАВА 9.

Дисперсионный анализ

В этой главе:

- Использование R для моделирования основных типов экспериментов.
- Подбор и интерпретация ANOVA-моделей.
- Оценка выполнимости допущений, лежащих в основе модели.

В главе 7 мы рассмотрели регрессионные модели для предсказания значений количественной зависимой переменной по значениям количественных независимых переменных. Однако нет никаких причин, по которым мы также не могли бы использовать номинальные или порядковые факторы в качестве независимых переменных. Если в набор независимых переменных входят факторы, акцент обычно смещается с предсказания на исследование межгрупповых различий, а такой метод называется *дисперсионный анализ* (analysis of variance, ANOVA). Этот метод используется для анализа самых разных экспериментов и квази-экспериментов. В этой главе представлен обзор функций, которые используются в R для анализа результатов исследований, проведенных по стандартным планам.

Для начала познакомимся с терминологией и обсудим общие принципы подгонки ANOVA-моделей в R. Затем мы рассмотрим несколько примеров, иллюстрирующих анализ распространенных типов экспериментов. По ходу изложения мы будем иметь дело с состояниями тревоги и пониженным уровнем холестерина в крови, а также поможем беременным мышам обзавестись упитанным потомством, убедимся, что у свиней вырастают достаточно длинные зубы, облегчим

дыхание растениям и узнаем, каких полок в бакалейных магазинах стоит избегать.

В дополнение к базовой версии программы при рассмотрении примеров мы будем использовать пакеты `car`, `gplots`, `HH`, `rrcov` и `mvoutlier`. Убедитесь в том, что они установлены, прежде чем пробовать запустить приведенный программный код.

9.1. Ускоренный курс терминологии

Планирование экспериментов в целом и дисперсионный анализ в частности имеют свой собственный язык. Перед тем как обсуждать анализ по-разному построенных экспериментов, мы быстро вспомним некоторые важные термины. Мы рассмотрим наиболее значимые концепции на примере ряда усложняющихся экспериментов.

Допустим, вы интересуетесь лечением состояний тревоги. Два широко распространенных метода лечения таких состояний – это когнитивная трудотерапия (КТ) и десенсибилизация и переработка движением глаз (ДПДГ). Вы берете десять человек, страдающих повышенной тревожностью, и случайным образом распределяете их по двум равным группам. В течение пяти недель одна группа подвергается КТ, а другая – ДПДГ. По окончании лечения каждого пациента просят заполнить специальную анкету самооценки уровня тревожности. Дизайн (план) эксперимента представлен в табл. 9.1.

При таком дизайне эксперимента способ лечения – это *межгрупповой* фактор (*between-groups factor*) с двумя уровнями (КТ и ДПДГ). Этот фактор называется *межгрупповым*, поскольку каждый пациент может быть отнесен к одной и только одной группе. Ни один из пациентов не подвергался действию и КТ, и ДПДГ. Буква «п» в таблице обозначает разных пациентов. Результат анкетирования – это *зависимая переменная*, а тип лечения – *независимая переменная*. Поскольку число наблюдений для каждого способа лечения равно, мы имеем дело со

Таблица 9.1. Однофакторный дисперсионный анализ для независимых переменных

Способ лечения	
КТ	ДПДГ
п1	п6
п2	п7
п3	п8
п4	п9
п5	п10

сбалансированным дизайном. Когда размер выборок для разных типов воздействия не одинаков, говорят о *несбалансированном дизайне* эксперимента.

Статистический дизайн, представленный в табл. 9.1, называется *однофакторным дисперсионным анализом*, поскольку имеется только одна классифицирующая переменная. Конкретнее в данном случае мы имеем дело с однофакторным дисперсионным анализом для независимых переменных (one-way between-groups ANOVA). Эффекты в разных дизайнах ANOVA сначала оцениваются при помощи F-тестов. Если такой тест для способа лечения дал значимый результат, можно считать, что средние баллы тревожности после пяти недель лечения разными способами различались.

Если бы вы заинтересовались результатом применения КТ по прошествии времени, вы могли бы поместить всех пациентов в одну группу с КТ и опросить их по окончании терапии и еще через шесть месяцев. Дизайн эксперимента представлен в табл. 9.2.

Время – это *внутригрупповой фактор* (within-groups factor) с двумя уровнями (пять недель, шесть месяцев). Этот фактор называется внутригрупповым, поскольку каждый пациент исследуется при обоих значениях этого фактора. Соответствующая статистическая процедура называется *однофакторный дисперсионный анализ для зависимых наблюдений* (one-way within-groups ANOVA). Поскольку каждый объект исследования измеряется более чем один раз, такой дизайн еще называют *дисперсионным анализом для повторных измерений* (repeated measures ANOVA). Если F-тест для переменной «время» дал значимый результат, то это значит, что средний уровень тревожности, отмеченный у пациентов через пять недель и шесть месяцев, различается.

Таблица 9.2. Однофакторный дисперсионный анализ для зависимых переменных

Пациент	Время	
	5 недель	6 месяцев
п1		
п2		
п3		
п4		
п5		
п6		
п7		
п8		
п9		
п10		

Если бы вы интересовались различиями как между способами лечения, так и изменениями результатов со временем, вы могли бы объединить описанные выше подходы, случайно распределив пациентов по двум равным группам (с применением КТ и ДПДГ) и проанализировав их уровень тревожности по завершении терапии (пять недель) и через шесть месяцев (см. табл. 9.3).

Таблица 9.3. Двухфакторный дисперсионный анализ с одной парой зависимых и одной парой независимых переменных

		Пациент	Время	
			5 недель	6 месяцев
Способ лечения	КТ	п1		
		п2		
		п3		
		п4		
		п5		
ДПДГ	ДПДГ	п6		
		п7		
		п8		
		п9		
		п10		

Используя и время, и способ лечения как факторы, можно изучить влияние способа лечения (усредненного по времени), времени (усредненного по способам лечения) и взаимодействие между способом лечения и временем. Первые два подхода называются *главными эффектами* (main effects), а взаимодействие (что неудивительно) называется *эффектом взаимодействия* (interaction effect).

Такое исследование сочетания нескольких факторов называется *многофакторным дизайном дисперсионного анализа* (factorial ANOVA design). Сочетание двух факторов – это двухфакторный дисперсионный анализ, сочетание трех факторов – трехфакторный и т. д. Когда многофакторный дизайн включает и межгрупповые, и внутригрупповые факторы, он также называется *смешанной моделью дисперсионного анализа* (mixed-model ANOVA). Описанный выше эксперимент – это двухфакторный дисперсионный анализ со смешанными эффектами (уф!).

В этом случае вы проводите три F-теста: один – для способа лечения, один – для временного интервала и один – для взаимодействия между этими двумя переменными. Достоверный результат для способа лечения означает, что КТ и ДПДГ различаются по их влиянию на уровень тревожности. Достоверный результат для временного интервала свидетельствует о том, что тревожность после окончания лечения и через шесть месяцев после этого различается. Достоверное взаимодействие между временем и лечением значит, что два способа терапии имеют разные временные эффекты (то есть изменения уровня тревожности за время между исследованиями различаются для двух способов лечения).

Теперь мы немного расширим дизайн эксперимента. Известно, что депрессия влияет на ход лечения, а также что депрессия и тревожность часто сопутствуют друг другу. Хотя пациенты распределялись по группам случайно, возможно, пациенты, получавшие разную терапию, исходно различались по уровню депрессии. Любые различия между этими группами могут объясняться исходными различиями в уровне депрессивности пациентов, а не экспериментальным воздействием. Поскольку уровень депрессии также может объяснять межгрупповые различия зависимой переменной, это *смешивающий фактор* (confounding factor). А поскольку вы не интересуетесь депрессией, это *мешающая переменная* (nuisance variable).

Если бы вы регистрировали уровень депрессивности при помощи анкеты для самооценки, вы могли бы учесть межгрупповые различия, которые имеют место из-за разницы в уровне депрессии, перед оценкой различий, связанных со способом лечения. В таком случае уровень депрессивности назывался бы *ковариатой* (covariate), а дизайн исследования носил бы название *ковариационный анализ* (analysis of covariance, ANCOVA).

Наконец, в этом исследовании вы изучали одну зависимую переменную (уровень тревожности). Можно увеличить ценность работы, добавив дополнительные оценки уровня тревожности (по мнению семьи, по отзывам терапевта, показатель влияния тревожности на повседневную жизнь). При наличии более чем одной зависимой переменной дизайн называется *многомерный дисперсионный анализ* (multivariate analysis of variance, MANOVA). Если в анализ входят ковариаты, то он будет называться *многомерный ковариационный анализ* (multivariate analysis of covariance, MANCOVA).

Теперь, вооружившись новой терминологией, вы готовы удивлять друзей, изумлять новых знакомых и обсуждать, как подгонять ANOVA/ANCOVA/MANOVA-модели в R.

9.2. Подгонка ANOVA-моделей

Хотя методологии ANOVA и регрессии развивались по отдельности, с функциональной точки зрения, это два частных случая общей линейной модели. Мы можем анализировать модели ANOVA при помощи той же функции `lm()`, которая использовалась для регрессионного анализа в главе 7. Однако в этой главе мы сначала обсудим функцию `aov()`. Результаты применения этих функций равнозначны, но `aov()` представляет результаты в более привычном для людей, работающих с ANOVA, формате. Для полноты изложения я приведу пример использования функции `lm()` в конце этой главы.

9.2.1. Функция `aov()`

Синтаксис применения функции `aov()` таков: `aov(формула, data=таблица_данных)`. В табл. 9.4 перечислены специальные символы, которые могут быть использованы в формулах. В этой таблице зависимая переменная обозначена как y , а буквы A , B и C соответствуют факторам.

Таблица 9.4. Специальные символы, используемые в формулах R

Символ	Использование
\sim	Разделяет зависимые переменные в левой части уравнения и независимые – в правой. Например, предсказание значений y по значениям A , B и C будет закодировано в виде $y \sim A + B + C$
$+$	Разделяет независимые переменные
$:$	Обозначает взаимодействие между переменными. Предсказание значений y по значениям A , B и взаимодействия между A и B будет закодировано в виде $y \sim A + B + A:B$
$*$	Обозначает все возможные сочетания переменных. Выражение $y \sim A*B*C$ расшифровывается как $y \sim A + B + C + A:B + A:C + B:C + A:B:C$
$^$	Обозначает сочетание определенного числа переменных. Выражение $y \sim (A+B+C)^2$ расшифровывается как $y \sim A + B + C + A:B + A:C + A:B:C$
$.$	Символ-заполнитель для всех остальных переменных в таблице данных, за исключением зависимой переменной. Например, если таблица данных содержит переменные y , A , B и C , то выражение $y \sim .$ расшифровывается как $y \sim A + B + C$

В табл. 9.5 приведены формулы для нескольких наиболее востребованных дизайнов исследования. В этой таблице строчные буквы – это количественные переменные, прописные буквы – это группирующие факторы, а *Объект* – это переменная, содержащая уникальные идентификаторы для объектов.

Таблица 9.5. Формулы для распространенных дизайнов экспериментов

Дизайн	Формула
Однофакторный дисперсионный анализ	$y \sim A$
Однофакторный ковариационный анализ с одной ковариатой	$y \sim x + A$
Двухфакторный дисперсионный анализ	$y \sim A * B$
Двухфакторный ковариационный анализ с двумя ковариатами	$y \sim x_1 + x_2 + A * B$
Случайный блоковый	$y \sim B + A$ (где B – определяющий блоки фактор)
Однофакторный дисперсионный анализ для зависимых переменных	$y \sim A + Error(Объект/A)$
Дисперсионный анализ с повторными изменениями с одним внутригрупповым фактором (W) и одним межгрупповым фактором (B)	$y \sim B * W + Error(Объект/W)$

Ниже в этой главе мы детально рассмотрим примеры для некоторых из этих дизайнов экспериментов.

9.2.2. Порядок членов в формуле

Порядок членов в формуле важен, если (а) есть больше одного фактора и дизайн несбалансирован или (б) есть ковариаты. Если одно из этих двух условий соблюдено, переменные с правой стороны уравнения будут коррелированы друг с другом. В таком случае не существует однозначного способа разделить их влияние на зависимую переменную.

Например, при двухфакторном дисперсионном анализе с неравным числом наблюдений для разных комбинаций факторов модель $y \sim A * B$ не даст того же результата, что и модель $y \sim B * A$.

По умолчанию в R применяется последовательный подход (I типа) для вычисления эффектов ANOVA (см. врезку «Упорядочите вычисления!»). Первая модель может быть сформулирована как

$y \sim A + B + A:B$. Итоговая таблица дисперсионного анализа в R будет оценивать:

- влияние А на y ;
- влияние В на y при постоянных значениях А;
- взаимодействие А и В при постоянных значениях главных эффектов А и В.

Упорядочите вычисления!

Если независимые переменные скоррелированы друг с другом или с ковариатами, не существует однозначного метода оценки влияния каждой из этих переменных на зависимую переменную. Рассмотрим несбалансированный двухфакторный дисперсионный анализ с факторами А и В и зависимой переменной y . У этого эксперимента три эффекта – главные эффекты А и В и взаимодействие между ними. Если вы моделируете данные при помощи формулы

$$Y \sim A + B + A:B,$$

существует три основных подхода для распределения дисперсии y между эффектами, перечисленными в правой части уравнения.

Тип I (последовательный)

Эффекты масштабируются по эффектам, которые указаны в формуле раньше. А не масштабируется, В масштабируется по А. Взаимодействие $A:B$ масштабируется по А и В.

Тип II (иерархический)

Эффекты масштабируются по тем эффектам, которые имеют такой же или более низкий уровень. А масштабируется по В, В масштабируется по А. Взаимодействие $A:B$ масштабируется и по А, и по В.

Тип III (краевой)

Каждый эффект масштабируется по любому другому эффекту в модели. А масштабируется по В и по $A:B$. В масштабируется по А и $A:B$. Взаимодействие $A:B$ масштабируется по А и В. В R по умолчанию реализуется тип I. В других программах, таких как SAS и SPSS, по умолчанию реализуется тип III.

Чем больше разница в размерах выборок, тем больше влияние порядка членов уравнения на результаты. В общем случае более важные эффекты должны стоять в формуле раньше. В частности, сначала нужно указывать ковариаты, затем главные эффекты, потом парные взаимодействия, далее – взаимодействия трех переменных и т. д. Из главных эффектов в первую очередь нужно указывать наиболее существенные. Вот главная идея: если дизайн исследования не ортогональный (то есть факторы и/или ковариаты скоррелированы), будьте аккуратны при определении порядка эффектов.

Перед тем как перейти к частным примерам, укажем, что функция Anova() из пакета car (не путайте с обычной функцией anova()) позволяет использовать подходы типа II или III, а не типа I, который реализован в функции aov(). Вы можете захотеть использовать функцию Anova(), если заинтересованы в соответствии ваших результатов тем, которые генерируются другими статистическими программами, такими как SAS или SPSS. Введите help(Anova, package="car"), чтобы узнать подробности.

9.3. Однофакторный дисперсионный анализ

При однофакторном дисперсионном анализе вы заинтересованы в сравнении средних значений зависимой переменной для двух и более групп, заданных категориальной группирующей переменной. Наш следующий пример основан на наборе заимствованных из публикации Westfall, Tobias, Rom, & Hochberg (1999) данных cholesterol из пакета multcomp. Пятьдесят пациентов проходили один из пяти снижающих уровень холестерина курсов лечения (переменная trt). Три из этих курсов заключались в использовании одного и того же препарата в количестве 20 мг один раз в день (1time), в количестве 10 мг дважды в день (2times) или 5 мг четыре раза в день (4times). Два других курса лечения заключались в приеме альтернативных препаратов (drugD и drugE). Какой из курсов лечения привел к наиболее заметному снижению уровня холестерина (переменная response)? Проведенный анализ представлен ниже.

Программный код 9.1. Однофакторный дисперсионный анализ

```
> library(multcomp)
> attach(cholesterol)
> table(trt)
trt
  1time 2times 4times drugD drugE
    10      10      10     10     10

> aggregate(response, by=list(trt), FUN=mean) ◀➁❶
  Group.1      x
1   1time  5.78
2   2times  9.22
3   4times 12.37
4   drugD 15.36
5   drugE 20.95
```

❶ **Объем выборки в каждой группе**

❷ **Средние значения в каждой группе**

```

> aggregate(response, by=list(trt), FUN=sd)
  Group.1      x
1   1time 2.88
2   2times 3.48
3   4times 2.92
4   drugD 3.45
5   drugE 3.35

```

3 Стандартные отклонения в каждой группе


```

> fit <- aov(response ~ trt)    ← 4 Тест на межгрупповые различия (ANOVA)
> summary(fit)

```

Графическое изображение средних значений с доверительными интервалами для каждой группы

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
trt	4	1351	338	32.4	9.8e-13 ***
Residuals	45	469	10		

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1					


```

> library(gplots)
> plotmeans(response ~ trt, xlab="Лечение", ylab="Эффект",
             main="Средние значения с 95%-ми доверительными интервалами")
> detach(cholesterol)

```

5

По выведенным на экран результатам можно видеть, что каждый способ лечения был испробован на 10 пациентах ①. Сравнение средних значений показывает, что препарат E (drugE) позволяет добиться наиболее заметного снижения уровня холестерина, а протокол time1 наименее эффективен ②. Стандартные отклонения во всех пяти группах были примерно одинаковыми, колеблясь от 2.88 до 3.48 ③. F-тест для типа лечения (trt) был статистически значимым ($p < .0001$), что свидетельствует о неодинаковой эффективности этих пяти способов лечения ④.

При помощи функции `plotmeans()` из пакета `gplots` можно построить диаграмму, отображающую средние значения и их доверительные интервалы для каждой группы ⑤. Такая диаграмма с 95% доверительными интервалами представлена на рис. 9.1. На ней хорошо видны межгрупповые различия.

9.3. 1. Множественные сравнения

Результат F-теста для способа лечения свидетельствует, что пять испробованных методов лечения неодинаково эффективны, однако отсюда нельзя понять, *какие именно* способы различаются между собой. Для ответа на этот вопрос можно использовать множественные сравнения. К примеру, функция `TukeyHSD()` позволяет провести тест на попарные различия между средними значениями для всех групп (см. программный код 9.2). Обратите внимание на то, что существуют

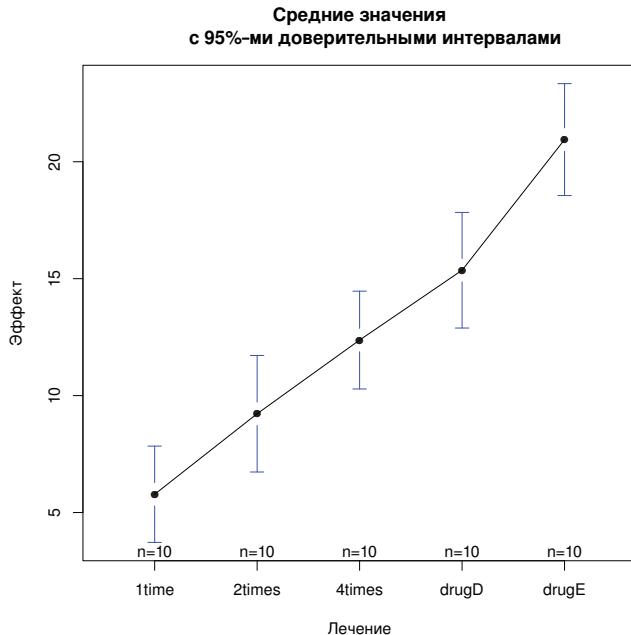


Рис. 9.1. Средние значения и 95%-ные доверительные интервалы для пяти курсов лечения, снижающих уровень холестерина

проблемы совместимости функции TukeyHSD() с пакетом HH, также используемым в этой главе; если этот пакет загружен, то функция не будет работать. В таком случае для удаления пакета из пути поиска используйте команду detach("package:HH"), а затем уже запускайте функцию TukeyHSD().

Программный код 9.2. Попарные межгрупповые сравнения с использованием функции TukeyHSD()

```
> TukeyHSD(fit)
Tukey multiple comparisons of means
 95% family-wise confidence level
Fit: aov(formula = response ~ trt)
$trt
            diff      lwr      upr p adj
2times-1time  3.44 -0.658   7.54 0.138
4times-1time  6.59  2.492 10.69 0.000
drugD-1time   9.58  5.478 13.68 0.000
drugE-1time  15.17 11.064 19.27 0.000
4times-2times  3.15 -0.951   7.25 0.205
drugD-2times   6.14  2.035 10.24 0.001
```

```
drugE-2times  11.72  7.621 15.82 0.000
drugD-4times   2.99 -1.115  7.09 0.251
drugE-4times   8.57  4.471 12.67 0.000
drugE-drugD    5.59  1.485  9.69 0.003
> par(las=2)
> par(mar=c(5,8,4,2))
> plot(TukeyHSD(fit))
```

Например, отсюда ясно, что средние уровни холестерина для протоколов лечения `1time` и `2times` статистически не отличается друг от друга ($p = 0.138$), тогда как разница между протоколами `1time` и `4times` значима ($p < .001$).

Попарные сравнения изображены на рис. 9.2. Первое выражение `par` разворачивает подписи по осям, а второе – увеличивает ширину левого поля так, чтобы подписи поместились на диаграмме (параметр `par` обсуждается в главе 3). На этой диаграмме доверительные интервалы, в которые входит 0, означают, что между данными двумя типами лечения нет статистически значимой разницы ($p > 0.05$).

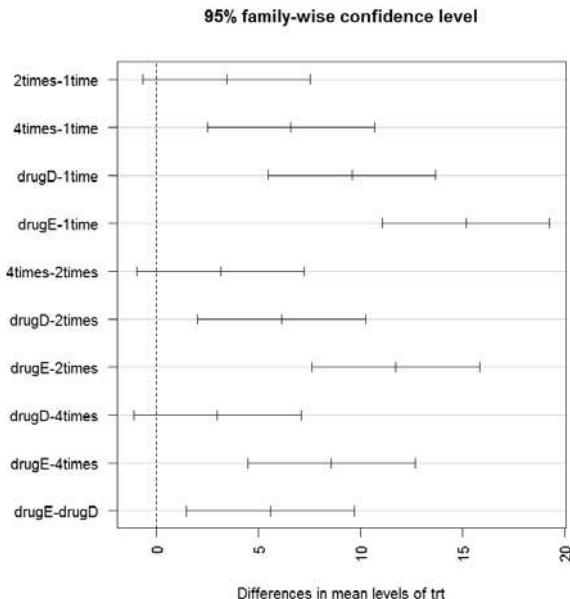


Рис. 9.2. Диаграмма для иллюстрации результатов попарных сравнений средних значений при помощи теста Тьюки

Функция `glht()` из пакета `multcomp` реализует набор более сложных методов сравнения нескольких средних значений. Эти методы

можно использовать как для линейных моделей (которые обсуждаются в этой главе), так и для обобщенных линейных моделей (описанных в главе 13). При помощи приведенного ниже программного кода можно провести тест честных оценок достоверности различий Тьюки (Tukey honest significant differences test) и графически представить его результаты (рис. 9.3).

```
> library(multcomp)
> par(mar=c(5,4,6,2))
> tuk <- glht(fit, linfct=mcp(trt="Tukey"))
> plot(cld(tuk, level=.05), col="lightgrey")
```

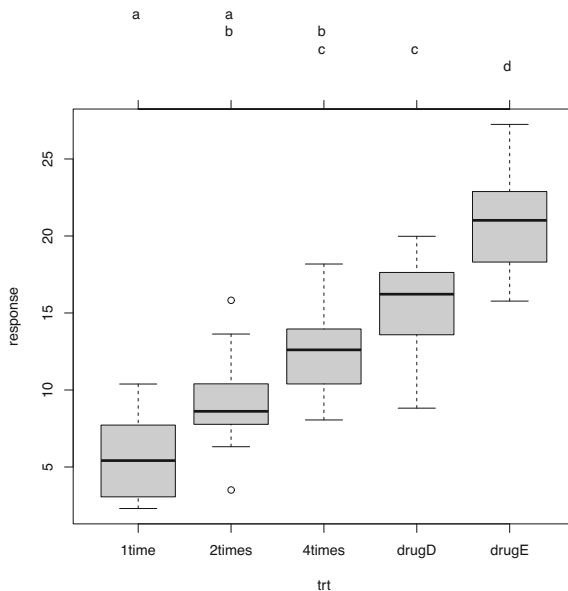


Рис. 9.3. Диаграмма для иллюстрации результатов теста Тьюки, реализованная в пакете `multcomp`

В данном программном коде высота верхнего поля диаграммы была увеличена при помощи параметра `par`. Параметр `level` функции `cld()` позволяет задать уровень значимости (в данном случае 0.05 или 95% уверенности).

Группы (которые представлены в виде диаграмм размахов) обозначены одинаковыми буквами, если их средние значения статистически не различаются. Отсюда можно видеть, что протоколы лечения `1time` и `2times` не различаются (они оба обозначены буквой *a*). Протоколы

2times и 4times также не различаются (обе обозначены буквой *b*). Лично мне легче интерпретировать рис. 9.3, чем рис. 9.2. Диаграмма, представленная на рис. 9.3, имеет дополнительное преимущество – на ней показано распределение значений признака в каждой группе.

Из этих результатов видно, что лучше принимать снижающее уро- вень холестерина лекарство по 5 мг четыре раза в день, чем по 20 мг один раз в день. Альтернативный препарат drugD не более эффе-ктивен, по сравнению с четырехразовым приемом первого препарата. Однако лекарство drugE действует лучше, чем все остальные методы лечения.

Методология множественных сравнений – это сложная и быстро изменяющаяся область исследований. Для получения дальнейшей информации ознакомьтесь с публикацией Bretz, Hothorn и Westfall (2010).

9.3.2. Проверка справедливости допущений, лежащих в основе теста

Как мы узнали из предыдущей главы, степень нашей уверенности в результатах зависит от степени соответствия данных допущениям, лежащим в основе статистических тестов. В случае однофакторного дисперсионного анализа предполагается, что значения зависимой пе-ременной распределены нормально и имеют одинаковую дисперсию в каждой группе. Для проверки нормальности распределения нужно использовать Q-Q-диаграмму:

```
> library(car)
> qqPlot(lm(response ~ trt, data=cholesterol),
  simulate=TRUE, main="Q-Q Plot", labels=FALSE)
```

Обратите внимание на то, что для создания этой диаграммы (рис. 9.4) необходимо построить регрессионную прямую при помо-щи команды `lm()`.

Данные хорошо укладываются в 95%-ные доверительные границы, это значит, что требование нормального распределения выполняется достаточно хорошо.

В R реализовано несколько тестов на однородность (гомогенность) дисперсий. Например, при помощи этого программного кода можно провести тест Бартлетта (Bartlett's test):

```
> bartlett.test(response ~ trt, data=cholesterol)
Bartlett test of homogeneity of variances
```

```
data: response by trt
Bartlett's K-squared = 0.5797, df = 4, p-value = 0.9653
```

Этот тест показал, что дисперсии в пяти группах статистически не различаются ($p = 0.97$). Другие возможные тесты – это тест Флигнера-Киллина (Fligner-Killeen test), который проводится при помощи функции fligner.test(), и тест Брауна-Форсайта (Brown-Forsythe test) – для него нужна функция hov() из пакета нн. Хотя результаты этих двух тестов и не приведены здесь, они позволяют прийти к тому же заключению.

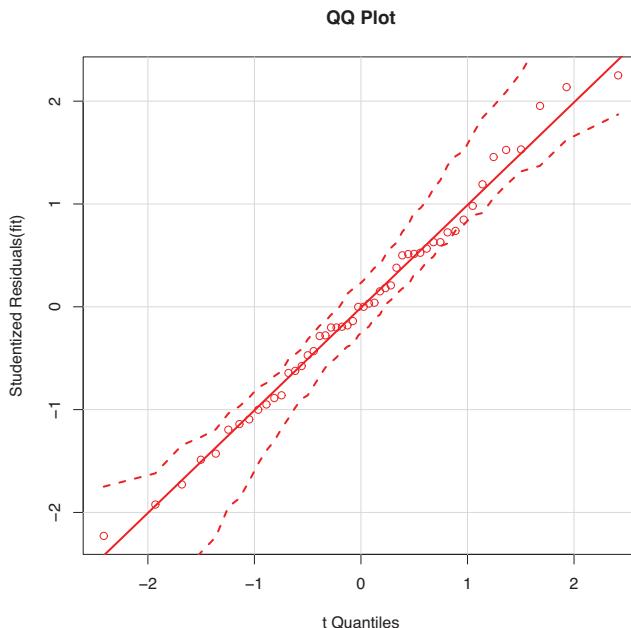


Рис. 9.4. Тест на нормальность распределения

Наконец, методы дисперсионного анализа могут быть чувствительны к выбросам. Проверить данные на их наличие можно при помощи функции outlierTest() из пакета car:

```
> library(car)
> outlierTest(fit)
No Studentized residuals with Bonferroni p < 0.05
Largest |rstudent|:
      rstudent unadjusted p-value Bonferroni p
19  2.251149          0.029422        NA
```

Полученные результаты означают, что в наборе данных cholesterol выбросы отсутствуют (NA приводится при $p > 1$). Рассмотрев вместе Q-Q-диаграмму, тест Барлетта и тест на наличие выбросов, можно сказать, что наши данные достаточно хорошо соответствуют модели дисперсионного анализа. Это, в свою очередь, добавляет уверенности в полученных результатах.

9.4. Однофакторный ковариационный анализ

Однофакторный ковариационный анализ расширяет однофакторный дисперсионный анализ добавлением одной или нескольких количественных ковариат. Приведенный пример основан на наборе данных litter из пакета multcomp (см. Westfall et al., 1999). Беременные мыши были разделены на четыре экспериментальные группы. Мышам из каждой группы давали разные дозы лекарства (0, 5, 50 или 500). Средний вес новорожденных мышат каждого помета был зависимой переменной, а время беременности было включено в модель как ковариата. Проведенный анализ представлен в следующем программном коде.

Программный код 9.3. Однофакторный ковариационный анализ

```
> data(litter, package="multcomp")
> attach(litter)
> table(dose)
dose
  0   5  50 500
 20 19 18 17
> aggregate(weight, by=list(dose), FUN=mean)
  Group.1    x
 1          0 32.3
 2          5 29.3
 3         50 29.9
 4        500 29.6
> fit <- aov(weight ~ gesttime + dose)
> summary(fit)
              Df  Sum Sq Mean Sq F value    Pr(>F)
gesttime      1 134.30 134.30  8.0493 0.005971 **
dose          3 137.12  45.71  2.7394 0.049883 *
Residuals     69 1151.27   16.69
---
Signif. codes:  0 '****' 0.001 '***' 0.01 '**' 0.05 '*' 0.1 '.' 1
```

При помощи функции `table()` можно узнать, что мыши из разных групп получали разные дозы препарата: 20 животных не подвергались действию лекарства, а 17 мышей получили 500 единиц препарата. На основании средних значений по группам можно сказать, что для не получавших лекарство мышей в целом характерен наибольший вес потомства (32.3). Результаты F-теста свидетельствуют о том, что (а) срок беременности связан с весом мышат при рождении и (б) количество лекарства влияло на вес мышат при постоянных значениях срока беременности. Средний вес мышат из разных экспериментальных групп неодинаков при постоянных значениях срока беременности.

Поскольку мы используем ковариату, вы можете захотеть выяснить скорректированные средние по группам – за вычетом эффекта ковариаты. Для вычисления скорректированных средних значений можно использовать функцию `effect()` из пакета `effects`:

```
> library(effects)
> effect("dose", fit)
dose effect
dose
  0    5   50   500
32.4 28.9 30.6 29.3
```

В данном случае скорректированные средние значения сходны с обычными средними, которые вычисляются при помощи функции `aggregate()`, но это не всегда бывает так. В пакете `effects` реализован мощный метод вычисления и графического представления скорректированных средних значений для исследований со сложным дизайном. Более подробная информация приведена в информационной базе CRAN.

Как и в примере с однофакторным дисперсионным анализом из предыдущего раздела, результаты F-теста для дозы препарата свидетельствуют, что средний вес мышат в разных экспериментальных группах неодинаков, однако не ясно, какие именно пары средних значений различаются. Вам вновь нужно использовать множественные попарные сравнения средних значений при помощи пакета `multcomp`. Кроме того, этот пакет можно использовать для проверки определенных пользовательских гипотез о средних значениях.

Предположим, вы интересуетесь, отличается ли группа, не получавшая лекарства, от трех групп, которым давали три разные дозы препарата. Для проверки этой гипотезы можно использовать приведенный ниже программный код.

Программный код 9.4. Множественные сравнения с использованием заданных пользователем контрастов

```
> library(multcomp)
> contrast <- rbind("no drug vs. drug" = c(3, -1, -1, -1))
> summary(glht(fit, linfct=mcp(dose=contrast)))
```

Multiple Comparisons of Means: User-defined Contrasts
Fit: aov(formula = weight ~ gesttime + dose)

Linear Hypotheses:

	Estimate	Std. Error	t value	Pr(> t)
no drug vs. drug == 0	8.284	3.209	2.581	0.0120 *

Signif. codes: 0 '****' 0.001 '***' 0.01 '**' 0.05 '*' 0.1 '.' 1

Контраст $c(3, -1, -1, -1)$ означает сравнение первой группы с усредненными по трем остальным группам значениями. Гипотеза проверяется при помощи t-статистики (ее значение в этом случае равно 2.581). Таким образом, вы можете утверждать, что не получавшая лекарств группа характеризуется большим весом новорожденных мышат, чем все другие экспериментальные группы. Другие контрасты можно добавлять при помощи команды `rbind()` (см. `help(glht)` для получения более подробной информации).

9.4.1. Проверка допущений, лежащих в основе теста

Ковариационный анализ основан на тех же допущениях о нормальности и гомогенности дисперсии, что и дисперсионный анализ. Соблюдение этих допущений можно проверить при помощи тех же методов, что были описаны в разделе 9.3.2. Кроме того, стандартный дизайн ковариационного анализа предполагает однородность углов наклона линий регрессии. В данном случае предполагается, что наклон регрессионной линии, отражающей зависимость веса мышат от срока беременности, одинаков для всех четырех экспериментальных групп. Проверку на однородность углов наклонов линий регрессии можно провести, включив взаимодействие между сроком беременности и дозой препарата в вашу ковариационную модель. Значимое взаимодействие будет означать, что характер связи между длительностью беременности и весом мышат зависит от дозы препарата. Программный код и результаты представлены в приведенном ниже программном коде.

Программный код 9.5. Проверка на гомогенность наклонов линий регрессии

```
> library(multcomp)
> fit2 <- aov(weight ~ gesttime*dose, data=litter)
> summary(fit2)

   Df Sum Sq Mean Sq F value Pr(>F)
gesttime      1    134     134    8.29 0.0054 **
dose          3    137      46    2.82 0.0456 *
gesttime:dose 3     82      27    1.68 0.1789
Residuals     66   1069     16
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Взаимодействие незначимо, что подтверждает справедливость предположения о равенстве наклонов линий регрессии. Если бы предположение не выполнялось, вы могли бы попробовать преобразовать ковариату или зависимую переменную с использованием модели, которая учитывает только отдельные углы наклона¹. Также можно применить непараметрический ковариационный анализ, для которого однородность углов наклона линий регрессии не обязательна. Этот метод может быть реализован при помощи функции `sm.ancova()` из пакета `sm`.

9.4.2. Визуализация результатов

Функция `ancova()` из пакета `HH` позволяет графически отобразить зависимость между зависимой переменной, ковариатой и фактором (независимой переменной). Например, команды

```
> library(HH)
> ancova(weight ~ gesttime + dose, data=litter)
```

создают диаграмму, приведенную на рис. 9.5. Учтите, что диаграмма была видоизменена, чтобы лучше воспроизвестись в черно-белом виде, и поэтому будет выглядеть несколько иначе, если вы самостоятельно выполните приведенные команды.

Здесь можно видеть, что регрессионные линии, аппроксимирующие зависимость веса новорожденных мышат от длительности беременности в разных экспериментальных группах, параллельны, но имеют разные свободные члены. С увеличением длительности беременности возрастает вес мышат. Кроме того, можно видеть, что в группе, где препарат не применялся, свободный член максимальен, а в группе с

1 На самом деле в одну ковариационную модель могут быть включены разные регрессионные коэффициенты. – Прим. пер.

наибольшей дозой препарата свободный член минимален. Линии параллельны, поскольку вы их так задали. Если бы вместо этого вы использовали выражение `ancova(weight ~ gesttime*dose)`, можно было бы наблюдать межгрупповую изменчивость как наклонов, так и свободных членов. Такой подход полезен в случае, когда требование однородности углов наклона линий регрессии не выполняется.

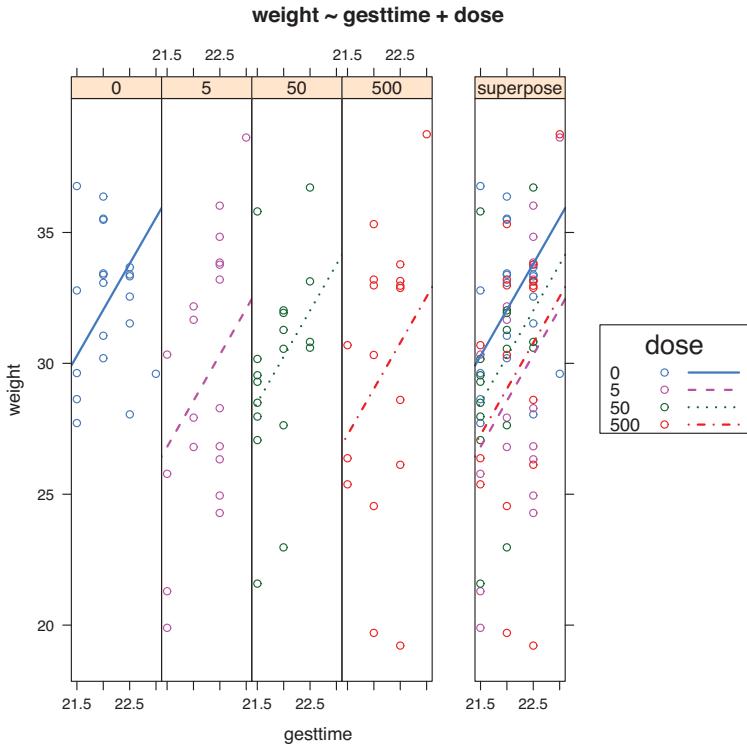


Рис. 9.5. Графическое отображение зависимости между длительностью беременности и весом новорожденных мышат для каждой экспериментальной группы

9.5. Двухфакторный дисперсионный анализ

При двухфакторном дисперсионном анализе объекты распределяются по группам, которые задаются комбинацией двух факторов.

Мы познакомимся с двухфакторным дисперсионным анализом для зависимых переменных на примере набора данных ToothGrowth, поставляющегося с базовой версией программы. Шестьдесят морских свинок были случайно распределены по группам, получавшим три разных количества аскорбиновой кислоты (`dose`: 0.5, 1 или 2 мг) двумя способами (`supp`: апельсиновый сок – OJ или витамин C – VC) – по 10 свинок в каждой из шести групп. Зависимая переменная – это длина зубов (`len`). Ниже приведен программный код для анализа.

Программный код 9.6. Двухфакторный дисперсионный анализ

```
> attach(ToothGrowth)
> table(supp, dose)
  dose
supp 0.5   1   2
  OJ  10 10 10
  VC  10 10 10
> aggregate(len, by=list(supp, dose), FUN=mean)
  Group.1 Group.2      x
1       OJ     0.5 13.23
2       VC     0.5  7.98
3       OJ     1.0 22.70
4       VC     1.0 16.77
5       OJ     2.0 26.06
6       VC     2.0 26.14
> aggregate(len, by=list(supp, dose), FUN=sd)
  Group.1 Group.2      x
1       OJ     0.5 4.46
2       VC     0.5 2.75
3       OJ     1.0 3.91
4       VC     1.0 2.52
5       OJ     2.0 2.66
6       VC     2.0 4.80
> fit <- aov(len ~ supp*dose)
> summary(fit)

Df Sum Sq Mean Sq F value Pr(>F)
supp        1    205     205   12.32 0.0009 ***
dose        1   2224    2224  133.42 <2e-16 ***
supp:dose   1      89      89    5.33 0.0246 *
Residuals  56    934     17
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Результат применения команды `table` указывает, что дизайн эксперимента сбалансирован (одинаковое число объектов во всех экспериментальных группах), а функция `aggregate` позволяет вычислить средние и стандартные отклонения для каждой группы. Результаты дисперсионного анализа, представленные в виде таблицы при по-

мощи функции `summary`, указывают на то, что оба главных эффекта (`supp` и `dose`) и взаимодействие между ними значимы.

Эти результаты можно визуализировать несколькими способами. Взаимодействие между факторами можно проиллюстрировать при помощи функции `interaction.plot()`. Программный код таков:

```
interaction.plot(dose, supp, len, type="b",
                  col=c("red","blue"), pch=c(16, 18),
                  main = "Взаимодействие между дозой и способом
→ ее получения")
```

Полученная диаграмма представлена на рис. 9.6. На ней изображена средняя длина зубов при всех трех дозах аскорбиновой кислоты и двух способах ее получения.

**Взаимодействие между дозой аскорбиновой кислоты
и способом ее получения**

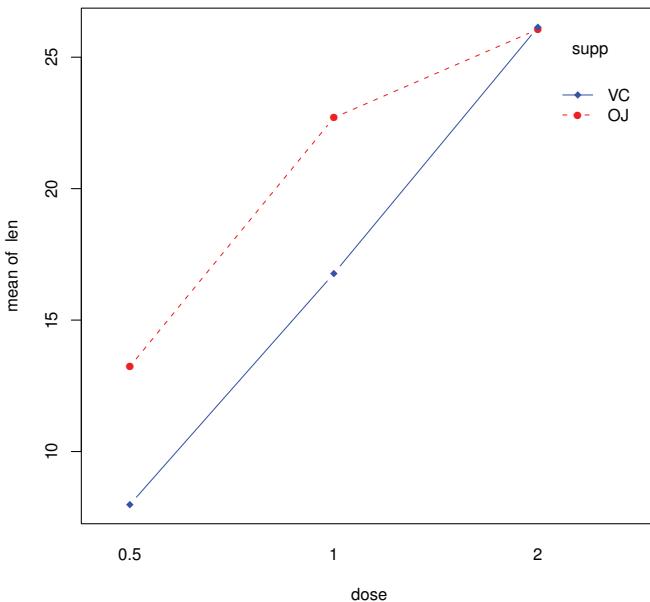


Рис. 9.6. Взаимодействие между дозой аскорбиновой кислоты и способом ее получения, оцененное по результатам измерения длины зубов (mean of len). Графическое отображение средних значений получено при помощи функции `interaction.plot()`

Приложив немного усилий, можно построить диаграмму взаимодействий при помощи команды `plotmeans()` из пакета `gplots`. При-

веденный ниже программный код позволяет получить диаграмму, представленную на рис. 9.7:

```
library(gplots)
plotmeans(len ~ interaction(supp, dose, sep=" "),
          connect=list(c(1,3,5),c(2,4,6)),
          col=c("red", "darkgreen"),
          main = "Диаграмма взаимодействия с 95%-ми доверительными
        ↪интервалами",
          xlab= "Комбинация дозы и способа ее получения")
```

На этой диаграмме отображены средние значения наряду с диапазонами точности (95%-ми доверительный интервал) и размерами выборок.

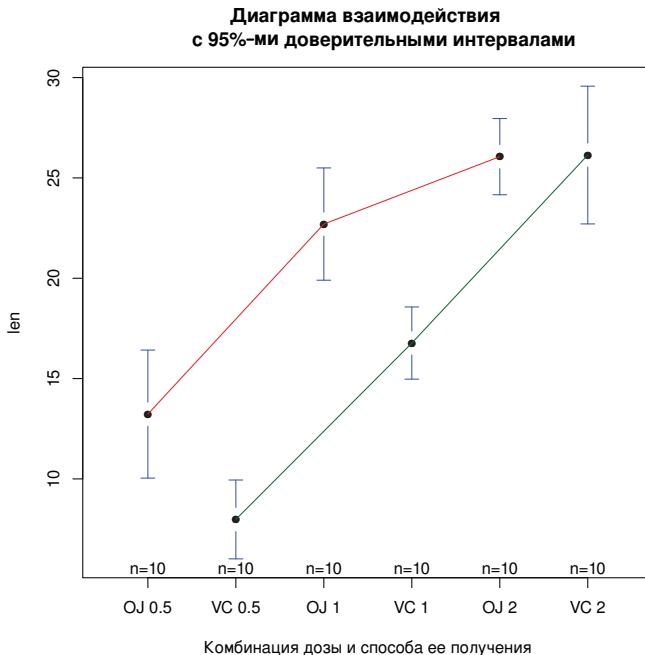


Рис. 9.7. Взаимодействие между дозой и способом ее получения, оцененное по результатам измерения длины зубов. Графическое отображение средних значений с 95%-ми доверительными интервалами получено при помощи функции `plotmeans()`

Наконец, для одновременного отображения главных эффектов и попарных взаимодействий между ними для эксперимента любого дизайна (рис. 9.8) можно использовать функцию `interaction2wt()` из пакета `HH`:

```
library(HH)
interaction2wt(len~supp*dose)
```

Опять же, эта диаграмма была видоизменена, чтобы она лучше выглядела в черно-белом варианте. Она будет иметь немного другой вид, если вы выполните данные команды самостоятельно.

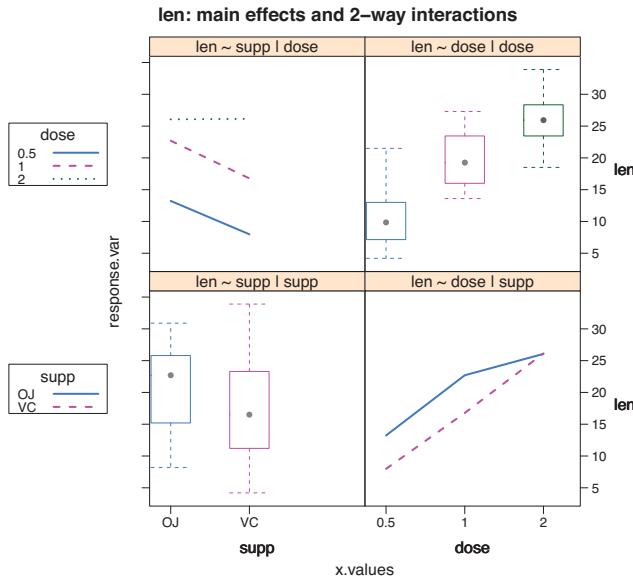


Рис. 9.8. Два главных эффекта и взаимодействие между ними в наборе данных ToothGrowth.

Эта диаграмма создана при помощи функции `interaction2way()`

Из всех трех диаграмм видно, что рост зубов ускоряется при увеличении дозы аскорбиновой кислоты, поступившей в форме как апельсинового сока, так и витамина С. Для доз в 0.5 и 1 мг апельсиновый сок оказывает более выраженный положительный эффект, по сравнению с витамином С. Для дозы в 2 мг обе формы потребления аскорбиновой кислоты одинаково хороши. Из всех трех представленных методов я предпочитаю функцию `interaction2wt()` из пакета HH, поскольку она отображает и главные эффекты, и попарные взаимодействия между ними для экспериментальных дизайнов любой сложности (двухфакторный дисперсионный анализ, трехфакторный дисперсионный анализ и т. д.).

Хотя я не обсуждал проверку справедливости допущений, лежащих в основе моделей, и сравнение средних, эти процедуры представ-

ляют собой естественное расширение методов, с которыми вы уже познакомились. Кроме того, если дизайн вашего эксперимента сбалансирован, вам не нужно беспокоиться о порядке эффектов.

9.6. Дисперсионный анализ для повторных измерений

При дисперсионном анализе для повторных измерений объекты измеряются более чем один раз. В этом разделе обсуждается дисперсионный анализ для повторных измерений с одним определяющим группы фактором и одним внутригрупповым фактором (обычный дизайн эксперимента). Мы рассмотрим пример из области экологической физиологии. Этот раздел биологии исследует, как протекание физиологических и биохимических процессов в живых системах зависит от условий обитания (важнейшая тема исследований, учитывая реалии глобального потепления). Набор данных CO2, входящий в базовую версию программы, содержит результаты исследования холодоустойчивости северных и южных популяций злака ежовника (*Echinochloa crus-galli*: Potvin, Lechowicz, Tardif, 1990). Сравнивали интенсивность фотосинтеза охлажденных и неохлажденных растений при разных концентрациях углекислого газа в окружающей среде. Половина растений происходила из Квебека, а половина – из штата Миссисипи.

В этом примере мы сосредоточимся на охлажденных растениях. Зависимая переменная – это уровень потребления углекислого газа (`uptake`) в мл/л, а независимые переменные (факторы) – это штат (`type`: Квебек или Миссисипи) и концентрация углекислого газа в окружающей среде (`conc`: семь концентраций от 95 до 1000 $\mu\text{моль}/\text{м}^2\text{ сек}$). Штат – это разделяющий группы фактор, а концентрация – внутригрупповой фактор. Анализ представлен в следующем программном коде.

Программный код 9.7. Дисперсионный анализ для повторных измерений с одним разделяющим группы фактором и одним внутригрупповым фактором

```
> w1b1 <- subset(CO2, Treatment=='chilled')
> fit <- aov(uptake ~ conc*Type + Error(Plant/(conc), w1b1))
> summary(fit)
Error: Plant
      Df  Sum Sq Mean Sq F value    Pr(>F)
Type       1 2667.24 2667.24  60.414 0.001477 ***
Residuals  4   176.60   44.15
---
Signif. codes:  0 '****' 0.001 '***' 0.01 '**' 0.05 '*' 0.1 '.' 1
```

```
Error: Plant:conc
      Df Sum Sq Mean Sq F value    Pr(>F)
conc       1 888.57 888.57 215.46 0.0001253 ***
conc:Type  1 239.24 239.24  58.01 0.0015952 **
Residuals  4   16.50     4.12
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Error: Within
      Df Sum Sq Mean Sq F value    Pr(>F)
Residuals 30 869.05   28.97
> par(las=2)
> par(mar=c(10,4,4,2))
> with(wlbl1, interaction.plot(conc, Type, uptake,
+ type="b", col=c("red","blue"), pch=c(16,18),
+ main="Диаграмма взаимодействий для региона и концентрации"))
> boxplot(uptake ~ Type*conc, data=wlbl1, col=(c("gold", "green")),
+ main="Охлажденные растения из Квебека и Миссисипи",
+ ylab="Интенсивность потребления CO2 (ммоль/м^2 сек)")


```

Результаты дисперсионного анализа, представленные в таблице, свидетельствуют о том, что главные эффекты (штат и концентрация), а также взаимодействие между ними значимы на уровне $p < 0.01$. Диаграмма, иллюстрирующая взаимодействия между факторами, представлена на рис. 9.9.

Диаграмма взаимодействий для региона и концентрации

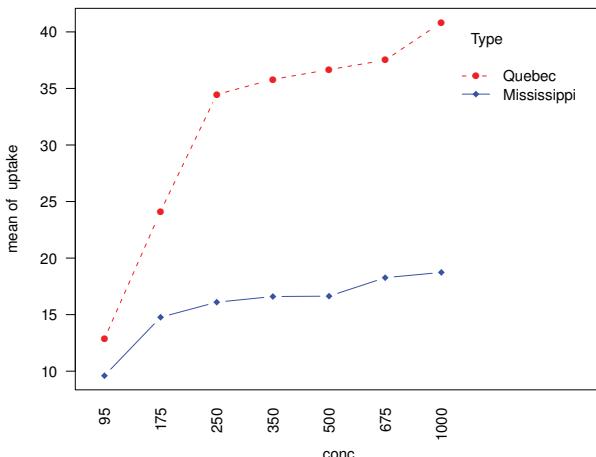


Рис. 9.9. Зависимость потребления углекислого газа от взаимодействия между концентрацией углекислого газа в окружающей среде и географическим происхождением растения. Диаграмма построена при помощи функции `interaction.plot()`

Для демонстрации другого способа графического отображения взаимодействий для тех же данных я использовал функцию `boxplot()` (рис. 9.10).

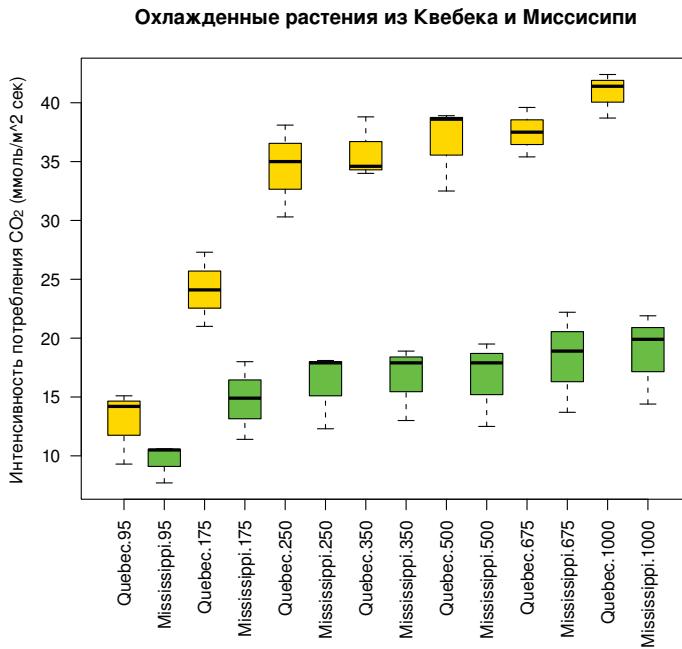


Рис. 9.10. Зависимость потребления углекислого газа от взаимодействия между концентрацией углекислого газа в окружающей среде и географическим происхождением растения. Диаграмма построена при помощи функции `boxplot()`

На обеих диаграммах видно, что у растений из Квебека интенсивность потребления углекислого газа выше, чем у растений из Миссисипи. Различия наиболее ярко выражены при высоких концентрациях углекислого газа в окружающей среде.

Замечание. Обычно вы работаете с данными в широком формате, когда столбцы – это переменные, строки – наблюдения, а каждому объекту соответствует отдельная строка. Хороший пример такого формата – набор данных `litter`, обсуждавшийся в разделе 9.4. Когда имеешь дело с повторными измерениями, обычно для подбора моделей нужны данные в длинном формате. В таком формате каждому измерению зависимой переменной соответствует своя строка. Набор данных CO_2 представлен именно в таком формате. К счастью, при помощи пакета `reshape`, описанного в главе 5 (раздел 5.6.3), можно с легкостью преобразовать данные в нужный формат.

Многообразие подходов к анализу экспериментов со смешанными эффектами

Пример с углекислым газом был проанализирован с использованием традиционного дисперсионного анализа для повторных наблюдений. Этот подход подразумевает, что ковариационная матрица для любого внутригруппового фактора обладает определенным свойством, известным под названием *сферичность*. Это значит, что дисперсии разниц между любыми двумя уровнями внутригруппового фактора одинаковы. Маловероятно, чтобы такое допущение выполнялось в реальном мире. Это соображение привело к возникновению ряда альтернативных подходов, включая такие:

- использование функции `lmer()` из пакета `lme4` для подгонки линейных моделей со смешанными эффектами (Bates, 2005);
- использование функции `Anova()` из пакета `cag` для корректировки обычных статистик с учетом отсутствия сферичности, например поправка Гейссера-Гринхауса (Geisser-Greenhouse);
- использование функции `gls()` из пакета `nlme` для подгонки обобщенных моделей методом наименьших квадратов с заданной структурой дисперсии и ковариации (UCLA, 2009);
- использование многомерного дисперсионного анализа для моделирования результатов повторных измерений (Hand, 1987).

Обсуждение этих подходов выходит за рамки данного пособия. Если вы хотите узнать больше, обратитесь к публикациям Pinheiro, Bates (2000) и Zuur et al. (2009).

До этого момента все методы подразумевали наличие одной зависимой переменной. В следующем разделе мы кратко рассмотрим дизайны экспериментов с более чем одной зависимой переменной.

9.7. Многомерный дисперсионный анализ

Можно исследовать несколько зависимых переменных одновременно при помощи многомерного дисперсионного анализа (multivariate analysis of variance, MANOVA). Мы рассмотрим этот подход на примере набора данных `UScereal` из пакета `MASS` (Venables, Ripley, 1999). В этом примере мы узнаем, как зависит содержание калорий (`calories`), жиров (`fat`) и углеводов (`sugars`) в американских сухих завтраках от того, на какой полке в магазине они стоят (`shelf`: 1 – нижняя полка, 2 – средняя, 3 – верхняя). Количество калорий, жиров и углеводов – это зависимые переменные, а номер полки – независимая переменная с тремя уровнями (1, 2 и 3). Анализ представлен в следующем программном коде.

Программный код 9.8. Однофакторный многомерный дисперсионный анализ

```

> library(MASS)
> attach(UScereal)
> y <- cbind(calories, fat, sugars)
> aggregate(y, by=list(shelf), FUN=mean)
  Group.1 calories      fat sugars
1          1     119 0.662    6.3
2          2     130 1.341   12.5
3          3     180 1.945   10.9
> cov(y)
            calories      fat sugars
calories   3895.2 60.67 180.38
fat         60.7  2.71  4.00
sugars     180.4  4.00 34.05
> fit <- manova(y ~ shelf)
> summary(fit)

  Df Pillai approx F num Df den Df Pr(>F)
shelf      1 0.1959   4.9550      3     61 0.00383 ***
Residuals 63
---
Signif. codes: 0 '****' 0.001 '***' 0.01 '**' 0.05 '*' 0.1 '.' 1

> summary.aov(fit)           ◀— Вывод результатов для каждого признака
  Response calories :
  Df Sum Sq Mean Sq F value Pr(>F)
shelf      1 45313 45313 13.995 0.0003983 ***
Residuals 63 203982 3238
---
Signif. codes: 0 '****' 0.001 '***' 0.01 '**' 0.05 '*' 0.1 '.' 1

  Response fat :
  Df Sum Sq Mean Sq F value Pr(>F)
shelf      1 18.421 18.421 7.476 0.008108 **
Residuals 63 155.236 2.464
---
Signif. codes: 0 '****' 0.001 '***' 0.01 '**' 0.05 '*' 0.1 '.' 1

  Response sugars :
  Df Sum Sq Mean Sq F value Pr(>F)
shelf      1 183.34 183.34 5.787 0.01909 *
Residuals 63 1995.87 31.68
---
Signif. codes: 0 '****' 0.001 '***' 0.01 '**' 0.05 '*' 0.1 '.' 1

```

В этом программном коде для создания матрицы трех зависимых переменных (калории, жиры, углеводы) использована функция `cbind()`. При помощи функции `aggregate()` рассчитываются средние значения для полок, а функция `cov()` вычисляет дисперсию и ковариацию по злакам.

Функция `manova()` осуществляет многомерный тест на межгрупповые различия. Статистически значимая величина F-статистики свидетельствует о том, что наши три группы злаков различаются по питательной ценности.

Поскольку результаты многомерного теста значимы, можно использовать функцию `summary.aov()` для выполнения ряда однофакторных дисперсионных анализов. В этом случае различия по каждому показателю пищевой ценности анализируются отдельно. Наконец, можно сравнить средние значения (при помощи функции `TukeyHSD`), чтобы выяснить, какие полки отличаются от каких по каждой из зависимых переменных (результат этого теста не приводится здесь для экономии места).

9.7.1. Проверка предположений, лежащих в основе теста

В основе однофакторного многомерного дисперсионного анализа лежат два допущения – многомерное нормальное распределение и однородность матриц дисперсии-ковариации.

Первое допущение заключается в том, что все зависимые переменные одновременно демонстрируют многомерное нормальное распределение. Для проверки выполнимости этого требования можно использовать Q-Q-диаграмму (для статистического обоснования см. врезку «Немного теории»).

Немного теории

Если у вас есть нормально распределенный многомерный вектор x размерности $r \times 1$ со средним значением μ и ковариационной матрицей Σ , то квадрат расстояния Махalanобиса между x и μ имеет распределение хи-квадрат с r степенями свободы. Q-Q-диаграмма отображает зависимость выборочных квантилей распределения хи-квадрат от квадрата расстояний Махalanобиса. Мы можем быть уверенными в многомерном нормальном распределении данных в той степени, в которой точки ложатся на линию $x=y$.

Ниже приведен необходимый для подобной проверки программный код, а полученная диаграмма представлена на рис. 9.11.

Программный код 9.9. Проверка нормальности многомерного распределения

```
> center <- colMeans(y)
> n <- nrow(y)
```

```

> p <- ncol(y)
> cov <- cov(y)
> d <- mahalanobis(y, center, cov)
> coord <- qqplot(qchisq(ppoints(n), df=p),
+ d, main="Q-Q диаграмма для проверки данных на многомерную
+ нормальность",
+ ylab="Расстояние Махаланобиса")
> abline(a=0, b=1)
> identify(coord$x, coord$y, labels=row.names(UScereal))

```

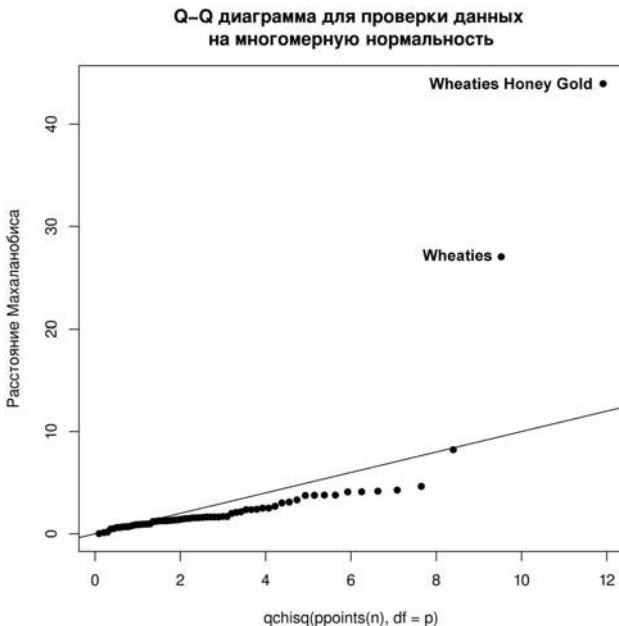


Рис. 9.11. Q-Q-диаграмма для проверки данных
на многомерную нормальность

Если данные соответствуют многомерному нормальному распределению, точки ложатся на линию. Функция `identify()` позволяет подписать точки на диаграмме в интерактивном режиме (эта функция обсуждается в главе 16, раздел 16.4). В нашем случае обнаруживается отклонение от многомерной нормальности в основном из-за сухих завтраков «Уитиз» (*Wheaties Honey Gold* и *Wheaties*). Возможно, эти случаи стоит удалить, а затем повторить анализ.

Однородность матриц дисперсии-ковариации подразумевает, что ковариационные матрицы для каждой из групп равны. Это предположение обычно проверяют при помощи M-теста Бокса (Box's M test).

В R нет функции для проведения этого теста, но в Интернете можно найти нужный программный код. К сожалению, этот тест чувствителен к отклонениям от нормальности, что в большинстве случаев приводит к отрицательным результатам. Это значит, что пока мы не располагаем хорошим рабочим методом для проверки этого важного допущения (однако описания альтернативных подходов, еще не реализованных в R, можно найти в публикациях Anderson (2006) и Silva et al. (2008)).

Наконец, можно провести тест на многомерные выбросы, используя функцию `aq.plot()` из пакета `mvoutlier`. Программный код в данном случае будет выглядеть так:

```
library(mvoutlier)
outliers <- aq.plot(y)
outliers
```

Попробуйте применить его и посмотрите, что получится!

9.7.2. Устойчивый многомерный дисперсионный анализ

Если требования многомерного нормального распределения или гомогенности матриц дисперсии-ковариации не выполняются, или вы беспокоитесь о многомерных выбросах, можно попробовать применить устойчивую (робастную) или непараметрическую версию многомерного дисперсионного анализа. Для этого можно использовать функцию `Wilks.test()` из пакета `rrcov`. Функция `adonis()` из пакета `vegan` реализует непараметрический аналог многомерного дисперсионного анализа. Программный код 9.10 демонстрирует применение функции `Wilks.test()` к нашему примеру.

Программный код 9.10. Устойчивый однофакторный многомерный дисперсионный анализ

```
library(rrcov)
> Wilks.test(y, shelf, method="mcd")
      Robust One-way MANOVA (Bartlett Chi2)
data: x
Wilks' Lambda = 0.511, Chi2-Value = 23.71, DF = 4.85, p-value =
0.0002143
sample estimates:
calories   fat sugars
1       120 0.701   5.66
2       128 1.185  12.54
3       161 1.652  10.35
```

Судя по результатам, устойчивый тест, нечувствительный и к выбросам, и к отклонениям от требований многомерного дисперсионного анализа, по-прежнему указывает на неравноценность сухих завтраков на верхней, средней и нижней полках магазинов по их питательной ценности.

9.8. Дисперсионный анализ как регрессия

В разделе 9.2 указано, что дисперсионный анализ и регрессия – это два частных случая одной и той же общей линейной модели. В таком случае все примеры в этой главе могли быть проанализированы при помощи функции `lm()`. Однако, чтобы понять результаты применения этой функции, вам нужно узнать, как R поступает с категориальными данными при подгонке моделей.

Рассмотрим задачу с одномерным дисперсионным анализом из раздела 9.3, в которой сравниваются пять снижающих уровень холестерина способов лечения (закодированные в переменной `trt`).

```
> library(multcomp)
> levels(cholesterol$trt)
[1] "ltime"   "2times"  "4times" "drugD"   "drugE"
```

Сначала подгоним модель при помощи функции `aov()`:

```
> fit.aov <- aov(response ~ trt, data=cholesterol)
> summary(fit.aov)
    Df  Sum Sq Mean Sq F value    Pr(>F)
trt      4 1351.37  337.84  32.433 9.819e-13 ***
Residuals 45  468.75   10.42
```

Теперь подгоним ту же модель при помощи функции `lm()`. В этом случае вы получите приведенные ниже результаты.

Программный код 9.11. Регрессионный подход к задаче по дисперсионному анализу из раздела 9.3

```
> fit.lm <- lm(response ~ trt, data=cholesterol)
> summary(fit.lm)
Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  5.782     1.021   5.665 9.78e-07 ***
trt2times    3.443     1.443   2.385  0.0213 *
trt4times    6.593     1.443   4.568 3.82e-05 ***
trtdrugD     9.579     1.443   6.637 3.53e-08 ***
trtdrugE    15.166     1.443  10.507 1.08e-13 ***
```

```
Residual standard error: 3.227 on 45 degrees of freedom
Multiple R-squared: 0.7425, Adjusted R-squared: 0.7196
F-statistic: 32.43 on 4 and 45 DF, p-value: 9.819e-13
```

Что мы тут видим? Поскольку для линейных моделей нужны числовые независимые переменные, то если функция `lm()` обнаруживает фактор, она замещает его набором числовых переменных, соответствующих контрастам между уровнями исходного фактора. Если у этого фактора k уровней, то будет создана $k - 1$ переменная. В R есть встроенные способы для создания таких контрастных переменных (см. табл. 9.6). Вы также можете создать свой способ (мы не будем обсуждать это здесь). По умолчанию контрасты комбинаций условий (`contr.treatment`) применяются для неупорядоченных факторов, а ортогональные полиномы (`contr.poly`) используются в случае упорядоченных факторов.

Таблица 9.6. Встроенные контрасты

Контраст	Описание
<code>contr.helmert</code>	Контрасты между первым уровнем и вторым, между третьим уровнем и средним значением для первых двух, между четвертым уровнем и средним значением для первых трех и т. д.
<code>contr.poly</code>	Контрасты, используемые для анализа трендов (линейных, квадратичных, кубических и т. д.), основаны на ортогональных полиномах. Применяются для упорядоченных факторов с одинаково различающимися уровнями
<code>contr.sum</code>	Контрасты ограничены условием равенства их суммы нулю. Также называемые контрастами отклонения, они сравнивают среднее значение для каждого уровня с общим средним по всем уровням
<code>contr.treatment</code>	Контрасты каждого уровня с базовым уровнем (по умолчанию первым). Также называется кодированием при помощи индикаторных переменных (<code>dummy coding</code>)
<code>contr.SAS</code>	Сходен с предыдущим способом, однако в качестве базового используется последний уровень. В результате получаются коэффициенты, сходные с контрастами в большинстве методов, реализованных в SAS

В случае контрастов комбинаций условий первый уровень фактора становится эталонной группой и каждый следующий уровень сравнивается с ним. С этой схемой перекодирования можно познакомиться при помощи функции `contrasts()`:

```
> contrasts(cholesterol$trt)
   2times 4times drugD drugE
1time      0      0      0      0
2times      1      0      0      0
4times      0      1      0      0
drugD       0      0      1      0
drugE       0      0      0      1
```

Если пациент находится в группе drugD, то эта переменная равна 1, а переменные 2times, 4times и drugE будут равны 0. Для первой группы переменная не нужна, поскольку нули во всех четырех индикаторных переменных однозначно указывают на то, что такой пациент находится в группе 1times.

В программном коде 9.11 переменная trt2times соответствует контрасту между уровнями фактора 1time и 2times. Аналогично trt4times – это контраст между 1time и 4times и т. д. По значениям вероятностей видно, что все способы лечения статистически значимо отличаются от первого (1time).

Контрасты, используемые функцией lm() по умолчанию, можно изменить при помощи опции contrasts(). Например, вы можете задать контрасты Хелмерта таким образом:

```
fit.lm <- lm(response ~ trt, data=cholesterol, contrasts="contr.helmert")
```

Изменять контрасты по умолчанию, используемые на протяжении всей сессии работы в R, можно при помощи функции options(). Например, выражение

```
options(contrasts = c(`contr.SAS`, `contr.helmert`))
```

привело бы к вычислению контрастов по умолчанию для неупорядоченных факторов по алгоритму contr.SAS, а для упорядоченных факторов – по алгоритму contr.helmert. Хотя мы ограничили наше обсуждение использованием контрастов в линейных моделях, учите, что они применяются и в других подгоняющих модели функциях R. Это справедливо и для обобщенных линейных моделей, обсуждаемых в главе 13.

9.9. Резюме

В этой главе мы рассмотрели анализ основных экспериментальных и квазиэкспериментальных дизайнов с использованием методологий дисперсионного, ковариационного и многомерного дисперсионного анализа. Мы обсудили основную используемую терминологию и по-

знакомились с примерами анализа для зависимых и независимых переменных, включая одно- и двухфакторный дисперсионный анализ, однофакторный ковариационный анализ, дисперсионный анализ для повторных измерений и однофакторный многомерный дисперсионный анализ.

В дополнение к этим основным методам мы рассмотрели способы проверки допущений, лежащих в основе моделей, и применение множественных сравнений вслед за получением статистически значимой величины универсального статистического критерия. Наконец, мы исследовали обширный набор методов графического представления результатов. Если вы хотите узнать больше о дизайне экспериментов в R, не забудьте прочесть соответствующий обзор в онлайн-архиве CRAN (Groemping, 2009).

В главах 8 и 9 были рассмотрены наиболее часто используемые учеными всех специальностей статистические методы. В следующей главе мы обсудим вопрос анализа мощности – важной составляющей планирования исследований. Анализ мощности помогает определить размеры выборок, необходимые для обнаружения эффекта заданной величины с заданной степенью уверенности.



ГЛАВА 10.

Анализ мощности

В этой главе:

- Определение необходимого размера выборки.
- Вычисление размера эффектов.
- Оценка статистической мощности.

Будучи консультантом по статистике, я часто слышу вопрос: «Сколько объектов мне нужно исследовать в своей работе?» Иногда вопрос принимает такую форму: «Я могу обследовать x человек. Есть ли смысл проводить исследование?» На вопросы вроде этого можно ответить при помощи *анализа мощности* – набора важных для планирования эксперимента методов.

Анализ мощности позволяет определить размер выборки, необходимый для выявления эффекта заданной величины с заданной долей уверенности. Верно и обратное, этот анализ позволяет оценить вероятность обнаружения эффекта заданной величины с заданной долей уверенности при данном объеме выборки. Если вероятность непримлемо мала, будет разумно изменить эксперимент или отказаться от него.

В этой главе вы узнаете, как провести анализ мощности для разных статистических тестов, включая тесты пропорций, критерий Стьюдента, хи-квадрат, сбалансированный однофакторный дисперсионный анализ, корреляционные тесты и линейные модели. Поскольку анализ мощности используется при проверке статистических гипотез, мы начнем с короткого обзора проверки значимости нулевых гипотез. Затем мы рассмотрим выполнение анализа мощности в R, в основном при помощи пакета `pwr`. Наконец, мы обсудим другие подходы к анализу мощности, которые реализованы в R.

10.1. Краткий обзор процедуры проверки гипотез

Для того чтобы помочь вам понять этапы анализа мощности, мы кратко обсудим общие вопросы проверки статистических гипотез. Если у вас уже есть представление об этом, можете спокойно перейти к разделу 10.2.

При проверке статистических гипотез вы формулируете гипотезу о параметре генеральной совокупности (вашу нулевую гипотезу, H_0). Затем вы создаете выборку из этой генеральной совокупности и вычисляете статистику, которая применяется для формирования заключений об этом параметре. Допуская, что нулевая гипотеза справедлива, вы вычисляете вероятность получения для генеральной совокупности наблюдаемого в выборке или большего значения статистики. Если вероятность достаточно мала, вы отвергаете нулевую гипотезу в пользу противоположной (ее называют альтернативной, или исследовательской, H_1).

Пример поможет прояснить ситуацию. Допустим, вы интересуетесь влиянием разговоров по мобильному телефону на время реакции водителя. Ваша нулевая гипотеза будет следующей: $\mu_1 - \mu_2 = 0$, где μ_1 – это среднее время реакции водителей, использующих мобильный телефон, а μ_2 – это среднее время реакции водителей без мобильника (в данном случае $\mu_1 - \mu_2$ – это интересующий нас параметр генеральной совокупности). Если вы отвергаете эту нулевую гипотезу, то остаетесь с альтернативной, а именно: $\mu_1 - \mu_2 \neq 0$. Это означает то же самое, что $\mu_1 \neq \mu_2$, то есть среднее время реакции при разных условиях не одинаково.

Далее создается выборка из людей, случайно отнесенных к той или иной экспериментальной группе. В первом случае участники исследования при работе на тренажере реагировали на ряд сложных для водителя ситуаций, разговаривая по мобильному телефону. Во втором случае люди выполняли то же задание, но уже без мобильного телефона. Для каждого человека регистрировали общее время реакции.

На основании выборочных данных можно вычислить статистику

$$(\bar{X}_1 - \bar{X}_2) / \left(\frac{S}{\sqrt{n}} \right),$$

где \bar{X}_1 и \bar{X}_2 – это средние для каждой выборки значения времени реакции, S – это стандартное отклонение для объединенной выбор-

ки, а n – число участников в каждой группе. Если нулевая гипотеза справедлива и вы можете утверждать, что значения времени реакции нормально распределены, значения выборочной статистики будут подчиняться распределению Стьюдента с $2n - 2$ степенями свободы. Используя этот факт, можно вычислить вероятность того, что выборочная статистика будет не меньше значения этой статистики для генеральной совокупности. Если вероятность не достигает некоторого принятого критического значения (скажем, $p < 0.05$), то вы отвергаете нулевую гипотезу в пользу альтернативной. Заранее определенное критическое значение (0.05) называется *уровнем значимости* теста.

Обратите внимание на то, что мы используем *выборочные данные*, чтобы делать заключения в отношении *генеральной совокупности*, из которой происходит эта выборка. В данном случае нулевая гипотеза заключается в том, что среднее время реакции *всех* водителей, которые говорили по мобильному телефону, не отличается от среднего времени реакции *всех* водителей, которые говорили по нему. Здесь не идет речь только о водителях из вашей выборки. У принятого решения может быть четыре возможных последствия:

- Если нулевая гипотеза не выполняется и мы отвергли ее на основании статистического теста, мы приняли верное решение. Мы правильно заключили, что использование мобильного телефона влияет на время реакции.
- Если нулевая гипотеза справедлива и мы не отвергаем ее, мы снова правы. В таком случае время реакции не зависит от использования мобильного телефона.
- Если нулевая гипотеза справедлива, но мы отвергли ее, мы совершили статистическую ошибку первого типа. Мы решили, что использование мобильного телефона влияет на время реакции, а это не так.
- Если нулевая гипотеза ложна, а мы не сумели ее отвергнуть, мы совершили статистическую ошибку второго типа. Мобильные телефоны влияют на время реакции, но мы не смогли это выявить.

Каждый из этих случаев проиллюстрирован ниже в таблице.

		Решение	
		Отвергаем H_0	Не смогли отвергнуть H_0
На самом деле	H_0 справедлива	Ошибка I типа	Верно
	H_0 ложна	Верно	Ошибка II типа

Противоречия, сопутствующие проверке нулевой гипотезы на значимость

Существуют разногласия по поводу проверки нулевой гипотезы на значимость. Скептики выдвигают многочисленные возражения по поводу этого подхода, особенно в психологии. Они указывают на непонимание значений p широкими массами, опору на статистическую значимость, а не значимость, вытекающую из здравого смысла, на тот факт, что нулевая гипотеза никогда не выполняется абсолютно точно и будет обязательно отвергнута при достаточно большом объеме выборки, а также на целый ряд логических неувязок при проверке нулевой гипотезы на значимость.

Подробное обсуждение этой темы выходит за рамки данной книги. Заинтересованные читатели могут обратиться к публикации Harlow, Mulaik, Steiger (1997).

При планировании исследования ученый обычно обращает особое внимание на четыре величины: размер выборки, уровень значимости, мощность теста и размер эффекта (см. рис. 10.1).

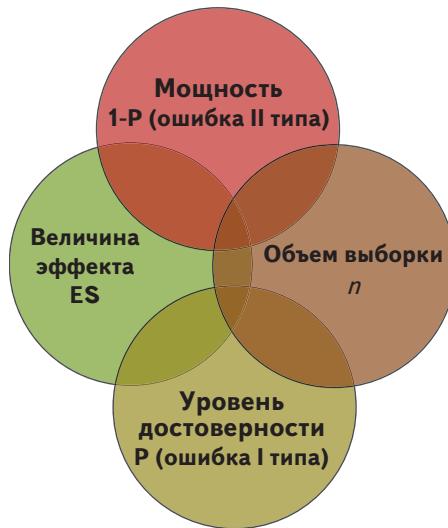


Рис. 10.1. Четыре основных параметра, которые

рассматриваются при анализе мощности.

Зная любые три, можно вычислить четвертый

Если быть более точными:

- размер выборки – число наблюдений при каждом типе воздействия или в каждой группе;

- *уровень значимости* (часто обозначаемый как альфа) – вероятность совершения статистической ошибки первого типа. Уровень значимости можно также трактовать как вероятность нахождения несуществующей закономерности;
- *мощность теста* (1 – вероятность совершения статистической ошибки второго типа). Мощность можно также понимать как вероятность обнаружения *существующей* закономерности;
- *размер эффекта* – величина эффекта, который является предметом альтернативной гипотезы. Формула для нахождения размера эффекта зависит от статистической методологии, используемой при проверке гипотезы.

Хотя размер выборки и уровень значимости находятся под непосредственным контролем исследователя, мощность и размер эффекта определяются не так легко. Например, если вы повышаете критическое значение уровня значимости (другими словами, опровергнуть нулевую гипотезу становится легче), то мощность возрастает. Сходным образом увеличение размера выборки повышает мощность.

Обычно ваша задача при исследовании – повысить мощность статистических тестов, сохранив приемлемый уровень значимости и работая с минимально возможной выборкой. Таким образом вы хотите увеличить шансы обнаружения существующей закономерности и свести к минимуму вероятность обнаружить отсутствующую закономерность при разумном бюджете.

Эти четыре величины (размер выборки, уровень значимости, мощность и величина эффекта) тесно взаимосвязаны. *Зная значения любых трех, можно определить четвертую.* Мы используем этот факт при выполнении различных анализов мощности на протяжении всей главы. В следующем разделе мы познакомимся со способами проведения анализа мощности с использованием пакета *rwr*. Позже мы кратко рассмотрим некоторые узкоспециализированные функции, которые используются в биологии и генетике.

10.2. Проведение анализа мощности при помощи пакета *rwr*

Пакет *rwr*, созданный Стефаном Чемпили (Stéphane Champely), позволяет выполнить анализ мощности по алгоритмам, описанным в публикации Cohen (1988). Некоторые важные функции приведены

в табл. 10.1. Для каждой функции пользователь может указать три из четырех величин (размер выборки, уровень значимости, мощность и размер эффекта), а четвертая будет вычислена.

Таблица 10.1. Функции из пакета `pwr`

Функция	Вычисляет мощность для
<code>pwr.2p.test()</code>	Двух пропорций (равные объемы выборок)
<code>pwr.2p2n.test()</code>	Двух пропорций (неодинаковые объемы выборок)
<code>pwr.anova.test()</code>	Сбалансированного однофакторного дисперсионного анализа
<code>pwr.chisq.test()</code>	Теста хи-квадрат
<code>pwr.f2.test()</code>	Общей линейной модели
<code>pwr.p.test()</code>	Пропорции (одна выборка)
<code>pwr.r.test()</code>	Корреляции
<code>pwr.t.test()</code>	Тестов Стьюдента (одна выборка, две выборки, для зависимых переменных)
<code>pwr.t2n.test()</code>	Тестов Стьюдента (две выборки разного объема)

10.2.1. Тесты Стьюдента

Когда нужно использовать тест Стьюдента, функция `pwr.t.test` предоставляет много полезных для анализа мощности возможностей. Формат ее применения таков:

`pwr.t.test(n=, d=, sig.level=, power=, type=, alternative=)`,

где:

- n – размер выборки;
- d – величина эффекта, выраженная в стандартизованной разнице средних значений, которая рассчитывается по формуле

$$d = \frac{(\mu_1 - \mu_2)}{\sigma}$$
, где μ_1 – среднее значение для первой группы,
 μ_2 – среднее значение для второй группы,
 σ – дисперсия общей ошибки;
- `sig.level` – уровень значимости (по умолчанию 0.05);
- `power` – уровень мощности;
- `type` – тип теста: для двух выборок ("two.sample", по умолчанию), для одной выборки ("one.sample"), для зависимых выборок ("paired");
- `alternative` – двухсторонний ("two.sided", по умолчанию) или односторонний ("less" или "greater").

Рассмотрим это подробнее на примере. Продолжив обсуждение эксперимента с мобильным телефоном и скоростью реакции из раздела 10.1, допустим, что вы используете двухсторонний тест Стьюдента для независимых выборок, чтобы сравнить среднее время реакции для водителей с мобильным телефоном и для водителей без отвлекающих факторов.

Предположим, по предыдущим исследованиям вам известно, что стандартное отклонение времени реакции составляет 1.25 секунды. Также предположим, что различие во времени реакции на одну секунду считается существенным. Таким образом, вам хотелось бы провести исследование, в котором можно было обнаружить эффект величиной $d = 1/1.25 = 0.8$ или больше. В дополнение к этому вы хотите быть уверенными на 90% в том, что действительно обнаружите различия, если они существуют, и на 95% – в том, что вы не назовете значимым то различие, которое на самом деле является результатом случайных флюктуаций. Сколько человек вам нужно обследовать?

Введя эту информацию в функцию `pwr.t.test()`, вы получите следующий результат:

```
> library(pwr)
> pwr.t.test(d=.8, sig.level=.05, power=.9, type="two.sample",
  alternative="two.sided")
      Two-sample t test power calculation
        n = 34
        d = 0.8
      sig.level = 0.05
      power = 0.9
    alternative = two.sided
NOTE: n is number in *each* group
```

Отсюда следует, что вам нужно включить по 34 человека в каждую группу (всего 68 человек), чтобы обнаружить эффект размером 0.8 с 90%-ной уверенностью и вероятностью обнаружить несуществующую закономерность не больше 5%.

Давайте изменим вопрос. Представим, что, сравнивая две группы, вы хотите, чтобы можно было обнаружить различие между выборочными средними в 0.5 стандартного отклонения. Вы хотите снизить шансы обнаружения различий по ошибке до 1 из 100. Кроме того, вы можете исследовать только 20 человек. С какой вероятностью вы сможете выявить различия между выборочными средними при таких условиях?

Если в обоих экспериментальных группах будет одинаковое число участников, то результат будет следующим:

```
> pwr.t.test(n=20, d=.5, sig.level=.01, type="two.sample",
  alternative="two.sided")
  Two-sample t test power calculation
    n = 20
    d = 0.5
  sig.level = 0.01
  power = 0.14
  alternative = two.sided
NOTE: n is number in *each* group
```

При 20 участниках в группе, заданном уровне значимости 0.01 и стандартном отклонении зависимой величины в 1.25 секунды вероятность того, что различия на 0.625 секунды ($d = 0.5 = 0.625/1.25$) или менее будут значимыми, составляет 14%. Соответственно, вероятность пропустить искомый эффект составляет 86%. Возможно, вам нужно еще раз серьезно подумать, стоит ли тратить время и силы на решение этой задачи в том виде, в котором она сформулирована.

В предыдущих примерах подразумевалось, что размеры групп одинаковы. Если размер выборок различается, можно использовать функцию

```
pwr.t2n.test(n1=, n2=, d=, sig.level=, power=, alternative=)
```

Здесь n_1 и n_2 – размеры выборок, а остальные параметры такие же, как в функции `pwr.t.test`. Попробуйте использовать разные значения в качестве аргументов функции `pwr.t2n` и посмотреть, что получится.

10.2.2. Дисперсионный анализ

Функция `pwr.anova.test()` позволяет провести анализ мощности для сбалансированного однофакторного дисперсионного анализа. Формат ее применения таков:

```
pwr.anova.test(k=, n=, f=, sig.level=, power=),
```

где k – число групп, а n – размер выборки в каждой группе.

Для однофакторного дисперсионного анализа величина эффекта отражена в параметре f , который рассчитывается по формуле

$$f = \sqrt{\frac{\sum_{i=1}^k p_i \times (\mu_i - \mu)^2}{\sigma^2}},$$

где $p_i = n_i/N_p$,
 n_i – число наблюдений в группе i ,
 N – общее число наблюдений,
 μ_i – среднее значение для группы i ,
 μ – общее среднее,
 σ^2 – дисперсия ошибок внутри групп.

Попробуем разобрать пример. Для однофакторного дисперсионного анализа, включающего пять групп, нужно найти размер групп, при котором мощность составляет 0.8, размер эффекта – 0.25 и уровень значимости – 0.05. Программный код будет выглядеть так:

```
> pwr.anova.test(k=5, f=.25, sig.level=.05, power=.8)
  Balanced one-way analysis of variance power calculation
    k = 5
    n = 39
    f = 0.25
  sig.level = 0.05
  power = 0.8
NOTE: n is number in each group
```

Таким образом, общий размер выборки составляет 5×39 , или 195. Учтите, что для проведения такого теста нужно оценить средние значения для групп и общую дисперсию. Если у вас нет ни малейшего представления об этих значениях, вам могут пригодиться подходы, описанные в разделе 10.2.7.

10.2.3. Корреляции

Функция `pwr.r.test()` позволяет провести анализ мощности для коэффициентов корреляции. Формат ее применения таков:

```
pwr.r.test(n=, r=, sig.level=, power=, alternative=),
```

где `n` – это число наблюдений, `r` – это размер эффекта (величина линейного коэффициента корреляции), `sig.level` – уровень значимости, `power` – уровень мощности, а параметр `alternative` определяет тип теста: двухсторонний ("two.sided") или односторонний ("less" или "greater").

Например, допустим, что мы изучаем связь между депрессией и одиночеством. Ваши нулевая и альтернативная гипотезы будут следующими:

$$H_0: \rho \leq 0.25 \text{ и } H_1: \rho > 0.25,$$

где ρ – это коэффициент корреляции между этими двумя психологическими параметрами для генеральной совокупности. Вы устанавливаете уровень значимости в 0.05 и хотите быть на 90% уверенными, что отвергнете нулевую гипотезу, если она не выполняется для генеральной совокупности. Сколько наблюдений нужно сделать? Ответ содержится в программном коде:

```
> pwr.r.test(r=.25, sig.level=.05, power=.90, alternative="greater")
  approximate correlation power calculation (arctanh transformation)
```

```
n = 134
r = 0.25
sig.level = 0.05
power = 0.9
alternative = greater
```

Отсюда следует, что вам нужно установить степень депрессии и одиночества у 134 человек, чтобы быть на 90% уверенными в том, что вы отвергнете нулевую гипотезу, если она действительно окажется ложной.

10.2.4. Линейные модели

Для проведения анализа мощности для линейных моделей (таких как множественная регрессия) можно использовать функцию `pwr.f2.test()`. Формат ее применения таков:

```
pwr.f2.test(u=, v=, f2=, sig.level=, power)= ,
```

где `u` и `v` – это числитель и знаменатель степеней свободы, а `f2` – размер эффекта, который вычисляется по формуле

$$f^2 = \frac{R^2}{(1-R^2)}, \quad \text{где } R^2 \text{ – это квадрат множественного коэффициента корреляции для генеральной совокупности, или по формуле}$$

$$f^2 = \frac{(R_{AB}^2 - R_A^2)}{(1 - R_{AB}^2)}, \quad \begin{aligned} &\text{где } R_A^2 \text{ – дисперсия в генеральной совокупно-} \\ &\text{сти, учтенная набором переменных A,} \\ &R_{AB}^2 \text{ – дисперсия в генеральной совокупности,} \\ &\text{учтенная переменными из наборов A и B.} \end{aligned}$$

Первая формула для расчета `f2` подходит в случае оценки влияния набора независимых переменных на одну зависимую. Вторую формулу нужно применять, если вас интересует влияние одного набора независимых переменных наряду с влиянием другого набора независимых переменных (или ковариат).

Скажем, вы интересуетесь, влияет ли стиль руководства на удовлетворенность сотрудников своей работой наряду с уровнем зарплаты и чаевыми. Стиль руководства описан четырьмя переменными, а размер заплаты и чаевых – тремя. На основании имеющегося опыта можно сказать, что удовлетворение от работы примерно на 30% определяется доходом. С практической точки зрения интересно, добавит ли стиль руководства хотя бы 5% к этому значению. Сколько людей нужно опросить, чтобы выявить такой размер эффекта с 90% уверенности при равном 0.05 уровне значимости?

В данном случае $\text{sig.level}=0.05$, $\text{power}=0.90$, $u=3$ (число независимых переменных за вычетом независимых переменных во втором наборе), а размер эффекта составляет $f_2 = (.35-.30)/(1-.35) = 0.0769$. Использование этих данных в качестве аргументов функции позволяет получить следующее:

```
> pwr.f2.test(u=3, f2=0.0769, sig.level=0.05, power=0.90)
Multiple regression power calculation
  u = 3
  v = 184.2426
  f2 = 0.0769
  sig.level = 0.05
  power = 0.9
```

В случае множественной регрессии знаменатель степеней свободы равен $N - k - 1$, где N – это число наблюдений, а k – число независимых переменных. В данном случае $N - 7 - 1 = 185$; значит, необходимый размер выборки составляет $N = 185 + 7 + 1 = 193$.

10.2.5. Сравнение пропорций

Функцию `pwr.2p.test()` можно применять для проведения анализа мощности при сравнении двух пропорций. Формат функции таков:

```
pwr.2p.test(h=, n=, sig.level=, power=),
```

где h – величина эффекта, а n – общий объем выборки для двух групп. Величина эффекта рассчитывается по формуле

$$h = 2 \arcsin\left(\sqrt{p_1}\right) - 2 \arcsin\left(\sqrt{p_2}\right)$$

и может быть вычислена при помощи функции `ES.h(p1, p2)`¹.

Для выборок разного объема подходит функция

```
pwr.2p2n.test(h =, n1 =, n2 =, sig.level=, power=).
```

Опцию `alternative=` можно использовать для указания типа теста: двухсторонний ("two.sided", по умолчанию) или односторонний ("less" или "greater").

Допустим, вы ожидаете, что обычное лекарство облегчает симптомы болезни у 60% пациентов. Новое (и более дорогое) средство будет иметь успех на рынке, если оно улучшит состояние 65% больных. Сколько испытуемых нужно исследовать, если вы хотите обнаружить указанное выше различие между лекарствами по эффективности?

1 Обозначения $p1$ и $p2$ соответствуют сравниваемым пропорциям. – Прим. пер.

Предположим, вы хотите быть на 90% уверенными в том, что новое лекарство зарекомендовало себя лучше, и с 95% вероятностью прийти к этому заключению не по ошибке. Вы будете использовать односторонний тест, поскольку хотите проверить лишь наличие пре-восходства нового лекарства над обычным. Программный код будет выглядеть примерно так:

```
> pwr.2p.test(h=ES.h(.65, .6), sig.level=.05, power=.9,
               alternative="greater")
Difference of proportion power calculation for binomial
distribution (arcsine transformation)
h = 0.1033347
n = 1604.007
sig.level = 0.05
power = 0.9
alternative = greater
NOTE: same sample sizes
```

Из полученных результатов следует, что вам нужно исследовать 1605 человек, принимающих новый препарат, и 1605 человек, получающих обычное лекарство, чтобы проверить гипотезу при заданных условиях.

10.2.6. Тесты хи-квадрат

Тесты хи-квадрат часто используются для оценки связи между двумя категориальными переменными. Нулевая гипотеза обычно подразумевает, что переменные независимы, а альтернативная гипотеза – что они зависимы. При проведении теста хи-квадрат функцию `pwr.chisq.test()` можно использовать для оценки мощности, размера эффекта или требуемого объема выборки. Формат ее применения таков:

```
pwr.chisq.test(w= , N= , df= , sig.level= , power= ),
```

где w – размер эффекта, N – общий объем выборки, а df – число степеней свободы. В данном случае размер эффекта рассчитывается по формуле

$$w = \sqrt{\sum_{i=1}^m \frac{(p0_i - p2_i)^2}{p0_i}},$$

где $p0_i$ – вероятность в ячейке таблицы сопряженности при справедливой H_0 , $p1_i$ – вероятность в ячейке таблицы сопряженности при справедливой H_1 .

Суммирование происходит от 1 до m , где m – число ячеек в таблице сопряженности. Для вычисления размера эффекта, соответствующе-

го альтернативной гипотезе для двухмерной таблицы сопряженности, можно использовать функцию `ES.w2(P)`, где `P` – это двухмерная таблица вероятностей.

В качестве простого примера предположим, что вы ищете связь между этнической принадлежностью и продвижением по службе. Вы ожидаете, что 70% выборки будет представлено белыми, 10% – афроамериканцами и 20% – латиноамериканцами. Далее вы предполагаете, что 60% белых продвинутся по службе, для афроамериканцев эта величина составит 30%, а для латиноамериканцев – 50%. Ваша рабочая гипотеза состоит в том, что вероятность продвижения по службе будет соответствовать значениям, представленным в табл. 10.2.

Таблица 10.2. Предполагаемая доля людей, которые продвинутся по службе, в соответствии с рабочей гипотезой

Этническая принадлежность	Продвинулись по службе	Не продвинулись
Белые	0.42	0.28
Афроамериканцы	0.03	0.07
Латиноамериканцы	0.10	0.10

К примеру, вы ожидаете, что 42% генеральной совокупности будет представлено получившими повышение белыми ($0.42 = 0.10 \times 0.60$), а 7% генеральной совокупности – это не продвинувшиеся по службе афроамериканцы. Давайте примем уровень значимости 0.05 и необходимый уровень мощности 0.90. Число степеней свободы в двухмерной таблице сопряженности рассчитывается как $(r - 1) * (c - 1)$, где r – это число строк, а c – число столбцов. Ожидаемый размер эффекта можно рассчитать при помощи следующего программного кода:

```
> prob <- matrix(c(.42, .28, .03, .07, .10, .10), byrow=TRUE, nrow=3)
> ES.w2(prob)
[1] 0.1853198
```

Используя эту информацию, можно рассчитать необходимый размер выборки следующим образом:

```
> pwr.chisq.test(w=.1853, df=2, sig.level=.05, power=.9)
    Chi squared power calculation
      w = 0.1853
      N = 368.5317
      df = 2
      sig.level = 0.05
      power = 0.9
NOTE: N is the number of observations
```

Полученный результат свидетельствует о том, что 369 человек достаточно, чтобы обнаружить взаимосвязь между этнической принадлежностью и продвижением по службе при заданном размере эффекта, мощности и уровне значимости.

10.2.7. Выбор подходящего размера эффекта в незнакомых ситуациях

Размер эффекта – это тот параметр, который сложнее всего определить при анализе мощности. Обычно для этого вы должны иметь опыт работы с объектом исследования и с используемыми метриками. Например, для вычисления размера эффектов используются данные предыдущих исследований, а полученные результаты затем применяются для планирования новых работ.

Но что же делать, если вы оказываетесь в совершенно новой для вас ситуации и не существует прошлого опыта, к которому можно было бы обратиться? В области бихевиоризма Cohen (1988) предложил ориентировочные значения для «малых», «средних» и «больших» размеров эффектов для разных статистических тестов. Они приведены в табл. 10.3.

Таблица 10.3. Ориентировочные значения размеров эффектов, предложенные Коэном (Cohen)

Статистический метод	Мера размера эффекта	Предлагаемые ориентировочные значения для разных размеров эффектов		
		Малый	Средний	Большой
Тест Стьюдента	<i>d</i>	0.20	0.50	0.80
Дисперсионный анализ	<i>f</i>	0.10	0.25	0.40
Линейные модели	<i>f²</i>	0.02	0.15	0.35
Тест пропорций	<i>h</i>	0.20	0.50	0.80
Тест хи-квадрат	<i>w</i>	0.10	0.30	0.50

Если у вас нет ни малейшего представления о возможном размере эффекта, можно руководствоваться этой таблицей. Например, какова вероятность отвергнуть ложную нулевую гипотезу (то есть найти существующую закономерность) при использовании однофакторного

дисперсионного анализа с пятью группами, 25 объектами в группе и уровне значимости 0.05?

Используя функцию `pwr.anova.test()` и данные, приведенные в строке `f` табл. 10.3, мы узнаем, что для малого эффекта мощность будет 0.118, для умеренного эффекта – 0.574, а для большого – 0.957. Учитывая ограничения в размере выборки, наиболее вероятно, что вы обнаружите эффект, только если он будет большим.

Важно помнить, что предложенные Коэном ориентировочные значения – это только общие идеи, сформулированные на основании ряда социологических исследований, и они могут быть неприменимыми в вашей области исследования. Альтернативный подход заключается в изменении исследуемых параметров и наблюдении за тем, как это отразится на таких вещах, как размер выборки и мощность. Для примера предположим, что нам опять нужно сравнить пять групп при помощи однофакторного дисперсионного анализа с уровнем значимости 0.05. Ниже продемонстрировано, как определить объем выборок, необходимых для обнаружения эффектов разной величины, и графически отобразить результаты (рис. 10.2).

Однофакторный дисперсионный анализ
с мощностью .90 и $p = .05$

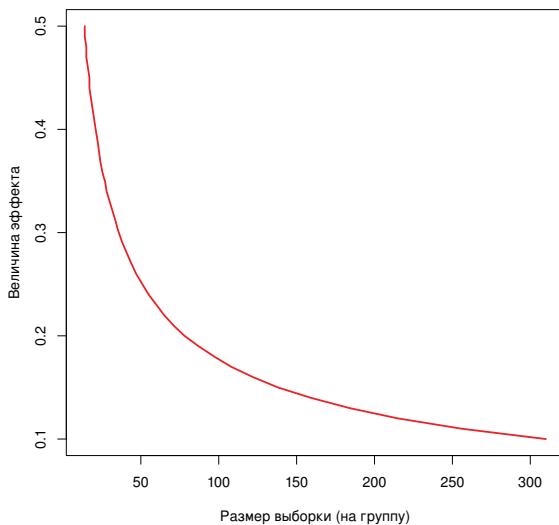


Рис. 10.2. Размеры выборок, необходимые для обнаружения эффектов разной величины при однофакторном дисперсионном анализе с пятью группами (при заданной мощности 0.9 и уровне значимости 0.05)

Программный код 10.1. Размеры выборок для обнаружения статистически значимых эффектов при однофакторном дисперсионном анализе

```
library(pwr)
es <- seq(.1, .5, .01)
nes <- length(es)
samsize <- NULL
for (i in 1:nes) {
  result <- pwr.anova.test(k=5, f=es[i], sig.level=.05, power=.9)
  samsize[i] <- ceiling(result$n)
}
plot(samsize, es, type="l", lwd=2, col="red",
      ylab="Величина эффекта",
      xlab="Размер выборки (на группу)",
      main="Однофакторный дисперсионный анализ с мощностью .90 и p=.05")
```

Диаграммы вроде той, что представлена на рис. 10.2, могут быть полезными при оценке разных параметров дизайна вашего эксперимента. Например, похоже, что вы не получите много дополнительных преимуществ, создав выборку из более чем 200 наблюдений. В следующем разделе мы увидим другой пример графического изображения результатов.

10.3. Графический анализ мощности

Прежде чем мы расстанемся с пакетом `pwr`, рассмотрим более сложный пример графического представления результатов. Предположим, вам бы хотелось узнать размер выборки, необходимый для обнаружения значимого коэффициента корреляции, при разных размерах эффекта и уровнях значимости. Для выполнения этой задачи можно использовать функцию `pwr.r.test()` и цикл `for`, как показано в приведенном ниже программном коде.

Программный код 10.2. Графическое изображение зависимости объема выборки от искомой величины корреляции

```
library(pwr)
r <- seq(.1,.5,.01)
nr <- length(r)
```

←❶ **Определяем диапазон значений
мощности и коэффициентов
корреляции**

```
p <- seq(.4,.9,.1)
np <- length(p)
```

←❷ **Вычисляем
объем выборок**

```
samsize <- array(numeric(nr*np), dim=c(nr,np))
```

```

for (j in 1:nr){
  result <- pwr.r.test(n = NULL, r = r[j],
  sig.level = .05, power = p[i],
  alternative = "two.sided")
  samsize[j,i] <- ceiling(result$n)
}
}

xrange <- range(r)
yrange <- round(range(samsize))
colors <- rainbow(length(p))
plot(xrange, yrange, type="n",
      xlab="Коэффициент корреляции (r)",
      ylab="Размер выборки (n)",

for (i in 1:np){
  lines(r, samsize[,i], type="l", lwd=2, col=colors[i])
}
abline(v=0, h=seq(0,yrange[2],50), lty=2, col="grey89")
abline(h=0, v=seq(xrange[1],xrange[2],.02), lty=2,
       col="gray89")
title("Оценка размера выборки для исследований\n
      корреляции при p=0.05 (двухсторонний тест)")
legend("topright", title=" Мощность", as.character(p),
       fill=colors)

```

← ③ Добавляем кривые изменения мощности

Добавляем линии ④

Добавляем аннотации ⑤

В программном коде 10.2 для создания ряда значений размера эффектов r (коэффициентов корреляции при H_1) и уровней мощности p ① использована функция `seq`. Затем при помощи двух циклов `for` перебираются все эти размеры эффектов и уровни мощности с вычислением необходимых объемов выборки, сохраненных впоследствии в виде матрицы `samsize` ②. Затем создается график с соответствующими горизонтальной и вертикальной осями и подписями ③. Кривые для разных значений мощности представлены в виде линий, а не точек ④. На последнем этапе для облегчения работы с графиком добавляются координатная сетка и условные обозначения ⑤. Полученный график представлен на рис. 10.3.

Как видно из графика, для обнаружения корреляции с коэффициентом 0.2 с 40%-ной уверенностью требуется выборка объемом примерно в 75 наблюдений. Вам понадобится еще примерно 185 наблюдений ($n = 260$) для обнаружения корреляции такой же силы с 90%-ной вероятностью. После несложных модификаций такой подход можно использовать для оценки необходимого размера выборки при разной мощности для широкого диапазона статистических тестов.

Мы закончим эту главу кратким обзором других функций R, полезных при проведении анализа мощности.

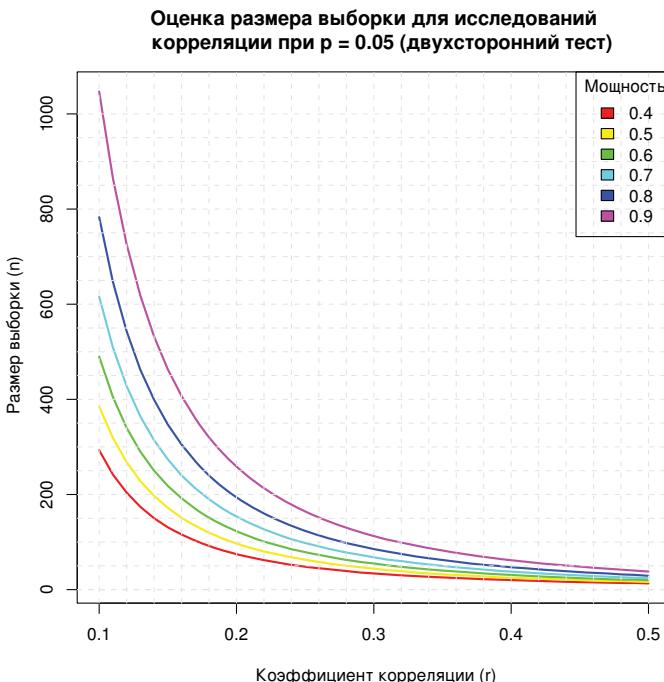


Рис. 10.3. Зависимость объема выборки от искомой величины корреляции при разных уровнях мощности

10.4. Другие пакеты

В R существует несколько других пакетов, которые могут быть полезны при планировании исследований. В некоторых из них реализованы общие подходы, а некоторые узкоспециализированы.

Пакет `piface` (см. рис. 10.4) запускает взаимодействующий с R графический пользовательский интерфейс, основанный на языке Java и предназначенный для оценки объема выборки.

Хотя этот пакет заявлен как пробный, с ним определенно стоит познакомиться. Необходимые файлы для Windows и Mac OS X можно загрузить по адресу <http://r-forge.r-project.org/projects/piface/>. Загружая пакет из R, наберите в командной строке

```
install.packages("piface", repos="http://R-Forge.R-project.org")
library(piface)
piface()
```

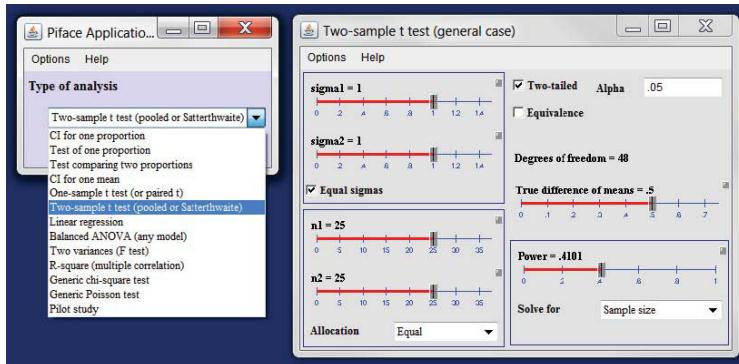


Рис. 10.4. Диалоговые окна для определения объема выборок в программе piface

Этот пакет особенно полезен для анализа влияния объема выборки, величины эффекта, уровней значимости и необходимой мощности на другие параметры.

Другие связанные с анализом мощности пакеты перечислены в табл. 10.4. Последние пять предназначены для анализа мощности в генетических исследованиях. Исследования геномных ассоциаций проводятся для определения генетической обусловленности наблюдаемых признаков. К примеру, такие исследования могут быть направлены на выяснение причины существования определенного типа порока сердца у некоторых людей.

Таблица 10.4. Специализированные пакеты для проведения анализа мощности

Пакет	Назначение
asypow	Расчет мощности при помощи методов асимптотического отношения правдоподобия
PwrGSD	Анализ мощности для группового последовательного дизайна
Ramm	Анализ мощности для случайных эффектов в смешанных моделях
powerSurvEpi	Расчет мощности и объема выборок для анализа выживаемости в эпидемиологических исследованиях
powerpkg	Анализ мощности для пар потомков одних и тех же родителей, на одного из которых оказывается воздействие, и для тестов неравновесной передачи аллелей

Пакет	Назначение
powerGWAInteraction	Расчет мощности для исследований геномных ассоциаций
pedantics	Функции, облегчающие анализ мощности при генетических исследованиях природных популяций
Gap	Функции для расчета мощности и объема выборок для дизайна обсервационных исследований (case-cohort design)
ssize.fdr	Расчет объема выборок для экспериментов с микрочипами

10.5. Резюме

В главах 7, 8 и 9 мы рассмотрели множество функций R, предназначенных для проверки статистических гипотез. В этой главе мы сосредоточились на этапе планирования исследования. Анализ мощности помогает определить объем выборки, необходимый для выявления эффекта заданного размера с определенной долей уверенности. Анализ мощности также используется для оценки вероятности обнаружения такого эффекта при заданном объеме выборки. При этом можно в явном виде наблюдать обратную связь между вероятностью нахождения значимой закономерности по ошибке (статистическая ошибка I рода) и вероятностью правильного обнаружения реального эффекта (мощность теста).

Основная часть этой главы была посвящена использованию функций из пакета `pwr`. Они могут применяться для оценки объема выборки и мощности при использовании стандартных статистических методов (включая тесты Стьюдента, хи-квадрат, пропорций, а также дисперсионный анализ и регрессию). Более специфические методы были упомянуты в последнем разделе.

Обычно анализ мощности – это интерактивный процесс. Исследователь изменяет объем выборки, размер эффекта, необходимый уровень значимости, чтобы узнать, как они влияют друг на друга. Полученные результаты используются для планирования исследований, в которых высока вероятность получения содержательных результатов.

Важная дополнительная польза от анализа мощности заключается в том, что он инициирует изменение образа мышления от направленного только на проверку бинарных гипотез (существует эффект

или нет) к оценке *размера* изучаемого эффекта. Редакторы научных журналов все чаще требуют от авторов указывать размеры эффектов наряду со значением p при описании результатов исследований. Это помогает определить практическую значимость результатов исследования и дает информацию, которая может быть использована для планирования дальнейших исследований.

В следующей главе мы познакомимся с дополнениями к уже известным подходам и с новыми способами визуализации многомерных взаимосвязей. Эти графические методы могут дополнить и улучшить те подходы, с которыми вы уже успели познакомиться, и подготовят вас к изучению продвинутых методов, описанных в части III.



ГЛАВА 11.

Диаграммы средней сложности

В этой главе:

- Визуализация взаимосвязей между двумя и более переменными.
- Работа с диаграммами рассеяния и линейными графиками.
- Осмысление коррелограмм.
- Использование мозаичных диаграмм.

В главе 6 (базовые диаграммы) мы рассмотрели различные типы диаграмм, отображающих распределение значений одной категориальной или непрерывной переменной. Глава 8 (регрессия) содержит обзор графических методов, которые полезны для предсказания непрерывной зависимой переменной по набору независимых. В главе 9 (дисперсионный анализ) мы рассмотрели методы, которые позволяют визуализировать межгрупповые различия в значениях непрерывной переменной. Эта глава во многом продолжает и расширяет затронутые раньше темы.

В этой главе мы сосредоточимся на графических способах отображения взаимосвязи между двумя переменными (двухмерные взаимосвязи) и между несколькими переменными (многомерные взаимосвязи). Например:

- Какова связь между расходом топлива и весом автомобиля? Изменяется ли характер этой связи в зависимости от числа цилиндров?
- Можно ли отразить взаимосвязи между расходом топлива, весом автомобиля, объемом двигателя и передаточным числом заднего моста на одной диаграмме?

- Как справиться с наложением многих точек друг на друга, которое часто наблюдается при изображении взаимосвязи между двумя переменными с большим числом (скажем, 10 000) наблюдений? Иными словами, что делать, если диаграмма представляет собой одну большую кляксу?
- Как можно одновременно изобразить взаимосвязи между тремя переменными (учитывая, что мы располагаем двухмерным экраном компьютера или листом бумаги и бюджетом, немного меньшим, чем у фильма «Аватар»)?
- Как можно изобразить динамику роста нескольких деревьев?
- Как можно визуально представить корреляции между десятком переменных на одной диаграмме? Как это поможет понять структуру данных?
- Как можно визуализировать взаимоотношения между полом и возрастом пассажиров Титаника, классом, в котором они путешествовали, и вероятностью их выживания? Что можно узнать из такой диаграммы?

На все вопросы вроде этих можно ответить при помощи описанных в данной главе методов. Наборы данных, которые мы будем использовать, – это лишь примеры того, что можно обрабатывать. Главное здесь – это принципиальные подходы. Если темы характеристик автомобиля или роста деревьев не интересны вам, используйте собственные данные!

Мы начнем с диаграмм рассеяния и составленных из них матриц. Затем мы исследуем разнообразные графики. Эти подходы хорошо известны и широко используются в научных исследованиях. Затем мы рассмотрим использование коррелограмм для визуализации корреляций и мозаичных диаграмм для иллюстрации многомерных связей между категориальными переменными. Это также полезные методы, но уже совсем не такие известные среди исследователей. Вы увидите примеры применения каждого из этих подходов для лучшего понимания ваших данных и доведения ваших результатов до сведения других людей.

11.1. Диаграммы рассеяния

Как вы узнали из предыдущих глав, диаграммы рассеяния иллюстрируют взаимосвязь между двумя непрерывными переменными. Этот раздел мы начнем с изображения взаимосвязи между одной парой переменных (x и y). Затем мы исследуем способы усовершенствования

этой диаграммы путем наложения на нее дополнительной информации. Потом мы узнаем, как совмещать несколько диаграмм рассеяния в общую матрицу так, чтобы можно было анализировать несколько двухмерных взаимосвязей одновременно. Мы также затронем такую ситуацию, когда много точек на диаграмме накладываются друг на друга, ограничивая возможности визуального анализа данных, и обсудим несколько способов обойти это затруднение. Наконец, мы расширим двухмерную диаграмму в третье измерение, добавив третью непрерывную переменную. К таким диаграммам относятся трехмерные диаграммы рассеяния и пузырьковые диаграммы. С их помощью можно одновременно исследовать взаимосвязи между тремя переменными.

Основная функция для создания диаграмм рассеяния в R – это `plot(x, y)`, где *x* и *y* – это числовые векторы, определяющие *x* и *y* координаты точек на диаграмме. Пример представлен в программном коде 11.1.

Программный код 11.1. Диаграмма рассеяния с линиями наилучшей аппроксимации

```
attach(mtcars)
plot(wt, mpg,
      main="Простая диаграмма рассеяния для зависимости расхода
      топлива от веса автомобиля",
      xlab="Вес автомобиля (тысячи фунтов)",
      ylab="Мили на галлон", pch=19)
abline(lm(mpg~wt), col="red", lwd=2, lty=1)
lines(lowess(wt,mpg), col="blue", lwd=2, lty=2)
```

Полученная диаграмма представлена на рис. 11.1.

Команды в программном коде 11.1 загружают таблицу данных `mtcars` и создают базовую версию диаграммы рассеяния, на которой наблюдения обозначены заполненными кружками. Как и ожидалось, расход топлива уменьшается с увеличением веса машины, хотя эта связь не идеально линейная. Функция `abline()` использована для добавления аппроксимирующей регрессионной линии (linear line of best fit), а сглаживающая кривая добавлена при помощи функции `lowess()`. Сглаживающая кривая – это непараметрическая аппроксимирующая кривая, основанная на локально взвешенной полиномиальной регрессии. Детально этот алгоритм описан в публикации Cleveland (1981).

Примечание. В R есть две функции для создания сглаженных кривых (*lowess fit*): `lowess()` и `loess()`. Функция `loess()` – это более мощная и современная версия функции `lowess()`.

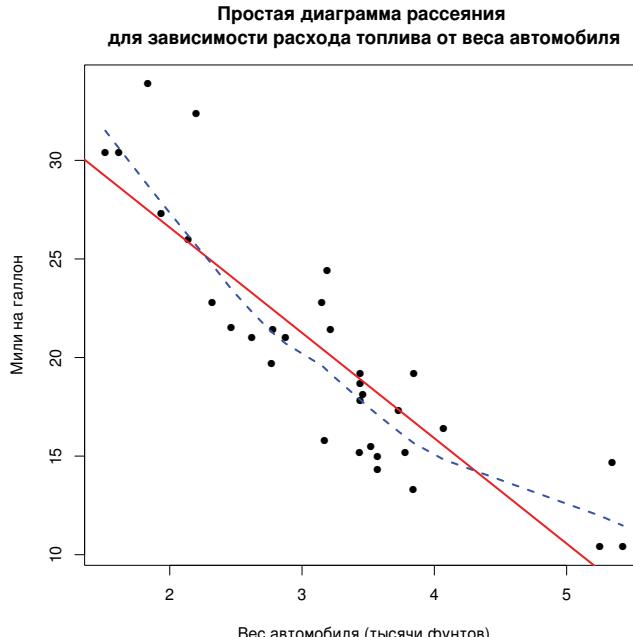


Рис. 11.1. Диаграмма рассеяния для зависимости между расходом топлива и весом машины с наложенными аппроксимирующими и сглаженной линиями

Функция `scatterplot()` из пакета `car` предоставляет много возможностей для создания диаграмм рассеяния, включая построение аппроксимирующих линий, диаграмм рассеяния по краям рисунка, эллипсов доверительных интервалов, обозначение подгрупп, создание подписей к точкам в интерактивном режиме. Например, более сложную версию предыдущей диаграммы можно создать при помощи следующего программного кода:

```
library(car)
scatterplot(mpg ~ wt | cyl, data=mtcars, lwd=2,
           main="Диаграмма рассеяния для зависимости расхода
           топлива от веса автомобилей с разным числом цилиндров",
           xlab="Вес автомобиля (тысячи фунтов)",
           ylab="Мили на галлон",
           legend.plot=TRUE,
           id.method="identify",
           labels=row.names(mtcars),
           boxplots="xy")
)
```

Здесь функция `scatterplot()` использована для отражения зависимости расхода топлива от веса автомобилей с четырьмя, шестью и восемью цилиндрами. Формула `mpg ~ wt | cyl` означает условие (то есть нужно привести отдельно диаграммы рассеяния зависимости `mpg` от `wt` для каждого значения `cyl`). Полученная диаграмма представлена на рис. 11.2.

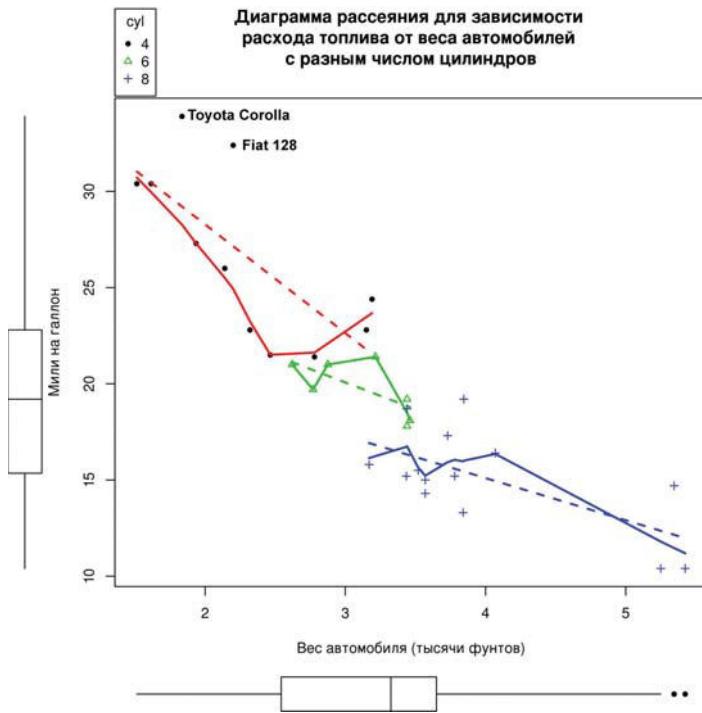


Рис. 11.2. Диаграмма рассеяния с обозначенными подгруппами и аппроксимирующими линиями для каждой подгруппы

По умолчанию подгруппы обозначаются разными цветами и символами, и для каждой подгруппы строится своя аппроксимирующая линия. Для построения слаживающей кривой по умолчанию нужно как минимум пять неповторяющихся точек, поэтому такая линия не нарисована для автомобилей с шестью цилиндрами. Опция `i.d.method` определяет, будут ли точки подписаны в интерактивном режиме при помощи щелчков мыши, пока пользователь не нажмет **Stop** (в графическом или контекстном меню) или клавишу **Esc**. Опция `labels` указывает, что точки будут подписаны названиями строк. Из диаграммы

видно, что Toyota Corolla и Fiat 128 необычайно экономны для своего веса. Опция `legend.plot` размещает условные обозначения в верхнем левом углу, за диаграммы размахов для `mpg` и `weight` по краям диаграммы отвечает опция `boxplots`. Многие параметры функции `scatterplot()` заслуживают дальнейшего изучения, включая опции, связанные с логистической регрессией, и эллипсы концентрации данных, которые здесь не рассматриваются. Введите `help(scatterplot)`, чтобы получить дальнейшую информацию.

Диаграммы рассеяния помогают визуализировать взаимосвязи между одной парой количественных переменных одновременно. Но что, если вы хотели изучить попарные взаимосвязи между расходом топлива (`mpg`), весом автомобиля (`wt`), объемом двигателя (`disp`) и передаточным числом заднего моста (`drat`)? Одна из возможностей заключается в том, чтобы объединить эти шесть диаграмм рассеяния в одну матрицу, о которой мы сейчас поговорим.

11.1.1. Матрицы диаграмм рассеяния

В R существуют по меньшей мере четыре полезные функции для создания матриц диаграмм рассеяния. Должно быть, аналитики обожают их! Базовый вариант матрицы диаграмм рассеяния можно получить посредством функции `pairs()`. Построить эту матрицу для переменных `mpg`, `disp`, `drat` и `wt` можно при помощи такого программного кода:

```
pairs(~mpg+disp+drat+wt, data=mtcars,  
      main="Базовый вариант матрицы диаграмм рассеяния")
```

Все переменные, перечисленные справа от знака `~`, попадают на диаграмму. Она представлена на рис. 11.3.

На этой диаграмме отражены взаимосвязи между всеми парами переменных. Например, диаграмма рассеяния для `mpg` и `disp` расположена на пересечении строки и столбца с названиями этих переменных. Обратите внимание на то, что шесть диаграмм рассеяния сверху от главной диагонали точно такие же, как и снизу. Они так расположены просто для удобства. Изменяя параметры команды, можно отображать только верхний или только нижний треугольник. Например, опция `upper.panel=NULL` позволит оставить только нижний треугольник с диаграммами.

Функция `scatterplotMatrix()` пакета `car` также создает матрицы диаграмм рассеяния и способна в качестве дополнительных опций делать следующее:

Базовый вариант матрицы диаграмм рассеяния

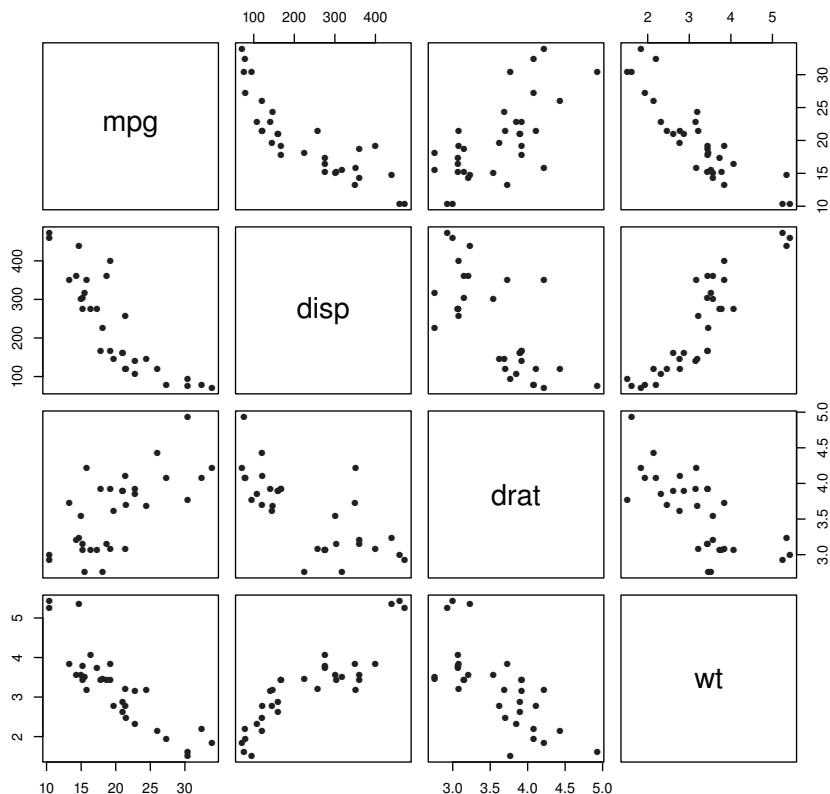


Рис. 11.3. Матрица диаграмм рассеяния, созданная при помощи функции `pairs()`

- выделять подгруппы по значениям заданного фактора;
- добавлять аппроксимирующие линии и сглаженные кривые;
- размещать диаграммы размахов и плотности рассеяния или гистограммы на главной диагонали;
- добавлять графики-щетки по краям ячеек матрицы.

Вот пример:

```
library(car)
scatterplotMatrix(~ mpg + disp + drat + wt, data=mtcars, spread=FALSE,
lty.smooth=2, main="Матрица диаграмм рассеяния,
созданная в пакете car")
```

Диаграмма представлена на рис. 11.4. Можно видеть, что по умолчанию на диаграммы рассеяния наложены аппроксимирующие линии и сглаживающие кривые, а на главной диагонали изображены диаграммы ядерной оценки функции плотности и графики-щетки. Опция `spread=FALSE` предотвращает появление линий, иллюстрирующих разброс и асимметрию, а параметр `lty.smooth=2` делает сглаженные кривые пунктирными, а не сплошными.

Матрица диаграмм рассеяния, созданная в пакете `car`

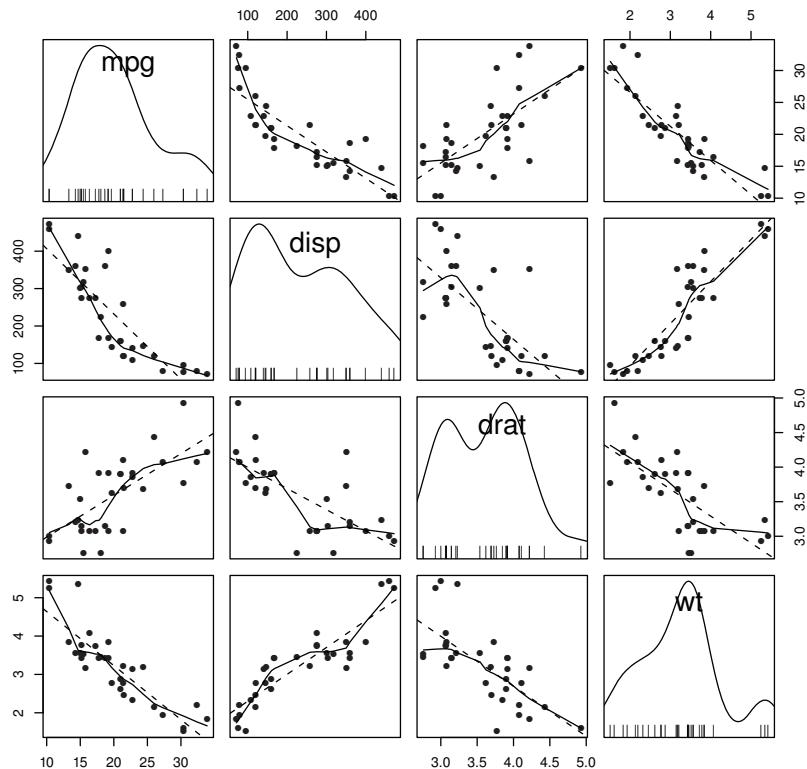


Рис. 11.4. Матрица диаграмм рассеяния, созданная посредством функции `scatterplotMatrix()`. На диаграммы рассеяния наложены аппроксимирующие линии и сглаженные кривые, а на главной диагонали изображены диаграммы ядерной оценки функции плотности и графики-щетки

Второй пример использования функции `scatterplotMatrix()` представлен в этом программном коде:

```
library(car)
scatterplotMatrix(~ mpg + disp + drat + wt | cyl, data=mtcars,
    spread=FALSE, diagonal="histogram",
    main="Матрица диаграмм рассеяния, созданная в пакете car")
```

Здесь диаграммы ядерной оценки функции плотности заменены гистограммами, а наблюдения разделены на подгруппы по числу цилиндров в двигателе. Результат представлен на рис. 11.5.

Матрица диаграмм рассеяния, созданная в пакете car

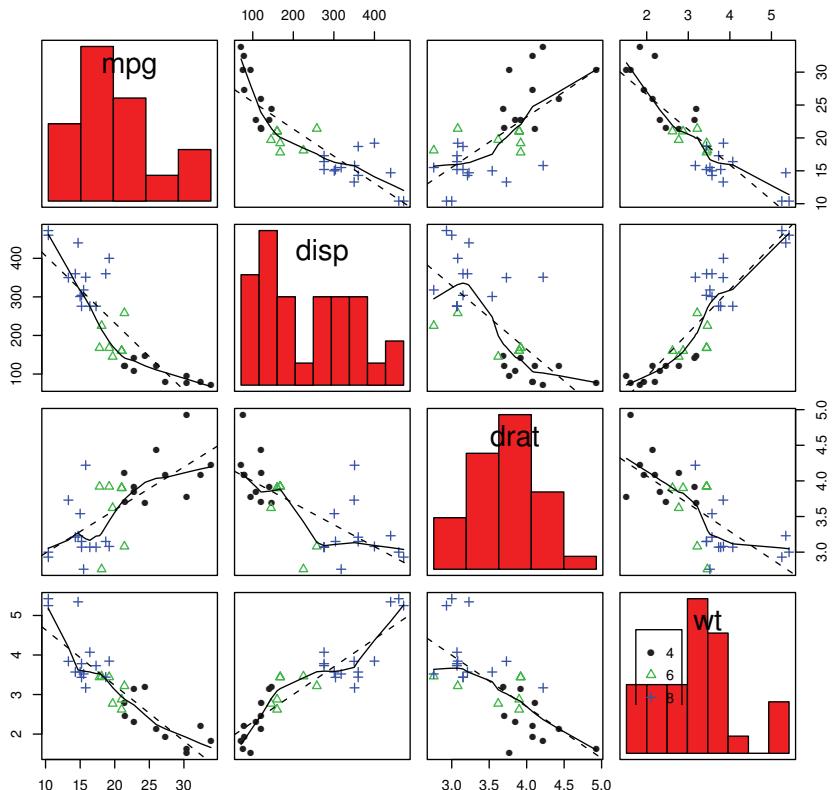


Рис. 11.5. Матрица диаграмм рассеяния, созданная посредством функции `scatterplotMatrix()`. На диаграммы рассеяния наложены аппроксимирующие линии и сглаживающие кривые, а на главной диагонали изображены гистограммы. Кроме того, подгруппы (определенные по числу цилиндров) обозначены разными цветами и символами

По умолчанию регрессионные линии строятся по всей выборке. Параметр `by.groups = TRUE` позволяет построить отдельные регрессионные линии для каждой подгруппы.

Интересная модификация матрицы диаграмм рассеяния реализована при помощи функции `cpairs()` из пакета `gclus`. Эта функция позволяет изменить порядок размещения переменных так, чтобы диаграммы рассеяния для пар наиболее сильно скоррелированных переменных оказались ближе к главной диагонали. Также можно цветом ячеек обозначить силу корреляции. Рассмотрим корреляции между переменными `mpg`, `disp`, `drat` и `wt`:

```
> cor(mtcars[c("mpg", "wt", "disp", "drat")])
      mpg      wt      disp      drat
mpg   1.000 -0.868 -0.848  0.681
wt    -0.868  1.000  0.888 -0.712
disp  -0.848  0.888  1.000 -0.710
drat  0.681 -0.712 -0.710  1.000
```

Видно, что наибольшая¹ корреляция наблюдается между весом автомобиля и объемом двигателя (0.89) и между весом машины и расходом топлива (-0.87). Наиболее низкая корреляция отмечена между расходом топлива и передаточным числом заднего моста (0.68). Расположение и цвет ячеек матрицы диаграмм рассеяния для этих переменных можно изменить при помощи команд, приведенных в следующем программном коде.

Программный код 11.2. Матрица диаграмм рассеяния, созданная при помощи команд из пакета `gclus`

```
library(gclus)
mydata <- mtcars[c(1, 3, 5, 6)]
mydata.corr <- abs(cor(mydata))
mycolors <- dmat.color(mydata.corr)
myorder <- order.single(mydata.corr)
cpairs(mydata,
       myorder,
       panel.colors=mycolors,
       gap=.5,
       main="Упорядоченные и раскрашенные в соответствии
             с коэффициентами корреляции переменные"
)
```

В этом программном коде использованы функции `dmat.color()`, `order.single()` и `cpairs()` из пакета `gclus`. Сначала вы выбираете нужные переменные из таблицы данных `mtcars` и вычисляете модули

¹ По модулю. – Прим. пер.

коэффициентов корреляции между ними. Затем вы генерируете цвета для ячеек при помощи функции `dmat.color()`. Эта функция принимает в качестве аргумента определенную симметричную матрицу (корреляционную в данном случае) и возвращает матрицу цветов. Вы также сортируете переменные для отображения их на диаграмме. Функция `order.single()` сортирует объекты так, что сходные пары объектов оказываются рядом. В данном случае сортировка происходит на основании коэффициентов корреляций. Наконец, мы создаем матрицу диаграмм рассеяния и раскрашиваем ее, используя новый порядок переменных (`myorder`) и список цветов (`mycolors`). Параметр `gap` позволяет добавить немного свободного пространства между ячейками матрицы. Полученная диаграмма приведена на рис. 11.6.

Упорядоченные и раскрашенные в соответствии с коэффициентами корреляции переменные

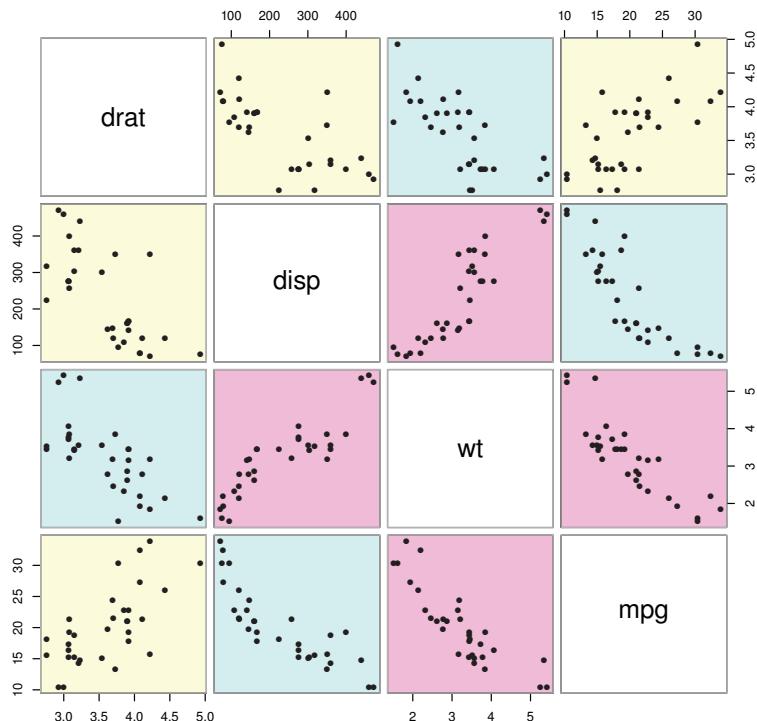


Рис. 11.6. Матрица диаграмм рассеяния, созданная при помощи функции `spairs()` из пакета `gclus`. Пары переменных, диаграммы для которых расположены ближе к главной оси, скоррелированы сильнее

На рисунке видно, что наиболее высокие корреляции наблюдаются между весом автомобиля и объемом двигателя, а также между весом автомобиля и расходом топлива (красные ячейки, ближе всего расположенные к главной диагонали). Наименьшая корреляция наблюдается между расходом топлива и передаточным числом заднего моста (желтая ячейка, наиболее удаленная от главной оси). Этот подход особенно полезен при анализе большого числа переменных, если корреляции между разными их парами сильно варьируют. Другие примеры использования диаграмм рассеяния приведены в главе 16.

11.1.2. Диаграммы рассеяния высокой плотности

На диаграммах рассеяния со значительным наложением точек друг на друга сложнее увидеть взаимосвязи. Рассмотрим такой пример с 10 000 наблюдений, образующих два перекрывающихся кластера:

```
set.seed(1234)
n <- 10000
c1 <- matrix(rnorm(n, mean=0, sd=.5), ncol=2)
c2 <- matrix(rnorm(n, mean=3, sd=2), ncol=2)
mydata <- rbind(c1, c2)
mydata <- as.data.frame(mydata)
names(mydata) <- c("x", "y")
```

Если вы построите диаграмму рассеяния для этих переменных при помощи команды

```
with(mydata,
     plot(x, y, pch=19, main="Диаграмма рассеяния для 10,000 наблюдений")),
```

то получите диаграмму вроде той, что представлена на рис. 11.7.

Наложение точек на рис. 11.7 затрудняет понимание характера взаимосвязи между x и y . В R реализовано несколько графических подходов, которые можно использовать в таких случаях. К таким подходам относится объединение точек, цвета и прозрачности для обозначения количества наложенных данных в заданной точке диаграммы.

Функция `smoothScatter()` использует ядерную оценку функции плотности для изображения плотности точек на диаграмме при помощи оттенков цвета. Команда

```
with(mydata,
     smoothScatter(x, y, main="Диаграмма рассеяния, раскрашенная
     согласно слаженной оценке плотности точек"))
создает диаграмму, приведенную на рис. 11.8.
```

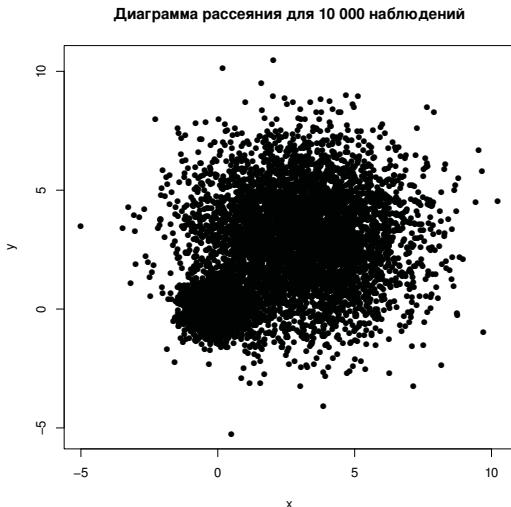


Рис. 11.7. Диаграмма рассеяния для 10 000 наблюдений и с заметным наложением точек друг на друга. Обратите внимание на то, что наложение точек затрудняет обнаружение области с наивысшей их концентрацией

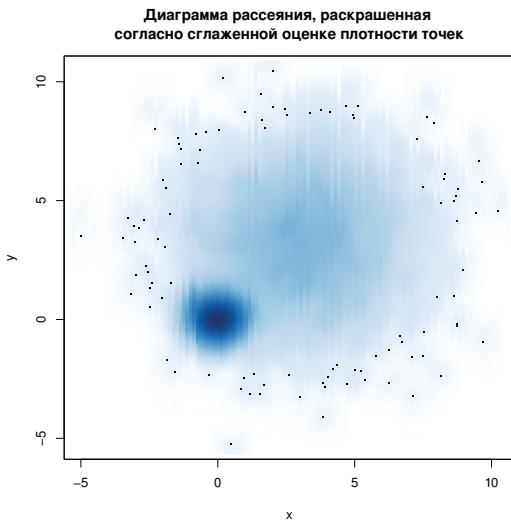


Рис. 11.8. Диаграмма рассеяния, построенная при помощи функции `smoothScatter()`, на которой отражена сглаженная оценка плотности точек. Плотность точек на графике легко оценить

Другой подход заключается в двухмерном объединении данных в шестиугольные ячейки при помощи функции `hexbin()` из пакета `hexbin` (это выглядит лучше, чем звучит). Применив эту функцию к нашему набору данных

```
library(hexbin)
with(mydata, {
  bin <- hexbin(x, y, xbins=50)
  plot(bin, main="Объединение 10 000 наблюдений в шестиугольные ячейки")
})
```

вы получите диаграмму рассеяния, представленную на рис. 11.9.

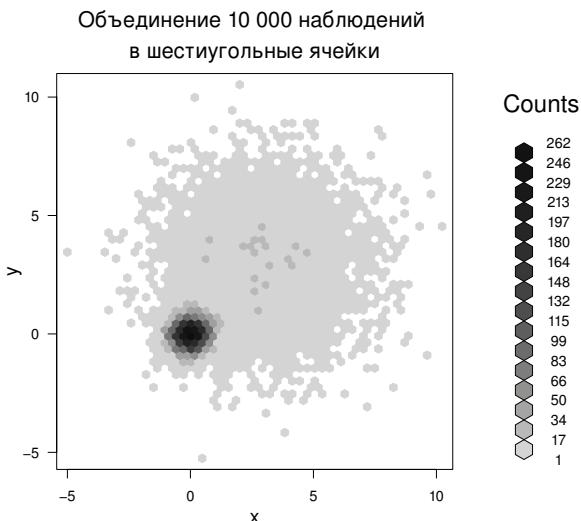


Рис. 11.9. Диаграмма рассеяния с объединением точек в шестиугольные ячейки для наглядного изображения числа наблюдений в каждой точке диаграммы. Скопления данных легко обнаружить, информация об их числе содержится в условных обозначениях

Наконец, для отображения на диаграмме плотности точек (числа точек в определенной области) при помощи цвета можно использовать функцию `iplot()` из пакета `IDPmisc`. Команда

```
library(IDPmisc)
with(mydata,
  iplot(x, y, main="Диаграмма рассеяния, на которой плотность точек  
показана цветом"))
```

создает диаграмму, показанную на рис. 11.10.

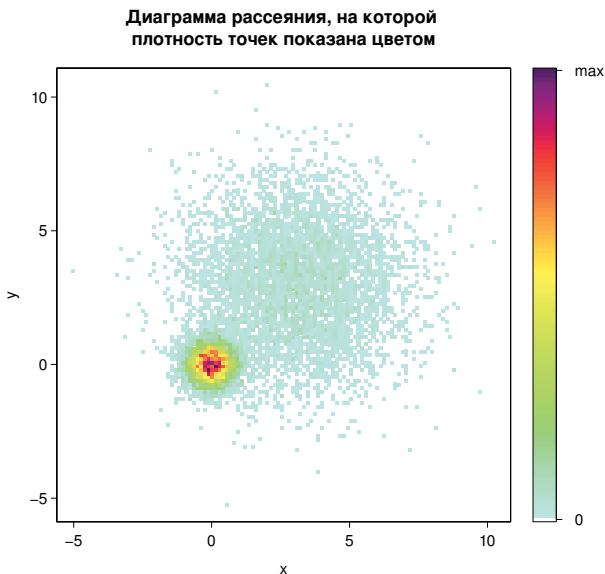


Рис. 11.10. Диаграмма рассеяния для 10 000 наблюдений, на которой плотность точек показана цветом. Области высокой концентрации данных легко выявить

Нужно отметить, что функция `smoothScatter()`, реализованная в базовой версии программы, наряду с функцией `ipairs()` из пакета `IDPmisc` может также использоваться для создания легко интерпретируемых матриц диаграмм рассеяния для больших наборов данных. Наберите `?smoothScatter` и `?ipairs`, чтобы получить дальнейшую информацию.

11.1.3. Трехмерные диаграммы рассеяния

Диаграммы рассеяния и матрицы диаграмм рассеяния отображают попарные взаимодействия, а что, если вы хотите визуализировать взаимосвязь между тремя количественными переменными одновременно? В этом случае можно использовать трехмерные диаграммы рассеяния.

К примеру, вас заинтересовала взаимосвязь между расходом топлива, объемом двигателя и весом автомобилей. Для графического отображения этой взаимосвязи можно использовать функцию

scatterplot3d() из пакета scatterplot3d. Формат ее применения таков:

```
scatterplot3d(x, y, z),
```

где x – переменная, отложенная по горизонтальной оси, y – по вертикальной, а z – в перспективе. Эти команды:

```
library(scatterplot3d)
attach(mtcars)
scatterplot3d(wt, disp, mpg,
    main="Базовый вариант трехмерной диаграммы рассеяния")
```

позволяют получить трехмерную диаграмму рассеяния, представленную на рис. 11.11.

Базовый вариант трехмерной диаграммы рассеяния

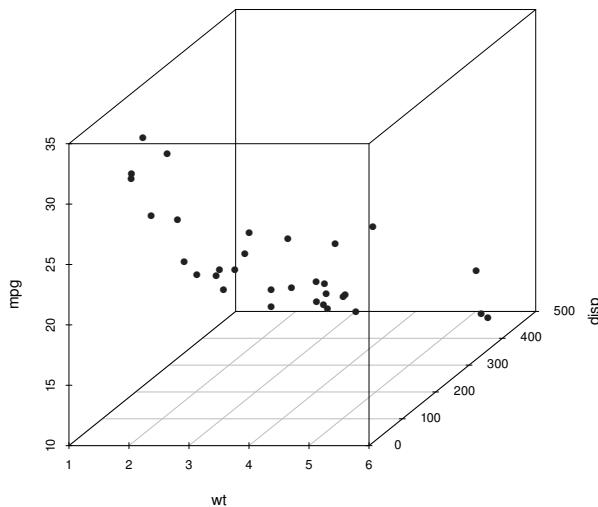


Рис. 11.11. Трехмерная диаграмма рассеяния
для расхода топлива, объема двигателя и веса автомобилей

Функция scatterplot3d имеет много параметров, включая возможность задавать тип символов, оси, цвета, линии, координатной сетки, выделение яркостью и углы обзора. Например, команды

```
library(scatterplot3d)
attach(mtcars)
scatterplot3d(wt, disp, mpg,
    pch=16,
    highlight.3d=TRUE,
```

```
type="h",
main="Трехмерная диаграмма рассеяния с вертикальными
• отрезками")
```

создают трехмерную диаграмму рассеяния с градиентом яркости для увеличения иллюзии перспективы и вертикальными линиями, соединяющими точки с горизонтальной плоскостью (см. рис. 11.12).

**Трехмерная диаграмма рассеяния
с вертикальными отрезками**

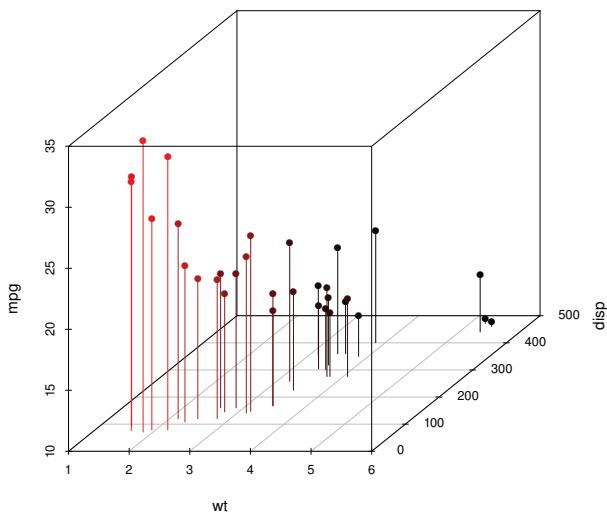


Рис. 11.12. Трехмерная диаграмма рассеяния с вертикальными линиями и градиентом яркости

В качестве последнего примера мы возьмем предыдущую диаграмму и добавим к ней регрессионную плоскость. Для этого понадобятся следующие команды:

```
library(scatterplot3d)
attach(mtcars)
s3d <- scatterplot3d(wt, disp, mpg,
    pch=16,
    highlight.3d=TRUE,
    type="h",
    main="Трехмерная диаграмма рассеяния с вертикальными отрезками
• и регрессионной плоскостью")
fit <- lm(mpg ~ wt+disp)
s3d$plane3d(fit)
```

Полученная диаграмма представлена на рис. 11.13.

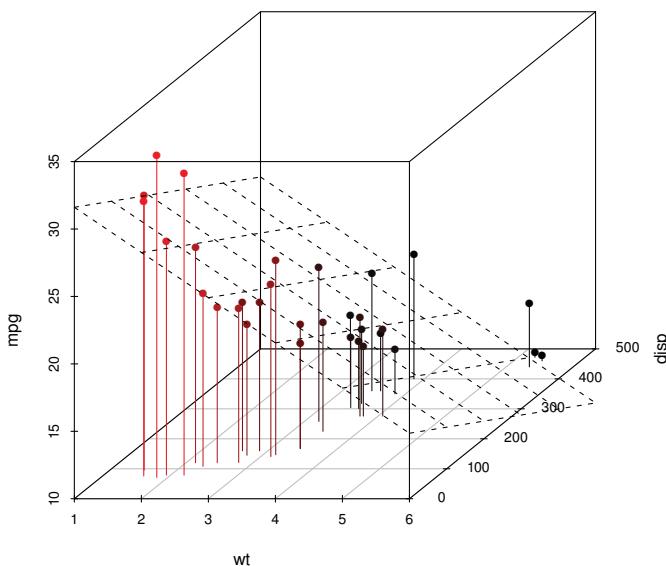
Трехмерная диаграмма рассеяния с вертикальными отрезками и регрессионной плоскостью


Рис. 11.13. Трехмерная диаграмма рассеяния с вертикальными линиями, градиентом яркости и наложенной регрессионной плоскостью

Эта диаграмма позволяет вам визуализировать прогноз расхода топлива на основе веса автомобиля и объема двигателя, сделанный при помощи уравнения множественной регрессии. Плоскость представляет собой предсказанные значения, а точки – это реальные наблюдения. Длина вертикальных отрезков между точками и прямой – это остатки. Регрессионные оценки значений лежащих над плоскостью точек занижены, а для лежащих под плоскостью точек – завышены. Множественная регрессия обсуждается в главе 8.

Вращение трехмерных диаграмм рассеяния

Трехмерные диаграммы рассеяния гораздо легче интерпретировать, если вы можете с ними взаимодействовать. В R реализовано несколько способов вращения диаграмм, так что точки можно видеть под разными углами.

Например, можно создать интерактивную трехмерную диаграмму рассеяния при помощи функции `plot3d()` из пакета `rgl`. Эта функ-

ция создает трехмерную диаграмму рассеяния, которую можно вращать при помощи мыши. Формат применения этой функции таков:

```
plot3d(x, y, z),
```

где x , y и z – это числовые векторы, определяющие координаты точек. Можно также добавить параметры типа `col` и `size`, чтобы задать цвет и размер точек. Продолжая наш пример, попробуйте ввести команды

```
library(rgl)
attach(mtcars)
plot3d(wt, disp, mpg, col="red", size=5)
```

У вас должна получиться диаграмма вроде той, что представлена на рис. 11.14. Используйте мышь, чтобы вращать оси. Я надеюсь, вы придетете к выводу, что возможность вращения диаграммы рассеяния в трехмерном пространстве делает ее гораздо более доступной для понимания.

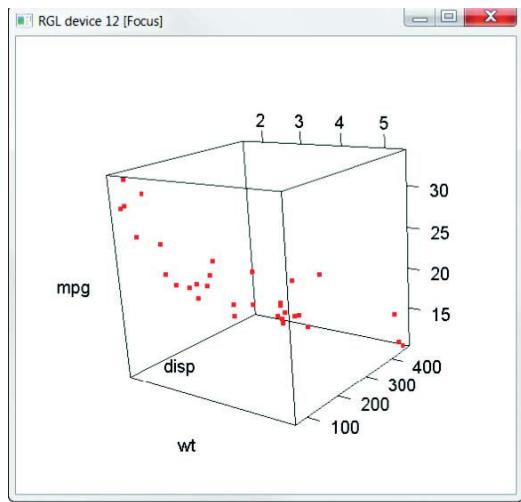


Рис. 11.14. Вращающаяся диаграмма рассеяния, полученная при помощи функции `plot3d()` из пакета `rgl`

Примерно такого же результата можно добиться, используя функцию `scatter3d()` из пакета `Rcmdr`:

```
library(Rcmdr)
attach(mtcars)
scatter3d(wt, disp, mpg)
```

Результат представлен на рис. 11.15.

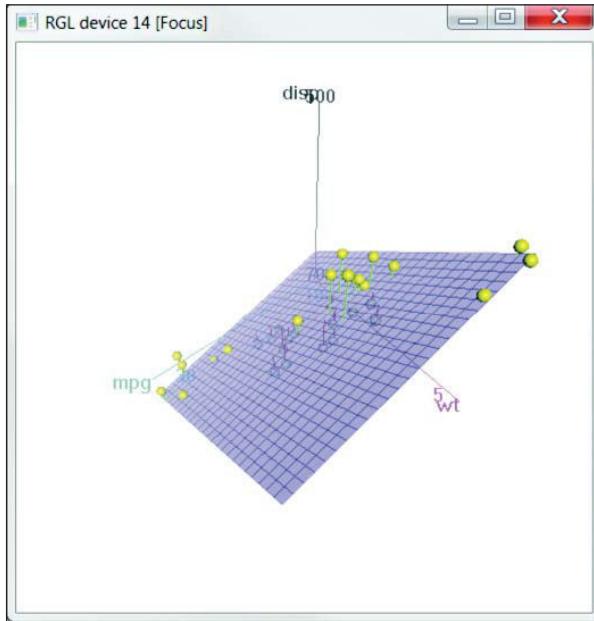


Рис. 11.15. Вращающаяся диаграмма рассеяния, полученная при помощи функции `scatter3d()` из пакета Rcmdr

При помощи функции `scatter3d()` можно изображать поверхности для разных типов регрессии, таких как линейная, квадратичная, сглаженная и аддитивная. По умолчанию изображается поверхность для линейной регрессии. Существуют дополнительные возможности обозначения точек в интерактивном режиме. Наберите `help(scatter3d)` для более подробной информации. Я подробнее расскажу о пакете Rcmdr в приложении А.

11.1.4. Пузырьковые диаграммы

В предыдущем разделе мы изображали взаимосвязи между тремя количественными переменными при помощи трехмерной диаграммы рассеяния. Другой подход – это сделать двухмерную диаграмму рассеяния и использовать размер точек для отражения значений третьей переменной. В результате получится так называемая *пузырьковая диаграмма* (bubble plot).

Эту диаграмму можно создать при помощи функции `symbols()`. Данную функцию можно использовать для изображения кругов,

квадратов, звезд, «термометров» и диаграмм рассеяния с заданными координатами (x, y). Круги изображаются при помощи следующей команды:

```
symbols(x, y, circle=radius),
```

где x, y и $radius$ – это векторы, которые задают x и y -координаты и диаметры кругов соответственно.

Если вы хотите, чтобы значения третьей переменной были пропорциональны площади, а не радиусу кругов, то нужно будет использовать такую команду:

```
symbols(x, y, circle=sqrt(z/pi)),
```

где z – это третья переменная, значения которой мы хотим отобразить.

Давайте применим эту команду к набору данных `mtcars`, отобразив значения веса автомобилей по оси x , расход топлива – по оси y , а объем двигателя – при помощи радиуса кругов. Такие команды

```
attach(mtcars)
r <- sqrt(disp/pi)
symbols(wt, mpg, circle=r, inches=0.30,
        fg="white", bg="lightblue",
        main="Пузырьковая диаграмма, на которой размер кругов
→ пропорционален объему двигателя",
        ylab="Расход топлива (мили на галлон)",
        xlab="Вес автомобиля (тысячи фунтов)")
text(wt, mpg, rownames(mtcars), cex=0.6)
detach(mtcars)
```

позволяют получить представленную на рис. 11.16 диаграмму. Параметр `inches` – это масштабный коэффициент, который можно использовать для определения размера кругов (по умолчанию радиус самого большого круга равен одному дюйму). Функция `text()` не обязательна. Здесь она используется для того, чтобы подписать названия автомобилей. Из диаграммы видно, что увеличение расхода топлива связано с уменьшением как веса машин, так и объема двигателя.

В целом специалисты, вовлеченные в работу над программой R, стремятся избегать пузырьковых диаграмм по тем же причинам, по которым они не любят круговых диаграмм. Людям, как правило, труднее сравнивать объемы², чем расстояния. Однако пузырьковые диаграммы весьма популярны в деловом мире, поэтому я поместил их в эту книгу для полноты изложения.

2 И площади. – Прим. пер.

Пузырьковая диаграмма, на которой размер кругов пропорционален объему двигателя

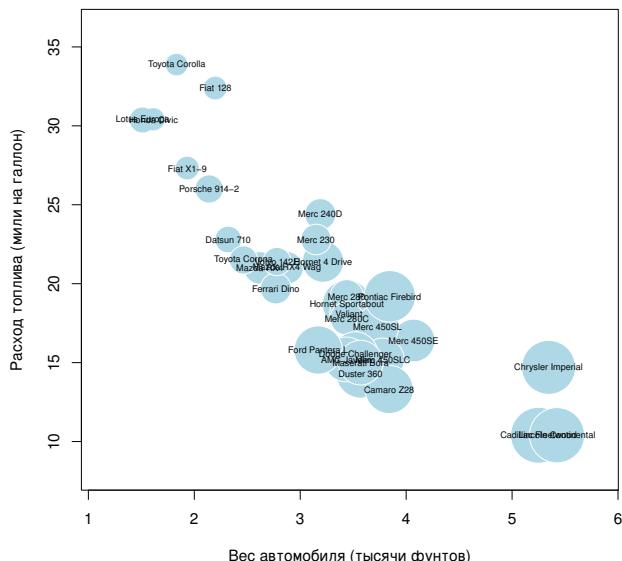


Рис. 11.16. Пузырьковая диаграмма для зависимости между весом автомобиля и расходом топлива, где размер кругов пропорционален объему двигателя

Мне, конечно, пришлось долго рассказывать про диаграммы рассеяния. Это внимание к деталям частично обусловлено главенствующим положением, которое занимают диаграммы рассеяния в анализе данных. Они, несмотря на свою простоту, помогают наглядно представить данные, демонстрируя связи, которые иначе могли остаться незамеченными.

11.2. Линейные графики

Если вы последовательно соедините точки диаграммы рассеяния, двигаясь слева направо, то у вас получится линейный график. Набор данных Orange, поставляемый с базовой версией программы, содержит данные о возрасте и обхвате пяти апельсиновых деревьев. Изучим рост первого дерева, данные о котором представлены на рис. 11.7. Слева дана диаграмма рассеяния, а справа – линейный график. Видно, что линейные графики особенно полезны, когда нужно проследить за изменениями значений анализируемой величины.

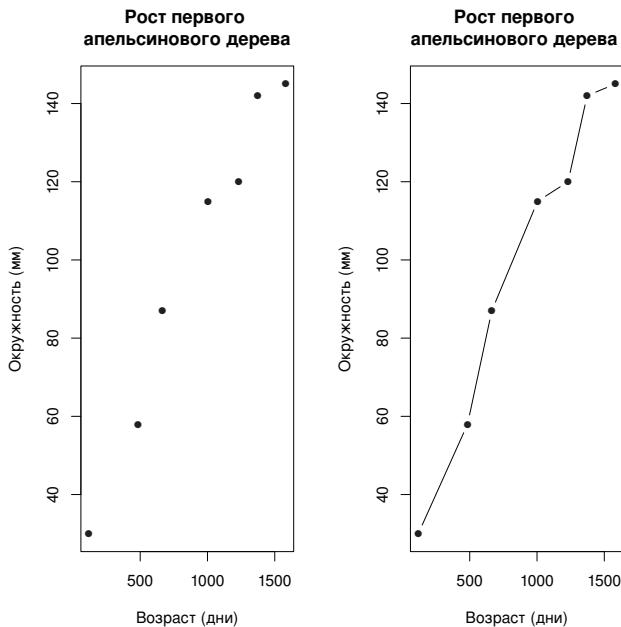


Рис. 11.17. Сравнение диаграммы рассеяния и линейного графика

Диаграммы на рис. 11.17 созданы при помощи команд, содержащихся в программном коде 11.3.

Программный код 11.3. Создание диаграммы рассеяния и линейного графика, расположенных рядом друг с другом

```
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))
t1 <- subset(Orange, Tree==1)
plot(t1$age, t1$circumference,
     xlab="Возраст (дни)",
     ylab="Окружность (мм)",
     main="Рост первого апельсинового дерева")
plot(t1$age, t1$circumference,
     xlab="Возраст (дни)",
     ylab="Окружность (мм)",
     main="Рост первого апельсинового дерева",
     type="b")
par(opar)
```

Вы видели все составные части этого кода в главе 3, так что здесь я не буду вдаваться в детали. Основное различие между двумя диаграммами на рис. 11.17 обусловлено добавлением параметра `type="b"`.

В общем случае графики создаются при помощи одной из этих функций:

```
plot(x, y, type=),
lines(x, y, type=),
```

где x и y – это числовые векторы, содержащие координаты точек, которые нужно соединить. Параметр `type`= может принимать значения, приведенные в табл. 11.1.

Таблица 11.1. Значения графического параметра `type`

Значения параметра	Что отображается
p	Только точки
l	Только линии
o	Точки, наложенные на линию
b, c	Точки (отсутствуют в случае c), соединенные отрезками
s, S	Ступенчатая линия
h	Вертикальные отрезки, как на гистограмме
n	Никакие точки или линии не отображаются (используется, чтобы задать оси для последующих построений)

Примеры для каждого значения этого параметра представлены на рис. 11.18. Можно видеть, что `type="p"` позволяет получить обычную диаграмму рассеяния. Параметр `type="b"` наиболее часто используется для построения графиков. Различие между `b` и `c` заключается в том, что точки либо отображаются, либо на их месте остается пробел. Параметры `type="s"` и `type="S"` позволяют получить ступенчатую линию (для соответствующих функций). В первом случае сначала идет горизонтальный отрезок, а потом – вертикальный, а во втором случае – наоборот.

Между функциями `plot()` и `lines()` существует одно важное различие. Функция `plot()` создает новую диаграмму. Функция `lines()` добавляет информацию на существующую диаграмму, но *не может* сама создать график.

Из-за этого функция `lines()` обычно применяется после того, как диаграмма уже создана посредством команды `plot()`. При необходимости можно использовать аргумент `type="n"` для команды `plot()`, чтобы создать оси, подписи и прочие графические элементы, а затем использовать функцию `lines()` для изображения различных линий на диаграмме.

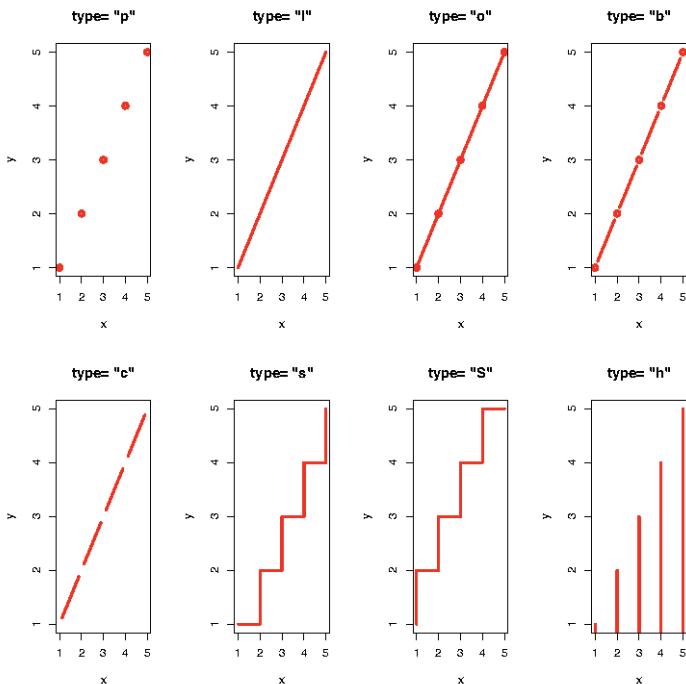


Рис. 11.18. Диаграммы, полученные при разных значениях параметра `type` функций `plot()` и `lines()`

В качестве примера создания более сложного графика давайте одновременно изобразим рост всех пяти апельсиновых деревьев. Каждому дереву будет соответствовать своя линия. Команды перечислены в следующем программном коде, а результаты представлены на рис. 11.19.

Программный код 11.4. График, на котором отображен рост пяти апельсиновых деревьев

```
Orange$Tree <- as.numeric(Orange$Tree)
ntrees <- max(Orange$Tree)
xrange <- range(Orange$age)
yrange <- range(Orange$circumference)
plot(xrange, yrange,
      type="n",
      xlab="Возраст (дни)",
      ylab="Окружность (мм)"
)
```

← **Преобразуем фактор в численный вид для удобства**

Создаем диаграмму

```

colors <- rainbow(ntrees)
linetype <- c(1:ntrees)
plotchar <- seq(18, 18+ntrees, 1)
for (i in 1:ntrees) {
  tree <- subset(Orange, Tree==i)
  lines(tree$circumference,
        type="b",
        lwd=2,
        lty=linetype[i],
        col=colors[i],
        pch=plotchar[i])
}
title("Рост деревьев", "пример линейного графика")
legend(xrange[1], yrange[2],
       1:ntrees,
       cex=0.8,
       col=colors,
       pch=plotchar,
       lty=linetype,
       title="Дерево"
)

```

Добавляем
линии

Добавляем
легенду

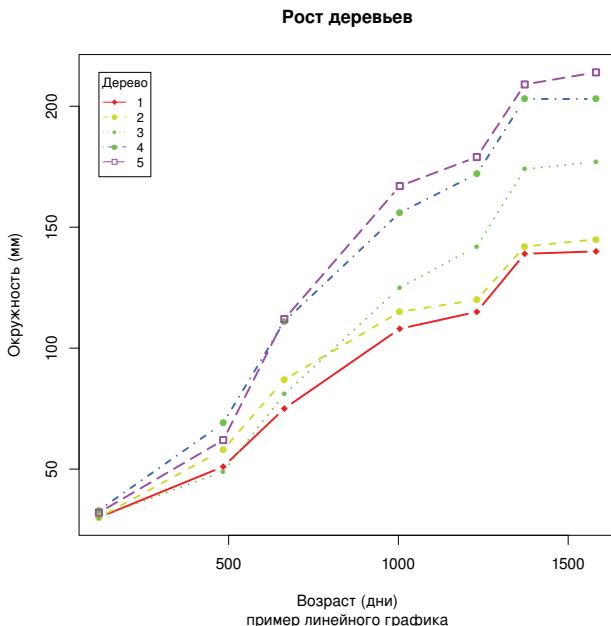


Рис. 11.19. График, отображающий рост пяти апельсиновых деревьев

В программном коде 11.4 функция `plot()` используется, чтобы создать «заготовку» для графика, подписать оси и определить диапазон их значений, но никакие данные при этом не отображаются. Затем для добавления отдельной линии и набора точек для каждого апельсинового дерева применяется функция `lines()`. Видно, что деревья 4 и 5 за время наблюдений выросли больше остальных и что дерево 5 переросло дерево 4 примерно на 664 дня.

В программном коде 11.4 нашли отражение многие особенности команд R, обсуждавшиеся в главах 2–4. Возможно, стоит проверить, как вы усвоили материал, разобрав этот код строчка за строчкой и узнав, что делает каждая из них. Если у вас получится, вы находитесь на пути к тому, чтобы стать настоящим программистом на языке R (слава и удача уже близко!). В следующем разделе мы рассмотрим способы визуализации многих коэффициентов корреляции одновременно.

11.3. Кореллограммы

Матрицы корреляции – это один из основных элементов многомерной статистики. Какие переменные из рассматриваемых сильно коррелируют друг с другом, а какие – нет? Существуют ли кластеры переменных, которые связаны между собой определенным способом? С увеличением числа переменных ответить на такие вопросы становится все сложнее. Кореллограммы – это сравнительно недавно появившийся способ для визуализации корреляционных матриц.

Проще всего объяснить, что такое кореллограмма, если вы уже однажды ее видели. Рассмотрим корреляции между переменными в наборе данных `mtcars`. В нем есть 11 переменных, каждая из которых отвечает какой-нибудь характеристике 32 автомобилей. Вы можете вычислить корреляционную матрицу³ при помощи следующих команд:

```
> options(digits=2)
> cor(mtcars)
   mpg cyl disp hp drat wt qsec vs am gear carb
mpg  1.00 -0.85 -0.85 -0.78  0.681 -0.87  0.419  0.66  0.600  0.48 -0.551
cyl -0.85  1.00  0.90  0.83 -0.700  0.78 -0.591 -0.81 -0.523 -0.49  0.527
disp -0.85  0.90  1.00  0.79 -0.710  0.89 -0.434 -0.71 -0.591 -0.56  0.395
hp   -0.78  0.83  0.79  1.00 -0.449  0.66 -0.708 -0.72 -0.243 -0.13  0.750
drat  0.68 -0.70 -0.71 -0.45  1.000 -0.71  0.091  0.44  0.713  0.70 -0.091
wt   -0.87  0.78  0.89  0.66 -0.712  1.00 -0.175 -0.55 -0.692 -0.58  0.428
```

³ Конечно, рассчитывать коэффициент корреляции Пирсона (а именно он, как вы помните, вычисляется по умолчанию при помощи команды `cor`) для дихотомических переменных, таких как `am`, некорректно. – *Прим. пер.*

qsec	0.42	-0.59	-0.43	-0.71	0.091	-0.17	1.000	0.74	-0.230	-0.21	-0.656
vs	0.66	-0.81	-0.71	-0.72	0.440	-0.55	0.745	1.00	0.168	0.21	-0.570
am	0.60	-0.52	-0.59	-0.24	0.713	-0.69	-0.230	0.17	1.000	0.79	0.058
gear	0.48	-0.49	-0.56	-0.13	0.700	-0.58	-0.213	0.21	0.794	1.00	0.274
carb	-0.55	0.53	0.39	0.75	-0.091	0.43	-0.656	-0.57	0.058	0.27	1.000

Какие из переменных больше всего связаны между собой? Какие переменные относительно независимы? Есть ли тут какая-нибудь структура? Не так-то легко ответить на такие вопросы, просто глядя на эту корреляционную матрицу и не затратив много времени и усилий (и, возможно, не вооружившись набором цветных ручек для заметок).

Эту же самую корреляционную матрицу можно представить графически при помощи функции `corrgram()` из пакета `corrgram` (см. рис. 11.20). Необходимые команды таковы:

```
library(corrgram)
corrgram(mtcars, order=TRUE, lower.panel=panel.shade,
         upper.panel=panel.pie, text.panel=panel.txt,
         main="Кореллограмма для набора данных mtcars")
```

Кореллограмма для набора данных `mtcars`

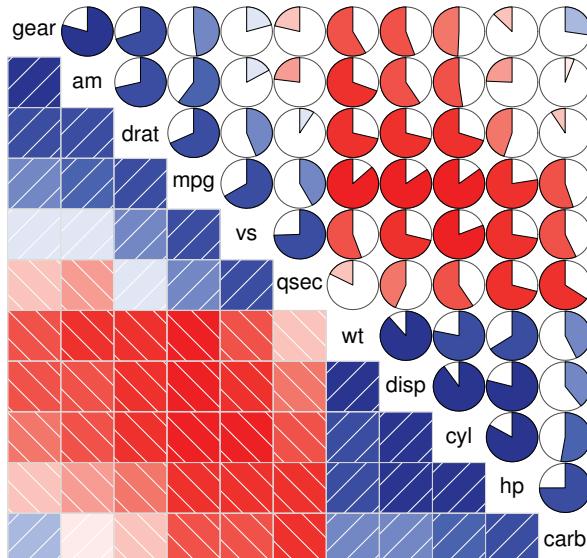


Рис. 11.20. Кореллограмма для корреляций между переменными из таблицы данных `mtcars`. Порядок строк и столбцов был изменен по результатам анализа главных компонент

Чтобы понять, что означает эта диаграмма, начнем с нижнего треугольника ячеек (ячейки ниже главной диагонали). По умолчанию голубой цвет и штриховка из левого нижнего угла к правому верхнему соответствуют положительной корреляции между двумя переменными, на пересечении которых находится данная ячейка. Напротив, красный цвет и штриховка из верхнего левого угла к правому нижнему соответствуют отрицательной корреляции. Чем темнее и насыщеннее цвет, тем сильнее корреляция. Слабые, близкие к нулю корреляции будут представлены «выцветшими» ячейками. На представленной диаграмме порядок строк и столбцов был автоматически изменен по результатам анализа главных компонент так, чтобы переменные со сходной корреляционной структурой формировали кластеры.

Можно видеть, что переменные gear, am, drat и mpg положительно коррелируют друг с другом. Также заметно, что переменные wt, disp, cyl, hp и carb тоже положительно коррелируют между собой. Однако переменные из первой группы отрицательно коррелируют с переменными второй группы. Очевидно и то, что корреляция между переменными carb и am слабая; это же справедливо для таких пар переменных, как vs и gear, vs и am, drat и qsec.

На верхнем треугольнике диаграммы та же информация представлена в виде круговых диаграмм. Здесь цвета имеют такое же значение, а сила корреляции выражена в размере закрашенного сегмента круговой диаграммы. Сегменты, соответствующие положительным корреляциям, начинаются от положения «12 часов» и заполняют круг по часовой стрелке. Сегменты, соответствующие отрицательным корреляциям, заполняют круг против часовой стрелки. Формат применения функции corrrgram() таков:

```
corrrgram(x, order=, panel=, text.panel=, diag.panel=),
```

где x – это таблица данных с одним наблюдением на строку. Если order=TRUE, то порядок переменных изменяется согласно результатам анализа главных компонент корреляционной матрицы. Такая сортировка переменных помогает обнаружить закономерности в корреляционной структуре.

Параметр panel определяет вид диаграммы (кроме главной диагонали – ее свойства задаются отдельно). Вместо него можно использовать параметры lower.panel и upper.panel, чтобы отдельно определять вид нижней и верхней (по отношению к главной диагонали) половин диаграммы. Параметры text.panel и diag.panel относятся

к главной диагонали. Допустимые значения параметра `panel` приведены в табл. 11.2.

Таблица 11.2. Значения параметра `panel` для функции `corrgram()`

Положение	Значение параметра	Описание
Не на главной диагонали	<code>panel.pie</code>	Закрашенный сегмент круговой диаграммы соответствует силе корреляции
	<code>panel.shade</code>	Интенсивность цвета соответствует силе корреляции
	<code>panel.ellipse</code>	Изображаются доверительный эллипс и сглаженная линия
	<code>panel.pts</code>	Изображается диаграмма рассеяния
	<code>panel.minmax</code>	Приводятся минимальное и максимальное значения переменной
	<code>panel.txt</code>	Отображается название переменной

Давайте попробуем рассмотреть следующий пример. Команды

```
library(corrgram)
corrgram(mtcars, order=TRUE, lower.panel=panel.ellipse,
         upper.panel=panel.pts, text.panel=panel.txt,
         diag.panel=panel.minmax,
         main="Кореллограмма для набора данных mtcars с использованием
         диаграмм рассеяния и эллипсов")
```

позволяют получить диаграмму, представленную на рис. 11.21. Здесь использованы доверительные линии вместе со сглаживающими линиями для нижнего треугольника и диаграммы рассеяния – для верхнего.

Почему диаграммы рассеяния выглядят странно?

Некоторые переменные, представленные на рис. 11.21, имеют ограниченный набор допустимых значений. Например, число передач может быть равным 3, 4 или 5. Число цилиндров равно 4, 6 или 8. Переменные `am` и `vs` – дихотомические. Это объясняет наличие необычно выглядящих диаграмм рассеяния на верхней панели.

Всегда проверяйте соответствие применяемых статистических методов типу ваших данных. Преобразование таких переменных в упорядоченные или неупорядоченные факторы может помочь вам в этом. Если R «знает», что переменная – категориальная или порядковая, то она «постарается» применить статистические методы, которые подходят для данного типа переменных.

**Коррелограмма для набора данных mtcars
с использованием диаграмм рассеяния и эллипсов**

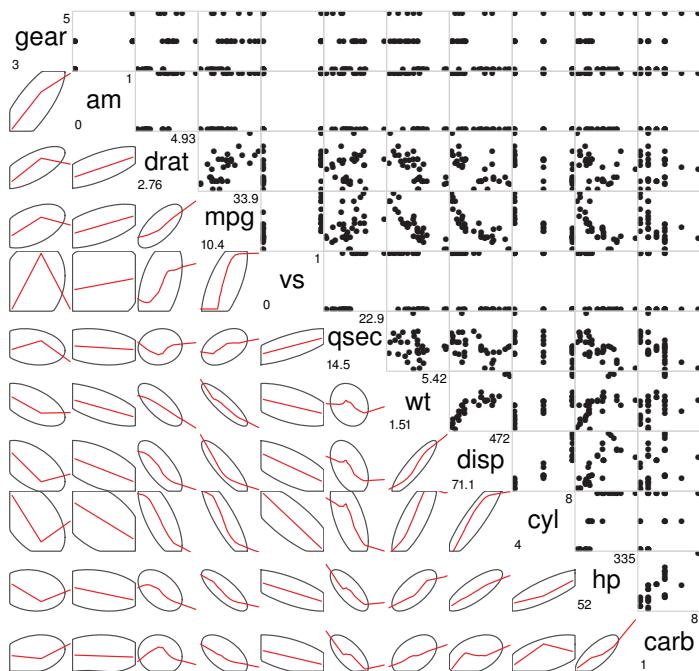


Рис. 11.21. Коррелограмма для корреляционной матрицы между переменными из таблицы данных mtcars. В нижнем треугольнике представлены сглаживающие линии вместе с доверительными эллипсами, а в верхнем треугольнике приведены диаграммы рассеяния. На диагонали даны минимальные и максимальные значения. Строки и столбцы упорядочены согласно результатам анализа главных компонент

В заключение приведем еще один пример. Команды

```
library(corrgram)
corrgram(mtcars, lower.panel=panel.shade,
         upper.panel=NULL, text.panel=panel.txt,
         main="Данные о расходе топлива (несортированные)")
```

позволяют получить диаграмму, представленную на рис. 11.22. В данном случае мы используем оттенки цвета в нижнем треугольнике, сохраняем исходный порядок переменных и оставляем верхний треугольник пустым.

Данные о расходе топлива (несортированные)

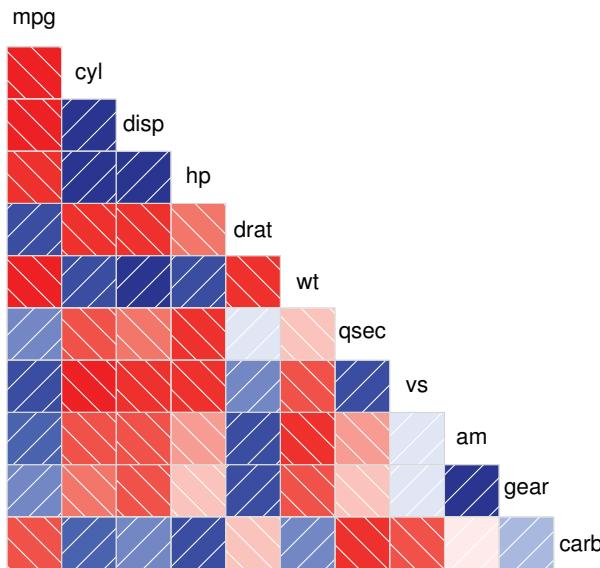


Рис. 11.22. Коррелограмма для корреляционной матрицы между переменными из таблицы данных mtcars. Оттенки цвета в нижнем треугольнике отражают силу и направление корреляций. Исходный порядок переменных сохранен

Прежде чем двигаться дальше, я должен отметить, что цветами, которые использует функция `corrgram()`, можно управлять. Для этого нужно указать четыре цвета для функции `colorRampPalette()` внутри функции `col.corrgram()`. Вот пример:

```
library(corrgram)
col.corrgram <- function(ncol){
  colorRampPalette(c("darkgoldenrod4", "burlywood1",
    "darkkhaki", "darkgreen"))(ncol)}
corrgram(mtcars, order=TRUE, lower.panel=panel.shade,
  upper.panel=panel.pie, text.panel=panel.txt,
  main="Коррелограмма другого цвета (совсем другое дело!)")
```

Попробуйте применить эти команды и посмотрите, что получится.

Коррелограммы могут быть хорошим способом изучения большого числа парных связей между количественными переменными. Поскольку это относительно новый метод, самое сложное – объяснить людям, как эти диаграммы нужно интерпретировать.

Чтобы узнать больше, обратитесь к статье Майкла Фрэндли (Michael Friendly) «Кореллограммы: визуальное исследование корреляционных матриц» («Corrrgrams: Exploratory Displays for Correlation Matrices»), доступной по адресу <http://www.math.yorku.ca/SCS/Papers/corrrgram.pdf>.

11.4. Мозаичные диаграммы

До сих пор мы исследовали методы визуализации взаимоотношений между количественными или непрерывными переменными. А что, если ваши переменные – категориальные? Когда вы исследуете одну категориальную переменную, можно использовать столбчатую или круговую диаграмму. Если у вас есть две категориальные переменные, можно построить для них трехмерную столбчатую диаграмму (которую, кстати, не так и просто сделать в R). Но что же вы будете делать с более чем двумя категориальными переменными?

Один из подходов заключается в использовании мозаичных диаграмм. В этих диаграммах частоты из многомерной таблицы сопряженности представлены в виде вложенных прямоугольников, размер которых пропорционален частотам. Для отображения остатков от подобранный модели могут быть использованы цвета и/или оттенки. Подробности изложены в публикации Meyer, Zeileis и Hornick (2006) или на страничке статистической графики Майкла Фрэндли (<http://datavis.ca>). Стив Симон (Steve Simon) написал хорошее пособие по созданию мозаичных диаграмм, доступное по адресу <http://www.childrensmercy.org/stats/definitions/mosaic.htm>.

Мозаичные диаграммы можно создавать при помощи функции `mosaic()` из пакета `vcd` (в базовой версии программы есть функция `mosaicplot()`, но я рекомендую использовать пакет `vcd` из-за его больших возможностей). В качестве примера рассмотрим набор данных `titanic`, который включен в базовую версию программы. Он содержит сведения о пассажирах, которые выжили или погибли, а также о классе, которым они путешествовали (первый, второй, третий, экипаж – Crew), поле (мужской – Male, женский – Female) и возрасте (ребенок – Child, взрослый – Adult). Это хорошо изученный набор данных. Число сочетаний разных признаков можно подсчитать при помощи такой команды:

```
> ftable(Titanic)
          Survived   No Yes
Class Sex      Age
```

1st	Male	Child	0	5
		Adult	118	57
	Female	Child	0	1
		Adult	4	140
2nd	Male	Child	0	11
		Adult	154	14
	Female	Child	0	13
		Adult	13	80
3rd	Male	Child	35	13
		Adult	387	75
	Female	Child	17	14
		Adult	89	76
Crew	Male	Child	0	0
		Adult	670	192
	Female	Child	0	0
		Adult	3	20

Функция `mosaic()` применяется следующим образом:

```
mosaic(table),
```

где `table` – это таблица сопряженности в виде массива данных, или

```
mosaic(formula, data=),
```

где `formula` – это формула в принятом в R формате, а параметр `data` задает или набор данных, или таблицу. Добавление параметра `shade=TRUE` позволит раскрасить диаграмму в соответствии со значениями пирсоновских остатков от подобранный модели (по умолчанию подразумевается независимость переменных), параметр `legend=TRUE` создает легенду, расшифровывающую значения использованных цветов.

К примеру, команды

```
library(vcd)
mosaic(Titanic, shade=TRUE, legend=TRUE),
```

а также команды

```
library(vcd)
mosaic(~Class+Sex+Age+Survived, data=Titanic, shade=TRUE, legend=TRUE)
```

позволяют получить диаграмму, представленную на рис. 11.23. Версия команды с формулой позволяет обрести больший контроль над расположением переменных на диаграмме.

В этой одной диаграмме содержится уйма информации. Например, вероятность выжить резко возрастает при перемещении из состава экипажа в первый класс. Большинство детей находилось во втором и третьем классах. Большинство женщин, путешествующих в первом

классе, выжили, а в третьем классе выжило только около половины женщин. В составе экипажа было очень мало женщин, и из-за этого на диаграмме индикаторы выживания (нет – No и да – Yes внизу диаграммы) наложились друг на друга. Продолжайте рассматривать диаграмму, и вы обнаружите много интересных фактов. Не забывайте сопоставлять относительную ширину и высоту прямоугольников. Что еще вы смогли узнать о той ночи?

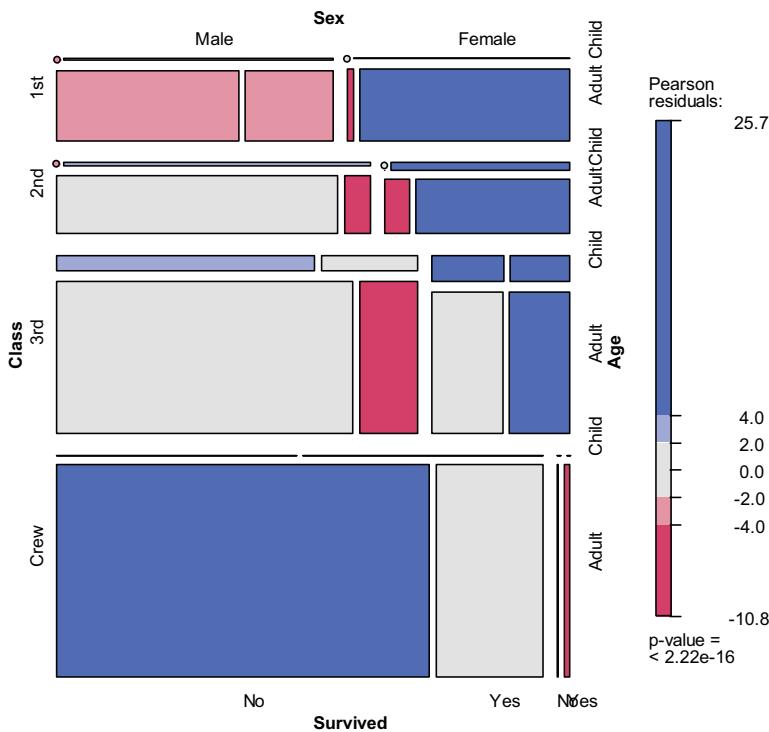


Рис. 11.23. Мозаичная диаграмма, которая характеризует пол, возраст людей, выживших при крушении Титаника, и класс, в котором они путешествовали

На усовершенствованных модификациях мозаичных диаграмм цвета и оттенки используются для отображения остатков от подобранной модели. На этой диаграмме оттенки синего соответствуют сочетаниям значений, которые имели место чаще, чем ожидалось в том случае, когда вероятность выжить не зависела от класса, возраста и пола. Оттенки красного соответствуют сочетаниям значений, ко-

торые наблюдались реже, чем ожидалось, исходя из модели о независимости переменных. Не забудьте запустить этот пример на своем компьютере, чтобы увидеть диаграмму в цвете. На ней видно, что выжило больше женщин из первого класса и погибло больше мужчин из экипажа, чем ожидалось, исходя из того, что вероятность выживания не зависела от класса, возраста и пола. Также выжило меньше мужчин из третьего класса, по сравнению с ожидаемыми значениями. Если вы хотите изучить мозаичные диаграммы более подробно, введите `example(mosaic)`.

11.5. Резюме

В этой главе мы рассмотрели разнообразные способы графического отображения взаимосвязей между двумя и более переменными, включая двух- и трехмерные диаграммы рассеяния, матрицы диаграмм рассеяния, пузырьковые диаграммы, линейные графики, коррелограммы и мозаичные диаграммы. Некоторые из этих методов стандартные, а некоторые – сравнительно новые.

Вместе с методами, которые позволяют настраивать диаграммы (глава 3), изображать распределение значений одномерных переменных (глава 6), исследовать регрессионные модели (глава 8) и визуализировать межгрупповые различия (глава 9), рассмотренные в этой главе подходы снабдили вас необходимым инструментарием для визуализации ваших данных и извлечения информации из них.

В последующих главах вы расширите свои умения за счет дополнительных методов, включая диаграммы для моделей с латентными переменными (глава 14), методы визуализации структуры пропущенных значений (глава 15) и диаграммы повышенной сложности (глава 16).

В следующей главе мы рассмотрим статистику повторных выборок и бутстреп-анализ. Это методы, которые требуют интенсивных вычислений и позволяют исследовать данные при помощи новых, не имеющих себе равных методов.



ГЛАВА 12.

Статистика повторных выборок и бутстреп-анализ

В этой главе:

- Понимание логики тестов с перестановками.
- Применение тестов с перестановками к линейным моделям.
- Использование бутстреп-анализа для вычисления доверительных интервалов.

В главах 7, 8 и 9 мы рассматривали статистические методы проверки гипотез и вычисление доверительных интервалов для параметров выборки, подразумевая, что полученные данные происходят из нормального распределения или другого типа распределения с хорошо известными свойствами. Однако существует много случаев, в которых выполнение этого допущения не гарантировано. Статистические подходы, основанные на рандомизации и повторных выборках, могут быть использованы в случаях, когда данные происходят из неизвестного или смешанного распределения, когда размеры выборок малы, когда выбросы представляют проблему или когда разработка подходящего теста, основанного на теоретическом распределении, слишком сложна и затруднительна с математической точки зрения.

В этой главе мы рассмотрим два общих статистических подхода с использованием рандомизации: перестановочные тесты и бутстреп-анализ. Раньше эти методы были доступны только опытным программистам и экспертам в области статистики. Теперь появление созданных пользователями пакетов для R сделало эти методы легко доступными для всех, кто обрабатывает данные.

Мы также вернемся к задачам, которые раньше анализировали при помощи традиционных методов (например, тестов Стьюдента и хиквадрат, дисперсионного и регрессионного анализа), и посмотрим, как можно решить эти задачи при помощи устойчивых к ошибкам методов, которые требуют большого объема вычислений. Для лучшего понимания раздела 12.2 необходимо сначала прочесть главу 7. Главы 8 и 9 содержат информацию, нужную для понимания раздела 12.3. Остальные разделы можно читать без предварительной подготовки.

12.1. Перестановочные тесты

Перестановочные тесты (permutation tests), также известные как тесты с рандомизацией, были известны в течение десятилетий, но стали доступными лишь с изобретением высокопроизводительных компьютеров.

Для того чтобы понять логику тестов с перестановками, рассмотрим следующую воображаемую задачу. Десять объектов были случайно подвергнуты одному из двух воздействий (A или B), а затем были зарегистрированы значения результирующей переменной (*score*). Результаты эксперимента представлены в табл. 12.1.

Таблица 12.1. Воображаемая задача с двумя группами

Воздействие A	Воздействие B
40	57
57	64
45	55
55	62
58	65

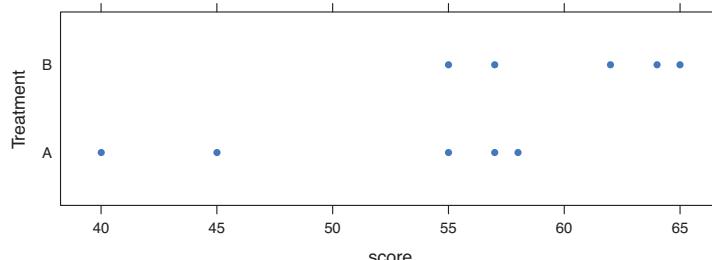


Рис. 12.1. Одномерная диаграмма рассеяния для воображаемых данных, представленных в табл. 12.1

При параметрическом подходе можно предположить, что данные происходят из нормальных распределений с одинаковой дисперсией, и применить тест Стьюдента для двух независимых групп. Нулевая гипотеза заключается в том, что среднее значение для выборки А равно среднему для выборки В. Вы бы вычислили t -статистику для данных и сравнили ее с теоретическим распределением. Если бы вычисленное значение t -статистики достаточно сильно отличалось от теоретического, скажем, находилось бы вне 95% значений теоретического распределения, вы бы отвергли нулевую гипотезу и заявили, что выборочные средние для этих двух групп неодинаковы с вероятностью 95%.

При teste с перестановками применяется другой подход. Если два воздействия действительно равнозначны, то названия (воздействие А или воздействие В) присвоены им случайным образом. Мы можем проверить, существуют ли различия между воздействиями, при помощи следующего алгоритма.

1. Вычислить выборочную t -статистику как при параметрическом подходе, назовем ее t_0 .
2. Поместить все 10 значений в одну группу.
3. Случайно поместить пять значений в группу А и пять – в группу В.
4. Вычислить новую t -статистику.
5. Повторить шаги 3–4 для всех возможных способов размещения пяти значений в группу А и пяти – в группу В. Всего существует 252 способа.
6. Расположить 252 значения t -статистики в порядке возрастания. Это эмпирическое распределение, основанное на выборках.
7. Если t_0 не входит в центральные 95% значений эмпирического распределения, то следует с вероятностью 95% отвергнуть нулевую гипотезу о равенстве средних значений в двух группах.

Обратите внимание на то, что одна и та же t -статистика рассчитывается и при перестановочном, и при параметрическом подходах. Однако вместо сравнения статистики с теоретическим распределением для выяснения того, достаточно ли она отличается от ожидаемых значений, чтобы можно было отклонить нулевую гипотезу, статистика сравнивается с эмпирическим распределением, полученным при перестановках данных. Эта логика может быть распространена на большинство классических статистических тестов и линейных моделей.

В только что приведенном примере эмпирическое распределение было основано на всех возможных перестановках данных. В таких случаях тест с перестановками называется «точным» тестом. С увеличением размеров выборок необходимое для проведения всех возможных перестановок время может стать непомерно большим. В таких случаях можно использовать моделирование по методу Монте-Карло для создания выборки из всех возможных перестановок. При таком образе действий тест получается приближенным.

Если вы не уверены в том, что данные распределены нормально, задумываетесь о влиянии выбросов или чувствуете, что набор данных может быть недостаточно большим для стандартных параметрических подходов, перестановочный тест послужит отличной альтернативой параметрическим методам.

В R реализованы самые сложные и широко применимые методы проведения перестановочных тестов из имеющихся в настоящее время. В этом разделе мы рассмотрим два пакета: `coin` и `lmPerm`. Не забудьте установить их перед тем, как использовать в первый раз:

```
install.packages(c("coin", "lmPerm"))
```

Пакет `coin` представляет собой обширный набор тестов с перестановками для проверки данных на независимость, тогда как в пакете `lmPerm` реализованы тесты с перестановками для дисперсионного и регрессионного анализов. Мы рассмотрим их по очереди, а в конце раздела кратко упомянем другие пакеты для проведения тестов с перестановками в R.

Прежде чем двигаться дальше, нужно вспомнить, что перестановочные тесты используют псевдослучайные числа для создания выборки из всех возможных перестановок (для приближенных тестов). Таким образом, результаты тестов каждый раз будут немного разными. Для получения одинаковых результатов нужно задать начальное число для генерации случайных величин. Это особенно полезно, если вы хотите поделиться своими результатами с другими. Установка начального числа, равного 1234 (то есть `set.seed(1234)`), позволит вам воспроизвести результаты, представленные в этой главе.

12.2. Перестановочные тесты в пакете `coin`

В пакете `coin` реализованы разнообразные тесты с перестановками для проверки данных на независимость. При помощи этого пакета мы можем ответить на такие вопросы, как:

- Зависят ли результаты от принадлежности к группе?
- Независимы ли две числовые переменные?
- Независимы ли две категориальные переменные?

Используя удобные функции, реализованные в этом пакете (см. табл. 12.2), мы можем провести перестановочные тесты, аналогичные большинству традиционных статистических тестов, описанных в главе 7.

Таблица 12.2. Функции, реализованные в пакете `coin`, которые позволяют выполнить перестановочные тесты, эквивалентные традиционным статистическим тестам

Тест	Функция пакета <code>coin</code> *
Перестановочный тест для двух и k выборок	<code>oneway_test(y ~ A)</code>
Перестановочный тест для двух и k выборок с группирующим фактором	<code>oneway_test(y ~ A C)</code>
Тест ранговых сумм Вилкоксона-Манна-Уитни	<code>wilcox_test(y ~ A)</code>
Тест Краскела-Уоллиса	<code>kruskal_test(y ~ A)</code>
Хи-квадрат тест Пирсона	<code>chisq_test(A ~ B)</code>
Тест Кохрана-Мантеля-Гензеля	<code>cmh_test(A ~ B C)</code>
Критерий линейной зависимости (Linear-by-linear association test)	<code>lbl_test(D ~ E)</code>
Тест Спирмена	<code>spearman_test(y ~ x)</code>
Тест Фридмана	<code>friedman_test(y ~ A C)</code>
Ранговый тест Вилкоксона	<code>wilcoxonsign_test(y1 ~ y2)</code>

* y и x – числовые переменные, A и B – категориальные переменные, C – категориальная группирующая переменная, D и E – упорядоченные факторы, y_1 и y_2 – парные числовые переменные.

Каждая из перечисленных в табл. 12.2 функций применяется в виде

`название_функции (formula, data= , distribution=),`

где

- `formula` описывает взаимосвязь между переменными, которую нужно проверить. Примеры приведены в таблице;
- `data` задает таблицу с данными;
- `distribution` указывает, как должно быть создано эмпирическое распределение для нулевой гипотезы. Возможные значения: `exact`, `asymptotic` и `approximate`.

Если `distribution="exact"`, распределение для нулевой гипотезы рассчитывается точно (то есть на основании всех возможных перестановок). Распределение может также быть приближенно вычислено (аппроксимировано) по асимптотическому распределению (`distribution="asymptotic"`) или методом перестановок Монте-Карло (`distribution="approximate(B=#)"`), где `#` означает число повторностей, использованных для приближенного вычисления точного распределения. В настоящее время опция `distribution="exact"` доступна только при работе с двумя выборками.

Замечание. Для анализа в пакете `coin` категориальные и порядковые переменные должны быть закодированы как факторы и упорядоченные факторы соответственно. Кроме того, данные должны быть представлены в формате таблицы данных.

В оставшейся части этого раздела мы применим несколько перестановочных тестов, упомянутых в табл. 12.2, для решения задач из предыдущих глав. Это позволит вам сравнить результаты непараметрических и более традиционных параметрических тестов. Мы закончим обсуждение пакета `coin` рассмотрением более сложных методов анализа.

12.2.1. Тесты на независимость для двух и k выборок

Для начала сравним тест Стьюдента для независимых выборок с односторонним точным тестом на примере воображаемых данных из табл. 12.2. Результаты представлены ниже.

Программный код 12.1. Сравнение теста Стьюдента с односторонним перестановочным тестом для воображаемых данных

```
> library(coin)
> score <- c(40, 57, 45, 55, 58, 57, 64, 55, 62, 65)
> treatment <- factor(c(rep("A",5), rep("B",5)))
> mydata <- data.frame(treatment, score)
> t.test(score~treatment, data=mydata, var.equal=TRUE)
    Two Sample t-test
data: score by treatment
t = -2.3, df = 8, p-value = 0.04705
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
-19.04 -0.16
sample estimates:
mean in group A mean in group B
51                  61
```

```
> oneway_test(score~treatment, data=mydata, distribution="exact")
    Exact 2-Sample Permutation Test
data: score by treatment (A, B)
Z = -1.9, p-value = 0.07143
alternative hypothesis: true mu is not equal to 0
```

Традиционный тест Стьюдента свидетельствует о статистически значимом различии между группами ($p < 0.05$), тогда как точный тест – нет ($p > 0.072$). Имея только 10 наблюдений, я больше склонен поверить результатам перестановочного теста и попытаться собрать больше данных, прежде чем прийти к окончательному заключению.

Далее рассмотрим тест Вилкоксона-Манна-Уитни. В главе 7 мы исследовали различия вероятности попасть в тюрьму в южных и не-южных штатах США, используя функцию `wilcox.test()`. Применив точный тест ранговых сумм Вилкоксона, мы получим следующий результат:

```
> library(MASS)
> UScrime <- transform(UScrime, So = factor(So))
> wilcox_test(Prob ~ So, data=UScrime, distribution="exact")
    Exact Wilcoxon Mann-Whitney Rank Sum Test
data: Prob by So (0, 1)
Z = -3.7, p-value = 8.488e-05
alternative hypothesis: true mu is not equal to 0
```

который свидетельствует о том, что вероятность угодить за решетку выше в южных штатах. Обратите внимание на то, что в приведенном программном коде числовая переменная `So` была преобразована в фактор. Это произошло потому, что, как уже упоминалось выше, для анализа в пакете `coin` все категориальные переменные должны быть перекодированы в факторы. Кроме того, внимательный читатель мог заметить, что эти результаты идеально согласуются с результатами функции `wilcox.test()`, приведенными в главе 7. Это произошло потому, что по умолчанию данная функция также рассчитывает точное распределение.

Наконец, рассмотрим тест для k выборок. В главе 9 мы использовали однофакторный дисперсионный анализ для оценки эффекта пяти разных режимов приема лекарств на снижение уровня холестерина для выборки из 50 пациентов. Вместо этого можно провести приближенный перестановочный тест для k выборок при помощи следующего программного кода:

```
> library(multcomp)
> set.seed(1234)
> oneway_test(response~trt, data=cholesterol,
```

```
distribution=approximate(B=9999)
Approximative K-Sample Permutation Test
data: response by
      trt (1time, 2times, 4times, drugD, drugE)
maxT = 4.7623, p-value < 2.2e-16
```

В данном случае опорное распределение рассчитано на основе 9999 перестановок данных. Мы задали начальное число для генерации случайных значений, так что ваши результаты будут такими же, как и мои. Результаты для пациентов из разных групп заметно различаются.

12.2.2. Независимость в таблицах сопряженности

Тесты с перестановками можно использовать для оценки независимости двух категориальных переменных при помощи функций `chisq_test()` или `cmh_test()`. Вторая функция применяется, когда данные разделены на группы согласно значениям третьей категориальной переменной. Если обе переменные порядковые, мы можем использовать функцию `cmh_test()` для проверки наличия линейного тренда.

В седьмой главе мы использовали тест хи-квадрат для проверки связи между лечением артрита и улучшением состояния больных. Переменная `Treatment` (лечение) имела два уровня (плацебо и лечение), а переменная `Improved` (улучшение) – три уровня (отсутствует, некоторое, заметное) – она была закодирована как упорядоченный фактор.

Если вы хотите провести перестановочный тест, аналогичный тесту хи-квадрат, можете использовать следующий программный код:

```
> library(coin)
> library(vcd)
> Arthritis <- transform(Arthritis,
+   Improved=as.factor(as.numeric(Improved)))
> set.seed(1234)
> chisq_test(Treatment~Improved, data=Arthritis,
+   distribution=approximate(B=9999))
Approximative Pearson's Chi-Squared Test
data: Treatment by Improved (1, 2, 3)
chi-squared = 13.055, p-value = 0.0018
```

Этот код позволяет провести приближенный тест хи-квадрат, основанный на 9999 перестановках. Вы могли бы поинтересоваться,

зачем переменная `Improved` была преобразована из упорядоченного фактора в категориальную переменную (хороший вопрос!). Если бы вы оставили ее в виде упорядоченного фактора, то пакет `coin` провел бы тест на линейный тренд вместо теста хи-квадрат. Хотя тест на линейный тренд был бы хорошим решением в данной ситуации, проведение теста хи-квадрат позволяет сравнить результаты с приведенными в главе 7.

12.2.3. Независимость между числовыми переменными

Функция `spearman_test()` позволяет провести тест с перестановками на независимость двух числовых переменных. В главе 7 мы исследовали корреляцию между уровнем неграмотности и уровнем преступности в разных штатах США. Эту связь можно проверить при помощи теста с перестановками посредством следующего программного кода:

```
> states <- as.data.frame(state.x77)
> set.seed(1234)
> spearman_test(Illiteracy~Murder, data=states,
+                 distribution=approximate(B=9999))
Approximative Spearman Correlation Test
data: Illiteracy by Murder
Z = 4.7065, p-value < 2.2e-16
alternative hypothesis: true mu is not equal to 0
```

На основании приближенного теста с 9999 перестановками нулевая гипотеза может быть отвергнута. Обратите внимание на то, что набор данных `state.x77` – это матрица. Перед использованием пакета `coin` ее нужно преобразовать в таблицу данных.

12.2.4. Тесты для двух и к зависимым выборок

Тесты для зависимых выборок используются, когда пары наблюдений в разных группах соответствуют друг другу или в случаях повторных измерений. Для перестановочного теста для двух зависимых выборок может быть использована функция `wilcoxonsign_test()`. Для более чем двух групп используйте функцию `friedman_test()`.

В главе 7 мы сравнивали уровень безработицы для горожан мужского пола в возрасте 14–24 лет (переменная `U1`) и в возрасте 35–39

лет (U_2). Поскольку значения обеих этих переменных известны для 50 штатов Америки, у нас есть две зависимые группы (`state` – штат – это задающая пары значений переменная). Для сравнения уровня безработицы в этих двух возрастных группах можно использовать ранговый тест Вилкоксона:

```
> library(coin)
> library(MASS)
> wilcoxsign_test(U1~U2, data=UScrime, distribution="exact")
      Exact Wilcoxon-Signed-Rank Test
data: y by x (neg, pos)
stratified by block
z = 5.9691, p-value = 1.421e-14
alternative hypothesis: true mu is not equal to 0
```

На основании полученных результатов можно заключить, что уровни безработицы различаются.

12.2.5. Дополнительная информация

В пакете `coin` реализован основной набор методов проверки независимости одной группы переменных от другой (с возможным разделением на группы согласно значениям еще одной сторонней переменной) при помощи перестановочных тестов. В частности, функция `independence_test()` позволяет пользователю реализовать большинство обычных тестов в «перестановочном» виде, а также создать новые и нестандартные статистические тесты для ситуаций, в которых традиционные подходы не работают. Такая гибкость имеет свою цену: для правильного использования этой функции требуется отличное значение статистики. За дальнейшими деталями следует обратиться к сопроводительной документации к пакету (доступной после введения команды `vignette("coin")`).

В следующем разделе вы познакомитесь с пакетом `lmPerm`. В нем реализованы перестановочные тесты для линейных моделей, включая регрессионный и дисперсионный анализ.

12.3. Перестановочные тесты, реализованные в пакете `lmPerm`

В пакете `lmPerm` реализован «перестановочный» подход к аналиzu линейных моделей. В частности, функции `lmp()` и `aovp()` – это аналоги функций `lm()` и `aov()`, модифицированные для проведения перестановочных тестов.

Параметры функций `lmp()` и `aovp()` сходны с параметрами функций `lm()` и `aov()`. Добавляется только опция `perm=`, которая может принимать значения “*Exact*” (точный тест, основанный на всех возможных перестановках), “*Prob*” или “*SPR*”. Значение “*Prob*” позволяет провести тест на основании выборки из возможных перестановок. Создание выборки прекращается, когда оцененное стандартное отклонение становится меньше 0.1 от оцененного значения статистической ошибки первого рода. Критерий остановки определяется дополнительным параметром `ca`. Наконец, параметр `SPR` позволяет использовать тест отношений последовательных вероятностей в качестве критерия остановки. Учтите, что если число наблюдений превышает 10, параметр `Exact` автоматически заменяется параметром `Prob`; точные тесты доступны только для малых выборок.

Для того чтобы понять, как это работает, мы применим «перестановочный» подход для простой, полиномиальной и множественной регрессии, одно- и двухфакторного дисперсионного анализа и однократного ковариационного анализа.

12.3.1. Простая и полиномиальная регрессия

В восьмой главе мы использовали линейную регрессию для изучения связи между весом и ростом в группе из 15 женщин. Использование функции `lmp()` вместо функции `lm()` позволяет получить результаты теста с перестановками, которые приведены ниже.

Программный код 12.2. Тест с перестановками для простой линейной регрессии

```
> library(lmPerm)
> set.seed(1234)
> fit <- lmp(weight~height, data=women, perm="Prob")
[1] "Settings: unique SS : numeric variables centered"
> summary(fit)
Call:
lmp(formula = weight ~ height, data = women, perm = "Prob")
Residuals:
    Min      1Q  Median      3Q      Max 
-1.733 -1.133 -0.383  0.742  3.117 
Coefficients:
            Estimate Iter Pr(Prob)    
height      3.45 5000   <2e-16 ***  
---
Signif. codes:  0 '****' 0.001 '***' 0.01 '**' 0.05 '*' 0.1 ' ' 1
Residual standard error: 1.5 on 13 degrees of freedom
```

```
Multiple R-Squared: 0.991,      Adjusted R-squared: 0.99  
F-statistic: 1.43e+03 on 1 and 13 DF,  p-value: 1.09e-14
```

Для подгонки квадратичной модели можно воспользоваться командами, приведенными в следующем программном коде.

Программный код 12.3. Перестановочный тест для полиномиальной регрессии

```
> library(lmPerm)  
> set.seed(1234)  
> fit <- lmp(weight~height + I(height^2), data=women, perm="Prob")  
[1] "Settings: unique SS : numeric variables centered"  
> summary(fit)  
Call:  
lmp(formula = weight ~ height + I(height^2), data = women, perm = "Prob")  
Residuals:  
    Min      1Q  Median      3Q     Max  
-0.5094 -0.2961 -0.0094  0.2862  0.5971  
Coefficients:  
            Estimate Iter Pr(Prob)  
height       -7.3483 5000   <2e-16 ***  
I(height^2)   0.0831 5000   <2e-16 ***  
---  
Signif. codes:  0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1  
Residual standard error: 0.38 on 12 degrees of freedom  
Multiple R-Squared: 0.999,      Adjusted R-squared: 0.999  
F-statistic: 1.14e+04 on 2 and 12 DF,  p-value: <2e-16
```

Как вы могли заметить, применять перестановочные тесты просто – для этого нужны лишь небольшие изменения программного кода. Результаты также выводятся в формате, сходном с форматом результатов функции `lm()`. Обратите внимание на добавление столбца `Iter`, в котором указано число итераций, потребовавшихся для достижения нужного значения критерия остановки.

12.3.2. Множественная регрессия

В главе 8 множественная регрессия была использована для предсказания уровня преступности по значениям численности населения, уровня неграмотности, дохода и морозности для 50 штатов Америки. Применение функции `lmp()` для решения этой задачи позволит получить следующие результаты.

Программный код 12.4. Перестановочный тест для множественной регрессии

```
> library(lmPerm)  
> set.seed(1234)  
> states <- as.data.frame(state.x77)
```

```
> fit <- lm(Population ~ Murder + Illiteracy + Income + Frost,
  data=states, perm="Prob")
[1] "Settings: unique SS : numeric variables centered"
> summary(fit)
Call:
lm(formula = Population ~ Murder + Illiteracy + Income + Frost,
  data = states, perm = "Prob")
Residuals:
    Min      1Q  Median      3Q     Max 
-4.79597 -1.64946 -0.08112  1.48150  7.62104 
Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
Population 2.237e-04   51    1.00000    
Illiteracy 4.143e+00 5000   0.0004 ***  
Income     6.442e-05   51    1.00000    
Frost      5.813e-04   51    0.8627    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
Residual standard error: 2.535 on 45 degrees of freedom
Multiple R-Squared:  0.567,    Adjusted R-squared:  0.5285 
F-statistic: 14.73 on 4 and 45 DF,  p-value: 9.133e-08
```

Как указано в восьмой главе, по результатам классических тестов и численность населения (`Population`), и уровень неграмотности (`Illiteracy`) статистически значимо ($p < 0.05$) влияли на уровень преступности. По результатам перестановочного теста численность населения больше не оказывает существенного эффекта на уровень преступности. Когда результаты двух подходов не совпадают, нужно более внимательно посмотреть на данные. Возможно, в данном случае не соблюдается требование нормального распределения или присутствуют выбросы.

12.3.3. Однофакторные дисперсионный и ковариационный анализы

Каждый из типов дисперсионного анализа, описанных в главе 9, можно провести при помощи перестановочных тестов. Во-первых, давайте рассмотрим задачу с однофакторным дисперсионным анализом, описанную в разделе 9.1, о влиянии разных способов лечения на снижение уровня холестерина. Команды и результаты их применения представлены в следующем программном коде.

Программный код 12.5. Перестановочный тест для однофакторного дисперсионного анализа

```
> library(lmPerm)
> library(multcomp)
```

```
> set.seed(1234)
> fit <- aovp(response~trt, data=cholesterol, perm="Prob")
[1] "Settings: unique SS"
> summary(fit)
Component 1 :
  Df R Sum Sq R Mean Sq Iter Pr(Prob)
trt      4  1351.37    337.84 5000 < 2.2e-16 ***
Residuals 45   468.75     10.42
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Полученные результаты свидетельствуют о том, что эффект от разных способов лечения неодинаков.

Второй пример в этом разделе касается применения перестановочного теста для проведения однофакторного ковариационного анализа. Задача описана в главе 9: она заключается в исследовании влияния четырех доз препаратов на вес мышат при разной продолжительности беременности. В следующем программном коде приведены перестановочный тест и его результаты.

Программный код 12.6. Перестановочный тест для однофакторного ковариационного анализа

```
> library(lmPerm)
> set.seed(1234)
> fit <- aovp(weight ~ gesttime + dose, data=litter, perm="Prob")
[1] "Settings: unique SS : numeric variables centered"
> summary(fit)
Component 1 :
  Df R Sum Sq R Mean Sq Iter Pr(Prob)
gesttime      1   161.49   161.493 5000   0.0006 ***
dose          3   137.12    45.708 5000   0.0392 *
Residuals    69  1151.27    16.685
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Из полученных значений статистической ошибки первого рода следует, что разные дозы препарата не одинаково влияют на вес мышат при разных сроках беременности.

12.3.4. Двухфакторный дисперсионный анализ

Мы закончим этот раздел выполнением двухфакторного дисперсионного анализа при помощи перестановочных тестов. В главе 9 мы исследовали влияние витамина С на рост зубов у морских свинок.

Двумя изменяемыми факторами были доза (три уровня) и способ получения витамина (два уровня). Каждой комбинации воздействий подвергалось по десять свинок, в результате чего был получен сбалансированный 3×2 факторный дизайн эксперимента. Перестановочный тест представлен в следующем программном коде.

Программный код 12.7. Перестановочный тест для двухфакторного дисперсионного анализа

```
> library(lmPerm)
> set.seed(1234)
> fit <- aovp(len~supp*dose, data=ToothGrowth, perm="Prob")
[1] "Settings: unique SS : numeric variables centered"
> summary(fit)
Component 1 :
   Df R Sum Sq R Mean Sq Iter Pr(Prob)
supp      1  205.35    205.35 5000 < 2e-16 ***
dose      1 2224.30    2224.30 5000 < 2e-16 ***
supp:dose  1     88.92     88.92 2032  0.04724 *
Residuals 56   933.63    16.67
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

С вероятностью 95% все три эффекта отличаются от нуля. На уровне 99% значим только главный эффект.

Важно отметить, что когда функция `aovp()` применяется к разным типам дисперсионного анализа, по умолчанию каждый эффект корректируется по всем остальным эффектам (используется так называемая уникальная сумма квадратов, или сумма квадратов SAS III типа). По умолчанию для параметрического дисперсионного анализа в R применяется последовательная сумма квадратов (SAS I типа). Каждый эффект корректируется по эффектам, которые появляются в модели *раньше*. Для сбалансированного дизайна эксперимента оба подхода дадут одинаковый результат, а для несбалансированного дизайна с разным числом наблюдений для разных комбинаций значений факторов результаты, полученные разными методами, будут различаться. Чем сильнее несбалансированность дизайна, тем больше несогласованность между методами. При необходимости можно вычислить последовательную сумму квадратов, добавив параметр `seqs=TRUE` в функцию `aovp()`. Для получения более подробной информации о суммах квадратов I и III типов обратитесь к разделу 9.2 главы 9.

12.4. Дополнительные замечания о перестановочных тестах

В R, помимо `coin` и `lmPerm`, есть другие пакеты для проведения перестановочных тестов. В пакете `perm` реализованы некоторые методы из пакета `coin`, поэтому функции из первого пакета можно использовать для проверки результатов функций из второго пакета. В пакете `corrperm` реализованы корреляционные перестановочные тесты для повторных измерений. Пакет `logregperm` содержит перестановочные тесты для логистической регрессии. И наверное, самое важное – пакет `glmperrm` распространяет перестановочные тесты на обобщенные линейные модели, которые обсуждаются в следующей главе.

Перестановочные тесты – это достойная альтернатива тестам, основанным на знании распределения, из которого происходит выборка. При помощи каждого из описанных выше перестановочных тестов мы можем проверять статистические гипотезы, не рассчитывая на то, что распределение будет нормальное, Стьюдента, F или хи-квадрат.

Вы могли заметить, насколько хорошо в предыдущих разделах результаты классических тестов соответствовали результатам перестановочных тестов. Данные в этих задачах были «правильными», и степень согласия между методами – это показатель того, насколько хорошо в такой ситуации работают классические методы.

Перестановочные тесты блистательно показывают себя в случаях, когда распределение данных значительно отличается от нормального (например, сильно асимметричное), когда присутствуют выбросы, когда размеры выборок малы или когда не существует параметрических тестов нужного типа. Однако если имеющаяся выборка нерепрезентативна, никакой тест, включая перестановочный, не сможет сделать выводы более корректными.

Перестановочные тесты в основном используются для вычисления значений статистической ошибки первого рода (*p-value*). Они могут помочь ответить на вопрос «Существует ли закономерность?». Сложнее использовать такие тесты для вычисления доверительных интервалов и оценки точности измерения. К счастью, это область, в которой превосходно работает бутстреп-анализ.

12.5. Бутстреп-анализ

Бутстреп-анализ (bootstrapping) создает эмпирическое распределение тестовой статистики или набора статистик путем создания многих случайных выборок с возвратом из исходной выборки. Это позволяет вычислять доверительные интервалы и проверять статистические гипотезы без опоры на определенное теоретическое распределение данных.

Проще всего объяснить логику бутстреп-анализа на примере. Допустим, вы хотите рассчитать 95%-ный доверительный интервал для выборочного среднего. У вас есть 10 наблюдений, выборочное среднее равно 40, а выборочное стандартное отклонение равно 5. Если вы готовы допустить нормальное распределение выборочного среднего, то $(1 - \alpha/2)\%$ -ный доверительный интервал может быть вычислен с использованием следующего выражения:

$$\bar{X} - t \frac{s}{\sqrt{n}} < \mu < \bar{X} + t \frac{s}{\sqrt{n}},$$

где t – это верхнее $(1 - \alpha/2)$ критическое значение для распределения t с $n - 1$ степенями свободы. В случае 95%-го доверительного интервала вы получите $40 - 2.262(5/3.163) < \mu < 40 + 2.262(5/3.162)$, или $36.424 < \mu < 43.577$. Вы будете ожидать, что вычисленные таким образом значения доверительного интервала будут окружать реальное среднее значение для генеральной совокупности.

Однако что, если вы не готовы предположить нормальное распределение выборочных средних? Тогда вместо приведенных выше вычислений вы можете использовать бутстреп-анализ:

1. Случайно выбрать 10 наблюдений из выборки с возвратом значений после каждого выбора. Некоторые наблюдения могут быть выбраны больше одного раза, а некоторые – могут остаться вовсе невыбранными.
2. Вычислить среднее для полученной выборки из 10 значений.
3. Повторить шаги 1 и 2 тысячу раз.
4. Отсортировать тысячу выборочных средних по возрастанию.
5. Найти выборочные средние, которые представляют собой 2.5 и 97.5 процентили. В данном случае 25-е число с начала и с конца. Это границы вашего 95%-го доверительного интервала.

Похоже, что в данном случае выборочные средние распределены нормально, поэтому вы немного выиграете от применения бутстреп-анализа. Однако есть много ситуаций, когда такой подход дает пре-

имущество. Что, если вы хотели вычислить доверительные интервалы для выборочной медианы или для разницы между двумя выборочными медианами? Для данного случая нет простой «классической» формулы, так что следует воспользоваться бутстреп-анализом. Если распределение данных неизвестно, если выбросы представляют собой проблему, если размеры выборок малы или если не существует параметрического метода, бустреп-анализ часто является полезным способом вычисления доверительных интервалов и проверки гипотез.

12.6. Бутстреп-анализ при помощи пакета boot

Пакет `boot` предоставляет обширные возможности для бутстреп-анализа и связанных с ним методов создания повторных выборок. Вы можете применить бутстреп-анализ к одной статистике (например, к медиане) или к вектору из статистик (например, к набору регрессионных коэффициентов). Не забудьте скачать и установить пакет `boot` перед его первым использованием:

```
install.packages("boot")
```

Бутстреп-анализ покажется сложным, но после того, как вы рассмотрите примеры, все должно стать понятным.

В общем случае бутстреп-анализ состоит из трех этапов:

1. Напишите функцию, которая вычисляет нужную статистику или нужные статистики. Если имеется одна статистика (например, медиана), функция должна возвращать число. Если есть набор статистик (например, набор регрессионных коэффициентов), функция должна возвращать вектор.
2. Примените функцию `boot()` к вашей функции, чтобы создать бутстреп-повторности вашей статистики или ваших статистик.
3. Используйте функцию `boot.ci()`, чтобы вычислить доверительные интервалы для статистики или статистик, созданных на этапе 2.

Теперь перейдем к частностям.

Основная функция для бустреп-анализа – это `boot()`. Формат ее применения таков:

```
bootobject <- boot(data=, statistic=, R=, ...)
```

Все параметры функции описаны в табл. 12.3.

Таблица 12.3. Параметры функции `boot()`

Параметр	Описание
<code>data</code>	Вектор, матрица или таблица данных
<code>statistic</code>	Функция, которая создает k статистик, которые будут подвергнуты бутстреп-анализу ($k=1$, если есть только одна статистика)
<code>R</code>	Число бутстреп-выборок
<code>...</code>	Дополнительные параметры функции, которая создает нужную статистику или нужные статистики

Функция `boot()` запускает статистическую функцию `R` раз. Каждый раз она генерирует набор целых чисел, выбранных случайно с возвратом из диапазона `1:nrow(data)`. Эти числа используются для создания новой выборки. Статистики вычисляются для новой выборки, а результаты записываются в `bootobject`. Структура этого объекта описана в табл. 12.4.

Таблица 12.4. Элементы объекта, создаваемого функцией `boot()`

Элемент	Описание
<code>t0</code>	Наблюдаемые значения k статистик для исходных данных
<code>t</code>	Матрица $R \times k$, где каждый ряд – это одна из бутстреп-реализаций k статистик

Эти элементы можно вызвать как `bootobject$t0` и `bootobject$t`.

После создания новых выборок бутстреп-анализом можно изучить результаты при помощи функций `print()` и `plot()`. Если результаты выглядят разумно, можно вычислить доверительные интервалы для статистик(и) при помощи функции `boot.ci()`. Формат применения этой функции таков:

```
boot.ci(bootobject, conf=, type= )
```

Параметры описаны в табл. 12.5.

Таблица 12.5. Параметры функции `boot.ci()`

Параметр	Описание
<code>bootobject</code>	Объект, создаваемый функцией <code>boot()</code>
<code>conf</code>	Требуемый доверительный интервал (по умолчанию <code>conf=0.95</code>)
<code>type</code>	Тип вычисляемого доверительного интервала. Возможны следующие значения: "norm", "basic", "stud", "perc", "bca" и "all" (по умолчанию <code>type="all"</code>)

Параметр `type` задает способ вычисления границ доверительного интервала. Метод процентиелей (`perc`) был разобран в примере с выборочным средним. Метод `bca` рассчитывает интервал, который делает простые поправки на смещение. Я считаю, что `bca` – самый удачный метод в большинстве ситуаций. Обсуждение этих методов дано в публикации Mooney & Duval (1993).

В оставшихся разделах мы рассмотрим бутстреп-анализ для одной статистики и для вектора статистик.

12.6.1. Бутстреп-анализ для одной статистики

Набор данных `mtcars` содержит информацию о 32 автомобилях, опубликованную в журнале *Motor Trend* за 1974 год. Предположим, вы используете множественную регрессию для предсказания расхода топлива по весу машины и по рабочему объему цилиндров двигателя. В дополнение к стандартным регрессионным статистикам вы хотели бы получить 95%-ный доверительный интервал для коэффициента детерминации (доли дисперсии зависимой переменной, которая объясняется независимыми переменными). Доверительный интервал можно получить при помощи непараметрического бутстреп-анализа.

Первая задача – это написание функции для вычисления коэффициента детерминации:

```
rsq <- function(formula, data, indices) {  
  d <- data[indices,]  
  fit <- lm(formula, data=d)  
  return(summary(fit)$r.square)  
}
```

Выражение `d <- data[indices,]` нужно, чтобы функция `boot()` могла создавать выборки.

Затем вы можете получить большое число повторных бутстреп-выборок (скажем, 1000) при помощи следующих команд:

```
library(boot)  
set.seed(1234)  
results <- boot(data=mtcars, statistic=rsq,  
R=1000, formula=mpg~wt+disp)
```

Результаты можно вывести на экран так:

```
> print(results)  
ORDINARY NONPARAMETRIC BOOTSTRAP  
Call:
```

```
boot(data = mtcars, statistic = rsq, R = 1000, formula = mpg ~
    wt + disp)
Bootstrap Statistics :
      original     bias   std. error
t1*  0.7809306  0.01333670  0.05068926
```

и графически изобразить при помощи команды `plot(results)`. Полученная диаграмма представлена на рис. 12.2.

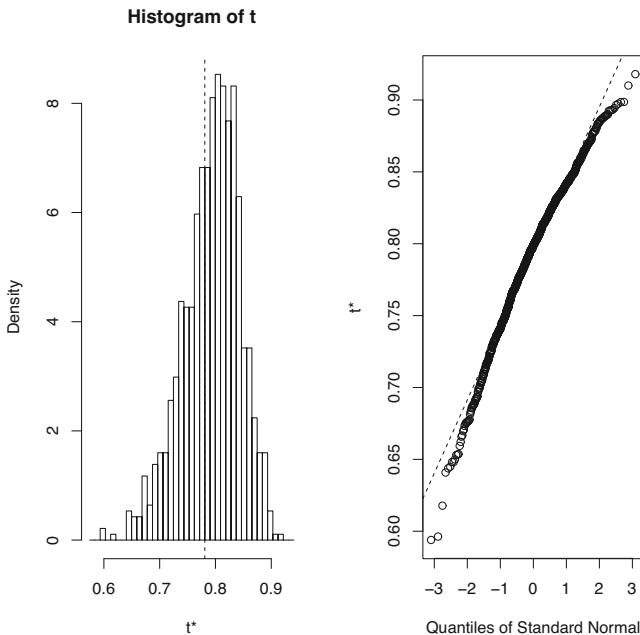


Рис. 12.2. Распределение значений коэффициентов детерминации, полученных при бутстреп-анализе

Из рис. 12.2 видно, что распределение значений коэффициентов детерминации, полученных при бутстреп-анализе, отлично от нормального. Можно вычислить 95%-ный доверительный интервал для коэффициента детерминации:

```
> boot.ci(results, type=c("perc", "bca"))
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates
CALL :
boot.ci(boot.out = results, type = c("perc", "bca"))
Intervals :
Level      Percentile          BCa

```

```
95%  ( 0.6838,  0.8833 )  ( 0.6344,  0.8549 )
Calculations and Intervals on Original Scale
Some BCa intervals may be unstable
```

Из этого примера видно, что разные способы вычисления доверительного интервала дают разные результаты. В данном случае два использованных метода дают умеренно различающиеся результаты.

В этом разделе мы вычислили границы доверительного интервала для одной статистики. В следующем разделе мы оценим доверительные интервалы для нескольких статистик одновременно.

12.6.2. Бутстреп-анализ для нескольких статистик

В предыдущем примере бутстреп-анализ был использован для оценки доверительного интервала для одной статистики (коэффициента детерминации). Продолжая этот пример, получим 95%-ные доверительные интервалы для вектора из статистик. А именно рассчитаем доверительные интервалы для трех коэффициентов регрессионной модели (свободный член, вес автомобиля и объем цилиндров).

Во-первых, создадим функцию, которая возвращает вектор регрессионных коэффициентов:

```
bs <- function(formula, data, indices) {
  d <- data[indices,]
  fit <- lm(formula, data=d)
  return(coef(fit))
}
```

Затем используем эту функцию для получения 1000 бутстреп-выборок:

```
library(boot)
set.seed(1234)
results <- boot(data=mtcars, statistic=bs,
                 R=1000, formula=mpg~wt+disp)
print(results)
ORDINARY NONPARAMETRIC BOOTSTRAP
Call:
boot(data = mtcars, statistic = bs, R = 1000, formula = mpg ~
      wt + disp)
Bootstrap Statistics :
    original     bias   std. error
t1*  34.9606  0.137873    2.48576
t2* -3.3508 -0.053904    1.17043
t3* -0.0177 -0.000121    0.00879
```

При бутстреп-анализе нескольких статистик нужно добавлять параметр индекса в функции `plot()` и `boot.ci()`, чтобы указать, какую колонку `bootobject$t` требуется проанализировать. В этом примере 1 соответствует свободному члену, 2 – весу машины, а 3 – объему цилиндров. Чтобы графически отобразить результаты для веса машины, используйте команду

```
plot(results, index=2)
```

Полученная диаграмма приведена на рис. 12.3.

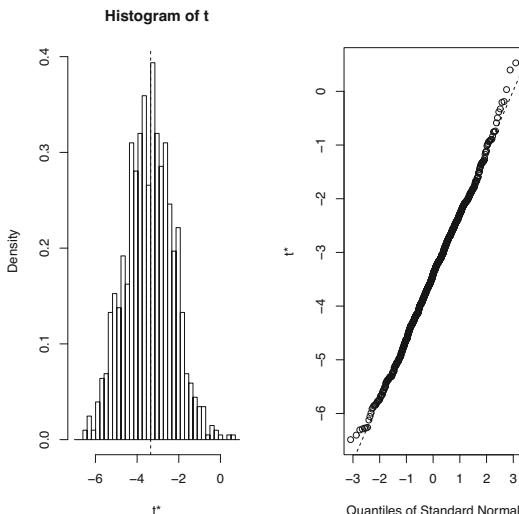


Рис. 12.3. Распределение регрессионных коэффициентов для веса автомобиля, полученных при бутстреп-анализе

Чтобы получить 95%-ные доверительные интервалы регрессионных коэффициентов для веса машины и объема двигателя, используйте следующие команды:

```
> boot.ci(results, type="bca", index=2)
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = results, type = "bca", index = 2)

Intervals :
Level      BCa
95%   (-5.66, -1.19 )
```

```
Calculations and Intervals on Original Scale

> boot.ci(results, type="bca", index=3)
BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
Based on 1000 bootstrap replicates

CALL :
boot.ci(boot.out = results, type = "bca", index = 3)

Intervals :
Level      BCa
95%   (-0.0331,  0.0010 )
Calculations and Intervals on Original Scale
```

Замечание. В предыдущем примере мы составляли повторные выборки из всей исходной выборки. Если мы допустим, что независимые переменные имеют фиксированные значения (обычно для спланированных экспериментов), лучше составлять повторные выборки только для остатков. См. публикацию Mooney & Duval (1993, стр. 16–17), где дано простое объяснение и алгоритм.

Прежде чем мы закончим с бутстреп-анализом, стоит ответить на два часто возникающих вопроса:

- Насколько велика должна быть исходная выборка?
- Сколько повторностей нужно?

На эти вопросы не существует простого ответа. Некоторые говорят, что размер исходной выборки в 20–30 наблюдений достаточен для получения хороших результатов, если выборка репрезентативна. Случайный выбор объектов из генеральной совокупности – наиболее надежный способ получения репрезентативной выборки. Что касается второго вопроса, я нахожу, что 1000 повторностей более чем достаточно в большинстве случаев. Компьютерное время дешево, и при необходимости вы всегда можете увеличить число повторностей.

Существует много полезных источников информации по перестановочным тестам и бутстреп-анализу. Прекрасный выбор для начала – статья Yu (2003), находящаяся в открытом доступе в интернете. В публикации Good (2006) представлен обширный общий обзор методов повторных выборок, включая программные коды R. Хорошее доступное введение в бутстреп-анализ написано Mooney & Duval (1993). Наиболее полный источник информации по бутстреп-анализу – это работа Efron & Tibshirani (1998). Наконец, существует множество замечательных онлайн-ресурсов, включая Simon (1997), Canty (2002), Shah (2005) и Fox (2002).

12.7. Резюме

В этой главе мы познакомились с набором требующих объемных вычислений методов, которые основаны на рандомизации и повторных выборках и позволяют проверять гипотезы и вычислять доверительные интервалы без опоры на известные теоретические распределения. Эти методы особенно ценные, когда ваши данные происходят из неизвестного распределения, есть заметные выбросы, размеры выборок малы, также когда нет параметрических методов для проверки интересующей вас гипотезы.

Методы, описанные в этой главе, особенно привлекательны, поскольку открывают широкие возможности для ответа на вопросы, когда нельзя четко доказать стандартные предположения о свойствах данных или когда у вас нет ни малейшей идеи о том, как решить проблему. Тем не менее перестановочные тесты и бутстреп-анализ – не панацея. Они не могут превратить плохие данные в хорошие. Если ваши исходные выборки нерепрезентативны, то эти методы не помогут.

В следующей главе мы рассмотрим модели для переменных, которые происходят из известных, но не обязательно нормальных распределений.



Часть IV.

ПРОДВИНУТЫЕ МЕТОДЫ

В этой, последней части мы рассмотрим продвинутые методы статистического и графического анализа, чтобы окончательно снабдить вас всеми инструментами анализа данных. Глава 13 расширяет представления о регрессионных методах, почерпнутые из восьмой главы, применением параметрических методов к данным, распределение которых отлично от нормального. Глава начинается описанием обобщенной линейной модели, а затем акцент смещается на случаи, когда мы стараемся предсказать зависимую переменную, которая является либо категориальной (логистическая регрессия), либо счетной (пуасоновская регрессия).

Иметь дело с большим числом переменных может быть затруднительно из-за сложностей, связанных с многомерными данными. В главе 14 описаны два распространенных метода исследования и упрощения многомерных данных. Анализ главных компонент можно использовать для преобразования большого числа скоррелированных переменных в меньший набор комбинированных признаков. Факторный анализ – это совокупность методов обнаружения скрытой структуры, лежащей в основе данного набора переменных. В главе 14 даны пошаговые инструкции для выполнения этих методов анализа.

Обычно исследователи имеют дело с неполными наборами данных. В главе 15 рассмотрены современные подходы к распространенной проблеме пропущенных данных. В R реализован ряд изящных методов для анализа наборов данных, в которых есть пропущенные по разным причинам значения. Здесь описаны несколько лучших подходов,



вместе с рекомендациями, какие методы лучше использовать, а каких стоит избегать.

Глава 16 завершает обсуждение графической обработки данных описанием наиболее сложных и полезных подходов к визуализации данных в R. Сюда входят графическое представление сложных данных при помощи пакета `lattice` и описание нового завоевывающего все большую популярность пакета `ggplot2`. Глава завершается обзором пакетов, которые позволяют взаимодействовать с диаграммами в реальном времени.

После прочтения части IV вы будете обладать достаточными умениями для того, чтобы справиться со всем разнообразием сложных задач анализа данных. К ним относятся моделирование зависимых переменных с отличным от нормального распределением, работа с большим числом скоррелированных переменных и обработка неупорядоченных неполных данных. К тому же вы овладеете полезными, новыми и нестандартными способами визуализации сложных данных.



ГЛАВА 13.

Обобщенные линейные модели

В этой главе:

- Задаем обобщенную линейную модель.
- Предсказываем значения категориальных зависимых переменных.
- Моделируем счетные данные.

В главах 8 (регрессия) и 9 (дисперсионный анализ) мы исследовали линейные модели, которые можно использовать для предсказания значений зависимых переменных с нормальным распределением по значениям набора непрерывных и/или категориальных независимых переменных. Однако часто у нас нет никаких оснований предполагать, что зависимые переменные нормально распределены (и даже непрерывны). Например:

- Зависимая переменная может быть категориальной. Бинарные переменные (например, да/нет, сдал/провалился, живой/мертвый) и политомические переменные (например, плохой/хороший/превосходный, республиканец/демократ/независимый) точно не характеризуются нормальным распределением.
- Зависимая переменная может быть счетной (например, число дорожно-транспортных происшествий за неделю, число порций спиртного в день). Такие переменные имеют ограниченное число значений и никогда не бывают отрицательными. Кроме того, их среднее и дисперсия часто связаны (это не выполняется для переменных с нормальным распределением).

Обобщенные линейные модели (generalized linear models) расширяют применимость линейных моделей, делая возможным анализ зависимых переменных, имеющих отличное от нормального распределение.

Эту главу мы начнем с краткого обзора обобщенных линейных моделей и используемой для их построения функции `glm()`. Затем мы сосредоточимся на двух распространенных частных случаях – логистической регрессии (зависимая переменная категориальная) и пуссоновской регрессии (зависимая переменная счетная).

Для того чтобы сделать обсуждение более интересным, мы применим обобщенные линейные модели к двум исследовательским задачам, которые непросто решить при помощи стандартных линейных моделей:

- Какие личные, демографические и психологические характеристики позволяют предсказать супружескую неверность? В этом случае зависимая переменная будет бинарной (был роман на стороне или нет).
- Какое воздействие прием лекарств оказывает на число припадков, произошедших за 8 недель? В данном случае зависимая переменная – счетная (число припадков).

Мы используем логистическую регрессию для ответа на первый вопрос и пуссоновскую регрессию для второй задачи. По ходу изложения мы обсудим дополнительные приложения этих методов.

13.1. Обобщенные линейные модели и функция `glm()`

К обобщенным линейным моделям относится множество распространенных методов анализа данных. В этом разделе мы кратко рассмотрим некоторые теоретические аспекты, лежащие в основе этого метода. Если хотите, можете спокойно пропустить данный раздел и вернуться к нему впоследствии.

Допустим, вы хотите смоделировать связь между зависимой переменной Y и набором из p независимых переменных $X_1 \dots X_p$. В стандартной линейной модели вы предполагали, что Y имеет нормальное распределение и тип связи описывается таким уравнением:

$$\mu_Y = \beta_0 + \sum_{j=1}^p \beta_j X_j .$$

Это значит, что условное среднее зависит от переменной, представляющей собой линейную комбинацию значений независимых переменных. Параметры, определяющие ожидаемое изменение переменной Y на единицу изменения X_j , обозначены как β_j , а β_0 – это ожидаемое значение переменной Y , когда все независимые переменные равны нулю. Вы можете предсказать среднее значение распределения значений Y для наблюдений с заданным набором значений X , сообщая определенные веса переменным X и суммируя их.

Обратите внимание, что мы не делали никаких предположений относительно распределения значений независимых переменных X_j . Они, в отличие от Y , не обязательно должны иметь нормальное распределение. На самом деле они часто бывают категориальными (например, при дисперсионном анализе). Кроме того, допускаются нелинейные комбинации независимых переменных. Нередко в уравнение входят такие независимые переменные как X^2 или $X_1 \times X_2$. Что здесь важно, так это линейность комбинаций параметров ($\beta_0, \beta_1, \dots, \beta_p$).

При создании обобщенных линейных моделей мы подбираем модели в виде

$$g(\mu_Y) = \beta_0 + \sum_{j=1}^p \beta_j X_j,$$

где $g(\mu_Y)$ – это функция условного среднего (называемая связующей функцией). Кроме того, вы отказываетесь от предположения о нормальном распределении значений Y . Вместо этого вы подразумеваете, что Y подчиняется распределению из семейства экспоненциальных распределений. Вы задаете связующую функцию и вероятностное распределение, а параметры оцениваются при помощи итеративного алгоритма максимального правдоподобия.

13.1.1. Функция `glm()`

Обобщенные линейные модели обычно подгоняются в R при помощи функции `glm()` (хотя доступны и другие функции). Формат применения этой функции схож с таковым для функции `lm()`, но имеет дополнительные параметры. Основной формат функции таков:

```
glm(formula, family=family(link=function), data=)
```

Типы распределений вероятностей (`family`) и соответствующие связующие функции (`function`) по умолчанию приведены в табл. 13.1.

Таблица 13.1. Параметры функции `glm()`

Семейство распределений	Связующая функция по умолчанию
<code>binomial</code>	<code>(link = "logit")</code>
<code>gaussian</code>	<code>(link = "identity")</code>
<code>gamma</code>	<code>(link = "inverse")</code>
<code>inverse.gaussian</code>	<code>(link = "1/mu^2")</code>
<code>poisson</code>	<code>(link = "log")</code>
<code>quasi</code>	<code>(link = "identity", variance = "constant")</code>
<code>quasibinomial</code>	<code>(link = "logit")</code>
<code>quasipoisson</code>	<code>(link = "log")</code>

Функция `glm()` позволяет подгонять разные распространенные модели, включая логистическую и пуассоновскую регрессии, а также модели для анализа выживания (здесь не рассматривается). Это можно продемонстрировать на двух первых моделях следующим образом. Предположим, что у вас есть одна зависимая переменная (y) и три независимых (x_1, x_2, x_3), которые находятся в одной таблице данных (`mydata`).

Логистическая регрессия подходит для дихотомических зависимых переменных (0, 1). Модель предполагает, что Y имеет биномимальное распределение и что можно подогнать линейную модель следующего вида:

$$\log_e \left(\frac{\pi}{1-\pi} \right) = \beta_0 + \sum_{j=1}^p \beta_j X_j,$$

где $\pi = \mu_y$ – это условное среднее Y (то есть вероятность того, что $Y = 1$ при данном наборе значений X), $(\pi/1 - \pi)$ – это отношение шансов того, что $Y = 1$, а $\log(\pi/1 - \pi)$ – это логарифм отношения шансов, или *логит*. В данном случае $\log(\pi/1 - \pi)$ – это связующая функция, вероятностное распределение биномиальное, а логистическая регрессионная модель может быть подогнана при помощи команды

```
glm(Y~X1+X2+X3, family=binomial(link="logit"), data=mydata)
```

Логистическая регрессия разобрана более подробно в разделе 13.2.

Пуассоновская регрессия применяется в случае счетной зависимой переменной. Пуассоновская регрессионная модель подразумевает, что Y имеет пуассоновское распределение и что можно подогнать линейную модель вида

$$\log_e(\lambda) = \beta_0 + \sum_{j=1}^p \beta_j X_j,$$

где λ – это среднее (и дисперсия Y). В данном случае связующая функция имеет вид $\log(\lambda)$, вероятностная функция пуассоновская, а пуассоновская регрессионная модель может быть подобрана при помощи команды

```
glm(Y~X1+X2+X3, family=poisson(link="log"), data=mydata)
```

Более подробно пуассоновская регрессия описана в разделе 13.3.

Стойт отметить, что стандартная линейная модель – это тоже частный случай обобщенной линейной модели. Если связующая функция будет $g(\mu_y) = \mu_y$ (тождественная функция), а распределение будет нормальным (гауссовым), то функция

```
glm(Y~X1+X2+X3, family=gaussian(link="identity"), data=mydata)
```

даст тот же результат, что и функция

```
lm(Y~X1+X2+X3, data=mydata)
```

Подводя итоги, можно отметить, что обобщенные линейные модели расширяют применимость стандартных линейных моделей, если подбирать *функцию* для условного среднего зависимой переменной (а не само условное среднее) и предполагать, что распределение зависимой переменной относится к семейству *экспоненциальных* распределений (а не ограничено нормальным распределением). Оценка параметров происходит методом максимального правдоподобия, а не методом наименьших квадратов.

13.1.2. Вспомогательные функции

Многие функции, которые вы использовали совместно с функцией `lm()`, когда анализировали стандартные линейные модели, имеют свои аналоги для функции `glm()`. Некоторые часто используемые функции приведены в табл. 13.2.

Таблица 13.2. Функции, которые используются вместе с функцией `glm()`

Функция	Описание
<code>summary()</code>	Выводит на экран подробные результаты для подогнанной модели
<code>coefficients()</code> , <code>coef()</code>	Выводит параметры модели (свободный член и регрессионные коэффициенты)

Функция	Описание
confint ()	Доверительные интервалы для параметров модели (по умолчанию 95%)
residuals ()	Выводит остатки подогнанной модели
anova ()	Создает таблицу дисперсионного анализа для сравнения двух моделей
plot ()	Создает диагностические диаграммы для оценки соответствия модели данным
predict ()	Использует подогнанную модель для предсказания значений зависимой переменной для нового набора данных

Мы познакомимся с примерами применения этих функций позднее. В следующем разделе мы кратко рассмотрим способы оценки адекватности модели.

13.1.3. Соответствие модели данным и регрессионная диагностика

Проверка адекватности модели также важна для обобщенных линейных моделей, как и для стандартных (МНК) линейных моделей. К сожалению, среди статистиков существует меньше единодушия по поводу способов проверки адекватности обобщенных линейных моделей. В целом вы можете использовать описанные в главе 8 приемы, обращая внимание на следующие предостережения.

При оценке адекватности модели вы, как правило, хотели бы изобразить зависимость предсказанных значений зависимой переменной в исходных единицах измерения от остатков в единицах стандартного отклонения. Например, обычную диагностическую диаграмму можно создать при помощи следующих команд:

```
plot(predict(model, type="response"),
      residuals(model, type="deviance")),
```

где `model` – это объект, созданный функцией `glm()`.

Вычисляемые в R стандартизованные по Стьюденту остатки, значения показателя влияния наблюдения и D-статистика Кука будут лишь примерными. Кроме того, не существует общего согласия по поводу пороговых значений для обнаружения проблемных наблюдений. Следует принимать решения по поводу значений этих статистик методом сравнения их друг с другом. Один подход заключается в том,

чтобы графически изобразить значения статистики для всех наблюдений и поискать необычно большие значения. Например, для создания трех диагностических диаграмм можно использовать следующие команды:

```
plot(hatvalues(model))
plot(rstudent(model))
plot(cooks.distance(model))
```

В качестве альтернативы можно использовать такие команды:

```
library(car)
influencePlot(model),
```

чтобы создать одну общую диаграмму. В ней горизонтальная ось – это напряженность (leverage), вертикальная ось – это стьюдентизированные остатки, а размер отображаемых символов пропорционален расстоянию Кука.

Диагностические диаграммы обычно наиболее полезны, когда зависимая переменная имеет много значений. Когда зависимая переменная принимает ограниченное число значений (например, логистическая регрессия), польза от диагностических диаграмм не так велика.

Для получения более подробной информации о регрессионной диагностике для обобщенных линейных моделей обратитесь к публикациям Fox (2008) и Faraway (2006). В оставшейся части этой главы мы подробно рассмотрим две наиболее популярные разновидности обобщенных линейных моделей: логистическую и пуассоновскую регрессию.

13.2. Логистическая регрессия

Логистическая регрессия полезна для предсказания значений бинарной зависимой переменной по набору непрерывных и/или категориальных зависимых переменных. Для демонстрации этого подхода мы исследуем данные по супружеской неверности, содержащиеся в наборе данных Affairs, поставляемом вместе с пакетом AER. Не забудьте скачать и установить этот пакет (при помощи команды `install.packages("AER")`) перед первым использованием.

Данные по супружеским изменениям, известные под названием «Измены Фейра»¹, основаны на перекрестном анализе, проведенном

1 Фейр (Fair) – ученый, впервые проанализировавший этот набор данных, см. ниже. – Прим. пер.

журналом «Психология сегодня» (Psychology Today) в 1969 году и описанном в публикациях Greene (2003) и Fair (1978). Он содержит 9 переменных, значения которых зарегистрированы для 601 человека. Вот эти переменные: как часто респондент вступал во внебрачные сексуальные связи за последний год, пол и возраст испытуемого, сколько лет он находится в браке, есть ли дети, степень религиозности (по пятибалльной шкале от 1 – антирелигиозность до 5 – очень религиозен), образование, род занятий (по семибальной классификации Холлингсгеда (Hollingshead) с обратной нумерацией) и числовая самооценка счастья в браке (от 1 – очень несчастлив до 5 – очень счастлив).

Давайте посмотрим на описательные статистики:

```
> data(Affairs, package="AER")
> summary(Affairs)

affairs      gender       age   yearsmarried   children
Min. : 0.000  female:315  Min. :17.50  Min. : 0.125 no :171
1st Qu.: 0.000  male : 286  1st Qu.:27.00  1st Qu.: 4.000 yes:430
Median : 0.000
Mean   : 1.456
3rd Qu.: 0.000
Max.   :12.000

religiousness   education   occupation   rating
Min.   :1.000  Min.   : 9.00  Min.   :1.000  Min.   :1.000
1st Qu.:2.000  1st Qu.:14.00  1st Qu.:3.000  1st Qu.:3.000
Median :3.000  Median :16.00  Median :5.000  Median :4.000
Mean   :3.116  Mean   :16.17  Mean   :4.195  Mean   :3.932
3rd Qu.:4.000  3rd Qu.:18.00  3rd Qu.:6.000  3rd Qu.:5.000
Max.   :5.000  Max.   :20.00  Max.   :7.000  Max.   :5.000

> table(Affairs$affairs)
  0   1   2   3   7  12 
451 34  17  19  42  38
```

Из приведенных значений видно, что 52% респондентов – это женщины, у 72% были дети, а медиана возраста составила 32 года. Если говорить о зависимой переменной, то 75% респондентов сказали, что они не имели связей на стороне за последний год (451 человек из 601). Наибольшее число супружеских измен за год составило 12 (у 6% испытуемых).

Хотя учитывалось абсолютное число опрометчивых поступков, в данном случае нам интересна бинарная переменная (была измена или нет). Можно преобразовать переменную `affairs` в дихотомическую переменную `ynaffair` при помощи следующего программного кода:

```
> Affairs$ynaffair[Affairs$affairs > 0] <- 1
> Affairs$ynaffair[Affairs$affairs == 0] <- 0
> Affairs$ynaffair <- factor(Affairs$ynaffair,
```

```

levels=c(0,1),
labels=c("No","Yes"))

> table(Affairs$ynaffair)
No Yes
451 150

Эту дихотомическую переменную теперь можно использовать как зависимую переменную в логистической регрессионной модели:

> fit.full <- glm(ynaffair ~ gender + age + yearsmarried + children +
+                   religiousness + education + occupation + rating,
+                   data=Affairs, family=binomial())
> summary(fit.full)
Call:
glm(formula = ynaffair ~ gender + age + yearsmarried + children +
    religiousness + education + occupation + rating, family = binomial(),
    data = Affairs)
Deviance Residuals:
    Min      1Q   Median      3Q      Max
-1.571 -0.750 -0.569 -0.254  2.519
Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) 1.3773    0.8878   1.55  0.12081
gendermale   0.2803    0.2391   1.17  0.24108
age          -0.0443   0.0182  -2.43  0.01530 *
yearsmarried 0.0948    0.0322   2.94  0.00326 **
childrenyes  0.3977   0.2915   1.36  0.17251
religiousness -0.3247   0.0898  -3.62  0.00030 ***
education     0.0211   0.0505   0.42  0.67685
occupation    0.0309   0.0718   0.43  0.66663
rating        -0.4685   0.0909  -5.15  2.6e-07 ***
---
Signif. codes:  0 '****' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
(Dispersion parameter for binomial family taken to be 1)
Null deviance: 675.38 on 600 degrees of freedom
Residual deviance: 609.51 on 592 degrees of freedom
AIC: 627.5
Number of Fisher Scoring iterations: 4

```

По значениям статистической ошибки первого рода (p) для коэффициентов регрессии (последний столбец) можно видеть, что пол, наличие детей, образование и род занятий не вносят значимого вклада в модель (мы не можем отвергнуть гипотезу о том, что эти параметры равны 0). Давайте подберем вторую модель без них и проверим, соответствует ли эта сокращенная модель данным настолько же хорошо:

```

> fit.reduced <- glm(ynaffair ~ age + yearsmarried + religiousness +
+                      rating, data=Affairs, family=binomial())
> summary(fit.reduced)
Call:
glm(formula = ynaffair ~ age + yearsmarried + religiousness + rating,
     family = binomial(), data = Affairs)

```

```

Deviance Residuals:
    Min      1Q  Median      3Q     Max
-1.628  -0.755  -0.570  -0.262   2.400

Coefficients:
            Estimate Std. Error z value Pr(>|z|)
(Intercept) 1.9308    0.6103   3.16  0.00156 **
age         -0.0353    0.0174  -2.03  0.04213 *
yearsmarried 0.1006    0.0292   3.44  0.00057 ***
religiousness -0.3290   0.0895  -3.68  0.00023 ***
rating       -0.4614    0.0888  -5.19  2.1e-07 ***
---
Signif. codes:  0 '****' 0.001 '***' 0.01 '**' 0.05 '*' 0.1 '.' 1
(Dispersion parameter for binomial family taken to be 1)
Null deviance: 675.38 on 600 degrees of freedom
Residual deviance: 615.36 on 596 degrees of freedom
AIC: 625.4
Number of Fisher Scoring iterations: 4

```

Каждый регрессионный коэффициент в сокращенной модели статистически значим ($p < 0.05$). Поскольку это две вложенные модели (`fit.reduced` – это часть `fit.full`), то для их сравнения можно использовать функцию `anova()`. Для обобщенных линейных моделей вам понадобится хи-квадрат-версия этого теста.

```

> anova(fit.reduced, fit.full, test="Chisq")
Analysis of Deviance Table
Model 1: ynaffair ~ age + yearsmarried + religiousness + rating
Model 2: ynaffair ~ gender + age + yearsmarried + children +
          religiousness + education + occupation + rating
Resid. Df Resid. Dev Df Deviance P(>|Chi|)
1        596      615
2        592      610  4      5.85      0.21

```

Незначимая величина критерия хи-квадрат ($p = 0.21$) свидетельствует о том, что сокращенная модель с четырьмя независимыми переменными соответствует данным так же хорошо, как и модель с восемью независимыми переменными. Это укрепляет вашу уверенность в том, что пол, наличие детей, образование и род занятий не оказывают существенного влияния на качество предсказания зависимой переменной. Так что вы можете интерпретировать данные на основании более простой модели.

13.2.1. Интерпретация параметров модели

Давайте посмотрим на регрессионные коэффициенты:

```
> coef(fit.reduced)
```

(Intercept)	age	yearsmarried	religiousness	rating
1.931	-0.035	0.101	-0.329	-0.461

При логистической регрессии в качестве моделируемых значений зависимой переменной используется логарифм отношения шансов (odds) того, что $Y = 1$ ². Регрессионный коэффициент – это изменение логарифма отношения шансов данной зависимой переменной на единицу изменения независимой переменной при постоянных значениях всех остальных независимых переменных.

Поскольку логарифм отношения шансов сложно интерпретировать, можно потенцировать коэффициенты, чтобы представить их в виде отношения шансов:

> exp(coef(fit.reduced))	age	yearsmarried	religiousness	rating
6.895	0.965	1.106	0.720	0.630

Теперь вы можете видеть, что отношение шансов внебрачных связей повышается в 1.106 раза при увеличении продолжительности брака на один год (при постоянных значениях возраста, религиозности и оценки счастья в браке). Отношение шансов супружеской измены нужно умножать на 0.965 при увеличении возраста испытуемых на год. Отношение шансов внебрачных связей повышается с увеличением продолжительности брака и снижается при увеличении возраста, религиозности и самооценки счастья в браке. Поскольку независимые переменные не могут быть равны нулю, то свободный член в данном случае лишен смысла.

При необходимости можно вычислить доверительные интервалы для коэффициентов при помощи функции `confint()`. Например, команда `exp(confint(fit.reduced))` позволит рассчитать доверительные интервалы для всех коэффициентов модели для каждого коэффициента в единицах отношения шансов.

Наконец, изменение независимой переменной на одну единицу может не быть интересным само по себе. Для бинарной логистической регрессии изменение отношения шансов того, что зависимая переменная примет более высокие значения при изменении независимой переменной на n единиц, равна $\exp(\beta_j)^n$. Если увеличение продолжительности брака на один год повышает отношение шансов супружеской измены в 1.106 раза, то увеличение продолжительности брака на 10 лет повысит отношение шансов неверности в $1.106^{10} = 2.7$ раза при постоянных значениях остальных переменных.

2 То есть $\ln(P(Y=1)/P(Y=0))$, где P – это вероятность. – Прим. пер.

13.2.2. Оценка влияния независимых переменных на вероятность исхода

Многим из нас легче думать в терминах вероятностей, а не отношений шансов. Для того чтобы понять, как изменение значений независимых переменных влияет на вероятность успешного исхода, можно использовать функцию `predict()`. Первый шаг – это создание искусственного набора данных, который содержит значения интересующих вас независимых переменных. Затем вы можете использовать этот набор данных вместе с функцией для предсказания вероятностей исхода.

Давайте применим данный подход для оценки влияния самооценки качества брака на вероятность супружеских измен. Сначала мы создаем искусственный набор данных, где возраст, продолжительность брака и религиозность представлены их средними по выборке значениями, а самооценка качества брака колеблется от 1 до 5.

```
>testdata <- data.frame(rating=c(1, 2, 3, 4, 5), age=mean(Affairs$age),
                           yearsmarried=mean(Affairs$yearsmarried),
                           religiousness=mean(Affairs$religiousness))

> testdata
   rating   age  yearsmarried religiousness
1       1 32.5          8.18        3.12
2       2 32.5          8.18        3.12
3       3 32.5          8.18        3.12
4       4 32.5          8.18        3.12
5       5 32.5          8.18        3.12
```

Затем мы используем этот набор данных и уравнение регрессии для вычисления вероятностей:

```
>testdata$prob <- predict(fit.reduced, newdata=testdata, type="response")
testdata
   rating   age  yearsmarried religiousness     prob
1       1 32.5          8.18        3.12  0.530
2       2 32.5          8.18        3.12  0.416
3       3 32.5          8.18        3.12  0.310
4       4 32.5          8.18        3.12  0.220
5       5 32.5          8.18        3.12  0.151
```

Из этих результатов ясно, что вероятность внебрачной связи уменьшается от 0.53, если человек считает свой брак очень несчастливым (1), до 0.15, когда брак считается очень счастливым (5) – при постоянных значениях возраста. Теперь посмотрим на эффект возраста:

```
>testdata <- data.frame(rating=mean(Affairs$rating),
                           age=seq(17, 57, 10),
                           yearsmarried=mean(Affairs$yearsmarried),
```

```

religiousness=mean(Affairs$religiousness))
> testdata
  rating age yearsmarried religiousness
1   3.93   17          8.18       3.12
2   3.93   27          8.18       3.12
3   3.93   37          8.18       3.12
4   3.93   47          8.18       3.12
5   3.93   57          8.18       3.12

> testdata$prob <- predict(fit.reduced, newdata=testdata, type="response")
> testdata
  rating age yearsmarried religiousness      prob
1   3.93   17          8.18       3.12  0.335
2   3.93   27          8.18       3.12  0.262
3   3.93   37          8.18       3.12  0.199
4   3.93   47          8.18       3.12  0.149
5   3.93   57          8.18       3.12  0.109

```

Здесь видно, что с увеличением возраста от 17 до 57 вероятность внебрачных связей снижается от 0.34 до 0.11 при постоянных значениях остальных переменных. При помощи этого подхода можно исследовать влияние каждой независимой переменной на вероятность исхода.

13.2.3. Избыточная дисперсия

Ожидаемая дисперсия для данных, происходящих из биномиально-го распределения, рассчитывается как $\sigma^2 = np(1 - \pi)$, где n – число наблюдений, а π – это вероятность принадлежности к группе $Y = 1$ ³. Избыточная дисперсия (overdispersion) отмечается, когда наблюдаемая дисперсия зависимой переменной превышает ожидаемую. Избыточная дисперсия может привести к искажению оценки среднеквадратичных ошибок и некорректным тестам значимости.

В случае избыточной дисперсии можно по-прежнему проводить логистическую регрессию при помощи функции `glm()`, однако в этом случае нужно использовать квазибиномиальное распределение, а не биномиальное.

Один из способов выявить избыточную дисперсию – это сравнить остаточную девиату (*residual deviance*)⁴ с остаточными степенями свободы (*residual df*) для вашей биномиальной модели. Если отношение

3 Вероятность «благоприятного исхода». – Прим. пер.

4 Общепринятый перевод термина *deviance* (обозначающего величину, производную от оценки максимального правдоподобия) на русский язык отсутствует. – Прим. пер.

$$\phi = \frac{\text{Остаточная девиатата}}{\text{Остаточные степени свободы}}$$

заметно больше единицы – это признак избыточной дисперсии. Применив этот критерий к примеру с супружескими изменениями, мы получим

$$\phi = \frac{\text{Остаточная девиатата}}{\text{Остаточные степени свободы}} = \frac{615.36}{596} = 1.03.$$

Полученное значение близко к единице, что свидетельствует об отсутствии избыточной дисперсии.

Существует также тест на избыточную дисперсию. Чтобы провести его, нужно подогнать модель дважды. В первый раз – с параметром `family="binomial"`, а во второй – с параметром `family="quasibinomial"`. Если объект, возвращенный функцией `glm()` в первом случае, называется `fit`, а во втором – `fit.od`, то команда

```
pchisq(summary(fit.od)$dispersion * fit$df.residual,
       fit$df.residual, lower = F)
```

позволяет вычислить значение статистической ошибки первого рода (p) для проверки нулевой гипотезы H_0 : $\phi = 1$ в противоположность альтернативной гипотезе H_1 : $\phi \neq 1$. Если значение p малое (скажем, меньше 0.05), вы отвергаете нулевую гипотезу.

Применяя эти соображения к данным о супружеских изменениях, мы получим

```
> fit <- glm(ynaffair ~ age + yearsmarried + religiousness +
              rating, family = binomial(), data = Affairs)
> fit.od <- glm(ynaffair ~ age + yearsmarried + religiousness +
                  rating, family = quasibinomial(), data = Affairs)
> pchisq(summary(fit.od)$dispersion * fit$df.residual,
           fit$df.residual, lower = F)
[1] 0.34
```

Полученное значение p (0.34) совершенно точно незначимо ($p > 0.05$), утверждая нас во мнении, что избыточная дисперсия в данном случае не представляет проблемы. Мы вернемся к вопросу о избыточной дисперсии, когда будем обсуждать пуассоновскую регрессию.

13.2.4. Дополнительные методы

В R реализовано несколько дополнительных методов и разновидностей логистической регрессии:

- *устойчивая (robust) логистическая регрессия.* Функцию `glmRob()` из пакета `robust` можно использовать для подгонки устойчивых обобщенных регрессионных моделей, в том числе и устойчивой логистической регрессии. Этот метод может быть полезен при подборе логистических регрессионных моделей для данных с выбросами и влиятельными наблюдениями;
- *мультиномиальная (multinomial) логистическая регрессия.* Если зависимая переменная имеет больше двух неупорядоченных значений (например, женат/вдов/разведен), можно рассчитать мультиномиальную логистическую регрессию при помощи функции `mlogit()` из пакета `mlogit`;
- *порядковая (ordinal) логистическая регрессия.* Если зависимая переменная представляет собой упорядоченный фактор (например, платежеспособность по кредиту: слабая/хорошая/отличная), можно использовать порядковую логистическую регрессию (функция `lrm()` из пакета `rms`).

Возможность моделировать категориальную зависимую переменную со многими значениями (и упорядоченными, и неупорядоченными) – это важное расширение применимости метода, но оно имеет свою цену – большую сложность интерпретации. Оценка соответствия модели данным и регрессионная диагностика в этих случаях тоже будут более сложными.

В примере с супружеской неверностью число внебрачных связей было преобразовано в дихотомическую (да/нет) переменную, поскольку мы интересовались наличием измен за текущий год. Если бы мы больше интересовались величиной эффекта – числом внебрачных связей за год, мы бы напрямую анализировали счетные данные. Один из распространенных подходов к анализу счетных данных – это пуассоновская регрессия – следующая тема, к которой мы обратимся.

13.3. Пуассоновская регрессия

Пуассоновская регрессия полезна, когда вы предсказываете счетную зависимую переменную по набору непрерывных и/или категориальных зависимых переменных. Всеобъемлющее и при этом доступное руководство по пуассоновской регрессии приведено в работе Соухе, West и Aiken (2009).

Для демонстрации процесса подгонки пуассоновской регрессионной модели, наряду с некоторыми проблемами, которые могут воз-

никнуть при таком анализе данных, мы используем опубликованные данные о припадках (Breslow, 1993), поставляющиеся с пакетом *robust*. А именно мы рассмотрим влияние приема лекарств от эпилептических припадков на число припадков, которые случились за 8 недель, прошедших со времени начала лечения. Не забудьте установить пакет *robust*, перед тем как продолжать.

Для пациентов, которые страдали от простых или сложных припадков, были получены данные о возрасте и о числе припадков, произошедших за 8 недель до и после начала лечения. Больные были случайным образом распределены на две группы, которые получали плацебо (*placebo*) или настоящее лекарство (*progabide*). Переменная *sumY* (число припадков за 8 недель после начала лечения) – это зависимая переменная. Тип лечения (*Trt*), возраст в годах (*Age*) и число припадков за 8 недель до начала лечения (*Base*) – это независимые переменные. Возраст и число припадков до начала лечения включены в анализ, поскольку они могут оказывать влияние на зависимую переменную. Нам интересно, есть ли свидетельство в пользу того, что применение лекарства снижает число припадков с учетом эффектов этих ковариат.

Сначала посмотрим на описательные статистики для этого набора данных:

```
> data(breslow.dat, package="robust")
> names(breslow.dat)
[1] "ID"      "Y1"      "Y2"      "Y3"      "Y4"      "Base"    "Age"     "Trt"     "Ysum"
[10] "sumY"   "Age10"   "Base4"
> summary(breslow.dat[c(6,7,8,10)])
      Base          Age          Trt          sumY      
Min. : 6.0   Min. :18.0   placebo :28   Min. :  0.0  
1st Qu.:12.0  1st Qu.:23.0  progabide:31  1st Qu.: 11.5 
Median :22.0  Median :28.0                    Median : 16.0 
Mean   :31.2  Mean   :28.3                    Mean   : 33.1 
3rd Qu.:41.0  3rd Qu.:32.0                   3rd Qu.: 36.0 
Max.   :151.0 Max.   :42.0                   Max.   :302.0
```

Обратите внимание на то, что хотя набор данных включает 12 переменных, мы сосредоточили свое внимание на четырех упомянутых выше. Распределение значений числа припадков до и после лечения характеризуется заметной асимметрией. Давайте проанализируем зависимую переменную более подробно. Следующие команды позволяют получить диаграмму, представленную на рис. 13.1.

```
opar <- par(no.readonly=TRUE)
par(mfrow=c(1,2))
attach(breslow.dat)
```

```
hist(sumY, breaks=20, xlab="Число припадков",
      main="Частота припадков")
boxplot(sumY ~ Trt, xlab="Способ лечения", main="Сравнение групп")
par(opar)
```

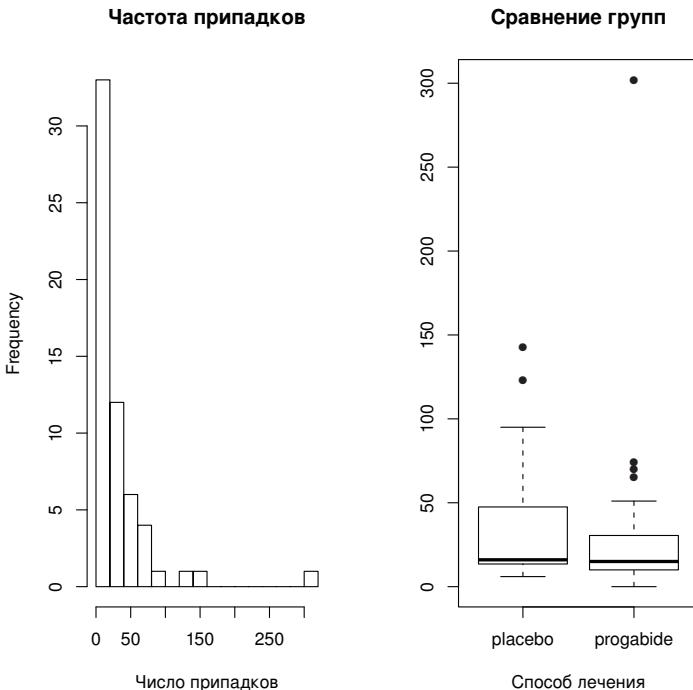


Рис. 13.1. Распределение значений числа припадков после лечения (источник данных: опубликованные данные о припадках (Breslow, 1993))

Хорошо заметно асимметричное распределение значений зависимой переменной, а также возможное наличие выбросов. На первый взгляд, число припадков у людей, принимающих настоящее лекарство, кажется меньшим и имеет меньший разброс данных. Для данных с пуассоновским распределением можно ожидать, что меньшая дисперсия будет сопряжена с меньшим средним значением. Гетерогенность дисперсии⁵ не представляет проблем для пуассоновской регрессии, в отличие от стандартной МНК-регрессии.

Следующий шаг – это подгонка пуассоновской регрессионной модели:

⁵ Межгрупповые различия дисперсии. – Прим. пер.

```
> fit <- glm(sumY ~ Base + Age + Trt, data=breslow.dat, family=poisson())
> summary(fit)
Call:
glm(formula = sumY ~ Base + Age + Trt, family = poisson(), data = breslow.dat)
Deviance Residuals:
    Min      1Q  Median      3Q     Max 
-6.057 -2.043 -0.940   0.793  11.006 
Coefficients:
            Estimate Std. Error z value Pr(>|z|)    
(Intercept) 1.948826  0.135619 14.37 < 2e-16 ***
Base        0.022652  0.000509 44.48 < 2e-16 ***
Age         0.022740  0.004024  5.65  1.6e-08 ***
Trtprogabide -0.152701  0.047805 -3.19   0.0014 ** 
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
Dispersion parameter for poisson family taken to be 1
Null deviance: 2122.73 on 58 degrees of freedom
Residual deviance: 559.44 on 55 degrees of freedom
AIC: 850.7
Number of Fisher Scoring iterations: 5
```

На экран выводятся девиаты, параметры регрессионной модели, среднеквадратичные отклонения и тесты, проверяющие, что параметры не равны нулю. Обратите внимание на то, что влияние каждой независимой переменной значимо на уровне $p < 0.05$.

13.3.1. Интерпретация параметров модели

Коэффициенты модели можно вывести на экран при помощи функции `coef()` или в составе таблицы `Coefficients`, создаваемой функцией `summary()`.

```
> coef(fit)
(Intercept)          Base           Age       Trtprogabide
      1.9488        0.0227        0.0227       -0.1527
```

В пуассоновской регрессии зависимая переменная моделируется как логарифм условного среднего $\log_e(\lambda)$. Регрессионный коэффициент для возраста, равный 0.0227, показывает, что каждый дополнительный год возраста при постоянных значениях числа припадков до лечения и способа лечения сопряжен с увеличением логарифма среднего значения числа припадков на 0.03. Поскольку не бывает нулевого возраста и никто из участников эксперимента не имел нулевого числа припадков до лечения, свободный коэффициент не имеет смысла.

Обычно гораздо проще интерпретировать регрессионные коэффициенты, когда они выражены в исходных единицах зависимой переменной (число припадков, а не логарифм этого числа). Для этого коэффициенты нужно потенцировать:

```
> exp(coef(fit))
(Intercept)      Base          Age  Trtprogabide
    7.020        1.023        1.023        0.858
```

Теперь видно, что увеличение возраста на один год *увеличивает* число припадков в 1.023 раза. Это значит, что старший возраст со-пряжен с большим числом припадков. Более важно, что изменение типа лечения на единицу (то есть переход от плацебо к настоящему лекарству) уменьшает число припадков в 0.86 раза. Можно ожидать снижения числа припадков на 14% у пациентов, принимающих настоящее лекарство, по сравнению с теми, кто принимает плацебо, при постоянных значениях возраста и числа припадков до лечения.

Важно помнить, что, как и экспоненциальные параметры в логистической регрессии, экспоненциальные параметры в пуассоновской регрессии оказывают мультипликативный, а не аддитивный эффект на зависимую переменную. Так же как и для логистической регрессии, нужно проверить модель на наличие избыточной дисперсии.

13.3.2. Избыточная дисперсия

В пуассоновском распределении дисперсия и среднее равны. Избыточная дисперсия при пуассоновской регрессии отмечается, когда наблюдаемая дисперсия зависит от переменной больше, чем ожидается, исходя из свойств распределения. Поскольку избыточная дисперсия часто встречается при работе со счетными данными и оказывает неблагоприятный эффект на интерпретацию результатов, мы потратим некоторое время на обсуждение этой проблемы.

Есть несколько причин, по которым может возникать избыточная дисперсия (Coxe et al., 2009):

- Удаление важной независимой переменной.
- Явление, известное под названием «зависимость от состояния». События считаются независимыми друг от друга. Для числа припадков это будет значить, что для любого пациента вероятность припадка не зависит от вероятности других припадков. Однако это допущение часто не выполняется. Неподожже, что для данного пациента вероятность первого припадка равна вероятности 40-го припадка, если их уже было 39.

- При продолжительных исследованиях избыточная дисперсия может быть вызвана группировкой данных, свойственной повторным измерениям. Мы не будем обсуждать здесь пуассоновские модели для продолжительных исследований.

Если избыточная дисперсия имеет место, а вы не учитываете ее в своей модели, вы получите слишком низкие среднеквадратичные ошибки и доверительные интервалы, а тесты на достоверность будут слишком либеральными (то есть вы найдете закономерности там, где их на самом деле нет).

Как и в случае логистической регрессии, признак избыточной дисперсии – это отношение остаточной девиаты к числу степеней свободы для остатков, заметно превышающее единицу. Для наших данных по припадкам это отношение составляет:

$$\frac{\text{Остаточная девиата}}{\text{Остаточные степени свободы}} = \frac{599.44}{55} = 10.73.$$

ясно, что это значение намного больше единицы.

Пакет `qcc` позволяет провести тест на избыточную дисперсию для пуассоновского распределения (не забудьте скачать и установить этот пакет перед первым использованием). Тест на избыточную дисперсию для данных по припадкам можно провести при помощи следующих команд:

```
> library(qcc)
> qcc.overdispersion.test(breslow.dat$sumY, type="poisson")
Overdispersion test Obs.Var/Theor.Var Statistic p-value
    poisson data           62.9      3646       0
```

Неудивительно, что значение p оказалось меньше 0.05, ясно указывая на наличие избыточной дисперсии.

Вы по-прежнему можете подгонять модель для ваших данных при помощи функции `glm()`, заменив опцию `family="poisson"` на `family="quasipoisson"`. Такая схема действий аналогична предложенной для логической регрессии с избыточной дисперсией данных.

```
> fit.od <- glm(sumY ~ Base + Age + Trt, data=breslow.dat,
                  family=quasipoisson())
> summary(fit.od)

Call:
glm(formula = sumY ~ Base + Age + Trt, family = quasipoisson(),
     data = breslow.dat)

Deviance Residuals:
```

```
Min      1Q Median      3Q    Max
-6.057 -2.043 -0.940  0.793 11.006
```

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)	
(Intercept)	1.94883	0.46509	4.19	0.00010	***
Base	0.02265	0.00175	12.97	< 2e-16	***
Age	0.02274	0.01380	1.65	0.10509	
Trtprogabide	-0.15270	0.16394	-0.93	0.35570	

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for quasipoisson family taken to be 11.8)

```
Null deviance: 2122.73 on 58 degrees of freedom
Residual deviance: 559.44 on 55 degrees of freedom
AIC: NA
```

Number of Fisher Scoring iterations: 5

Обратите внимание на то, что оценки параметров при квазипуассоновском подходе идентичны полученным при пуассоновской регрессии, но стандартные ошибки намного больше. В данном случае большие стандартные ошибки привели к тому, что значения p для переменных Trt и Age превысили 0.05. Когда вы учитываете избыточную дисперсию, у вас остается недостаточно оснований, чтобы утверждать, что настоящее лекарство снижает число припадков эффективнее, чем плацебо, при постоянных значениях числа припадков до лечения и возраста.

Пожалуйста, учтите, что этот пример приведен здесь только для демонстрации метода. Представленные результаты не должны быть использованы для каких-либо утверждений относительно эффективности лекарств в реальном мире. Я не доктор – по крайней мере, не доктор медицинских наук – и даже не изображал его в телепередачах.

Мы закончим это исследование пуассоновской регрессии обсуждением нескольких ее важных разновидностей и дополнительных методов.

13.3.3. Дополнительные методы

В R реализовано несколько полезных дополнений к основному варианту пуассоновской регрессионной модели, включая модели, которые позволяют изменять периоды времени, вводящие поправку на избыточное число нулей и устойчивые модели, которые полезны, когда данные содержат выбросы и влиятельные наблюдения. Я опишу каждую из них отдельно.

Пуассоновская регрессия с варьирующими временными периодами

Наше описание пуассоновской регрессии было ограничено зависимыми переменными, которые измеряются в числе событий за фиксированный отрезок времени (например, число припадков за 8 недель, число дорожно-транспортных происшествий за прошедший год, число хороших поступков за день). Длина временного отрезка в этом случае остается постоянной. Однако можно подгонять также регрессионные модели, которые допускают различия во временных отрезках, за которые происходят события. В таком случае зависимая переменная представлена частотой событий за единицу времени.

Для таких частот вы должны добавить переменную (которая будет называться, например, `time`), в которой будет указана длина временного отрезка, за который произошли события, отмеченные за данное наблюдение. Тогда ваша модель изменяется от такой:

$$\log_e(\lambda) = \beta_0 + \sum_{j=1}^p \beta_j X_j$$

к такой:

$$\log_e\left(\frac{\lambda}{time}\right) = \beta_0 + \sum_{j=1}^p \beta_j X_j$$

или, что то же самое, к такой:

$$\log_e(\lambda) = \log_e(time) + \beta_0 + \sum_{j=1}^p \beta_j X_j.$$

Для подгонки этой новой модели используйте опцию `offset` функции `glm()`. Предположим для примера, что период времени, в течение которого пациенты получали лекарство от эпилептических припадков, варьировал от 14 до 60 суток. Вы можете использовать относительную частоту припадков как зависимую переменную (при условии, что для каждого пациента вы фиксировали продолжительность периода лечения в сутках) и подогнать модель:

```
fit <- glm(sumY ~ Base + Age + Trt, data=breslow.dat,
            offset= log(time), family=poisson()),
```

где `sumY` – это число припадков, которые случились за время лечения. В этом случае вы предполагаете, что относительная частота припадков оставалась постоянной (то есть два припадка за четыре дня – это то же самое, что 10 припадков за 20 дней).

Пуассоновская регрессия для данных с избыточным числом нулей

Бывает так, что частота наблюдений с нулевым числом событий в наборе данных выше, чем предсказывает пуассоновская модель. Такое может случиться, если в одной из подгрупп генеральной совокупности регистрируемые события никогда не происходят. Например, в наборе данных по изменам, который обсуждался в разделе о логистической регрессии, исходная зависимая переменная (*affairs*) была выражена в числе внебрачных связей за прошедший год. Весьма вероятно, что существует группа верных супружеских пар, которые никогда не совершают измену, вне зависимости от продолжительности времени наблюдений. Такие наблюдения называются *структурными нулями* (в основном людьми свободных нравов из числа испытуемых).

В таких случаях можно анализировать данные при помощи подхода, который называется *пуассоновская регрессия для данных с избыточным числом нулей* (zero-inflated Poisson regression). В рамках этого подхода одновременно подгоняются две модели – одна предсказывает, кто будет, а кто не будет изменяться, а другая предсказывает, сколько раз вступят в связи на стороне тех участников, для которых зарегистрирована хотя бы одна измена. Это можно интерпретировать как модель, которая объединяет логистическую регрессию (для предсказания структурных нулей) и пуассоновскую регрессию (для предсказания числа наблюдений, которые не являются структурными нулями). Такой подход можно реализовать при помощи функции `zeroinfl()` из пакета `pscl`.

Устойчивая пуассоновская регрессия

Наконец, для подбора устойчивой обобщенной линейной модели, включая устойчивую пуассоновскую регрессию, можно использовать функцию `glmRob()` из пакета `robust`. Как уже было сказано выше, это может быть полезным при наличии выбросов и влиятельных наблюдений.

Дополнительные сведения

Обобщенные линейные модели – это сложный метод с замысловатым математическим аппаратом, однако существует много прекрасных источников для более детального знакомства с ним. Хорошее короткое введение в тему можно найти в работе Duntzman & Ho (2006). Классическое и непростое описание обобщенных линейных моделей представлено McCullagh & Nelder (1989). Исчерпывающие и доступные изложения материала сделаны Dobson & Barnett (2008) и Fox (2008). Прекрасное введение в тему в контексте работы в R можно найти в работах Faraway (2006) и Fox (2002).

13.4. Резюме

В этой главе мы использовали обобщенные линейные модели для того, чтобы расширить круг доступных вам методов анализа данных. В частности, теперь вы можете анализировать зависимые переменные, распределение которых отличается от нормального, включая категориальные и счетные переменные. После короткого описания общих принципов мы сосредоточились на логистической регрессии (для анализа дихотомических зависимых переменных) и на пуассоновской регрессии (для анализа зависимых переменных, которые выражены в «штуках» или представляют собой относительные частоты).

Мы также обсудили важные вопросы, связанные с избыточной дисперсией, включая то, как ее обнаружить и учесть.

Все статистические методы, которые мы обсуждали до сих пор, применялись непосредственно к наблюдаемым и регистрируемым переменным. В следующей главе мы рассмотрим статистические модели, которые имеют дело с латентными переменными – ненаблюдамыми теоретическими переменными, которые, как считается, лежат в основе наблюдаемых переменных и учитывают их поведение. В частности, вы узнаете, как можно использовать методы факторного анализа для формулировки и проверки гипотез об этих ненаблюдаемых переменных.



ГЛАВА 14.

Главные компоненты и факторный анализ

В этой главе:

- Анализ главных компонент.
- Факторный анализ.
- Другие модели для скрытых переменных.

Один из наиболее затруднительных аспектов многомерных данных – это очевидная сложность информации. Если в вашем наборе данных есть 100 переменных, как вы можете осмыслить взаимосвязи между ними? Даже в случае 20 переменных существует 190 попарных корреляций, которые нужно проанализировать, чтобы понять, как отдельные переменные связаны друг с другом. Есть две связанные между собой, но разные методологии для исследования и упрощения сложных многомерных данных – это анализ главных компонент и факторный анализ.

Анализ главных компонент (principal components analysis, PCA) – это способ снижения размерности данных, который преобразует большое число скоррелированных переменных в гораздо меньший набор нескоррелированных переменных, называемых главными компонентами. Например, можно использовать PCA, чтобы преобразовать 30 скоррелированных (и возможно, избыточных) характеристик состояния окружающей среды в пять нескоррелированных комбинированных переменных, которые сохраняют максимально возможную часть информации, которая содержалась в исходном наборе переменных.

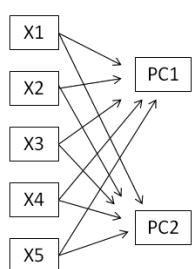
В отличие от PCA, под названием разведочный факторный анализ (exploratory factor analysis, EFA) объединены методы, которые обна-

режима скрытую структуру в имеющемся наборе переменных. Этот анализ позволяет найти меньший набор лежащих в основе, или латентных структурных компонентов которые могут объяснить взаимосвязи между наблюдаемыми, или явными, переменными. Например, набор данных Hartman74.csv содержит корреляции между результатами 24 психологических тестов, которые были предложены 145 школьникам 7 и 8 классов. Если вы примените EFA к этим данным, то в результате окажется, что имеющиеся 276 попарных корреляций между результатами разных тестов могут быть объяснены успехами детей в четырех лежащих в основе результатов тестов областях (развиваемость речи, скорость обработки информации, способность к логическим умозаключениям и память).

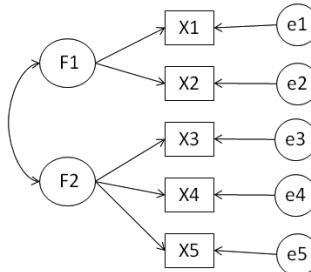
Различия между моделями PCA и EFA показаны на рис. 14.1¹. Главные компоненты (PC1 и PC2) – это линейные комбинации наблюдаемых переменных (с X1 по X5). Используемые для формирования линейных комбинаций переменных веса (коэффициенты) выбраны так, чтобы максимизировать долю дисперсии, объясненной каждой главной компонентой, при этом главные компоненты не коррелированы между собой.

В противоположность этому подразумевается, что факторы (F1 и F2) лежат в основе, или «обуславливают», наблюдаемые переменные, а не являются их линейными комбинациями. Ошибки (с e1 по e5) – это неучтенная факторами дисперсия наблюдаемых переменных². Кружки означают, что факторы и ошибки не наблюдаются непосредственно, а выводятся из корреляций между переменными. В данном примере изогнутая стрелка между факторами означает, что они не скоррелированы. Скоррелированные факторы обычны для EFA моделей, но не обязательны.

-
- 1 Ошибка в интерпретации автора заключается в том, что и в случае PCA, и в случае EFA вновь построенные оси являются линейными комбинациями исходных наблюдаемых переменных. Эвфемизм «лежат в основе, или обуславливают» означает в данном контексте описание линейного метода то же самое, что и «линейная комбинация». Ошибкой является также противопоставление этих двух методов, поскольку EFA фактически представляет из себя «надстройку» над PCA. Дело в том, что следует различать задачу построения ортогональной системы координат, решаемую PCA, и задачу поиска простой структуры, которая выполняется при помощи EFA – *Прим. пер.*
 - 2 Поскольку число выделяемых факторов всегда меньше числа исходных переменных, то ошибки в случае PCA – это дисперсия, заключенная в «отброшенных» компонентах. Каким бы способом ни строилась новая ортогональная система координат, в том случае, когда число ее осей станет равным или превысит число исходных показателей в выборке, никаких ошибок просто не останется, поскольку данные полностью «поместятся» в построенном пространстве. – *Прим. пер.*



(а) Модель анализа главных компонент



(б) Модель факторного анализа

Рис. 14.1. Сравнение моделей анализа главных компонент и факторного анализа. На схемах показаны наблюдаемые переменные (с X_1 по X_5), главные компоненты (PC_1 и PC_2) и ошибки (с e_1 по e_5)

Для получения устойчивых результатов при помощи описанных в этой главе методов нужны большие выборки³. Решить, какой размер выборки достаточен, несколько затруднительно. До последнего времени использовались эмпирические правила вроде этого: «при факторном анализе число объектов должно превышать число переменных в 5–10 раз». Недавние исследования показали, что необходимый размер выборки зависит от числа факторов, числа сопряженных с каждым фактором переменных и того, насколько хорошо набор факторов объясняет дисперсию переменных (Bandalos & Boehm-Kaufman, 2009). Я возьму на себя смелость сказать, что если у вас есть несколько сотен наблюдений, то вы, скорее всего, в безопасности.

Мы начнем с обзора функций, при помощи которых в R можно провести PCA или EFA, и сделаем краткий обзор необходимых этапов анализа. Затем мы тщательно разберем два примера применения PCA, за которыми последует расширенный пример применения EFA. В конце главы будет приведен краткий обзор других пакетов в R, которые можно использовать для подбора моделей с латентными переменными. Обсуждение коснется пакетов для конфирматорного (confirmatory) факторного анализа, моделирования структурных уравнений, анализа соответствий (correspondence analysis) и анализа скрытых (или латентных) классов.

3 Способы бутстреп-оценки, так хорошо описанные в данной книге, полностью применимы для оценки устойчивости полученного решения и вычисления интервальных оценок показателей полученной модели. – Прим. пер.

14.1. Выполнение анализа главных компонент и факторного анализа в R

В базовой версии R PCA и EFA проводятся при помощи функций `princomp()` и `factanal()` соответственно. В этой главе мы сосредоточимся на описании функций, реализованных в пакете `psych`. У этих функций есть намного больше полезных опций, чем у их аналогов из базовой версии программы. Кроме того, результаты этих функций представлены в формате, более привычном для социологов и более сходном со способом представления таких результатов в других статистических программах, таких как SAS и SPSS.

Наиболее актуальные в данном случае функции пакета `psych` представлены в табл. 14.1. Не забудьте установить данный пакет перед началом работы с примерами из этой главы.

Таблица 14.1. Полезные функции для проведения факторного анализа при помощи пакета `psych`

Функции	Описание
<code>principal()</code>	Анализ главных компонент с возможностью поворота осей*
<code>fa()</code>	Факторный анализ методом главных осей, минимальных остатков, взвешенных наименьших квадратов или наибольшего правдоподобия
<code>fa.parallel()</code>	График собственных значений (scree plot) с параллельным анализом
<code>factor.plot()</code>	Графическое изображение результатов факторного анализа или анализа главных компонент
<code>fa.diagram()</code>	Графическое изображения матриц нагрузок факторного анализа или анализа главных компонент
<code>scree()</code>	График собственных значений для факторного анализа и анализа главных компонент

* Это, по сути, процедура EFA, проводимого с первоначальным построением осей-факторов по методу PCA. Поскольку данный способ построения системы ортогональных осей, описывающих экспериментальные данные, наиболее обоснован математически, он вынесен в отдельную процедуру. – Прим. пер.

Факторный анализ (и в меньшей степени PCA) часто сбивает с толку неискушенных пользователей. Это происходит потому, что под этими названиями скрывается большое разнообразие методов, и что-

бы достигнуть результата при помощи каждого из них, нужно преодолеть несколько этапов (и принять несколько решений). Обычно нужно предпринимать следующие шаги:

1. *Подготовить данные.* Результаты и PCA, и EFA выводятся из корреляций между наблюдаемыми переменными. Пользователи могут использовать в качестве аргументов функций `principal()` и `fa()` либо исходную таблицу данных, либо корреляционную матрицу. Если используются исходные данные, то корреляционная матрица будет вычислена автоматически. Не забудьте перед обработкой проверить данные на наличие пропущенных значений.
2. *Выбрать факторную модель.* Нужно решить, что лучше подходит для ваших исследовательских задач – PCA (снижение размерности данных) или EFA (обнаружение скрытой структуры). Если вы выбрали EFA, нужно также выбрать определенный метод (например, наибольшего правдоподобия).
3. *Решить, сколько компонент/факторов выделять.*
4. *Выделить компоненты/факторы.*
5. *Повернуть компоненты/факторы.*
6. *Интерпретировать результаты.*
7. *Вычислить значения компонент или факторов.*

В оставшейся части этой главы мы внимательно рассмотрим каждый из этих шагов, начав с PCA. В конце главы вы найдете блок-схему с возможными этапами PCA/EFA (рис. 14.7). Вы лучше поймете эту блок-схему, когда ознакомитесь с предшествующим ей материалом.

14.2. Главные компоненты

Задача PCA – заменить большое число скореллированных переменных меньшим числом нескореллированных переменных, которые сохраняют как можно больше информации, содержащейся в исходных переменных. Производные переменные, называемые главными компонентами, – это линейные комбинации наблюдаемых переменных. В частности, первая главная компонента

$$PC_1 = a_1X_1 + a_2X_2 + \dots + a_kX_k$$

это взвешенная комбинация k наблюдаемых переменных, которая учитывает наибольшую дисперсию (долю изменчивости) исходного набора переменных. Вторая главная компонента – это линейная

комбинация, которая учитывает наибольшую дисперсию исходного набора переменных при условии, что она *ортогональна* первой главной компоненте (то есть не коррелирует с ней). Каждая следующая компонента учитывает наибольшую возможную дисперсию, оставаясь нескоррелированной с остальными компонентами. Теоретически вы можете выделить столько главных компонент, сколько у вас переменных. Однако из практических соображений вы надеетесь, что вы сможете охарактеризовать полный набор переменных посредством гораздо меньшего набора компонент. Давайте рассмотрим простой пример.

Набор данных USJudgeRatings содержит рейтинг судей разных штатов главного суда первой инстанции в США, составленный адвокатами. Таблица данных содержит данные по 43 судьям и 12 числовым переменным. Названия переменных расшифрованы в табл. 14.2.

Таблица 14.2. Переменные, содержащиеся в наборе данных USJudgeRatings

Переменная	Описание	Переменная	Описание
CONT	Число контактов адвоката с судьей	PREP	Подготовка к судебному процессу
INTG	Честность судьи	FAMI	Знание законов
DMNR	Поведение	ORAL	Верные устные решения
DILG	Старание	WRIT	Верные письменные решения
CFMG	Качество организации процесса вынесения судебных решений	PHYS	Физические возможности
DECI	Быстрота решений	RTEN	Достоин запоминания

Возникает практический вопрос: можно ли описать оценки по этим 11 параметрам (от INTG до RTEN) при помощи меньшего набора комбинированных переменных? А если да, то сколько таких переменных понадобится и какими они будут? Поскольку наша цель – это упрощение данных, то мы решим эту задачу при помощи РСА. Даные представлены в виде исходных баллов, пропущенные значения отсутствуют. Так что теперь нужно решить, сколько главных компонент вам понадобится.

14.2.1. Выбор необходимого числа компонент

Существует несколько критериев для определения числа компонент в РСА:

- имеющийся опыт и теоретические соображения;
- объяснение заданной доли дисперсии исходных переменных (например, 80%);
- изучение собственных значений матрицы корреляций между всеми переменными.

Наиболее распространенный подход основан на собственных значениях (eigenvalues). Каждая компонента связана с собственным значением корреляционной матрицы. Первой главной компоненте (principal component, PC) соответствует наибольшее собственное значение, второй компоненте – второе по величине собственное значение и т. д. Согласно критерию Кайзера-Харриса (Kaiser-Harris), следует использовать компоненты, у которых собственные значения превышают единицу. Компоненты с собственными значениями меньше единицы объясняют меньше дисперсии, чем содержится в одной исходной переменной. В тесте собственных значений (Cattell Scree test) собственные значения изображаются в соответствии с номерами компонент. На подобных графиках (иногда называемых графиками собственных значений, или каменистой осыпи, scree plot) обычно виден изгиб, и используются компоненты до этого изгиба. Наконец, вы можете заниматься моделированием, выделяя компоненты из случайных матриц данных той же размерности, что и исходная матрица. Если собственное значение, основанное на реальных данных, выше, чем соответствующие усредненные собственные значения для набора случайных матриц данных, тогда такая компонента используется. Подобный подход называется *параллельным анализом* (более подробную информацию можно найти в публикации Hayton, Allen & Scarpello, 2004).

Все три способа анализа собственных значений можно применить одновременно при помощи функции `fa.parallel()`. Для 11 шкал оценки (без переменной `cont`) необходимый программный код будет следующим:

```
library(psych)
fa.parallel(USJudgeRatings[,-1], fa="PC", n.trials=100,
            show.legend=FALSE, main="Диаграмма каменистой осыпи
            с параллельным анализом")
```

В результате получается график, представленный на рис. 14.2. Это графическое изображение результатов теста собственных значений (обозначены отрезками и крестиками), усредненных собственных значений, полученных из 100 случайных таблиц данных (пунктир) и критерия превышения собственными значениями единицы (горизонтальная линия $y = 1$).

Диаграмма каменистой осыпи с параллельным анализом

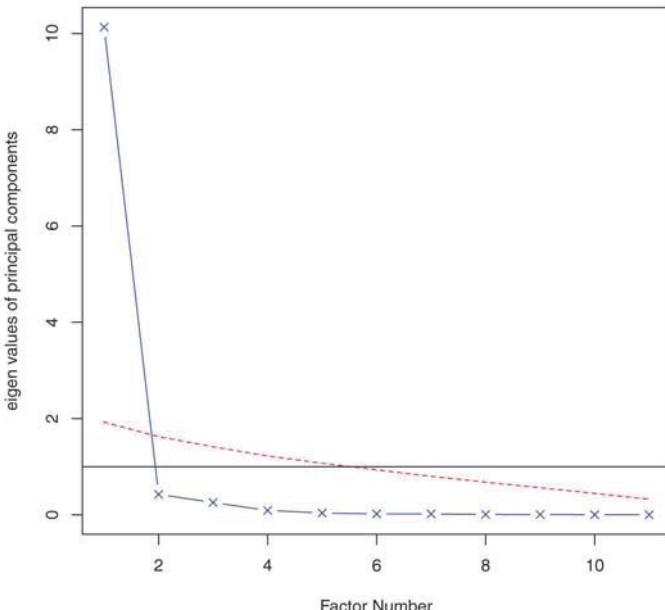


Рис. 14.2. Оценка числа главных компонент, достаточного для характеристики данных из примера про американских судей. График собственных значений (линия с крестиками), критерия превышения собственными значениями единицы (горизонтальная линия) и параллельный анализ со 100 симуляциями (пунктир) свидетельствуют о том, что достаточно использовать одну компоненту

Все три критерия свидетельствуют о том, что для характеристики этого набора данных достаточно одной компоненты. Следующий шаг – выделить эту компоненту при помощи функции `principal()`.

14.2.2. Выделение главных компонент

Как было указано раньше, анализ главных компонент для исходной матрицы данных или корреляционной матрицы можно провести при помощи функции `principal()`. Формат ее применения таков:

```
principal(r, nfactors=, rotate=, scores=),
```

где

- `r` – корреляционная матрица, или исходная таблица данных;
- `nfactors` – определяет число главных компонент, которые нужно выделить (по умолчанию одна);
- `rotate` – указывает, какой тип вращения нужно применить (по умолчанию варимакс, см. раздел 14.2.3);
- `scores` – определяет, нужно ли рассчитывать значения главных компонент (по умолчанию – нет).

Для выделения первой главной компоненты можно использовать приведенный ниже программный код.

Программный код 14.1. Анализ главных компонент для данных по рейтингу американских судей

```
> library(psych)
> pc <- principal(USJudgeRatings[,-1], nfactors=1)
> pc
Principal Components Analysis
Call: principal(r = USJudgeRatings[, -1], nfactors=1)
Standardized loadings based upon correlation matrix
      PC1    h2   u2
INTG  0.92  0.84  0.157
DMNR  0.91  0.83  0.166
DILG  0.97  0.94  0.061
CFMG  0.96  0.93  0.072
DECI  0.96  0.92  0.076
PREP  0.98  0.97  0.030
FAMI  0.98  0.95  0.047
ORAL  1.00  0.99  0.009
WRIT  0.99  0.98  0.020
PHYS  0.89  0.80  0.201
RTEN  0.99  0.97  0.028
                  PC1
SS loadings    10.13
Proportion Var  0.92
[... остальные сведения удалены ...]
```

Здесь анализ главных компонент проводится для исходной таблицы данных без переменной `CONT` с указанием, что должна быть выделена одна главная компонента без вращения (вращение компонент

обсуждается в разделе 14.2.3). Поскольку РСА проводится на основе корреляционной матрицы, перед извлечением компонент исходные данные автоматически преобразуются в такую матрицу.

Столбец с названием PC1 содержит *нагрузки* компоненты – корреляции наблюдаемых переменных с главной компонентой (или главными компонентами). Если бы вы извлекли больше одной главной компоненты, то там были бы еще столбцы PC2, PC3 и т. д. Нагрузки компонент используются для интерпретации их значений. Видно, что каждая переменная сильно коррелирует с первой компонентой (PC1). Поэтому ее можно использовать как общий показатель положения судей в рейтинге.

В столбце h2 указаны общности компонент (component communalities) – доля учтенной компонентой дисперсии каждой переменной. В колонке u2 приведены величины уникальности компонент (component uniquenesses) – доля дисперсии переменной, которая остается неучтеною компонентой (1 – h2). Например, 80% дисперсии рейтинга физических возможностей судей (*phys*) объясняется первой компонентой, а 20% – нет. Это переменная, которая хуже всего характеризуется этой единственной компонентой.

В строке с названием ss loadings приведены связанные с компонентами собственные значения. Собственные значения – это стандартизованная дисперсия, сопряженная с данной компонентой (в данном случае ее значение для первой компоненты составляет 10.13). Наконец, в строке Proportion Var указана доля дисперсии, которая учтена каждой компонентой. В данном случае видно, что первая главная компонента учитывает 92% дисперсии 11 переменных.

Давайте рассмотрим второй пример, такой, где извлекается более одной главной компоненты. Набор данных Harman23.cor содержит данные о восьми промерах тела у 305 девочек. В данном случае набор данных представляет собой корреляции между переменными, а не исходные данные (см. табл. 14.3).

Вам вновь нужно заменить исходные промеры меньшим числом производных переменных. Число компонент, которые нужно извлечь, можно определить при помощи следующего программного кода. В данном случае вам нужно указать, что в качестве исходных данных используется корреляционная матрица (компонент cov объекта Harman23.cor), и указать объем выборки (n.obs):

```
library(psych)
fa.parallel(Harman23.cor$cov, n.obs=302, fa="pc", ntrials=100,
            show.legend=FALSE, main="Диаграмма каменистой осипи
            ↵ с параллельным анализом")
```

Полученный график приведен на рис. 14.3.

Диаграмма каменистой осьпи с параллельным анализом

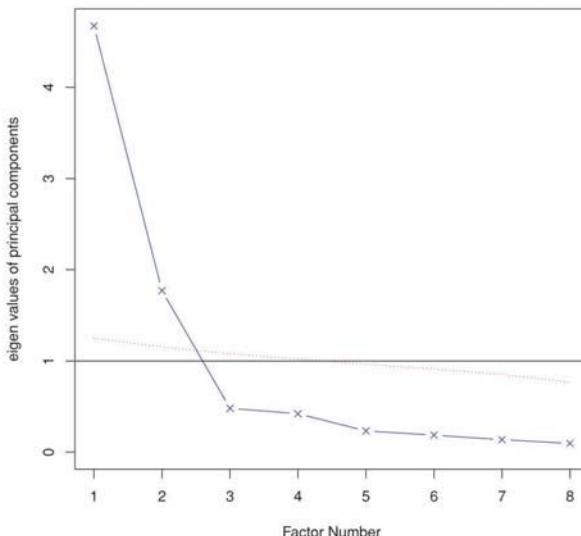


Рис. 14.3. Оценка числа главных компонент, достаточного для характеристики данных из примера с промерами тела. График собственных значений (линия с крестиками), критерия превышения собственными значениями единицы (горизонтальная линия) и параллельный анализ со 100 симуляциями (пунктир) свидетельствуют о том, что достаточно использовать две компоненты

Таблица 14.3. Корреляции между промерами тела у 305 девочек (набор данных Harman23.cor)

	height (рост)	arm span (раз- макс рук)	forearm (пред- плечье)	lower leg (го- лень)	weight (вес)	bitro diameter (диаметр бедер)	chest girth (объем груди)	chest width (ширина грудной клетки)
height	1.00	0.85	0.80	0.86	0.47	0.40	0.30	0.38
arm span	0.85	1.00	0.88	0.83	0.38	0.33	0.28	0.41
forearm	0.80	0.88	1.00	0.80	0.38	0.32	0.24	0.34
lower leg	0.86	0.83	0.8	1.00	0.44	0.33	0.33	0.36
weight	0.47	0.38	0.38	0.44	1.00	0.76	0.73	0.63
bitro diameter	0.40	0.33	0.32	0.33	0.76	1.00	0.58	0.58

	height (рост)	arm span (раз- макс рук)	forearm (пред- плечье)	lower leg (го- лень)	weight (вес)	bitro diameter (диаметр бедер)	chest girth (объем груди)	chest width (ширина грудной клетки)
chest girth	0.30	0.28	0.24	0.33	0.73	0.58	1.00	0.54
chest width	0.38	0.41	0.34	0.36	0.63	0.58	0.54	1.00

Источник: Harman, H. H. (1976) Modern Factor Analysis, Third Edition Revised, University of Chicago Press, Table 2.3.

Из диаграммы видно, что следует использовать две компоненты. Как и в первом примере, критерий Кайзера-Харриса, критерий собственных значений и параллельный анализ дают сходные результаты. Так бывает не всегда, и вам может понадобиться извлечь разное число компонент и выбрать решение, которое кажется наиболее полезным. Следующий программный код позволяет извлечь две первые компоненты из корреляционной матрицы.

Программный код 14.2. Анализ главных компонент для промеров тела

```
> library(psych)
> PC <- principal(Harman23.cor$cov, nfactors=2, rotate="none")
> PC
Principal Components Analysis
Call: principal(r = Harman23.cor$cov, nfactors = 2, rotate = "none")
Standardized loadings based upon correlation matrix
          PC1    PC2    h2    u2
height      0.86 -0.37  0.88  0.123
arm.span    0.84 -0.44  0.90  0.097
forearm     0.81 -0.46  0.87  0.128
lower.leg   0.84 -0.40  0.86  0.139
weight      0.76  0.52  0.85  0.150
bitro.diameter 0.67  0.53  0.74  0.261
chest.girth  0.62  0.58  0.72  0.283
chest.width   0.67  0.42  0.62  0.375
          PC1    PC2
SS loadings 4.67 1.77
Proportion Var 0.58 0.22
Cumulative Var 0.58 0.81
[... остальные сведения удалены ...]
```

Если вы внимательно просмотрите колонки PC1 и PC2 в программном коде 14.2, то заметите, что первая компонента учитывает 58% дисперсии промеров тела, а вторая компонента – 22%. Вместе эти две компоненты объясняют 81% общей дисперсии. Две компоненты учитывают 88% дисперсии значений роста.

Компоненты и факторы интерпретируются при помощи их нагрузок. Первая компонента положительно коррелирует со всеми физическими параметрами и служит общим показателем размера тела. Вторая компонента противопоставляет первые четыре переменные (`height`, `arm.span`, `forearm` и `lower.leg`) и вторые четыре (`weight`, `bitro.diameter`, `chest.girth` и `chest.width`). Таким образом, получается, что вторая компонента соответствует отношению длины к объему. С такой компонентой непросто работать. Если извлечены две компоненты и более, можно вращать их, чтобы эти компоненты было легче интерпретировать. Это тема, к которой мы сейчас обратимся.

14.2.3. Вращение главных компонент

Вращение – это набор математических приемов трансформации матрицы нагрузок компонент в другую, более легко интерпретируемую. Это делается путем как можно большей «очистки» компонент. Способы вращения различаются по тому, остаются ли получившиеся компоненты нескоррелированными (*ортогональное вращение*) или же допускается их корреляция (*наклонное вращение*). Они также различаются по способу «очистки». Наиболее распространенный тип ортогонального вращения – это *варимакс* (`varimax`), при котором делается попытка очистить столбцы матрицы нагрузок так, чтобы каждая компонента была определена ограниченным набором переменных (то есть в каждом столбце будет лишь несколько больших нагрузок и много очень малых). Применив такое вращение к данным промеров тела, вы получите результаты, которые представлены в следующем программном коде. Пример наклонного вращения будет дан в разделе 14.4.

Программный код 14.3. Анализ главных компонент с вращением варимакс

```
> rc <- principal(Harman23.cor$cov, nfactors=2, rotate="varimax")
> rc
Principal Components Analysis
Call: principal(r = Harman23.cor$cov, nfactors = 2, rotate = "varimax")
Standardized loadings based upon correlation matrix
            RC1   RC2    h2     u2
height       0.90  0.25  0.88  0.123
arm.span     0.93  0.19  0.90  0.097
forearm      0.92  0.16  0.87  0.128
lower.leg    0.90  0.22  0.86  0.139
weight       0.26  0.88  0.85  0.150
bitro.diameter 0.19  0.84  0.74  0.261
chest.girth   0.11  0.84  0.72  0.283
```

```
chest.width    0.26 0.75 0.62 0.375
               RC1 RC2
SS loadings   3.52 2.92
Proportion Var 0.44 0.37
Cumulative Var 0.44 0.81
[... остальные сведения удалены ...]
```

Названия колонок изменились с PC на RC (rotated components – повернутые компоненты), чтобы обозначить вращение. Рассматривая нагрузки в столбце RC1, можно заметить, что первая компонента в основном определяется первыми четырьмя переменными (размерные признаки). Значения нагрузки в столбце RC2 указывают на то, что вторая компонента в основном определяется переменными с 5 по 8 (характеристики объема). Обратите внимание на то, что эти две компоненты по-прежнему нескоррелированы, и вместе они так же хорошо объясняют изменчивость переменных, как и до вращения. Это происходит потому, что общности переменных не изменились. Общая доля объясненной дисперсии также осталась прежней (81%). Однако доля дисперсии, объясненной каждой переменной, изменилась (с 58% до 44% для первой компоненты и с 22% до 37% для второй компоненты). Такое уравнивание дисперсии, объясненной разными компонентами, обычно, и формально их теперь следует называть компонентами, а не главными компонентами (поскольку изменилась доля объясненной дисперсии, характеризующая отдельные компоненты).

Наша конечная цель – заменить больший набор скоррелированных переменных меньшим набором производных переменных. Для того чтобы это сделать, нужно рассчитать значения компонент для каждого наблюдения.

14.2.4. Вычисление значений главных компонент

В примере с рейтингом судей мы извлекли единственную главную компоненту из исходных данных, описывающих рейтинг судей по 11 шкалам. Значения производной переменной для каждого участника легко получить при помощи команды `principal()` (см. следующий программный код).

Программный код 14.4. Вычисление значений компонент для исходных данных

```
> library(psych)
> pc <- principal(USJudgeRatings[,-1], nfactors=1, score=TRUE)
```

```
> head(pc$scores)
```

	PC1
AARONSON, L.H.	-0.1857981
ALEXANDER, J.M.	0.7469865
ARMENTANO, A.J.	0.0704772
BERDON, R.I.	1.1358765
BRACKEN, J.J.	-2.1586211
BURNS, E.B.	0.7669406

Значения главных компонент сохраняются как элемент scores в объекте, который создается в результате работы функции principal() с опцией scores=TRUE. При желании теперь можно вычислить корреляцию между числом контактов адвоката и судьи и мнением адвокатов о судье при помощи команды

```
> cor(USJudgeRatings$CONT, PC$score)
      PC1
[1,] -0.008815895
```

Конечно же, не существует никакой связи между степенью знаменитства адвоката с судьей и мнением адвоката!⁴

Ясно, что невозможно вычислить значения главных компонент для каждого наблюдения, если анализ главных компонент основан на корреляционной матрице и исходные данные недоступны. Однако можно рассчитать коэффициенты, которые используются для получения значений главных компонент.

Данные о промерах тела представляли собой корреляции между этими промерами, но у нас не было отдельных промеров для каждой из 305 девочек. Коэффициенты можно получить при помощи следующего программного кода.

Программный код 14.5. Вычисление коэффициентов для главных компонент

```
> library(psych)
> rc <- principal(Harman23.cor$cov, nfactors=2, rotate="varimax")
> round(unclass(rc$weights), 2)
      RC1    RC2
height       0.28 -0.05
arm.span     0.30 -0.08
forearm      0.30 -0.09
lower.leg    0.28 -0.06
weight       -0.06  0.33
```

⁴ Исследуемые объекты (в данном случае это судьи) классифицируют по значениям главных компонент. Такую классификацию изображают в виде диаграммы рассеяния. Подробнее об этом можно прочесть в книге А. Б. Шипунова с соавторами «Наглядная статистика. Используем R!» (М.: ДМК Пресс, 2012). – Прим. пер.

```
bitro.diameter -0.08 0.32  
chest.girth    -0.10 0.34  
chest.width    -0.04 0.27
```

Значения компонент вычисляются при помощи формул

```
PC1 = 0.28*height + 0.30*arm.span + 0.30*forearm + 0.29*lower.leg -  
      0.06*weight - 0.08*bitro.diameter - 0.10*chest.girth -  
      0.04*chest.width
```

и

```
PC2 = -0.05*height - 0.08*arm.span - 0.09*forearm - 0.06*lower.leg +  
      0.33*weight + 0.32*bitro.diameter + 0.34*chest.girth +  
      0.27*chest.width
```

Подразумевается, что промеры тела были стандартизованы (среднее = 0, стандартное отклонение = 1). Обратите внимание, что коэффициенты для РС1 примерно равны или 0,3, или 0. То же самое верно и для РС2. С практической точки зрения, можно упростить дальнейший анализ, приняв значения первой компоненты равными среднему арифметическому стандартизованных значений четырех первых переменных. Сходным образом вторую компоненту можно определить как среднее стандартизованных значений вторых четырех переменных. Это то, что я в большинстве случаев сделал бы на практике.

«Маленькое мгновение» завоевывает мир

Люди, анализирующие данные, частенько путают РСА и ЕФА. Одна из причин этого – историческая, она берет начало с программы, которая называлась “Little Jiffy”⁵ (я не шучу). Эта была одна из распространенных старых программ для проведения факторного анализа, но по умолчанию она проводила анализ главных компонент, извлекая компоненты, собственные значения которых превышали единицу, и проводя вращение варимакс. Данная программа использовалась так широко, что многие социологи стали отождествлять эти настройки по умолчанию с разведочным факторным анализом (ЕФА). Многие более поздние статистические программы также использовали эти настройки для проведения ЕФА⁶.

5 «Маленькое мгновение». – Прим. пер.

6 Автор заблуждается, РСА как метод выделения факторов есть прямо в базовой процедуре fa(). Метод выделения осей тут совсем ни при чем. Суть различий между РСА и ЕФА заключается в проведении дополнительного вращения осей после отбрасывания выделенных факторов, которые не входят в «простую структуру». Процедура дополнительного вращения системы координат, да и само понятие «простой структуры» просто не имеют смысла с точки зрения математической статистики. Ортогональные вращения дают тождественные данные, а косоугольные вращения фактически разрушают данные. Следует также заметить, что обоснование процедуры РСА методами математической статистики вызвало другое часто встречающееся заблуждение. Поскольку обоснование процедуры построено вокруг трансформа-

Как, я надеюсь, вы поймете из следующего раздела, между РСА и ЕФА есть важные фундаментальные различия. Вы можете узнать больше о путанице между РСА и ЕФА, прочитав публикацию Hayton, Allen & Scarfello (2004).

Если ваша цель – это поиск скрытых переменных, которые объясняют наблюдаемые, тогда вам следует обратиться к факторному анализу. Это станет темой следующего раздела.

14.3. Разведочный факторный анализ

Цель ЕФА – объяснить корреляции внутри набора наблюдаемых переменных меньшим набором более фундаментальных ненаблюдаемых переменных, которые лежат в основе данных. Эти гипотетические ненаблюдаемые переменные называют *факторами*. Каждый фактор должен объяснять дисперсию, разделенную между двумя или более наблюдаемыми переменными, так что формально они называются *общими факторами*. Модель выглядит так:

$$X_i = a_1 F_1 + a_2 F_2 + \dots + a_p F_p + U_i$$

где X_i – это i -ая наблюдаемая переменная ($i = 1 \dots k$), F_j – это общие факторы ($j = 1 \dots p$), и $p < k$. U_i – это уникальная составляющая переменной X_i (не объясненная общими факторами). Параметр a_j можно интерпретировать как степень вклада каждого фактора в наблюдаемую переменную. Если вернуться к примеру с набором данных Harman74.cor из начала этой главы, то можно сказать, что результаты отдельных испытуемых по 24 психологическим тестам представляют собой взвешенные комбинации способностей испытуемых, оцененных по четырем лежащим в основе этих тестов психологическим параметрам.

Хотя модели РСА и ЕФА различаются, многие шаги окажутся сходными. Мы проиллюстрируем процесс, применив ЕФА к корреляциям между результатами шести психологических тестов для 112 человек. Тесты были следующими: невербальная оценка общего умственного

ций многомерного нормального распределения, то и сам метод РСА ошибочно ограничивают применением только к нормально распределенным данным. На самом деле метод РСА не налагает никаких ограничений на вид обрабатываемых данных (в этом легко убедиться, прочитав оригиналную работу: в исходной формулировке К. Пирсона ставится задача об аппроксимации конечного множества данных и отсутствует даже гипотеза об их статистической природе, не говоря уже о распределении). – Прим. пер.

развития (`general`), тест на завершение фигур (`picture`), тест блочных конструкций (`blocks`), тест с лабиринтом (`maze`), тест на понимание прочитанного (`reading`) и тест на словарный запас (`vocab`). Можем ли мы объяснить результаты, показанные участниками в этих тестах, при помощи небольшого числа основополагающих, или скрытых, психологических параметров?

Ковариационная матрица для наблюдаемых переменных содержится в наборе данных `ability.cov`. Ее можно преобразовать в корреляционную матрицу при помощи функции `cov2cor()`. Пропущенные данные в этом случае отсутствуют.

```
> options(digits=2)
> covariances <- ability.cov$cov
> correlations <- cov2cor(covariances)
> correlations
      general picture blocks maze reading vocab
general   1.00    0.47  0.55  0.34    0.58  0.51
picture    0.47    1.00  0.57  0.19    0.26  0.24
blocks     0.55    0.57  1.00  0.45    0.35  0.36
maze       0.34    0.19  0.45  1.00    0.18  0.22
reading    0.58    0.26  0.35  0.18    1.00  0.79
vocab      0.51    0.24  0.36  0.22    0.79  1.00
```

Поскольку вы ищете гипотетические составные переменные, которые объясняют данные, вы будете использовать EFA. Как и в случае PCA, следующая задача – это решить, сколько факторов извлечь.

14.3.1. Определение числа извлекаемых факторов

Для того чтобы решить, сколько факторов извлечь, воспользуйтесь функцией `fa.parallel()`:

```
> library(psych)
> covariances <- ability.cov$cov
> correlations <- cov2cor(covariances)
> fa.parallel(correlations, n.obs=112, fa="both", n.iter=100,
  main=" Диаграммы собственных значений с параллельным
  анализом")
```

Полученная диаграмма представлена на рис. 14.4. Обратите внимание на то, что функция выводит результаты и для анализа главных компонент, и для факторного анализа, так что вы можете сравнить эти два подхода (`fa="both"`).

Диаграммы собственных значений с параллельным анализом

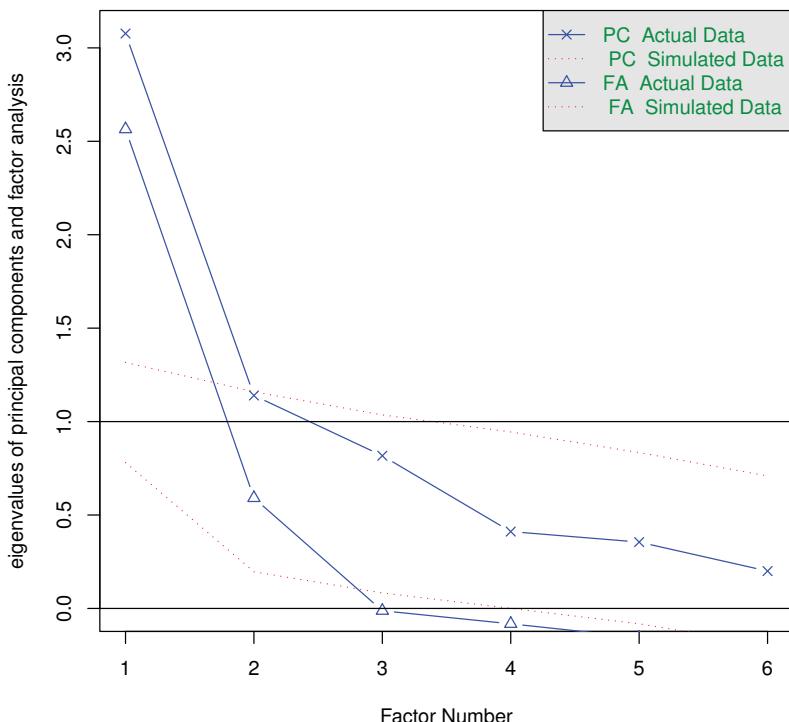


Рис. 14.4. Определение числа извлекаемых факторов для примера с психологическими тестами. Показаны результаты как для РСА (предлагается выделить две компоненты), так и для ЕФА (следует выделить два фактора)

На этом графике нужно обратить внимание на несколько вещей. Если бы вы проводили РСА, вам бы пришлось выбирать между одной компонентой (критерий собственных значений, параллельный анализ) или двумя компонентами (собственные значения больше единицы). Если вы сомневаетесь, всегда лучше выделить больше факторов, чем меньше. Выбор большего числа факторов, чем это необходимо, приводит к меньшему искажению «правильных» результатов анализа.

В случае ЕФА ясно видно, что нужно выделять два фактора. Первые два собственных значения (треугольники) находятся выше изгиба кривой собственных значений, а также превышают среднее значение

собственных значений, полученных на основании 100 симулированных матриц данных. Для EFA собственные значения, согласно критерию Кайзера-Харриса, должны превышать 0, а не 1 (многие этого не знают, так что это хороший способ выигрывать пари на вечеринках). В данном случае критерий Кайзера-Харриса также свидетельствует о необходимости выделения двух факторов.

14.3.2. Выделение общих факторов

Теперь, когда вы решили выделить два фактора, для достижения своей цели можете воспользоваться функцией `fa()`. Формат ее применения таков:

```
fa(r, nfactors=, n.obs=, rotate=, scores=, fm=),
```

где

- `r` – это корреляционная матрица или таблица исходных данных;
- `nfactors` определяет число факторов, которое нужно выделить (1 по умолчанию);
- `n.obs` – число наблюдений (если анализируется корреляционная матрица);
- `rotate` определяет тип вращения факторов (по умолчанию облимин, `oblimin`);
- `scores` указывает, нужно ли вычислять значения факторов (по умолчанию – нет);
- `fm` задает метод факторного анализа (по умолчанию минрез, `minres`).

В отличие от PCA, существует много способов извлечения общих факторов: максимальное правдоподобие (`m1`), повторные главные оси (*iterated principal axis*, `pa`), взвешенный наименьший квадрат (`wls`), обобщенные взвешенные наименьшие квадраты (`gls`) и наименьшие остатки (`minres`). Статистики в основном предпочитают способ максимального правдоподобия, поскольку в его основе лежит определенная статистическая модель. Иногда этот метод не сходится, и в таком случае часто хорошо работает метод повторных главных осей. Более подробное описание всех этих подходов дано в работах Mulaik (2009) и Gorsuch (1983).

В приведенном примере мы выделим факторы, не вращая их, при помощи метода повторных главных осей. Результаты приведены в следующем программном коде.

Программный код 14.6. Выделение факторов при помощи метода повторных главных осей без вращения

```
> fa <- fa(correlations, nfactors=2, rotate="none", fm="pa")
> fa
Factor Analysis using method = pa
Call: fa(r = correlations, nfactors = 2, rotate = "none", fm = "pa")
Standardized loadings based upon correlation matrix
      PA1    PA2    h2   u2
general 0.75  0.07  0.57  0.43
picture  0.52  0.32  0.38  0.62
blocks   0.75  0.52  0.83  0.17
maze     0.39  0.22  0.20  0.80
reading  0.81 -0.51  0.91  0.09
vocab    0.73 -0.39  0.69  0.31
      PA1    PA2
SS loadings  2.75  0.83
Proportion Var 0.46  0.14
Cumulative Var 0.46  0.60
[... остальные сведения удалены ...]
```

Отсюда видно, что два фактора учитывают 60% дисперсии результатов шести психологических тестов. Однако, как следует из факторных нагрузок, полученные факторы нелегко интерпретировать. Вращение факторов должно помочь.

14.3.3. Вращение факторов

Вращать два фактора, полученные в разделе 14.3.3, можно при помощи ортогонального или наклонного вращения. Давайте попробуем оба способа и посмотрим, чем они отличаются. Сначала проведем ортогональное вращение (следующий программный код).

Программный код 14.7. Выделение факторов с ортогональным вращением

```
> fa.varimax <- fa(correlations, nfactors=2, rotate="varimax", fm="pa")
> fa.varimax
Factor Analysis using method = pa
Call: fa(r = correlations, nfactors = 2, rotate = "varimax", fm = "pa")
Standardized loadings based upon correlation matrix
      PA1    PA2    h2   u2
general 0.49  0.57  0.57  0.43
picture  0.16  0.59  0.38  0.62
blocks   0.18  0.89  0.83  0.17
maze     0.13  0.43  0.20  0.80
reading  0.93  0.20  0.91  0.09
vocab    0.80  0.23  0.69  0.31
      PA1    PA2
SS loadings  1.83  1.75
```

```
Proportion Var 0.30 0.29
Cumulative Var 0.30 0.60
[... остальные сведения удалены ...]
```

Глядя на факторные нагрузки, можно сказать, что факторы стало легче интерпретировать. Чтение и словарь соответствуют первому фактору, а фигуры, лабиринт и блоки – второму. Общий показатель невербального интеллектуального развития связан с обоими факторами. На основании этого можно решить, что у нас есть фактор вербального интеллекта и фактор невербального интеллекта.

Использование ортогонального вращения «заставляет» факторы быть нескоррелированными. А что вы обнаружите, если допустите возможность корреляции факторов? Можно попробовать применить наклонное вращение, такое как промакс (promax, см. следующий программный код).

Программный код 14.8. Выделение факторов с наклонным вращением

```
> fa.promax <- fa(correlations, nfactors=2, rotate="promax", fm="pa")
> fa.promax
Factor Analysis using method = pa
Call: fa(r = correlations, nfactors = 2, rotate = "promax", fm = "pa")
Standardized loadings based upon correlation matrix
      PA1    PA2    h2   u2
general  0.36  0.49  0.57  0.43
picture -0.04  0.64  0.38  0.62
blocks  -0.12  0.98  0.83  0.17
maze    -0.01  0.45  0.20  0.80
reading  1.01 -0.11  0.91  0.09
vocab    0.84 -0.02  0.69  0.31
      PA1    PA2
SS loadings     1.82 1.76
Proportion Var 0.30 0.29
Cumulative Var 0.30 0.60
With factor correlations of
      PA1    PA2
PA1  1.00  0.57
PA2  0.57  1.00
[... остальные сведения удалены ...]
```

Этот результат отличается от полученного при ортогональном вращении несколькими аспектами. При ортогональном вращении внимание фокусируется на *матрице факторной структуры* (factor structure matrix, корреляциях переменных с факторами). При наклонном вращении рассматриваются три матрицы: факторной структуры, модели факторов и *корреляций между факторами* (factor intercorrelation matrix).

Матрица модели факторов (factor pattern matrix) – это матрица стандартизованных регрессионных коэффициентов. Они определяют веса (коэффициенты) для предсказания значений переменных по значениям факторов.

В программном коде 14.8 значения в колонках PA1 и PA2 формируют матрицу модели факторов. Это скорее стандартизованные регрессионные коэффициенты, чем коэффициенты корреляции. Анализ этих столбцов также используется для характеристики факторов (хотя не все сходятся во мнениях по этому поводу). Здесь мы вновь можем выделить вербальный и невербальный факторы. Корреляция между этими двумя факторами составляет 0.57. Это высокая корреляция. Если бы эта корреляция была низкой, нам нужно было бы вернуться к ортогональному вращению, чтобы не усложнять результаты анализа.

Матрица факторной структуры (или факторных нагрузок) не вычисляется. Однако вы можете без труда рассчитать ее при помощи формулы $F = P * \Phi$, где F – это матрица факторных нагрузок, P – это матрица модели факторов, а Φ – это матрица корреляций между факторами. Это умножение можно произвести при помощи следующей простой функции:

```
fsm <- function(oblique) {  
  if (class(oblique)[2] == "fa" & is.null(oblique$Phi)) {  
    warning("Объект не похож на результат наклонного вращения факторов")  
  } else {  
    P <- unclass(oblique$loading)  
    F <- P %*% oblique$Phi  
    colnames(F) <- c("PA1", "PA2")  
    return(F)  
  }  
}
```

Применив эту функцию к нашему примеру, мы получим:

```
> fsm(fa.promax)  
            PA1   PA2  
general  0.64  0.69  
picture   0.33  0.61  
blocks   0.44  0.91  
maze     0.25  0.45  
reading  0.95  0.47  
vocab    0.83  0.46
```

Теперь мы можем проанализировать корреляции между переменными и факторами. Сравнив их с матрицей факторных нагрузок после ортогонального вращения, вы увидите, что столбцы не такие

«чистые». Это происходит потому, что вы допустили возможность корреляции факторов друг с другом. Хотя результаты наклонного вращения более сложны для интерпретации, они часто представляют более реалистичную модель данных.

Вы можете графически изобразить результаты ортогонального или наклонного вращения при помощи команд `factor.plot()` или `fa.diagram()`. Команда

```
factor.plot(fa.promax, labels=rownames(fa.promax$loadings))
```

позволяет получить диаграмму, представленную на рис. 14.5.

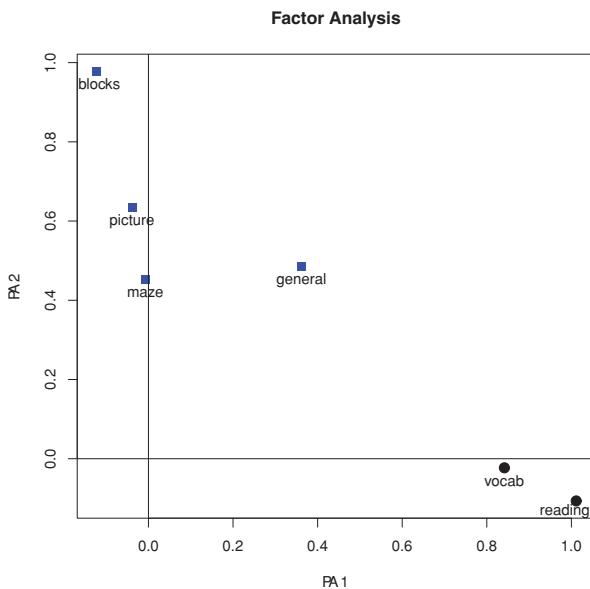


Рис. 14.5. Двухфакторная диаграмма для результатов психологических тестов из набора данных `ability.cov`. Словарь и чтение сопряжены с первым фактором (PA1), а блоки, картинки и лабиринт – со вторым (PA2). Результаты теста на общее умственное развитие связаны с обоими факторами

Команда

```
fa.diagram(fa.promax, simple=FALSE)
```

создает диаграмму, показанную на рис. 14.6. Если вы используете опцию `simple=TRUE`, для каждого объекта будет показана только наибольшая нагрузка. На этой диаграмме изображены нагрузки для

каждого фактора, а также корреляции между факторами. Такая диаграмма полезна, когда у вас есть несколько факторов.

В реальной жизни использование факторного анализа для такого небольшого числа переменных маловероятно. Мы сделали это здесь для того, чтобы с результатами было легко работать. Если вы хотите проверить свои умения, попробуйте применить факторный анализ к результатам 24 психологических тестов, содержащимся в наборе данных Harman74.cor. Команды

```
library(psych)
fa.24tests <- fa(Harman74.cor$cov, nfactors=4, rotate="promax")
```

помогут вам взяться за дело!

Factor Analysis

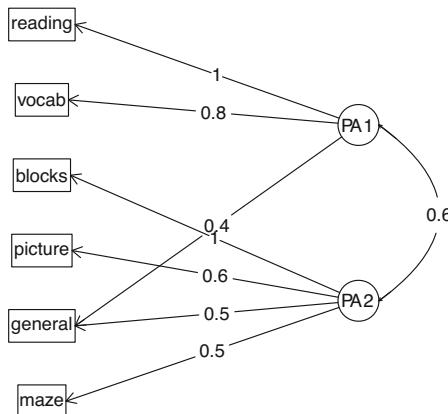


Рис. 14.6. Диаграмма для визуализации наклонного вращения двух факторов для результатов психологических тестов из набора данных ability.csv

14.3.4. Значения факторов

В отличие от PCA, EFA гораздо реже используется для вычисления значений факторов. Однако эти значения можно легко получить при помощи функции `fa()` с опцией `score=TRUE` (если доступны исходные данные). Кроме того, коэффициенты для вычисления значений факторов (стандартизованные регрессионные коэффициенты) содержатся в элементе `weights` объекта, который возвращает функция `fa()`.

Для набора данных `ability.csv` можно вычислить бета-коэффициенты для расчета значений двух выделенных факторов после наклонного вращения при помощи следующей команды:

```
> fa.promax$weights  
[,1] [,2]  
general 0.080 0.210  
picture 0.021 0.090  
blocks 0.044 0.695  
maze 0.027 0.035  
reading 0.739 0.044  
vocab 0.176 0.039
```

В отличие от значений компонент, которые вычисляются точно, значения факторов можно только оценить. Для этого существует несколько методов. Функция `fa()` основана на регрессионном подходе. Более подробная информация об оценке значений факторов приведена в публикации DiStefano, Zhu & Mondrila (2009).

Прежде чем двинуться дальше, давайте кратко рассмотрим другие пакеты для разведочного факторного анализа в R.

14.3.5. Другие пакеты для проведения факторного анализа

В R есть еще несколько пакетов, предназначенных для выполнения факторного анализа. В пакете `FactoMineR` реализованы методы PCA и EFA, а также другие модели для скрытых переменных. Этот пакет предоставляет много возможностей, которые мы не обсуждали в этой главе, включая совместное использование числовых и категориальных переменных. Пакет `FAIR` позволяет оценивать факторные модели при помощи генетического алгоритма, который обеспечивает возможность налагать ограничения на пространство возможных значений параметров модели. В пакете `GParotation` реализовано много дополнительных методов вращения факторов. Наконец, пакет `nFactors` дает возможность применять сложные техники для определения числа факторов, лежащих в основе данных.

14.4. Другие модели для латентных переменных

EFA – это одна из многих моделей скрытых переменных, которые используются в статистике. Мы закончим эту главу кратким описанием

других моделей, которые можно подгонять в R. К ним относятся модели для проверки априорных теорий, модели, которые могут работать со смешанными типами данных (числовые и категориальные) и модели, основанные лишь на многомерных таблицах сопряженности категориальных переменных.

При определении числа и значения факторов для EFA вы основывались на данных. Однако, начиная анализ, можно руководствоваться теоретическими представлениями о числе факторов, лежащих в основе переменных, и о том, как эти факторы связаны друг с другом. Затем вы можете проверить эту теорию при помощи собранных данных. Этот подход называется конфирматорным факторным анализом (confirmatory factor analysis, CFA).

CFA – это частный случай методического подхода, называемого моделированием структурных уравнений (structural equation modeling, SEM). SEM позволяет вам не только изначально задать число и состав факторов, которые лежат в основе данных, но и указать, как эти факторы влияют друг на друга. SEM можно определить как сочетание конфирматорного факторного анализа (для количественных переменных) и регрессионного анализа (для факторов). В составе результатов SEM выводятся статистические тесты и индексы соответствия моделей данным. В R есть несколько замечательных пакетов (включая `sem`, `openMx` и `lavaan`), при помощи которых можно выполнить SEM и CFA.

Пакет `ltm` можно использовать для подгонки моделей латентных переменных по результатам тестов и опросников. Эта методология обычно используется для создания крупномасштабных стандартизованных тестов, таких как американские отборочный тест при поступлении в колледж или в университет (Scholastic Aptitude Test, SAT) и тест при поступлении в магистратуру (Graduate Record Exam, GRE).

Модели латентных классов (latent class models), в которых предполагается, что лежащие в основе данных факторы будут категориальными, а не непрерывными, можно подобрать при помощи пакетов `FlexMix`, `lcmm`, `randomLCA` и `poLC`. Пакет `lcda` позволяет провести дискриминантный анализ для скрытых классов, а пакет `lsa` предназначен для проведения скрытого семантического анализа – методологии, используемой при обработке естественных языков.

В пакете `ca` реализованы функции для простого и множественного анализа соответствий. Этот метод позволяет исследовать структуру категориальных переменных в двух- и многомерных таблицах соответственно.

Наконец, в R реализовано много методов многомерного шкалирования (multidimensional scaling, MDS)⁷. MDS-анализ разработан для выявления измерений, лежащих в основе данных и объясняющих сходства и различия между исследованными объектами (например, странами). Функция `cmdscale()` из базовой версии программы позволяет провести классический MDS-анализ, функция `isoMDS()` из пакета `MASS` выполняет неметрический MDS-анализ. Пакет `vegan` также содержит функции для выполнения классического и неметрического MDS-анализа.

14.5. Резюме

В этой главе мы познакомились с методами анализа главных компонент и разведочного факторного анализа. Анализ главных компонент – это полезный метод снижения размерности данных, который заменяет множество скррелированных переменных небольшим набором несвязанных переменных, что упрощает анализ. Разведочный факторный анализ объединяет разнообразные методы обнаружения скрытых или ненаблюдаемых переменных (факторов), которые могут лежать в основе набора явных или наблюдаемых переменных.

В то время как цель анализа главных компонент – обобщить данные и снизить их размерность, разведочный факторный анализ может быть использован как инструмент для формулировки гипотез, полезный, когда вы пытаетесь выявить связи между большим числом переменных. Этот метод часто применяется в социологии для разработки теорий.

Хотя эти два подхода обладают многими поверхностными сходствами, между ними существуют и важные различия. В этой главе мы рассмотрели модели, которые лежат в основе каждого из этих методов, способы определения числа подлежащих выделению компонент/факторов, методы выделения компонент/факторов и их вращения (трансформации) с целью облегчения интерпретации, а также способы вычисления значений компонент или факторов. Основные этапы анализа главных компонент и разведочного факторного анализа представлены на рис. 14.7. Глава заканчивается кратким обсуждением других методов анализа скрытых переменных, реализованных в R.

7 Подробнее о многомерном шкалировании, анализе соответствий и других распространенных методах многомерного анализа, которые не упомянуты здесь (например, кластерном и дискриминантном), можно прочесть в книге А. Б. Шипунова с соавторами «Наглядная статистика. Используем R!» (М.: ДМК Пресс, 2012). – Прим. пер.

Поскольку анализ главных компонент и разведочный факторный анализ основаны на корреляционных матрицах, перед применением этих подходов важно избавиться от всех пропущенных значений. В разделе 4.5 мы кратко упомянули простые способы работы с пропущенными данными. В следующей главе мы разберем более совершенные методы для выявления пропущенных данных и избавления от них.

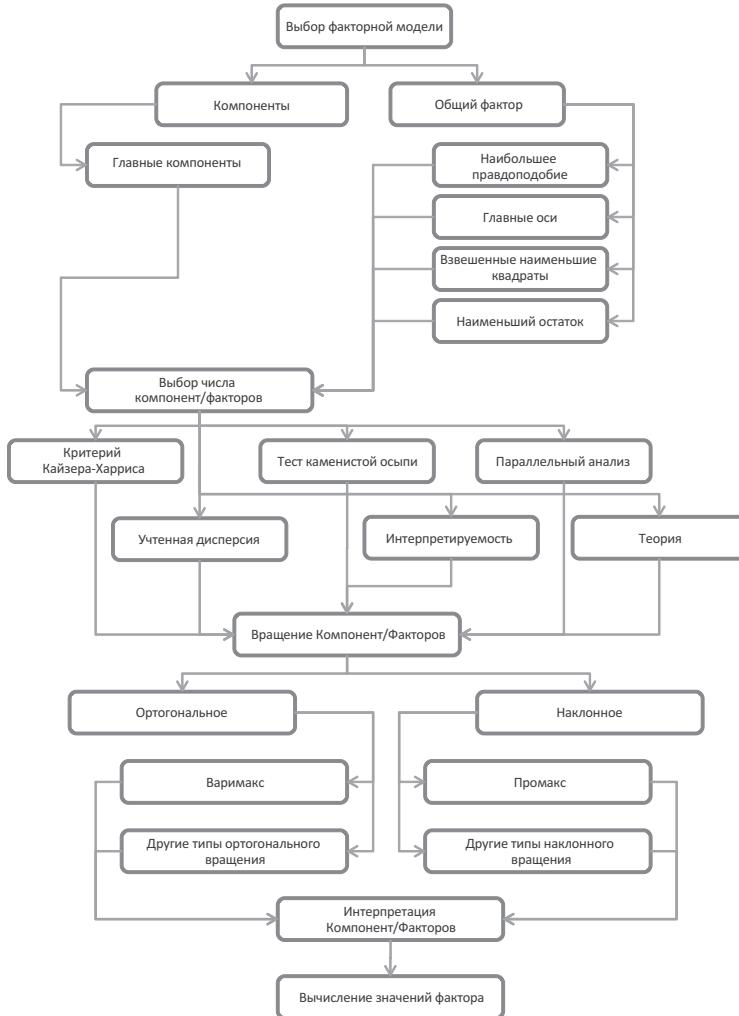


Рис. 14.7. Основные этапы анализа главных компонент и разведочного факторного анализа



ГЛАВА 15.

Продвинутые методы работы с пропущенными данными

В этой главе:

- Обнаружение пропущенных данных.
- Визуализация закономерностей в пропущенных данных.
- Анализ полных наблюдений.
- Множественное восстановление пропущенных данных.

В предыдущих главах мы занимались анализом полных наборов данных (без пропущенных значений). Хотя это упрощало описание статистических и графических методов, в реальном мире пропущенные данные неизбежны.

В некотором смысле влияние пропущенных данных – это то, чего каждый из нас хочет избежать. В книгах по статистике эта тема может вообще не затрагиваться, или же ее обсуждение сводится к нескольким абзацам. В статистических пакетах реализованы не всегда оптимальные методы работы с пропущенными данными. Даже учитывая, что обычно анализ данных (по крайней мере, в социологии) всегда включает работу с пропущенными наблюдениями, эта тема редко обсуждается при описании методов и результатов научных статей. Учитывая то, как часто пропущенные данные встречаются и насколько сильно их наличие может сделать несостоительными результаты исследования, можно сказать, что эта тема получила недостаточно внимания за пределами специализированных книг и курсов.

Данные могут отсутствовать по многим причинам. Участники анкетирования могут забыть ответить на один или более вопросов, отказаться отвечать на щекотливые вопросы или же утомиться и не закончить заполнение длинной анкеты. Испытуемые могут не выполнить инструкции или преждевременно выбыть из исследования. Записывающее оборудование может выйти из строя, подключение к Интернету – пропасть, а также данные могут быть неправильно за- кодированы. Наличие пропущенных данных даже может быть запланировано. Например, для увеличения эффективности исследования или снижения расходов вы можете решить не собирать полные данные обо всех участниках исследования. Наконец, данные могут быть потеряны по причинам, о которых вы никогда не узнаете.

К сожалению, большинство статистических методов рассчитано на то, что вы работаете с полными матрицами, векторами и таблицами данных. В большинстве случаев вам придется удалить пропущенные данные, перед тем как искать ответ на те вопросы, которые побудили вас начать исследование. От пропущенных данных вы можете избавиться, (1) удаляя наблюдения с такими данными или (2) заменяя пропущенные данные подходящими расчетными значениями. В любом случае, вы получите набор данных без пропущенных значений.

В этой главе мы рассмотрим как традиционные, так и новые подходы к работе с пропущенными данными. Мы будем в основном работать с пакетами `VIM` и `mice`. Установить оба этих пакета можно при помощи команды `install.packages(c("VIM", "mice"))`.

Чтобы сделать обсуждение более интересным, мы рассмотрим набор данных о сне млекопитающих (`sleep`) из пакета `VIM` (не перепутайте его с набором данных `sleep`, описывающим влияние лекарств на сон, из базовой версии программы). Данные описаны в работе Allison & Chichetti (1976), которая посвящена взаимосвязям между сном, экологией и морфологией 62 видов млекопитающих. Авторы исследовали, почему необходимая продолжительность сна различается от вида к виду. Параметры сна служили зависимыми переменными, а экологические и морфологические характеристики были независимыми переменными. К параметрам сна относились продолжительность сна со сновидениями (`Dream`) и без сновидений (`NonD`), а также их сумма (`Sleep`). Морфологические характеристики – это вес тела в килограммах (`BodyWgt`), вес мозга в граммах (`BrainWgt`), продолжительность жизни в годах (`span`) и продолжительность беременности в днях (`Gest`). Экологические характеристики – пресс хищников (`Pred`), степень уязвимости во время сна (`Exp`) и общая степень

опасности, которой подвергается животное (*Danger*). Экологические характеристики оценивались по пятибалльной шкале, принимавшей значения от 1 (низкий) до 5 (высокий).

В исходной статье анализ был ограничен только теми видами, для которых имелись полные данные. Мы же пойдем дальше, проанализировав все 62 вида при помощи множественного восстановления пропущенных данных.

15.1. Этапы работы с пропущенными данными

Новички в области исследования пропущенных значений обнаружат обескураживающее многообразие подходов, критических отзывов и методологий. Классическое руководство в этой области – это публикация Little & Rubin (2002). Прекрасные доступные обзоры написаны Allison (2001), Schafer & Graham (2002) и Schlomer, Bauman & Card (2010). Полный алгоритм работы с пропущенными данными, как правило, состоит из следующих этапов:

1. Обнаружить пропущенные данные.
2. Выявить причины их наличия.
3. Удалить наблюдения с пропущенными значениями или заменить пропущенные данные подходящими расчетными значениями.

К сожалению, обнаружение пропущенных данных – это единственный простой этап. Узнать, почему данные пропущены, можно, только если полностью понимать процесс их появления. Решить, как поступить с пропущенными данными, можно, если оценить какие методы дадут наиболее надежный и аккуратный результат.

Классификация типов пропущенных данных

В статистике обычно выделяют три типа пропущенных данных. Они, как правило, описываются в терминах вероятностей, но идеи, лежащие в основе выделения этих типов, очевидны. Мы рассмотрим их на примере продолжительности сна со сновидениями из набора данных `sleep` (где для 12 животных есть пропущенные значения).

(1) *Полностью случайный пропуск.* – Если наличие пропущенных значений в переменной не зависит от значений любой другой наблюдаемой или ненаблюдаемой переменной, тогда данные являются отсутствующими полностью случайно. Если бы пропуск данных по продолжительности сна со сновидениями не был бы ничем обусловлен, то данные подходили бы под

этую категорию. Отметим, что если пропуски во всех переменных полностью случайны, то наблюдения без пропусков – это просто случайная выборка из общего массива данных.

(2) *Случайный пропуск*. – Если наличие пропущенных значений в переменной зависит от других переменных, но не от самих неотмеченных значений, то данные являются отсутствующими случайно. Например, если пропуски значений продолжительности сна со сновидениями выше у животных с меньшим весом тела (возможно, потому, что за мелкими животными сложнее наблюдать) и если пропуски не зависят от продолжительности сновидений, то пропуски можно назвать случайными. В данном случае наличие или отсутствие данных по продолжительности сна со сновидениями будет случайным, если избавиться от влияния веса животного.

(3) *Неслучайный пропуск*. – Под эту категорию попадают пропущенные значения, которые не относятся к первым двум категориям. Например, если животные с меньшей продолжительностью сновидений с большей вероятностью имеют пропущенные значения этой величины (возможно, из-за трудностей регистрации более кратких событий), пропуски будут неслучайными.

Существует много методов для работы с пропущенными данными, и нет гарантии, что они дадут один и тот же результат. На рис. 15.1 показано все разнообразие методов, которые используются для работы с пропущенными данными, наряду с пакетами, в которых реализованы эти методы.

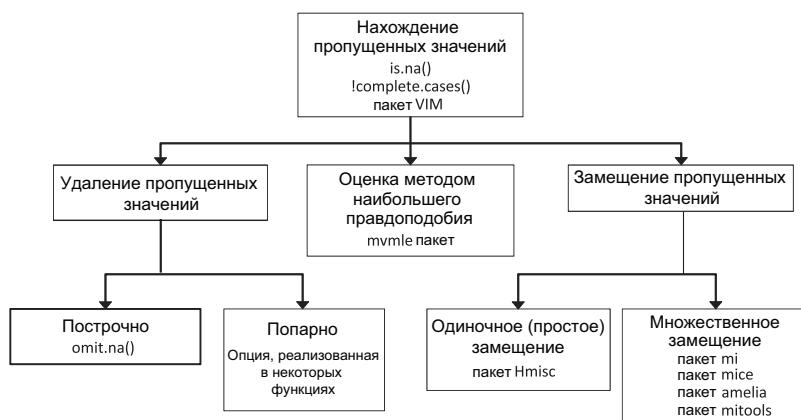


Рис. 15.1. Методы, которые используются для работы с пропущенными данными, и пакеты R, в которых реализованы эти методы

Для полного обзора методов работы с пропущенными данными понадобилась бы отдельная книга. В этой главе мы рассмотрим методы изучения закономерностей в пропущенных данных и сосредоточимся на трех наиболее распространенных методах работы с неполными

наблюдениями (вычислительный подход, удаление неполных наблюдений и множественное восстановление пропущенных данных). Мы закончим главу кратким обсуждением прочих методов, включая те, которые могут оказаться полезными в отдельных ситуациях.

15.2. Обнаружение пропущенных значений

Для начала давайте повторим сведения, изложенные в разделе 4.5 главы 4, и будем отталкиваться от них. В R пропущенные значения обозначаются как `NA` (нет в наличии – *not available*), а недопустимые значения – как `Nan` (не являются числом – *not a number*). В дополнение символы `Inf` и `-Inf` обозначают плюс и минус бесконечность соответственно. Для обнаружения пропущенных, недопустимых и бесконечных значений используются функции `is.na()`, `is.nan()` и `is.infinite()` соответственно. Каждая из них возвращает значения `TRUE` (правда) и `FALSE` (ложь). Примеры приведены в табл. 15.1.

Таблица 15.1. Примеры значений, которые возвращаются функциями `is.na()`, `is.nan()` и `is.infinite()`

x	is.na()	is.nan()	is.infinite()
<code>x <- NA</code>	TRUE	FALSE	FALSE
<code>x <- 0 / 0</code>	TRUE	TRUE	FALSE
<code>x <- 1 / 0</code>	FALSE	FALSE	TRUE

Эти функции возвращают объект того же размера, что и исходный, но при этом каждый элемент заменяется выражением `TRUE`, если это – элемент проверяемого типа, и `FALSE` – в противном случае. К примеру, пусть `y <- c(1, 2, 3, NA)`. Тогда функция `is.na(y)` возвратит вектор `c(FALSE, FALSE, FALSE, TRUE)`.

Функцию `complete.cases()` можно использовать для обнаружения строк в матрице или таблице данных, которые не содержат пропущенных значений. Эта функция возвращает логический вектор со значениями `TRUE` для всех полных строк и `FALSE` – для строк с одним и более пропущенными значениями.

Давайте применим эту функцию к нашему набору данных:

```
# загружаем набор данных
data(sleep, package="VIM")
# выводим строки без пропущенных значений
```

```
sleep[complete.cases(sleep),]  
# выводим строки, в которых хотя бы одно значение пропущено  
sleep[!complete.cases(sleep),]
```

Изучение результатов показывает, что 42 строки не имеют пропущенных значений, а в 20 строках хотя бы одно значение пропущено.

Поскольку логические значения TRUE и FALSE эквивалентны числам 1 и 0, функции sum() и mean() могут быть использованы для получения полезной информации о пропущенных данных. Рассмотрим следующие команды:

```
> sum(is.na(sleep$Dream))  
[1] 12  
> mean(is.na(sleep$Dream))  
[1] 0.19  
> mean(!complete.cases(sleep))  
[1] 0.32
```

Полученный результат свидетельствует о том, что в переменной Dream пропущено 12 значений. Пропущенные значения этой переменной содержатся в 19% строк. В 32% строк из набора данных есть хотя бы одно пропущенное значение.

При поиске пропущенных значений нужно помнить про две вещи. Во-первых, функция complete.cases() «воспринимает» как отсутствующие только значения NA и NaN. Бесконечные значения (`Inf` и `-Inf`) пропущенными не считаются. Во-вторых, упомянутые выше функции можно использовать лишь для обнаружения пропущенных значений в объектах R. Логические выражения типа `myvar == NA` никогда не будут верными.

Теперь, когда вы знаете, как обнаруживать пропущенные значения программными средствами, давайте познакомимся со способами исследования возможной *структуре* пропущенных данных.

15.3. Исследование структуры пропущенных данных

Перед тем как решить, что делать с пропущенными данными, стоит выяснить, в каких переменных содержатся пропущенные значения, в каких объемах и комбинациях. В этом разделе мы рассмотрим табличные, графические и корреляционные методы исследования структуры пропущенных значений. С их помощью вам нужно понять, *почему* данные отсутствуют. Ответ повлияет на дальнейший ход статистического анализа.

15.3.1. Представление пропущенных значений в виде таблицы

Вы уже видели простейший подход к поиску пропущенных значений. Вы можете использовать описанную в разделе 15.2 функцию `complete.cases()` для создания списка полных строк или, наоборот, строк, в которых есть хотя бы одно пропущенное значение. Хотя по мере увеличения объема данных этот подход становится наименее привлекательным. В таком случае вы можете обратиться к другим функциям R.

Функция `md.pattern()` из пакета `mice` представляет информацию о пропущенных значениях в табличной форме. Применив эту функцию к набору данных о сне, мы получим следующее:

```
> library(mice)
> data(sleep, package="VIM")
> md.pattern(sleep)

   BodyWgt BrainWgt Pred Exp Danger Sleep Span Gest Dream NonD
42       1       1     1     1       1     1     1     1     1     1     1     0
 2       1       1     1     1       1     1     0     1     1     1     1     1
 3       1       1     1     1       1     1     1     0     1     1     1     1
 9       1       1     1     1       1     1     1     1     0     0     0     2
 2       1       1     1     1       1     0     1     1     1     1     0     2
 1       1       1     1     1       1     1     0     0     1     1     1     2
 2       1       1     1     1       1     0     1     1     0     0     0     3
 1       1       1     1     1       1     1     0     1     0     0     0     3
 0       0       0     0     0       0     4     4     4     4     12    14    38
```

Единицы и нули в ячейках таблицы дают нам представление о структуре пропущенных значений, 0 обозначает пропущенное значение для данной переменной, а 1 – имеющееся значение. Первая строка содержит информацию о полных наблюдениях (все значения в ячейках равны 1). Вторая строка показывает число наблюдений, в которых нет пропущенных значений во всех переменных, кроме `Span`. В первом столбце приведено число наблюдений с данным типом структуры пропущенных данных, а последний столбец содержит данные о числе столбцов с пропущенными значениями для каждого типа структуры. В этом примере видно, что есть 42 наблюдения без пропущенных данных и два наблюдения, в которых пропущены только значения переменной `Span`. В девяти наблюдениях пропущены значения переменных `NonD` и `Dream`. Всего в наборе данных содержится $(42 \times 0) + (2 \times 1) + \dots + (1 \times 3) = 38$ пропущенных значений. В последней строке таблицы приведено общее число пропущенных значений для каждой переменной.

15.3.2. Визуальное исследование структуры пропущенных данных

Хотя табличное представление результатов при помощи функции `md.pattern()` компактно, мне всегда было легче исследовать закономерности визуально. К счастью, в пакете `VIM` реализованы многочисленные функции для визуализации структуры пропущенных данных. В этом разделе мы познакомимся с некоторыми из них, включая `aggr()`, `matrixplot()` и `scattMiss()`.

Функция `aggr()` графически отображает число наблюдений для каждой отдельной переменной и для каждой комбинации переменных. Например, программный код

```
library("VIM")
aggr(sleep, prop=FALSE, numbers=TRUE)
```

позволяет построить диаграмму, представленную на рис. 15.2. (Пакет `VIM` открывает GUI-интерфейс – вы можете закрыть его; для выполнения всех задач, описанных в этой главе, мы будем использовать программный код).

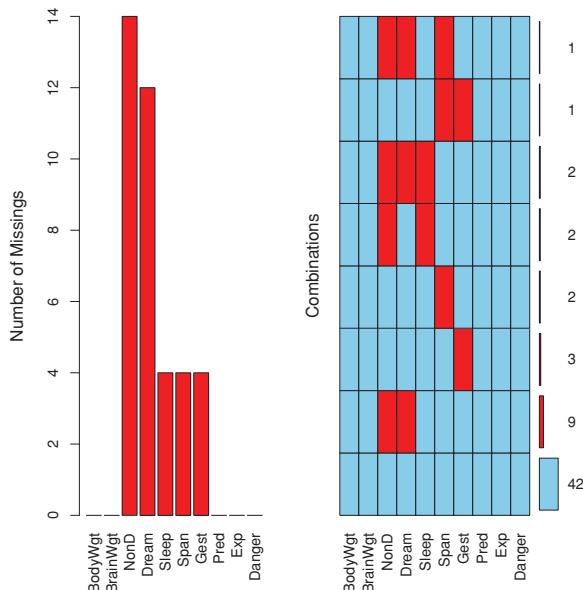


Рис. 15.2. Функция `aggr()` графически представляет структуру пропущенных значений в наборе данных `sleep`

Видно, что больше всего пропущено значений в переменной NonD (14) и что есть два животных, для которых нет данных по переменным NonD, Dream и Sleep. По 42 млекопитающим у нас есть полные данные.

Команда `aggr(sleep, prop=TRUE, numbers=TRUE)` создает такую же диаграмму, только в ней вместо чисел указаны проценты. Опция `numbers=FALSE` (по умолчанию) позволяет не выводить числовые подписи.

Функция `matrixplot()` графически отображает данные по каждой строке. Диаграмма, полученная при помощи этой функции, представлена на рис. 15.3. Здесь числовые данные нормализованы так, что они все находятся в интервале [0, 1] и представлены оттенками серого. Чем светлее оттенок, тем более низкому значению он соответствует. По умолчанию пропущенные значения обозначены красным.

Эта диаграмма интерактивная: матрица сортируется по значениям той переменной, название которой вы выбираете мышкой. Строки на рис. 15.3 отсортированы в порядке убывания значений переменной `BodyWgt`. Матричная диаграмма позволяет вам увидеть, зависит ли наличие пропущенных значений в одной или нескольких переменных от значений других переменных. В данном случае видно, что пропущенные значения отсутствуют в переменных, характеризующих сон (`Dream`, `NonD`, `Sleep`), при низких значениях веса тела или мозга (`BodyWgt`, `BrainWgt`).

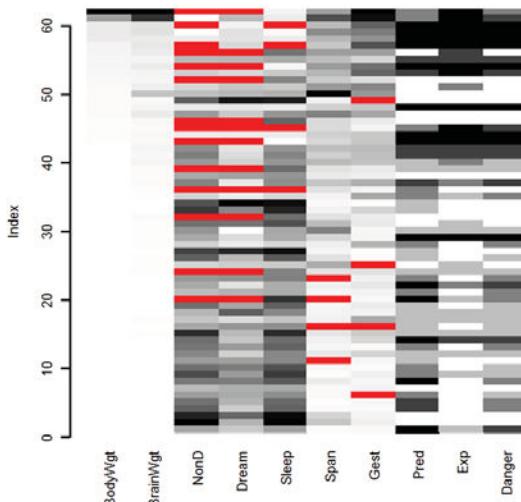


Рис. 15.3. Матричная диаграмма наблюдаемых и пропущенных значений в каждой строке набора данных `sleep`.

Матрица отсортирована по значениям переменной `BodyWgt`

Команда `marginplot()` позволяет получить диаграмму рассеяния для двух переменных, где информация о пропущенных значениях представлена на полях. Рассмотрим связь между длительностью сна со сновидениями и продолжительностью беременности. Команда

```
marginplot(sleep[c("Gest", "Dream")], pch=c(20),  
          col=c("darkgray", "red", "blue"))
```

позволяет создать диаграмму, представленную на рис. 15.4. Необязательные параметры `pch` и `col` определяют тип и цвет отображаемых символов.

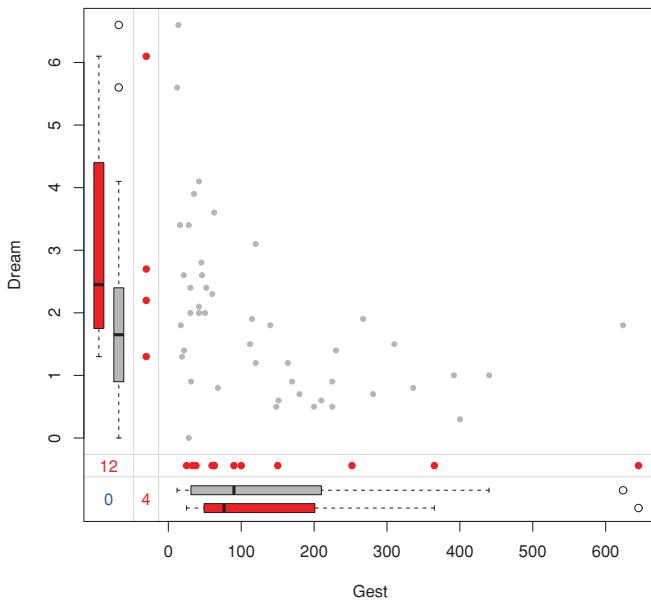


Рис. 15.4. Диаграмма рассеяния для длительности сна со сновидениями и продолжительности беременности с информацией о пропущенных значениях на полях

На основной части рисунка представлена диаграмма рассеяния для переменных `Gest` и `Dream` (на основании полных наблюдений для этих двух переменных). На левом поле расположены диаграммы размахов для значений переменной `Dream`, которым соответствуют наблюдаемые значения переменной `Gest` (темно-серый) и пропущенные (красный). Четыре красные точки соответствуют значениям переменной `Dream` для животных, у которых значения переменной `Gest` неизвестны. На нижнем поле эти две переменные «поменялись

местами». Видно, что существует отрицательная связь между длиной сна и продолжительностью беременности и что сведения о продолжительности беременности, как правило, отсутствуют у животных с более длительным сном. Число наблюдений с пропущенными значениями в обеих переменных дано синим цветом на пересечении полей (в нижнем левом углу).

Пакет `VIM` позволяет построить множество диаграмм, которые помогут вам понять роль пропущенных значений в наборе данных и с которыми стоит познакомиться. Там есть функции для создания диаграмм рассеяния, диаграмм размахов, гистограмм, матриц диаграмм рассеяния, параллельных диаграмм, графиков-щеток и пузырьковых диаграмм.

15.3.3. Использование корреляции для исследования пропущенных значений

Перед тем как двигаться дальше, нам нужно познакомиться с еще одним заслуживающим внимания подходом. Можно заменить ваши данные условными значениями: 1 – обозначает пропущенное значение, 0 – имеющееся. Полученную таблицу иногда называют *матрицей теней* (*shadow matrix*). Вычисление корреляций между этими преобразованными переменными и между ними и исходными переменными поможет узнать, значения каких переменных имеют тенденцию отсутствовать согласованно, а также выявить связи между отсутствием значений в одной переменной и значениями других переменных.

Рассмотрим следующую команду:

```
x <- as.data.frame(abs(is.na(sleep)))
```

Элементы таблицы данных `x` равны 1, если соответствующий элемент набора данных `sleep` отсутствует, и 0 – в противном случае. В этом можно убедиться, рассмотрев первые несколько строк каждой таблицы:

```
> head(sleep, n=5)
   BodyWgt BrainWgt NonD Dream Sleep Span Gest Pred Exp Danger
1 6654.000    5712.0   NA     NA    3.3  38.6   645    3    5     3
2    1.000      6.6    6.3    2.0    8.3   4.5    42    3    1     3
3    3.385     44.5   NA     NA   12.5  14.0    60    1    1     1
4    0.920      5.7   NA     NA   16.5    NA    25    5    2     3
5 2547.000    4603.0   2.1    1.8    3.9  69.0   624    3    5     4
> head(x, n=5)
   BodyWgt BrainWgt NonD Dream Sleep Span Gest Pred Exp Danger
```

1	0	0	1	1	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	1	1	0	0	0	0	0	0
4	0	0	1	1	0	1	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0

Выражение

```
y <- x[which(sd(x) > 0)]
```

позволяет извлечь переменные, в которых пропущено несколько значений (но не все), а команда

```
cor(y)
```

дает возможность вычислить корреляции между этими преобразованными переменными:

	NonD	Dream	Sleep	Span	Gest
NonD	1.000	0.907	0.486	0.015	-0.142
Dream	0.907	1.000	0.204	0.038	-0.129
Sleep	0.486	0.204	1.000	-0.069	-0.069
Span	0.015	0.038	-0.069	1.000	0.198
Gest	-0.142	-0.129	-0.069	0.198	1.000

Здесь видно, что переменные Dream и NonD чаще имеют пропущенные значения в одних и тех же строках ($r = 0.91$). Это в меньшей степени относится к переменным Sleep и NonD ($r = 0.49$) или Sleep и Dream ($r = 0.20$).

Наконец, можно посмотреть на связь между наличием пропущенных значений одной переменной и наблюдаемыми значениями других переменных:

```
> cor(sleep, y, use="pairwise.complete.obs")
      NonD   Dream   Sleep   Span   Gest
BodyWgt  0.227  0.223  0.0017 -0.058 -0.054
BrainWgt  0.179  0.163  0.0079 -0.079 -0.073
NonD       NA      NA      NA -0.043 -0.046
Dream     -0.189     NA -0.1890  0.117  0.228
Sleep     -0.080 -0.080      NA  0.096  0.040
Span      0.083  0.060  0.0052      NA -0.065
Gest      0.202  0.051  0.1597 -0.175      NA
Pred      0.048 -0.068  0.2025  0.023 -0.201
Exp       0.245  0.127  0.2608 -0.193 -0.193
Danger    0.065 -0.067  0.2089 -0.067 -0.204
Warning message:
In cor(sleep, y, use = "pairwise.complete.obs") :
  the standard deviation is zero
```

В этой корреляционной матрице строки – это наблюдаемые переменные, а столбцы – это преобразованные переменные, несущие

информацию о наличии пропущенных значений. Вы можете не обращать внимания на предупреждающее сообщение (warning message) и на значения NA – это побочные эффекты нашего метода.

Из первого столбца нашей матрицы видно, что значения продолжительности сна без сновидений чаще бывают пропущенными у животных с высокими значениями веса тела ($r = 0.227$), продолжительности беременности ($r = 0.202$) и уязвимости во время сна (0.245). Другие столбцы можно интерпретировать сходным образом. Все коэффициенты корреляции в этой матрице не слишком велики, что позволяет предположить, что пропуски не слишком отклоняются от полностью случайных и могут считаться случайными.

Обратите внимание на то, что вы никогда не можете исключить возможность того, что пропуски неслучайны, поскольку вы не знаете, какие значения в реальности принимали пропущенные данные. К примеру, вы не знаете, есть ли связь между продолжительностью сновидений и вероятностью пропуска значений этой переменной. В отсутствие убедительных свидетельств в пользу обратного мы обычно считаем пропуски полностью случайными или случайными.

15.4. Выявление источников пропущенных данных и эффекта от них

Мы исследуем количество, распределение и структуру пропущенных данных, чтобы получить представление (1) о возможных механизмах возникновения пропущенных данных и (2) о влиянии пропущенных данных на качество ответов на интересующие нас вопросы. В частности, мы хотим знать:

- Какая доля данных пропущена?
- Сосредоточены ли пропущенные данные в нескольких переменных или они широко распределены по всему набору данных?
- Можно ли их считать случайными?
- Позволяет ли ковариация пропущенных данных друг с другом или с наблюдаемыми данными обнаружить возможный механизм, лежащий в основе пропущенных значений?

Ответы на эти вопросы позволят определить, какие статистические методы лучше всего подходят для анализа ваших данных. Например, если пропущенные данные сосредоточены в нескольких второсте-

пенных переменных, вы можете удалить эти переменные и спокойно продолжить анализ данных. Если частота пропущенных значений невелика (скажем, меньше 10%) и они случайным образом распределены по всему набору данных (полностью случайные пропуски), то можно ограничиться полными строками и все равно получить надежные и обоснованные результаты. Если вы предполагаете, что данные пропущены случайно или полностью случайно, то можно выполнить множественное восстановление пропущенных данных, чтобы получить обоснованные выводы. Если значения пропущены неслучайно, нужно обратиться к специализированным методам, собрать новые данные или сменить профессию на более простую и благодарную.

Вот несколько примеров:

- В недавнем исследовании бумажных анкет я обнаружил, что некоторые вопросы пропускаются совместно, потому что многие респонденты не поняли, что у третьей страницы анкеты была обратная сторона (с этими вопросами). В данном случае это полностью случайные пропуски.
- В другом исследовании стилей руководства по всему миру выяснилось, что много данных отсутствует в графе «образование». Оказалось, что европейцы чаще оставляли это поле анкеты пустым. Это происходило потому, что предложенные значения не имели смысла для участников опроса из определенных стран. В этом случае пропуски, скорее всего, были случайными.
- Наконец, я участвовал в исследованиях депрессии, где пожилые пациенты чаще пропускали вопросы, посвященные подавленному настроению, по сравнению с молодыми. Выяснилось, что пожилые пациенты неохотно признавали такие симптомы, поскольку это шло вразрез с их стремлением не пасовать перед трудностями. К сожалению, было также выявлено, что пациенты с наиболее тяжелыми формами депрессии чаще пропускали эти вопросы из-за чувства безнадежности и проблем с концентрацией внимания. Так что эти значения были пропущены неслучайно.

Как вы могли понять из приведенных примеров, обнаружение закономерностей в пропущенных данных – только первый шаг. Для выявления источника пропущенных данных необходимо использовать ваши знания о предмете исследования и процессе сбора данных.

Теперь, когда мы рассмотрели источники и эффект от пропущенных данных, давайте разберемся, как в этом случае нужно изменить

стандартные статистические подходы. Мы сосредоточимся на самых распространенных методах: восстановление пропущенных данных, традиционный способ, который заключается в удалении пропущенных данных, и современный подход с использованием моделирования. Попутно мы кратко рассмотрим устаревшие методы, которые должны выйти из употребления. Наша цель остается неизменной: ответить настолько корректно, насколько это возможно, на основные вопросы, ради ответа на которые мы и собирали данные, учитывая, что полная информация нам недоступна.

15.5. Рациональный подход

При так называемом рациональном подходе (rational approach) при попытках замены или восстановления пропущенных значений используются математические или логические связи между переменными. Несколько примеров помогут прояснить ситуацию.

В наборе данных `sleep` переменная `Sleep` представляет собой сумму переменных `Dream` и `NonD`. Если вы знаете значения любых двух переменных для данного животного, вы можете вычислить третью. Таким образом, если есть какие-то наблюдения, где пропущены значения только одной из трех переменных, вы можете восстановить пропущенную информацию при помощи сложения или вычитания.

Во втором примере мы рассмотрим исследование различий в стиле жизни людей разных поколений, где возрастная группа определялась годом рождения. Участников исследования спрашивали и об их дате рождения, и о возрасте. Если дата рождения отсутствует, вы можете восстановить год рождения (и, таким образом, возрастную группу), зная возраст опрашиваемых и дату проведения исследования.

В качестве примера использования логических связей для восстановления пропущенных значений рассмотрим исследование стиля руководства. Во время опросов участникам нужно было ответить, являются ли они руководителем, и указать число непосредственных подчиненных. Если опрашиваемые игнорировали вопрос про руководство, но указывали, что у них есть один или более непосредственный подчиненный, было бы логичным предположить, что они являлись руководителями.

В качестве последнего примера я расскажу о сравнительных исследованиях стиля и эффективности руководства мужчин и женщин, в которые я часто бываю вовлечен. Респонденты заполняют анкеты, в которых они указывают свои имя и фамилию, пол и дают детальную

оценку своего стиля руководства и его эффективности. Если опрашиваемый игнорирует вопрос о поле, мне приходится восстанавливать это значение для того, чтобы можно было включить этого человека в исследование. В одном из последних опросов 66 000 руководителей 11 000 (17%) оставили вопрос о поле без ответа.

Для улучшения ситуации я пользуюсь следующим алгоритмом. Для начала я выясняю соответствие имени и пола¹. Некоторые имена ассоциированы только с мужчинами, некоторые – только с женщинами, и некоторые – с обоими полами. Например, имя Вильям встретилось 417 раз, и это всегда были мужчины. Напротив, имя Крис было у 237 человек, часть из них – мужчины (86%), а часть – женщины (14%). Если имя встречалось больше, чем 20 раз, и всегда принадлежало либо мужчинам, либо женщинам (но никогда – и тем, и другим), я предполагал, что это имя ассоциировано только с одним полом. Я использовал это допущение для составления таблицы соответствий полов и имен, характерных только для одного пола. Используя эту таблицу для участников с неизвестным полом, я смог восстановить 7000 пропущенных значений (63% всех пропущенных ответов).

Для применения рационального подхода обычно требуется творческое мышление, наряду с должностными навыками управления данными. Восстановление данных может быть точным (как в примере со сном) или приблизительным (как в примере с полами). В следующем разделе мы рассмотрим подход, при котором от пропущенных значений избавляются, удаляя строки.

15.6. Анализ полных строк (построчное удаление)

При анализе полных строк (complete-case analysis) работают только со строками без пропущенных значений. Многие широко используемые статистические пакеты по умолчанию используют построчное удаление (listwise/case-wise deletion) при работе с пропущенными данными. Такой подход настолько распространен, что многие аналитики, проводя регрессионный или дисперсионный анализы, могут даже не осознавать, что существует «проблема пропущенных данных», с которой нужно как-то справляться!

Функцию `complete.cases()` можно использовать для извлечения полных строк матрицы или таблицы данных:

1 Дело происходило в Америке, где очень редко можно определить пол человека по его фамилии. – *Прим. пер.*

```
newdata <- mydata[complete.cases(mydata), ]
```

Этого же результата можно добиться при помощи функции `na.omit`:

```
newdata <- na.omit(mydata)
```

В результате обеих команд все строки с пропущенными значениями будут удалены из объекта `mydata` перед его сохранением в виде объекта `newdata`.

Предположим, что вас интересуют корреляции между переменными в исследовании сна. Применив построчное удаление, вы исключите из анализа всех животных, для которых часть данных пропущена:

```
> options(digits=1)
> cor(na.omit(sleep))
   BodyWgt BrainWgt NonD Dream Sleep Span Gest Pred Exp Danger
BodyWgt    1.00     0.96 -0.4 -0.07 -0.3  0.47  0.71  0.10  0.4   0.26
BrainWgt    0.96     1.00 -0.4 -0.07 -0.3  0.63  0.73 -0.02  0.3   0.15
NonD       -0.39    -0.39  1.0  0.52  1.0 -0.37 -0.61 -0.35 -0.6  -0.53
Dream      -0.07    -0.07  0.5  1.00  0.7 -0.27 -0.41 -0.40 -0.5  -0.57
Sleep      -0.34    -0.34  1.0  0.72  1.0 -0.38 -0.61 -0.40 -0.6  -0.60
Span        0.47     0.63 -0.4 -0.27 -0.4  1.00  0.65 -0.17  0.3   0.01
Gest        0.71     0.73 -0.6 -0.41 -0.6  0.65  1.00  0.09  0.6   0.31
Pred        0.10    -0.02 -0.4 -0.40 -0.4 -0.17  0.09  1.00  0.6   0.93
Exp         0.41     0.32 -0.6 -0.50 -0.6  0.32  0.57  0.63  1.0   0.79
Danger      0.26     0.15 -0.5 -0.57 -0.6  0.01  0.31  0.93  0.8   1.00
```

Корреляции в этой таблице основаны только на 42 животных, для которых у нас есть полные данные по всем переменным. Обратите внимание на то, что команда `cor(sleep, use="complete.obs")` даст такие же результаты.

Если бы вы хотели исследовать влияние продолжительности жизни и срока беременности на длину сновидений, вы могли бы применить линейную регрессию с построчным удалением пропущенных значений:

```
> fit <- lm(Dream ~ Span + Gest, data=na.omit(sleep))
> summary(fit)
Call:
lm(formula = Dream ~ Span + Gest, data = na.omit(sleep))
Residuals:
   Min     1Q Median     3Q    Max 
-2.333 -0.915 -0.221  0.382  4.183 
Coefficients:
            Estimate Std. Error t value Pr(>|t|)    
(Intercept) 2.480122   0.298476    8.31 3.7e-10 ***
Span        -0.000472   0.013130   -0.04    0.971    
Gest        -0.004394   0.002081   -2.11    0.041 *  

```

```
Signif. codes: 0 '****' 0.001 '***' 0.01 '**' 0.05 '*' 0.1 '.' 1
Residual standard error: 1 on 39 degrees of freedom
Multiple R-squared: 0.167, Adjusted R-squared: 0.125
F-statistic: 3.92 on 2 and 39 DF, p-value: 0.0282
```

В данном случае видно, что млекопитающие с более короткой беременностью дольше видят сны (при постоянной продолжительности жизни) и что продолжительность жизни не связана с количеством сновидений при постоянных значениях срока беременности. Анализ был основан на 42 строках с полными данными.

Что будет, если в предыдущем примере выражение `data=na.omit(sleep)` заменить на `data=sleep?` Как многие функции R, `lm()` использует узкое определение построчного удаления значений. В данном случае будут удалены строки, в которых есть пропущенные значения в любой из переменных, используемых функцией (`Dream`, `Span` и `Gest`). Анализ будет основан на 44 строках.

При построчном удалении данных предполагается, что пропуски полностью случайны (то есть полные строки – это случайная выборка из всего набора данных). В данном примере мы предполагали, что 42 проанализированных животных – это случайная выборка из всех 62 видов. Полученные параметры регрессии исказятся настолько, насколько будет нарушено предположение о полной случайности пропусков. Удаление всех строк с пропущенными данными также может снизить статистическую мощность, уменьшив объем выборки. В приведенном примере построчное удаление сократило размер выборки на 32%. Далее мы рассмотрим подход, который позволяет использовать весь набор данных целиком (включая строки с пропущенными данными).

15.7. Метод множественного восстановления пропущенных данных

Метод множественного восстановления пропущенных данных (multiple imputation, MI) – это способ заполнения пропусков при помощи повторного моделирования. Множественное восстановление часто применяется для работы с пропущенными данными в сложных ситуациях. При этом подходе из существующего набора данных с пропущенными значениями создается несколько полных наборов данных (обычно от трех до десяти). Для замещения пропущенных значений

в производных наборах данных используются методы Монте-Карло. К каждому из производных наборов данных применяются стандартные статистические методы, а на основании их результатов формируются оценки окончательных результатов и доверительные интервалы, которые учитывают неопределенность, созданную пропущенными значениями. Эта идея хорошо реализована в таких пакетах R, как *Amelia*, *mice* и *mi*. В этом разделе мы сосредоточим свое внимание на подходе, реализованном в пакете *mice* (многомерное восстановление данных при помощи связанных уравнений – multivariate imputation by chained equations).

Для того чтобы понять, как работает этот пакет, рассмотрите схему на рис. 15.5.

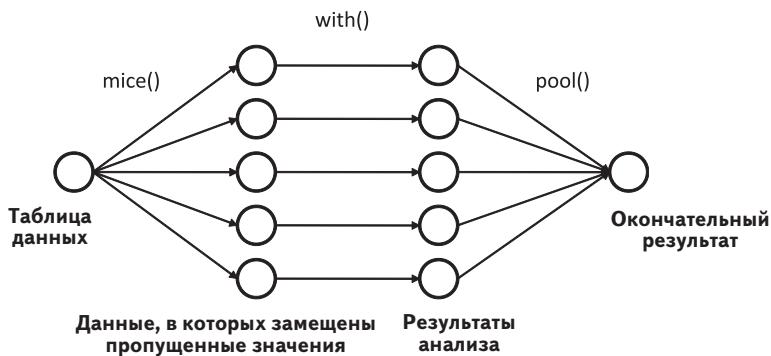


Рис. 15.5. Этапы множественного восстановления пропущенных данных при помощи подхода, реализованного в пакете *mice*

Функция *mice()* использует исходную таблицу данных с пропущенными значениями, а возвращает объект, содержащий несколько полных наборов данных (пять по умолчанию). Каждый такой полный набор данных получается при восстановлении пропущенных данных исходной таблицы. В алгоритме восстановления данных есть случайная составляющая, поэтому все производные полные наборы данных немного отличаются друг от друга. Затем при помощи функции *with()* применяется статистическая модель (например, линейная или обобщенная линейная). Наконец, функция *pool()* объединяет результаты, полученные для отдельных производных наборов данных. Стандартные ошибки и значения статистической ошибки первого рода в этой окончательной модели корректно отражают неопределенность, обусловленную как наличием пропущенных значений, так и алгоритмами множественной замены.

Как функция `mice()` замещает пропущенные значения?

Пропущенные значения замещаются при помощи выборок Гиббса. По умолчанию значения каждой переменной, содержащей пропущенные значения, предсказываются по значениям остальных переменных. Полученные уравнения используются для замещения пропущенных данных подходящими значениями. Этот процесс повторяется, пока значения для пропущенных данных не сойдутся. Пользователь может выбирать вид предсказательной модели (называемой простейшим методом замещения) для каждой переменной и определять переменные, которые в нее войдут.

По умолчанию для замены пропущенных значений непрерывных переменных используется метод соответствия предсказанного среднего (predictive mean matching), а логистическая или полиномиальная логистическая регрессия применяются для дихотомических (фактор с двумя уровнями) или полиномиальных (фактор с более чем двумя уровнями) переменных соответственно. К другим простейшим методам замещения относятся байесовская линейная регрессия, дискриминантный анализ, двухуровневое нормальное замещение и составление случайных выборок из наблюдаемых значений. Пользователи также могут использовать собственные методы.

Анализ данных в пакете `mice` обычно имеет такую структуру:

```
library(mice)
imp <- mice(mydata, m)
fit <- with(imp, analysis)
pooled <- pool(fit)
summary(pooled),
```

где

- `mydata` – это матрица или таблица данных с пропущенными значениями;
- `imp` – список, содержащий m наборов данных с восстановленными пропущенными значениями вместе с информацией о том, как это восстановление было проведено. По умолчанию $m = 5$;
- `analysis` – формула, определяющая тип статистического метода, который должен быть применен к каждому из m восстановленных наборов данных. К таким методам относятся `lm()` – линейная регрессия, `glm()` – обобщенная линейная регрессия, `gam()` – обобщенные аддитивные модели и `nbinom()` – отрицательные биномиальные модели. В формулах внутри скобок зависимая переменная указывается слева от знака `~`, а независимые (разделенные знаком `+`) – справа;
- `fit` – список, содержащий результаты m отдельных статистических анализов;

- *pooled* – список, содержащий усредненные результаты этих *t* отдельных статистических анализов.

Давайте применим метод множественного восстановления пропущенных данных к нашему набору данных о сне. Мы повторим анализ, описанный в разделе 15.6, только на этот раз используем данные обо всех 62 животных. Установите начальное значение для генератора случайных чисел, равное 1234, чтобы ваши результаты совпали с моими.

```
> library(mice)
> data(sleep, package="VIM")
> imp <- mice(sleep, seed=1234)
[...выводимая информация удалена для экономии места...]
> fit <- with(imp, lm(Dream ~ Span + Gest))
> pooled <- pool(fit)
> summary(pooled)

            est      se      t    df Pr(>|t|)    lo   95
(Intercept) 2.58858 0.27552 9.395 52.1 8.34e-13 2.03576
Span        -0.00276 0.01295 -0.213 52.9 8.32e-01 -0.02874
Gest        -0.00421 0.00157 -2.671 55.6 9.91e-03 -0.00736
                  hi   95 nmis     fmi
(Intercept) 3.14141   NA 0.0870
Span         0.02322     4 0.0806
Gest        -0.00105     4 0.0537
```

Отсюда видно, что коэффициент регрессии для переменной *Span* незначим ($p \cong 0.08$), а коэффициент для переменной *Gest* значим на уровне $p < 0.01$. Если вы сравните эти результаты с полученными при помощи анализа полных строк (раздел 15.6), вы увидите, что по данному вопросу мы пришли к таким же заключениям. Срок беременности (статистически) значимо отрицательно связан с продолжительностью сновидений при постоянных значениях длины жизни. Кстати, в столбце *fmi* содержатся сведения о количестве отсутствующей информации (то есть о доле изменчивости, которая вносит неопределенность, сопряженную с пропущенными значениями).

Вы можете получить больше информации о замещении пропущенных значений, исследовав объекты, созданные в ходе анализа. К примеру, давайте посмотрим на общую информацию об объекте *imp*:

```
> imp
Multiply imputed data set
Call:
mice(data = sleep, seed = 1234)
Number of multiple imputations: 5
Missing cells per column:
  BodyWgt BrainWgt NonD Dream Sleep Span Gest Pred
```

```

      0      0     14     12      4      4      4      0
Exp    Danger
      0      0
Imputation methods:
BodyWgt BrainWgt NonD Dream Sleep Span Gest Pred
      ""      "" "pmm" "pmm" "pmm" "pmm" "pmm" ""
Exp    Danger
      ""      ""
VisitSequence:
NonD Dream Sleep Span Gest
      3      4      5      6      7
PredictorMatrix:
      BodyWgt BrainWgt NonD Dream Sleep Span Gest Pred Exp Danger
BodyWgt      0      0      0      0      0      0      0      0      0      0
BrainWgt      0      0      0      0      0      0      0      0      0      0
NonD         1      1      0      1      1      1      1      1      1      1
Dream        1      1      1      0      1      1      1      1      1      1
Sleep         1      1      1      1      0      1      1      1      1      1
Span          1      1      1      1      1      0      1      1      1      1
Gest          1      1      1      1      1      1      0      1      1      1
Pred          0      0      0      0      0      0      0      0      0      0
Exp           0      0      0      0      0      0      0      0      0      0
Danger        0      0      0      0      0      0      0      0      0      0
Random generator seed value: 1234
  
```

Из представленной информации видно, что было создано пять искусственных наборов данных (Number of multiple imputations) и что к каждой переменной с пропущенными значениями был применен метод сочетания предсказанного среднего (pmm, predictive mean matching). Восстановление данных не применялось ("") к переменным BodyWgt, BrainWgt, Pred, Exp и Danger, поскольку в них не было пропущенных значений. Стока VisitSequence содержат информацию о последовательности замещения данных в переменных – справа налево, начиная с NonD и заканчивая Gest. Наконец, из матрицы PredictorMatrix следует, что при восстановлении пропущенных данных каждой переменной использовались все остальные переменные. (В этой матрице строки соответствуют переменным, значения которых восстанавливались, столбцы – переменным, которые использовались для расчета замещаемых значений, а 1/0 указывает, использовалась ли переменная: да/нет).

Восстановленные значения можно увидеть, вызывая отдельные компоненты объекта imp. Например, команда

```

> imp$imp$Dream
      1      2      3      4      5
1  0.5  0.5  0.5  0.5  0.0
3  2.3  2.4  1.9  1.5  2.4
  
```

```

4 1.2 1.3 5.6 2.3 1.3
14 0.6 1.0 0.0 0.3 0.5
24 1.2 1.0 5.6 1.0 6.6
26 1.9 6.6 0.9 2.2 2.0
30 1.0 1.2 2.6 2.3 1.4
31 5.6 0.5 1.2 0.5 1.4
47 0.7 0.6 1.4 1.8 3.6
53 0.7 0.5 0.7 0.5 0.5
55 0.5 2.4 0.7 2.6 2.6
62 1.9 1.4 3.6 5.6 6.6

```

позволяет отобразить все пять вариантов восстановленных значений для каждого из 12 животных, для которых отсутствовали значения переменной `Dream`. Обзор этих матриц позволит вам понять, имеют ли восстановленные значения смысл. Отрицательные значения продолжительности сна могут заставить вас остановиться (или станут причиной ночных кошмаров).

Каждый из *m* восстановленных наборов данных можно вызвать на экран при помощи команды `complete()`. Формат ее применения таков:

```
complete(imp, action=#),
```

где # указывает на номер одного из *m* искусственных наборов данных с замещенными значениями. Например, команды

```

> dataset3 <- complete(imp, action=3)
> dataset3
   BodyWgt BrainWgt NonD Dream Sleep Span Gest Pred Exp Danger
1 6654.00    5712.0  2.1   0.5   3.3  38.6  645     3    5      3
2    1.00      6.6   6.3   2.0   8.3   4.5   42     3    1      3
3    3.38     44.5  10.6   1.9  12.5  14.0   60     1    1      1
4    0.92      5.7  11.0   5.6  16.5   4.7   25     5    2      3
5 2547.00    4603.0  2.1   1.8   3.9  69.0  624     3    5      4
6   10.55     179.5   9.1   0.7   9.8  27.0  180     4    4      4
[...вывод удален для экономии места...]

```

позволяют увидеть третий (из пяти) полный набор данных, созданный посредством множественного восстановления пропущенных данных.

Из-за экономии места мы лишь коротко рассмотрели алгоритм восстановления пропущенных данных, реализованный в пакете `mice`. Пакеты `mi` и `Amelia` также содержат стоящие методы. Если вы интересуетесь множественным восстановлением пропущенных данных, рекомендую следующие источники информации:

- страница часто задаваемых вопросов по множественному восстановлению пропущенных данных (www.stat.psu.edu/~jls/mifaq.html);

- статьи Van Buuren & Groothuis-Oudshoorn (2010) и Yu-Sung, Gelman, Hill & Yajima (2010);
- электронный ресурс – AmeliaII: программа для работы с пропущенными данными (<http://gking.harvard.edu/amelia/>).

Каждый из этих источников информации поможет углубить и расширить ваши представления об этой важной, но недостаточно широко используемой методологии.

15.8. Другие подходы к пропущенным данным

В R реализовано и несколько других подходов к пропущенным данным. Пакеты, перечисленные в табл. 15.2, хотя и не так широко применимы, как описанные выше методы, содержат функции, которые могут быть весьма полезными при определенных обстоятельствах.

Таблица 15.2. Специализированные методы работы с пропущенными данными

Пакет	Описание
Hmisc	Содержит многочисленные функции для простого и множественного восстановления пропущенных значений и их восстановления с использованием канонических значений
Mvnmle	Оценка методом максимального правдоподобия для многомерных нормальных данных с пропущенными значениями
Cat	Множественное восстановление пропущенных данных для многомерных категориальных данных при помощи лог-линейных моделей
arrayImpute, arrayMissPattern, SeqKnn	Полезные функции для работы с пропущенными значениями в данных, получаемых с использованием технологии микрочипов
longitudinalData	Содержит полезные служебные функции, включая алгоритмы интерполяции, для восстановления пропущенных данных во временных сериях
kmi	Множественное восстановление пропущенных данных методом Каплана-Майера при анализе выживаемости
mix	Множественное восстановление пропущенных значений для смешанных категориальных и непрерывных данных при помощи общей модели положения (general location model)
pan	Множественное восстановление данных для многомерных сгруппированных данных

Наконец, существует два метода работы с пропущенными данными, которые до сих пор используются, но должны уже считаться устаревшими. Это попарное удаление и простое восстановление.

15.8.1. Попарное удаление

Попарное удаление (pairwise deletion) при работе с неполными наборами данных обычно рассматривается как альтернатива построчному удалению. При попарном удалении наблюдения удаляются только в том случае, если это пропущенные значения в переменных, которые используются в конкретном анализе данных. Рассмотрим следующий программный код:

```
> cor(sleep, use="pairwise.complete.obs")
```

	BodyWgt	BrainWgt	NonD	Dream	Sleep	Span	Gest	Pred	Exp	Danger
BodyWgt	1.00	0.93	-0.4	-0.1	-0.3	0.30	0.7	0.06	0.3	0.13
BrainWgt	0.93	1.00	-0.4	-0.1	-0.4	0.51	0.7	0.03	0.4	0.15
NonD	-0.38	-0.37	1.0	0.5	1.0	-0.38	-0.6	-0.32	-0.5	-0.48
Dream	-0.11	-0.11	0.5	1.0	0.7	-0.30	-0.5	-0.45	-0.5	-0.58
Sleep	-0.31	-0.36	1.0	0.7	1.0	-0.41	-0.6	-0.40	-0.6	-0.59
Span	0.30	0.51	-0.4	-0.3	-0.4	1.00	0.6	-0.10	0.4	0.06
Gest	0.65	0.75	-0.6	-0.5	-0.6	0.61	1.0	0.20	0.6	0.38
Pred	0.06	0.03	-0.3	-0.4	-0.4	-0.10	0.2	1.00	0.6	0.92
Exp	0.34	0.37	-0.5	-0.5	-0.6	0.36	0.6	0.62	1.0	0.79
Danger	0.13	0.15	-0.5	-0.6	-0.6	0.06	0.4	0.92	0.8	1.00

В этом примере корреляции между любыми двумя переменными вычислены на основе всех имеющихся наблюдений для этих двух переменных (не принимая во внимание все остальные). Корреляция между переменными BodyWgt и BrainWgt рассчитана для всех 62 животных (число животных, для которых имеются данные в обеих переменных). Корреляция между переменными BodyWgt и NonD основана на 42 млекопитающих, а корреляция между переменными Dream и NonDream – на 46 животных.

Хотя кажется, что попарное удаление позволяет использовать все имеющиеся данные, на самом деле расчет каждого коэффициента производится для разных выборок из данных. Это может привести к искаженным и сложно интерпретируемым результатам. Я рекомендую держаться подальше от этого подхода.

15.8.2. Простое (нестохастическое) восстановление данных

При простом восстановлении данных (simple imputation) пропущенные значения переменной заменяются одним и тем же числом

(например, средним, медианой или модой). Используя *замену средним значением*, вы могли бы заменить пропущенные значения переменной `Dream` числом 1.97, а пропущенные значения переменной `NonD` – значением 8.67 (средние арифметические для `Dream` и `NonD` соответственно). Обратите внимание на то, что замены нестochasticны, то есть в них не присутствует случайная составляющая (в отличие от множественного восстановления).

Преимущество простого восстановления заключается в том, что оно «решает проблему пропущенных значений» без уменьшения размера выборки. Ну что же, простое восстановление данных – это действительно просто, однако оно дает искаженные результаты для неслучайных пропусков. Если пропущенных значений достаточно много, их простое замещение, скорее всего, приведет к снижению среднеквадратичной ошибки, искажит корреляции между переменными и вызовет появление некорректных значений статистической ошибки первого рода. Как и в случае попарного удаления, я рекомендую избегать этого подхода в большинстве ситуаций с пропущенными данными.

15.9. Резюме

Многие статистические методы подразумевают, что исходные данные не содержат пропущенных значений (таких как `NA`, `NaN`, `Inf`). Однако в условиях реального мира большинство наборов данных содержит пропущенные значения. Поэтому вы должны или удалить пропущенные значения, или заменить их подходящими числами, перед тем как проводить статистический анализ. Часто в статистических пакетах по умолчанию используются не самые удачные методы работы с пропущенными значениями. Поэтому вам важно понимать суть доступных подходов и варианты использования каждого из них.

В этой главе мы обсудили методы обнаружения пропущенных значений и изучения структуры пропущенных данных. Нашей задачей было понять механизмы, которые привели к появлению пропущенных значений, и оценить их возможное влияние на последующий статистический анализ. Затем мы рассмотрели три распространенных способа работы с пропущенными данными: рациональный подход, построчное удаление и множественное восстановление данных.

Рациональный подход можно использовать для восстановления пропущенных значений, когда данные отчасти избыточны или когда для решения проблемы можно использовать дополнительную инфор-

мацию. Построчное удаление пропущенных значений полезно, если пропуски полностью случайны и последующее уменьшение объема выборки не окажет серьезного воздействия на мощность статистических тестов. Множественное восстановление пропущенных данных быстро завоевывает популярность как метод для решения сложных проблем с пропущенными данными, если можно предположить, что пропуски были случайными. Хотя многие аналитики могут быть незнакомы с методами множественного восстановления данных, созданные пользователями пакеты (`mice`, `mi`, `Amelia`) делают эти методы легкодоступными. Я верю в то, что в ближайшие несколько лет мы будем наблюдать быстрый рост популярности этих методов.

Мы закончили данную главу кратким обзором пакетов R, в которых реализованы специализированные подходы к работе с пропущенными данными, и указали на подходы к работе с пропущенными значениями (попарное удаление, простое восстановление), которых следует избегать.

В следующей главе мы рассмотрим «продвинутые» графические методы, включая использование категоризованных графиков, системы `ggplot2` и интерактивной графики.



ГЛАВА 16.

Продвинутые графические методы

В этой главе:

- Категоризованные графики и пакет `lattice`.
- Грамматика графических построений в пакете `ggplot2`.
- Интерактивная графика.

В предыдущих главах мы создавали самые разные общие и специализированные диаграммы (и хорошоенько при этом повеселились). Многие из них были созданы в базовой графической системе R. Учитывая общее разнообразие реализованных в R методов, наверное, вы не удивитесь, узнав, что на самом деле в этой программе сейчас существует четыре отдельных и самодостаточных графических системы.

В дополнение к базовой системе отдельные графические системы представлены в пакетах `grid`, `lattice` и `ggplot2`. Каждый из этих пакетов создан для расширения возможностей базовой графической системы и исправления ее недостатков.

Графическая система `grid` предоставляет доступ нижнего уровня к графическим базовым элементам, что дает программистам большую свободу в настройке графического вывода. В пакете `lattice` реализован интуитивный подход к исследованию многомерных взаимодействий при помощи своеобразных одно-, двух- или трехмерных диаграмм, называемых *категоризованными* (*trellis graphs*). В пакете `ggplot2` можно создавать инновационные диаграммы на основе всеобъемлющей графической «грамматики».

В этой главе мы начнем с обзора четырех графических систем. Затем мы сосредоточимся на диаграммах, которые можно создать в па-

кетах `lattice` и `ggplot2`. Эти пакеты значительно увеличивают разнообразие и качество диаграмм, которые вы можете создать в R.

Мы закончим эту главу знакомством с интерактивными диаграммами. Взаимодействие с диаграммами в реальном времени поможет вам лучше понять данные и разработать более совершенные гипотезы о взаимосвязях между переменными. В данном случае мы сосредоточимся на возможностях, предоставляемых пакетами `ipplots`, `playwith`, `latticist` и `rggobi`.

16.1. Четыре графические системы R

Как уже было сказано ранее, в R есть четыре графические системы. Базовая система, созданная Россом Ихакой (Ross Ihaka), уже включена в программу. Большинство диаграмм, обсуждаемых в предыдущих главах, созданы при помощи функций этой базовой графической системы.

Сеточная графическая система (`grid graphics system`), созданная Полом Мюррелом (Paul Murrell, 2006), реализована в пакете `grid`. Это альтернатива базовой графической системе на нижнем уровне. Пользователь может создавать произвольные прямоугольные области на графических устройствах, определять координатные системы для каждой области и использовать обширный набор изобразительных базовых элементов, для того чтобы контролировать устройство и облик графических объектов.

Подобная гибкость делает сеточную графическую систему ценным инструментом для разработчиков программного обеспечения. Однако в пакете `grid` нет функций для создания статистических диаграмм или графиков в окончательном виде. Поэтому данный пакет редко используется напрямую теми, кто анализирует данные.

Пакет `lattice`, созданный Дипаяном Саркаром (Deepayan Sarkar, 2008), реализует идеи категоризованных графиков, сформулированные Кливлендом (Cleveland, 1985, 1993) и описанные на специализированном сайте: <http://netlib.bell-abs.com/cm/ms/departments/sia/project/trellis/>. Пакет `lattice`, созданный при помощи пакета `grid`, теперь представляет собой больше, чем простую реализацию исходного подхода Кливленда к визуализации многомерных данных, и рассматривается сейчас как самодостаточная альтернативная система статистической графики в R.

В пакете *ggplot2*, созданном Хэдли Уикхэмом (Hadley Wickham, 2009a), реализована графическая система, которая основана на «графическом словаре», описанном в статье Уилкинсона (Wilkinson, 2005) и развитом в публикации Wickham (2009b). Цель этого пакета – предоставить всеобъемлющую основанную на определенной грамматике систему для построения диаграмм в единообразном и логическом стиле, что позволит пользователям создавать новые способы визуализации данных.

Способ доступа к этим четырем системам различается, как указано в табл. 16.1. Функции из базовой системы доступны автоматически. Для доступа к функциям пакетов *grid* и *lattice* вы должны вызывать эти пакеты в явном виде, например `library(lattice)`. Перед первым использованием функций пакета *ggplot2* нужно скачать его и установить: `install.packages("ggplot2")`, а затем уже вызвать: `library(ggplot2)`.

Таблица 16.1. Доступ к графическим системам

Система	Включена в базовую версию программы?	Должна быть вызвана в явном виде?
<code>base</code>	Да	Нет
<code>grid</code>	Да	Да
<code>lattice</code>	Да	Да
<code>ggplot2</code>	Нет	Да

Поскольку нас в основном интересуют практические аспекты анализа данных, в этой главе мы не будем детально рассматривать пакет *grid*. (Если вы интересуетесь этой темой, загляните на сайт доктора Мюррела (www.stat.auckland.ac.nz/~paul/grid/grid.html)). Вместо этого мы более подробно рассмотрим пакеты *lattice* и *ggplot2*. Каждый из них позволяет создать уникальные и полезные диаграммы, которые не так-то легко получить другими способами.

16.2. Пакет *lattice*

В пакете *lattice* реализована самодостаточная графическая система для визуализации одно- и многомерных данных. В частности, многие пользователи обращаются к этому пакету, поскольку он позволяет легко создавать категоризованные диаграммы.

Категоризованные диаграммы (называемые еще «графиками на решетке») показывают распределение одной переменной или взаимо-

связь между переменными отдельно для каждого уровня одной или более других переменных. Рассмотрим следующий вопрос: как рост членов нью-йоркского общества хорового пения варьирует в зависимости от их вокальных партий?

Данные о росте певцов и их вокальных партиях содержатся в наборе данных `singer` из пакета `lattice`. В следующем программном коде

```
library(lattice)
histogram(~height | voice.part, data = singer,
  main="Распределение значений роста исполнителей \\'разных
  вокальных партий',
  xlab="Рост (дюймы)")
```

`height` – это зависимая переменная, `voice.part` называется *задающей условие переменной* (*conditioning variable*), и гистограммы строятся отдельно для каждой из восьми вокальных партий. Полученная диаграмма представлена на рис. 16.1. Похоже, теноры и басы в среднем имеют более высокий рост, чем альты и сопрано.

Распределение значений роста исполнителей разных вокальных партий

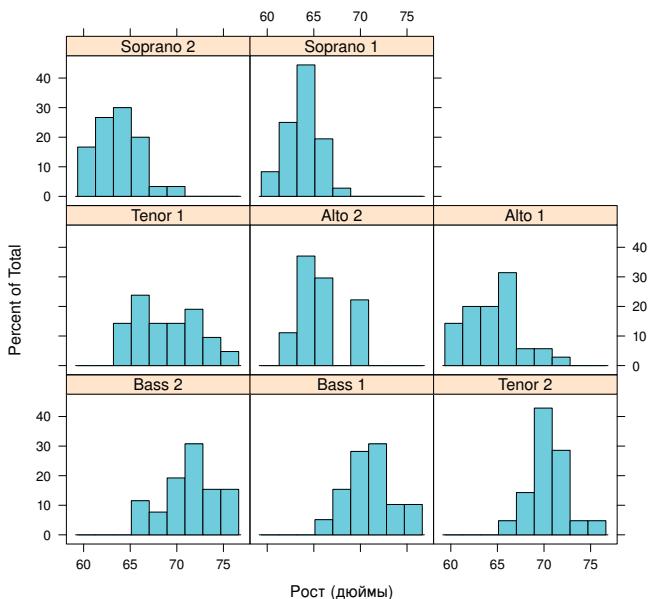


Рис. 16.1. Категоризованная диаграмма для роста исполнителей разных вокальных партий

На категоризованных диаграммах для каждого значения условной переменной создается своя *ячейка* (panel). Если используется более одной условной переменной, то отдельная ячейка создается для каждой комбинации значений факторов. Для облегчения сравнений ячейки объединяются в матрицу. Подпись в каждой ячейке находится в области, называемой *планкой* (strip). Как вы увидите позже, пользователь обретает полный контроль над диаграммами в каждой ячейке, форматом и расположением планок, порядком ячеек, расположением и содержанием легенды и многими другими графическими параметрами.

В пакете содержится много функций для создания одномерных (точечные диаграммы, диаграммы ядерной оценки функции плотности, гистограммы, столбчатые диаграммы, диаграммы размахов), двухмерных (диаграммы рассеяния, параллельные диаграммы рассеяния) и многомерных (3D-диаграммы, матрицы диаграмм рассеяния) диаграмм.

Каждая графическая функция более высокого уровня имеет вид
graph_function(*formula*, *data*=, *options*),

где

- *graph_function* – это одна из функций, указанных во втором столбце табл. 16.2;
- *formula* определяет переменные, которые нужно изобразить, наряду с задающими условия переменными;
- *data* – используемая таблица данных;
- *options* – разделяемые запятыми параметры, которые используются для изменения содержимого, расположения и аннотаций диаграммы. Наиболее распространенные параметры приведены в табл. 16.3.

Пусть строчные буквы соответствуют числовым переменным, а прописные буквы – категориальным переменным (факторам). Формула в графической функции высшего уровня обычно принимает вид

$$y \sim x | A * B,$$

где переменные слева от вертикальной черты называются *первичными*, а справа от черты – *условными*. Первичные переменные задают оси в каждой ячейке. В данном случае выражение $y \sim x$ описывает переменные, отложенные по вертикальной и горизонтальной осям соответственно. Для одномерных диаграмм это выражение заменяет

ся на $\sim x$. Для трехмерных диаграмм оно будет выглядеть как $z \sim x * y$. Наконец, для многомерных диаграмм (матрица диаграмм рассеяния или диаграмма параллельных координат) выражение $y \sim x$ заменяется на название таблицы данных. Учтите, что указывать условные переменные необязательно.

Следуя этой логике, $\sim x | A$ позволяет изобразить значения числовой переменной x при каждом значении фактора A . Выражение $y \sim x | A * B$ отображает связь между числовыми переменными y и x отдельно для каждой комбинации значений факторов A и B . Выражение $A \sim x$ отображает значения категориальной переменной A по вертикальной оси и числовой переменной x по горизонтальной. Выражение $\sim x$ отображает только значения числовой переменной x . Остальные примеры приведены в табл. 16.2.

Таблица 16.2. Типы диаграмм и соответствующие функции в пакете lattice

Тип диаграммы	Функция	Пример формулы
3D контурная диаграмма	contourplot()	$z \sim x * y$
3D уровневая диаграмма	levelplot()	$z \sim y * x$
3D диаграмма рассеяния	cloud()	$z \sim x * y A$
3D каркасная диаграмма	wireframe()	$z \sim y * x$
Столбчатая диаграмма	barchart()	$x \sim A$ ИЛИ $A \sim x$
Диаграммы рассеяния	bwplot()	$x \sim A$ ИЛИ $A \sim x$
Точечная диаграмма	dotplot()	$\sim x A$
Гистограмма	histogram()	$\sim x$
Диаграмма ядерной оценки функции плотности	densityplot()	$\sim x A * B$
Диаграмма параллельных координат	parallel()	таблица_данных
Диаграмма рассеяния	xyplot()	$y \sim x A$
Матрица диаграмм рассеяния	splom()	таблица_данных
Одномерная диаграмма рассеяния	stripplot()	$A \sim x$ ИЛИ $x \sim A$

Примечание: В приведенных формулах строчные буквы обозначают числовые переменные, а прописные – категориальные.

Таблица 16.3. Наиболее распространенные параметры графических функций высокого уровня в пакете lattice

Параметр	Описание
aspect	Число, определяющее соотношение размеров (высота/ширина) диаграммы в каждой ячейке
col, pch, lty, lwd	Векторы, которые определяют используемые на диаграммах цвета, символы, типы и ширину линий соответственно
groups	Группирующая переменная (фактор)
index.cond	Список, определяющий порядок расположения ячеек
key (или auto.key)	Функция, используемая для пояснения значений группирующей переменной (переменных)
layout	Числовой вектор из двух элементов, который определяет расположение ячеек (число колонок и число строк). При необходимости может быть введен третий элемент, указывающий число страниц
main, sub	Текстовые векторы, которые определяют заголовок и подзаголовок
panel	Функция, используемая для создания диаграммы в каждой ячейке
scales	Список, в котором содержится информация о разметке осей
strip	Функция, используемая для определения формата планок.
split, position	Числовые векторы, которые используются для того, чтобы расположить больше одной диаграммы на странице
type	Текстовый вектор, задающий один или более параметров диаграмм рассеяния (p = точки, l = линии, r = регрессионная линия, $smooth$ = сглаживание, g = сетка и т. д.)
xlab, ylab	Текстовые векторы, которые определяют подписи горизонтальных осей
xlim, ylim	Числовые векторы из двух элементов, которые определяют наименьшее и наибольшее значения по горизонтальной и вертикальной осям соответственно

Для того чтобы быстро получить общее представление о категоризованных диаграммах, попробуйте запустить программный код 16.1. Диаграммы основаны на данных об автомобилях (расход топлива, вес, число передач, число цилиндров и т. д.) из таблицы `mtcars`. Вы можете захотеть изменить формулы и посмотреть на результат. (Он здесь не приводится для экономии места.)

Программный код 16.1. Примеры категоризованных диаграмм

```

library(lattice)
attach(mtcars)

gear <- factor(gear, levels=c(3, 4, 5), 
               labels=c("3 передачи", "4 передачи", "5 передач"))
cyl <- factor(cyl, levels=c(4, 6, 8),
               labels=c("4 цилиндра", "6 цилиндра", "8 цилиндра"))

densityplot(~mpg,
            main="Диаграмма плотности распределения",
            xlab="Мили на галлон")

densityplot(~mpg | cyl,
            main="Диаграмма плотности распределения \n для разного числа
            цилиндров",
            xlab="Мили на галлон")

bwplot(cyl ~ mpg | gear,
       main="Диаграммы размахов для разного числа цилиндров и передач",
       xlab="Вес машины", ylab="Цилинды")

xyplot(mpg ~ wt | cyl * gear,
       main="Диаграммы рассеяния \n для разного числа цилиндров и передач",
       xlab="Вес машины", ylab="Мили на галлон")

cloud(mpg ~ wt * qsec | cyl,
      main="Трехмерные диаграммы рассеяния \n для разного числа цилиндров")

dotplot(cyl ~ mpg | gear,
        main="Точечные диаграммы \n для разного числа цилиндров и передач",
        xlab="Мили на галлон")

splom(mtcars[c(1, 3, 4, 5, 6)],
      main="Матрица диаграмм рассеяния для набора данных mtcars")

detach(mtcars)

```

Высокоуровневые графические функции в пакете *lattice* создают графические объекты, которые можно сохранять и видоизменять. Например, команда

```

library(lattice)
mygraph <- densityplot(~height|voice.part, data=singer)

```

создает категоризованную диаграмму плотности рассеяния и сохраняет ее в виде объекта *mygraph*. При этом на экран она не выводится. Ввод команды *plot (mygraph)* или просто *mygraph* позволит увидеть диаграмму.

**Создаем факторы
с подписями значений**

Категоризованные диаграммы легко видоизменять, указывая их параметры. Наиболее употребимые из них приведены в табл. 16.3. Эти параметры можно вводить в высокоуровневые функции или использовать в функциях, работающих с ячейками, которые описаны в разделе 16.2.2.

Для модификации графического объекта, созданного при помощи пакета *lattice*, можно также использовать функцию *update()*. Продолжая пример с певцами, команда

```
update(mygraph, col="red", pch=16, cex=.8, jitter=.05, lwd=2)
```

позволит нарисовать диаграмму заново, используя красные линии и символы (*color="red"*), заполненные цветом точки (*pch=16*), более мелкие (*cex=.8*) и сильнее смещенные друг относительно друга символы (*jitter=.05*), а также линии двойной толщины (*lwd=2*). Теперь, когда мы в общих чертах рассмотрели устройство высокоуровневых функций пакета, давайте подробнее поговорим об условных переменных.

16.2.1. Условные переменные

Как вы уже поняли, одна из наиболее впечатляющих особенностей категоризованных графиков – это возможность добавления условных переменных. Если есть одна условная переменная, для каждого ее значения создается отдельная ячейка. Если есть две условные переменные, отдельные ячейки создаются для каждого сочетания их значений. Использование более двух условных переменных редко бывает полезным.

Обычно условные переменные – это факторы. Но что, если вы хотите использовать непрерывную переменную в качестве условной? Одна из возможностей – это преобразовать непрерывную переменную в дискретную при помощи функции *cut()*. В качестве альтернативы можно использовать функции пакета *lattice*, которые преобразуют непрерывную переменную в объект типа *shingle* («галька»). В данном случае непрерывная переменная разделяется на ряд (возможно) перекрывающихся диапазонов значений. Например, функция

```
myshingle <- equal.count(x, number=#, overlap=proportion)
```

делит непрерывную переменную *x* на # интервалов с перекрытием *proportion* и одинаковым числом наблюдений в каждом интервале, сохраняя результат в виде объекта *myshingle* (класса *shingle*). Вызов этого объекта на экран: *plot(myshingle)* – позволит увидеть полученные интервалы.

Как только непрерывная переменная преобразуется в объект класса `shingle`, вы можете использовать ее в качестве условной переменной. Например, давайте исследуем связь между расходом топлива и весом машины у автомобилей с разным объемом двигателя, охарактеризованных в наборе данных `mtcars`. Поскольку объем двигателя – это непрерывная переменная, давайте сначала преобразуем ее в переменную с тремя уровнями:

```
displacement <- equal.count(mtcars$disp, number=3, overlap=0)
```

Затем используем полученную переменную как ввод для функции `xypplot()`:

```
xypplot(mpg~wt|displacement, data=mtcars,
  main = "Зависимость расхода топлива от веса автомобилей \nс
  разным объемом двигателя",
  xlab = "Вес", ylab = "Мили на галлон",
  layout=c(3, 1), aspect=1.5)
```

Результаты приведены на рис. 16.2. Обратите внимание на то, что мы также использовали параметры для изменения расположения ячеек (три столбца и одна строка) и соотношения их размеров (отношение высоты к ширине), чтобы упростить сравнение полученных трех групп данных.

**Зависимость расхода топлива от веса автомобилей
с разным объемом двигателя**

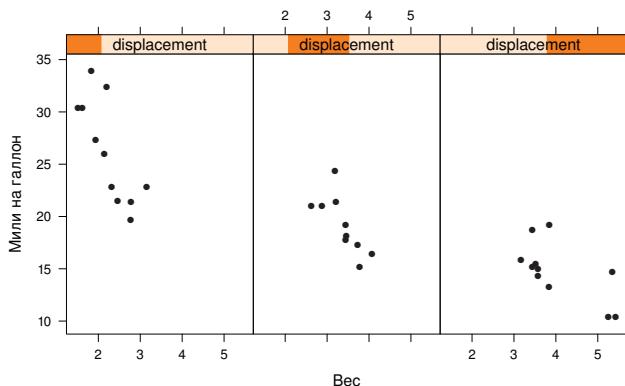


Рис. 16.2. Категоризованная диаграмма для зависимости расхода топлива и веса машины для разных объемов двигателей (`displacement`). Поскольку объем двигателя – это непрерывная переменная, она была разделена на три неперекрывающихся интервала с одинаковым числом значений в каждом

Можно заметить, что подписи на планках ячеек, приведенных на рис. 16.1 и рис. 16.2, различаются. Вид планок на рис. 16.2 указывает на непрерывность исходной условной переменной, диапазон ее значений в данной ячейке отмечен более темным цветом. В следующем разделе мы будем использовать специальные функции для изменения формата ячеек.

16.2.2. Функции для изменения формата ячеек

Каждая высокоуровневая функция в табл. 16.2 по умолчанию использует для создания ячеек определенную функцию. Эти функции имеют названия типа `panel.graph.function`,

где `graph.function` – это высокоуровневая функция. Например, функция

```
xyplot(mpg~wt|displacement, data=mtcars)
```

может также быть записана в виде

```
xyplot(mpg~wt|displacement, data=mtcars, panel=panel.xyplot)
```

Это заманчивая возможность, поскольку она позволяет заменять функцию, по умолчанию контролирующую формат ячеек, на вашу собственную. При создании этой функции вы можете использовать одну или несколько из более чем 50 встроенных функций пакета `lattice`, работающих с ячейками. Настройка функций для работы с ячейками предоставляет большие возможности для создания итоговой диаграммы, которая отвечает вашим потребностям. Рассмотрим некоторые примеры.

В предыдущем разделе мы рассмотрели зависимость расхода топлива от веса машины при разных объемах двигателя. Что, если вам понадобится добавить к этой диаграмме регрессионные линии, графики-щетки и координатную сетку? Это можно сделать, создав свою собственную функцию для управления форматом ячеек (см. приведенный ниже программный код). Полученная диаграмма представлена на рис. 16.3.

Программный код 16.2. Изменение формата ячеек для функции `xyplot`

```
displacement <- equal.count(mtcars$disp, number=3, overlap=0)
```

```
mypanel <- function(x, y) {  
  panel.xyplot(x, y, pch=19) } ◀➊ Функция, изменяющая  
параметры диаграммы
```

```

panel.rug(x, y)
panel.grid(h=-1, v=-1)
panel.lmline(x, y, col="red", lwd=1, lty=2)
}
xyplot(mpg~wt|displacement, data=mtcars,
       layout=c(3, 1),
       aspect=1.5,
       main = "Зависимость расхода топлива \nот веса автомобилей
       с разным объемом двигателя",
       xlab = "Вес",
       ylab = "Мили на галлон",
       panel = mypanel)

```

**Зависимость расхода топлива
от веса автомобилей с разным объемом двигателя**

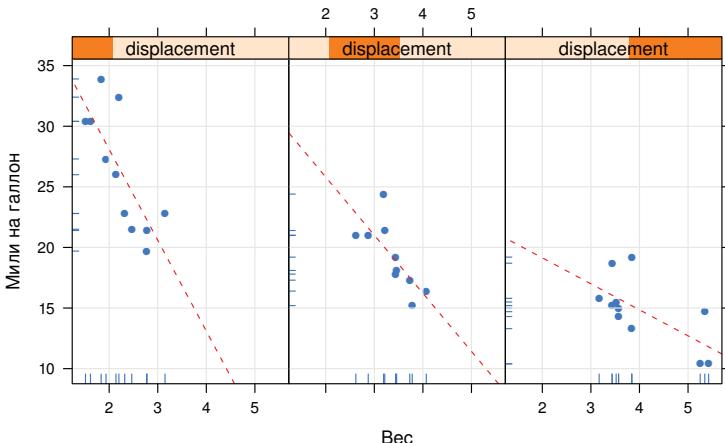


Рис. 16.3. Категоризованная диаграмма зависимости расхода топлива от веса машины для разного объема двигателя.

Для добавления регрессионных линий, графиков-щеток и координатной сетки была использована специально созданная функция изменения параметров ячеек

В данном случае мы объединили четыре функции в одной новой функции `mypanel()` и вызвали ее внутри функции `xyplot()` при помощи параметра `panel= ①`. Функция `panel.xyplot()` создает диаграмму рассеяния, на которой изображены заполненные кружочки (`pch=19`). Функция `panel.rug()` добавляет графики-щетки к `x` и `y` осям каждой ячейки. Функции `panel.rug(x, FALSE)` или `panel.rug(FALSE, y)` добавили бы графики щетки только к горизонтальной или вертикальной оси соответственно. Функция `panel.grid()`

добавляет горизонтальные и вертикальные линии координатной сетки (использование отрицательных чисел позволяет расположить эти линии напротив делений). Наконец, функция `panel.lmline()` добавляет регрессионную красную (`col="red"`) пунктирную (`lty=2`) линию стандартной толщины (`lwd=1`). Каждая функция, которая применяется по умолчанию для определения вида ячеек, имеет свою структуру и параметры. Прочтите страницу помощи для каждой из них, например `help(panel.abline)`, чтобы узнать больше.

В качестве второго примера мы графически отобразим связь между расходом топлива и объемом двигателя (в формате непрерывной переменной) для разных типов коробки передач. В дополнение к отдельному изображению этой связи для автоматической и ручной коробки передач мы добавим аппроксимирующие линии и горизонтальные линии, соответствующие средним значениям. Программный код приведен ниже.

Программный код 16.3. Изменение формата ячеек и дополнительные параметры функции `xyplot`

```
library(lattice)
mtcars$transmission <- factor(mtcars$am, levels=c(0,1),
                                labels=c("Автомат", "Ручная"))
panel.loess <- function(x, y) {
    panel.grid(h=-1, v=-1)
    panel.xyplot(x, y)
    panel.loess(x, y)
    panel.abline(h=mean(y), lwd=2, lty=2, col="green")
}
xyplot(mpg~disp|transmission, data=mtcars,
       scales=list(cex=.8, col="red"),
       panel=panel.loess,
       xlab="Объем двигателя", ylab="Мили на галлон",
       main="Зависимость расхода топлива от объема двигателя \n для
→ автомобилей с разными коробками передач",
       sub = "Среднее значение в каждой группе показано пунктирной
→ линией", aspect=1)
```

Диаграмма, полученная при помощи этих команд, представлена на рис. 16.4.

В этом новом коде стоит обратить внимание на несколько вещей. Функция `panel.xyplot()` изображает отдельные точки, а функция `panel.loess()` накладывает на точки аппроксимирующие линии. Функция `panel.abline()` добавляет на диаграмму горизонтальные линии, соответствующие средним значениям расхода топлива в каждой группе. (Если мы заменим выражение `h=mean(y)`

на `h=mean(mtcars$mpg)`, будет нарисована одна линия, соответствующая среднему значению в целой выборке.) Параметр `scales=list()` позволяет сделать деления осей красными и уменьшить их до 80% от размера по умолчанию.

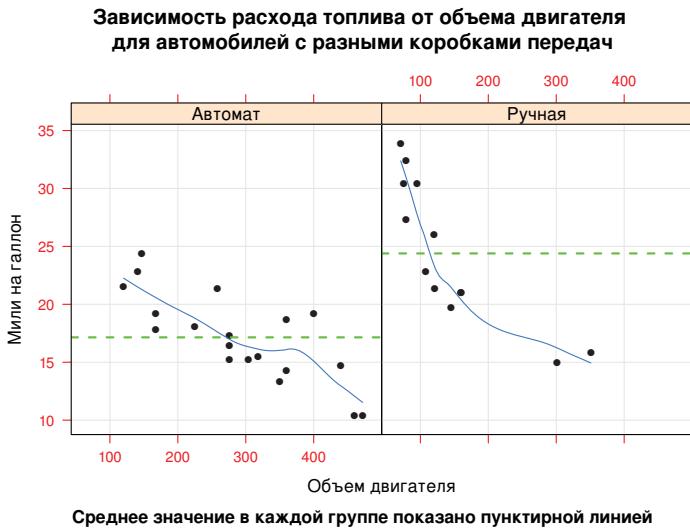


Рис. 16.4. Категоризованная диаграмма для зависимости расхода топлива от объема двигателя при разных типах коробки передач. Добавлены аппроксимирующие линии, координатные сетки и линии, соответствующие среднему значению в группе

В предыдущем примере мы могли бы использовать выражение `scales=list(x=list(), y=list())`, чтобы отдельно указать параметры горизонтальной и вертикальной осей. Введите `help(xyplot)`, чтобы получить подробную информацию о многих доступных для изменения параметрах осей. В следующем разделе вы узнаете, как представлять данные для разных групп объектов на одной диаграмме, вместо того чтобы изображать их в отдельных ячейках.

16.2.3. Группировка переменных

Когда вы включаете условную переменную в формулу графической функции пакета `lattice`, для каждого значения этой переменной создается отдельная ячейка. Если вы хотите вместо этого наложить результаты для разных групп данных друг на друга, можете обозначить эту переменную как группирующую.

Скажем, вам нужно отобразить распределение значений расхода топлива для автомобилей с ручной и автоматической коробкой передач на диаграмме ядерной оценки функции плотности. Вы можете наложить эти диаграммы при помощи следующих команд:

```
library(lattice)
mtcars$transmission <- factor(mtcars$am, levels=c(0, 1),
                                 labels=c("Автомат", "Ручная"))
densityplot(~mpg, data=mtcars,
            group=transmission,
            main="Расход топлива у автомобилей \nc разными коробками
            передач",
            xlab="Мили на галлон",
            auto.key=TRUE)
```

Полученная диаграмма представлена на рис. 16.5. По умолчанию опция `group=` позволяет наложить друг на друга диаграммы для всех уровней группирующей переменной. Наблюдения обозначаются незаполненными кружками, линии рисуются сплошными, а разным значениям группирующей переменной соответствуют разные цвета. Как вы можете видеть, цвета трудно различать при черно-белой печати. Позже вы узнаете, как изменить эти настройки по умолчанию.

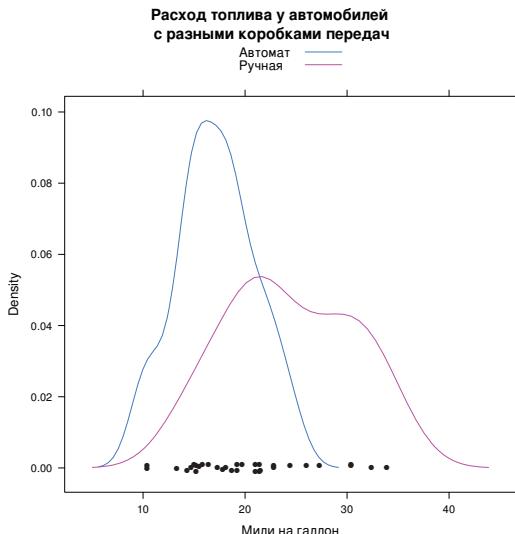


Рис. 16.5. Диаграммы ядерной оценки функции плотности значений расхода топлива (Miles per Gallon) для разных типов коробки передач. Смещенные относительно друг друга значения переменной изображены на горизонтальной оси

Обратите внимание на то, что условные обозначения не создаются автоматически. Параметр `auto.key=TRUE` позволяет поместить примитивную легенду сверху от диаграммы. Этую легенду можно немного изменять при помощи указания параметров в списке. Например, выражение

```
auto.key=list(space="right", columns=1, title="Тип коробки передач")  
позволит переместить легенду справа от диаграммы, перечислить значения группирующих переменных в одном столбце и добавить заголовок.
```

Если вы хотите обрести больший контроль над условными обозначениями, можете использовать параметр `key=`. Пример приведен в программном коде 16.4. Полученная диаграмма представлена на рис. 16.6.

Программный код 16.4. Диаграмма ядерной оценки функции плотности с группирующей переменной и улучшенной легендой

```
library(lattice)  
mtcars$transmission <- factor(mtcars$am, levels=c(0, 1),  
                                labels=c("Автомат", "Ручная"))  
  
colors = c("red", "blue")  
lines  = c(1,2)  
points = c(16,17)  
  
key.trans <- list(title="Тип коробки передач", space="bottom", columns=2,  
                    text=list(levels(mtcars$transmission)),  
                    points=list(pch=points, col=colors),  
                    lines=list(col=colors, lty=lines),  
                    cex.title=1, cex=.9)  
  
densityplot(~mpg, data=mtcars,  
            group=transmission,  
            main="Расход топлива у машин с разными коробками передач",  
            xlab="Мили на галлон",  
            pch=points, lty=lines, col=colors, lwd=2, jitter=.005,  
            key=key.trans)
```

◀—❶ Назначение цветов, типов линий и символов
❷ Изменение параметров легенды

❸ Изменение параметров диаграммы плотности распределения

В данном случае типы значков и линий, а также цвета заданы в виде векторов ❶. Первый элемент каждого вектора соответствует первому значению группирующей переменной, второй элемент – второму значению и т. д. Параметры легенды указаны в виде списка ❷. Эти параметры позволяют расположить легенду внизу диаграммы в виде

двух столбцов, включающих расшифровку значений переменной, типы значков и линий, а также цвета. Заголовок легенды напечатан немного крупнее, чем расшифровка значений переменной.

Те же самые типы символов и линий, а также цвета заданы в качестве параметров функции `densityplot()` ❸. Вдобавок для улучшения вида диаграммы увеличены ширина линий и степень смещения символов друг относительно друга. В завершение указано, что легенда должна быть основана на ранее созданном списке. Такой подход к созданию условных обозначений для группирующей переменной открывает большие возможности. В действительности вы можете создавать более одной легенды и размещать их в разных областях диаграммы (не показано).

Расход топлива у машин с разными коробками передач

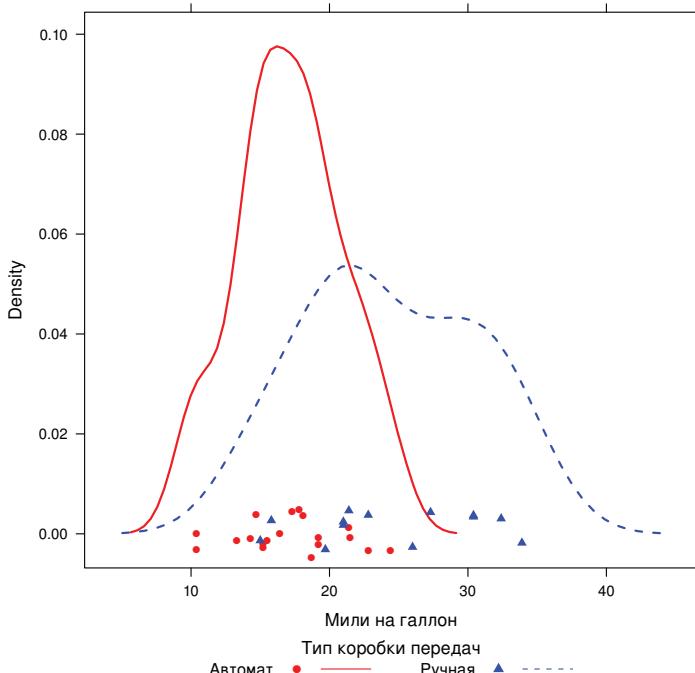


Рис. 16.6. Диаграммы ядерной оценки функции плотности значений расхода топлива для разных типов коробки передач.

Графические параметры были видоизменены, добавлена улучшенная легенда, в которой указаны цвет и форма символов, тип линий, размер текста и заголовок

Прежде чем завершить этот раздел, рассмотрим пример одновременного применения группирующей и условной переменной. Набор данных CO2, поставляющийся с базовой версией программы, описывает исследование холдоустойчивости злака ежовника обыкновенного (*Echinochloa crus-galli*).

В наборе данных содержится информация об интенсивности поглощения углекислого газа (переменная `uptake`) у 12 растений (`Plant`) при семи уровнях концентрации двуокиси углерода в окружающей среде (`conc`). Шесть растений были из Квебека, а шесть – из Миссисипи. По три растения из каждого географического района содержались в условиях пониженной температуры. В данном примере `Plant` – это группирующая переменная, а `Treatment` (охлажденные/неохлажденные) – условные. Следующий программный код позволяет получить диаграмму, представленную на рис. 16.7.

Программный код 16.5. Применение функции `xypplot` с группирующей и условными переменными и улучшенной легендой

```
library(lattice)
colors <- "darkgreen"
symbols <- c(1:12)
linetype <- c(1:3)
key.species <- list(title="Раст.",
                      space="right",
                      text=list(levels(CO2$Plant)),
                      points=list(pch=symbols, col=colors))
xypplot(uptake~conc|Type*Treatment, data=CO2,
        group=Plant,
        type="o",
        pch=symbols, col=colors, lty=linetype,
        main="Поглощение двуокиси углерода злаками",
        ylab=expression(paste("Поглощениe",
                               bgroup("(, italic(frac(\"ммоль\", \"м\"^2)), \"))"),
        xlab=expression(paste("Концентрация ",
                               bgroup("(, italic(frac(мл, л)), \"))")),
        sub = "Вид злака: Echinochloa crus-galli",
        key=key.species)
```

Обратите внимание на использование символа `\n` для создания заголовка из двух строк и функции `expression()` для добавления математических обозначений в подписи по осям. В данном случае обозначение разных значений группирующей переменной разными цветами не применяется – в параметре `col=` указан один цвет. Здесь 12 разных цветов были бы избыточными и не позволяли бы достичь цели наглядного изображения связей между переменными в каждой

ячейке. Хорошо видно, что охлажденные злаки из Миссисипи отличаются от остальных.

Поглощение двуокиси углерода злаками

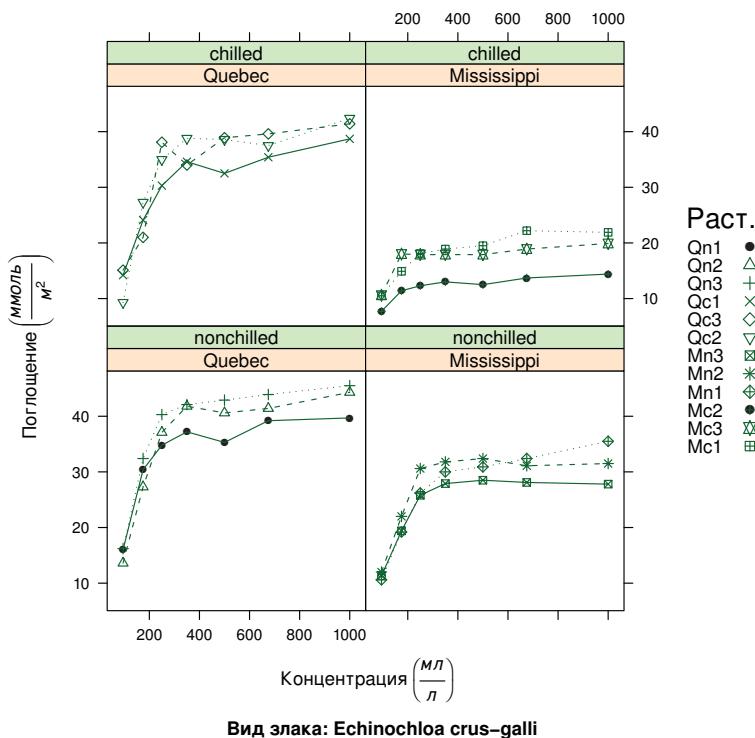


Рис. 16.7. Графики, которые демонстрируют влияние концентрации углекислого газа в окружающей среде на его потребление 12 растениями при разных экспериментальных условиях и в разных регионах. `Plant` – это группирующая переменная, а `Type` и `Treatment` – условные

До этого момента вы изменяли графические элементы при помощи параметров или высокоровневых функций, например `xypplot(pch=17)`, или используемых ими функций для изменения характеристик ячеек, например `panel.xypplot(pch=17)`. Однако такие изменения действуют только во время выполнения данной функции. В следующем разделе мы познакомимся со способами изменения графических параметров на все время интерактивной сессии или запуска программы в пакетном режиме.

16.2.4. Графические параметры

В третьей главе вы узнали, как просматривать и устанавливать графические параметры при помощи функции `par()`. Хоть это и работает для диаграмм, созданных в базовой графической системе R, на графику пакета `lattice` такой подход не действует. Вместо этого графические параметры, используемые в этом пакете по умолчанию, можно вызвать при помощи функции `trellis.par.get()` и изменить командой `trellis.par.set()`. Для визуального представления текущих графических параметров можно воспользоваться функцией `show.settings()`.

Давайте для примера изменим тип символов, которые используются при наложении нескольких групп данных (то есть точек на диаграмме с группирующей переменной). По умолчанию это незаполненный кружок. Вместо этого мы назначим свой символ для каждой группы.

Сначала выведем на экран текущие настройки по умолчанию и сохраним их в виде списка с именем `mysettings`:

```
show.settings()  
mysettings <- trellis.par.get()
```

Затем выведем на экран значения настроек по умолчанию, которые относятся к наложенным символам:

```
> mysettings$superpose.symbol  
$alpha  
[1] 1 1 1 1 1 1  
$cex  
[1] 0.8 0.8 0.8 0.8 0.8 0.8  
$col  
[1] "#0080ff"  "#ff00ff"  "darkgreen" "#ff0000"  "orange"    "#00ff00"  
[7] "brown"  
$fill  
[1] "#CCFFFF"  "#FFCCFF"  "#CCFFCC"  "#FFE5CC"  "#CCE6FF"  "#FFFFCC"  "#FFCCCC"  
$font  
[1] 1 1 1 1 1 1  
$pch  
[1] 1 1 1 1 1 1
```

Отсюда ясно, что для каждого уровня группирующей переменной используется символ в виде незаполненного кружочка (`pch=1`). Заданы семь уровней, после чего типы символов используются повторно.

Наконец, введем следующие команды:

```
mysettings$superpose.symbol$pch <- c(1:10)  
trellis.par.set(mysettings)  
show.settings()
```

Теперь в диаграммах пакета *lattice* символ 1 (незаполненный кружок) будет использоваться для первого значения группирующей переменной, символ 2 (незаполненный треугольник) – для второго и т. д. Кроме того, разные символы определены для 10 значений группирующей переменной, а не для семи. Произведенные изменения будут действовать, пока все графические устройства не будут закрыты. Таким образом можно изменить любой графический параметр.

16.2.5. Расположение диаграмм на странице

В третьей главе вы узнали, как размещать больше двух диаграмм на одной странице при помощи функции *par()*. Поскольку этот подход не действует на функции из пакета *lattice*, нам нужен другой способ. Самое простое – это сохранить диаграммы пакета *lattice* как объекты, а затем использовать функцию *plot()* с определением параметров *split=* или *position=*.

Параметр *split* разделяет страницу на заданное число строк и столбцов, размещая диаграммы в заданные ячейки получившейся матрицы. Формат применения этой опции таков:

```
split=c(нужная строка, нужный столбец,  
общее число строк, общее число столбцов)
```

К примеру, этот программный код

```
library(lattice)  
graph1 <- histogram(~height|voice.part,  
                      main="Рост исполнителей разных вокальных партий  
в хоре")  
graph2 <- densityplot(~height, data=singer, group=voice.part,  
                      plot.points=FALSE, auto.key=list(columns=4))  
plot(graph1, split=c(1, 1, 1, 2))  
plot(graph2, split=c(1, 2, 1, 2), newpage=FALSE)
```

позволяет разместить первую диаграмму сверху от второй. В частности, первая команда *plot()* разделяет страницу на один столбец и две строки, размещая диаграмму в первом (единственном) столбце и первой строке (считая сверху вниз и слева направо). Вторая команда *plot()* разделяет страницу таким же образом, но размещает диаграмму во второй строке. Поскольку по умолчанию функция *plot()* открывает новую страницу, вы предотвращаете это при помощи параметра *newpage=FALSE*. (Я не привожу диаграмму для экономии места.)

Больший контроль над размером диаграмм и их расположением можно обрести при помощи параметра `position=`. Рассмотрим следующий программный код:

```
library(lattice)
graph1 <- histogram(~height|voice.part, data=singer,
                     main="Рост исполнителей разных вокальных
                     партий в хоре")
graph2 <- densityplot(~height, data=singer, group=voice.part,
                      plot.points=FALSE, auto.key=list(columns=4))
plot(graph1, position=c(0, .3, 1, 1))
plot(graph2, position=c(0, 0, 1, .3), newpage=FALSE)
```

В данном случае параметр `position=` определяет значения с (`xmin`, `ymin`, `xmax`, `ymax`), где для каждой страницы применяется прямоугольная система координат (от 0 до 1 по осям абсцисс и ординат), где левый нижний угол имеет координаты (0, 0). (Диаграмма опять не приведена для экономии места.)

В категоризованных диаграммах можно также изменять порядок ячеек. Параметр `index.cond=` определяет последовательность уровней условной переменной в высокоуровневых функциях категоризованных графиков. Уровни фактора `voice.part` таковы:

```
> levels(singer$voice.part)
[1] "Bass 2"    "Bass 1"    "Tenor 2"   "Tenor 1"   "Alto 2"    "Alto 1"
[7] "Soprano 2" "Soprano 1"
```

Добавление параметра `index.cond=list(c(2, 4, 6, 8, 1, 3, 5, 7))` позволит сначала указать все первые партии, а потом – все вторые. Если у вас есть две условные переменные, включите в список два вектора. Если в программный код 16.5 добавить параметр `index.cond=list(c(1, 2), c(2, 1))`, то порядок экспериментальных воздействий на рис. 16.7 изменится на противоположный.

Для того чтобы узнать про категоризованные диаграммы больше, просмотрите на прекрасный текст Sarkar (2008) и соответствующий сайт <http://lmdvtr.r-forge.r-project.org>. Руководство по категоризованным графикам (<http://cm.bell-labs.com/cm/ms/departments/sia/doc/trellis.user.pdf>) также представляет собой отличный источник информации.

В следующем разделе мы познакомимся со второй мощной альтернативой базовой графической системой R. Она основана на пакете `ggplot2`.

16.3. Пакет `ggplot2`

Пакет `ggplot2` представляет собой систему для построения графиков в R, основанную на целостной и логичной грамматике. Это позволяет

сделать графические построения последовательными, чего часто недостает R, и дает возможность пользователям создавать новые типы диаграмм.

Простейший подход к созданию диаграмм в пакете `ggplot2` – это использование функции быстрого построения диаграмм `qplot()`¹. Формат ее применения таков:

```
qplot(x, y, data=, color=, shape=, size=, alpha=, geom=, method=,
      formula=, facets=, xlim=, ylim=, xlab=, ylab=, main=, sub=)
```

Все упомянутые параметры описаны в табл. 16.4.

Таблица 16.4. Параметры функции `qplot()`

Параметр	Описание
<code>alpha</code>	Коэффициент прозрачности для перекрывающихся элементов, принимающий значения от 0 (абсолютная прозрачность) до 1 (абсолютная непрозрачность)
<code>color</code> , <code>shape</code> , <code>size</code> , <code>fill</code>	Определяет соответствие цвета, формы и размера символов уровням переменной. Для диаграмм плотности и размахов параметр <code>fill</code> определяет соответствие заполняющего цвета значениям переменной. Условные обозначения отображаются автоматически
<code>data</code>	Назначает таблицу данных
<code>facets</code>	Создает категоризованную диаграмму, определяя условные переменные. Этот параметр принимает значения формата <code>rowvar ~ colvar</code> (пример приведен на рис. 16.10). Для построения категоризованной диаграммы только с одной условной переменной используйте выражения <code>rowvar~.</code> или <code>~colvar</code>
<code>geom</code>	Определяет геометрические объекты, которые задают тип диаграммы. Этот параметр задается в формате текстового вектора с одним или более элементами: "point", "smooth", "boxplot", "line", "histogram", "density", "bar" и "jitter"
<code>main</code> , <code>sub</code>	Текстовые векторы, определяющие заголовок и подзаголовок.

¹ От англ. *quick plot* – быстрая диаграмма. – Прим. пер.

Параметр	Описание
method, formula	<p>Если <code>geom="smooth"</code>, то к диаграмме по умолчанию добавляются аппроксимирующая линия (loess) и доверительные интервалы. Если число наблюдений превышает 1000, то применяется более эффективный сглаживающий алгоритм. Можно использовать следующие методы сглаживания (параметр <code>formula</code>): <code>"lm"</code> (регрессия), <code>"gam"</code> (обобщенные аддитивные модели) и <code>"rlm"</code> (устойчивая регрессия).</p> <p>Например, для добавления к диаграмме линии простой регрессии нужно использовать выражение <code>geom="smooth", method="lm", formula=y~x</code>. Если изменить формулу на <code>y~poly(x, 2)</code>, появится аппроксимирующая линия в виде полинома второй степени. Обратите внимание, что в параметре <code>formula</code> используются буквы <code>x</code> и <code>y</code>, а не названия переменных.</p> <p>Перед использованием параметра <code>method="gam"</code> не забудьте загрузить пакет <code>mgcv</code>. Для использования параметра <code>method="rlm"</code> необходим пакет <code>MASS</code>.</p>
<code>x, y</code>	Определяют переменные, значения которых будут отложены по горизонтальной и вертикальной осям. Для одномерных диаграмм (например, гистограммы) укажите только <code>x</code>
<code>xlab, ylab</code>	Текстовые векторы, определяющие подписи по горизонтальной и вертикальной осям
<code>xlim, ylim</code>	Числовые двухэлементные векторы, которые определяют наибольшее и наименьшее значения горизонтальной и вертикальной осей соответственно

Для того чтобы понять, как работает функция `qplot()`, давайте рассмотрим несколько примеров. Приведенный ниже программный код позволяет создать диаграммы размахов для расхода топлива автомобилями с разным числом цилиндров. Точки, которые соответствуют реальным данным, наложены на диаграмму (и смешены друг относительно друга, чтобы уменьшить перекрытие). Разные цвета диаграмм размахов соответствуют разному числу цилиндров.

```
library(ggplot2)
mtcars$cylinder <- as.factor(mtcars$cyl)
qplot(cylinder, mpg, data=mtcars, geom=c("boxplot", "jitter"),
```

```
fill=cylinder,
main="Диаграммы размахов и исходные данные",
xlab="Число цилиндров",
ylab="Мили на галлон")
```

Диаграмма представлена на рис. 16.8.

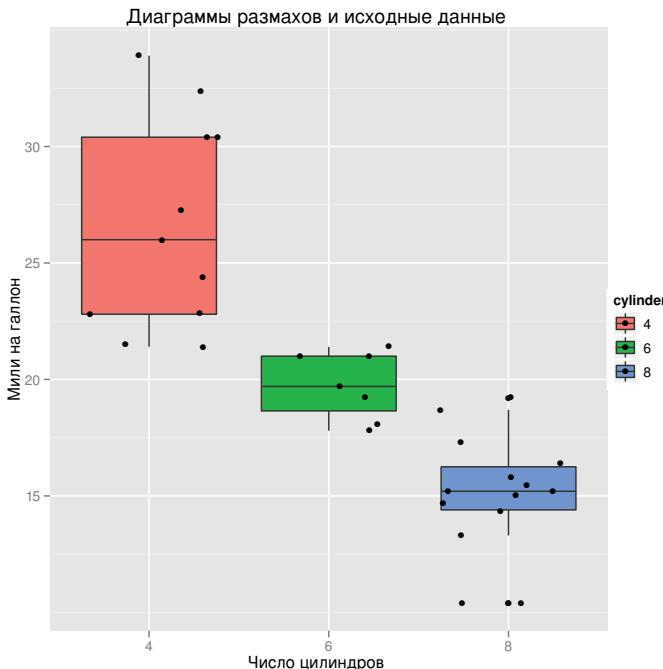


Рис. 16.8. Диаграммы размахов для значений расхода топлива автомобилями с разным числом цилиндров. На диаграмму наложены смещенные относительно друг от друга точки, которые отражают реальные данные

В качестве следующего примера построим диаграмму рассеяния для зависимости расхода топлива от веса машины, обозначив автомобили с разным типом коробки передач разным цветом и формой символов. Кроме того, добавим отдельные регрессионные линии и доверительный интервал для каждого типа коробки передач.

```
library(ggplot2)
transmission <- factor(mtcars$am, levels=c(0, 1),
                       labels=c("Автомат", "Ручная"))
qplot(wt, mpg, data=mtcars,
      color=transmission, shape=transmission,
```

```
geom="point", "smooth"),
method="lm", formula=y~x,
xlab="Вес",
ylab="Мили на галлон",
main="Пример с регрессией")
```

Полученная диаграмма представлена на рис. 16.9. Это полезный тип диаграммы, который непросто создать при помощи других пакетов.

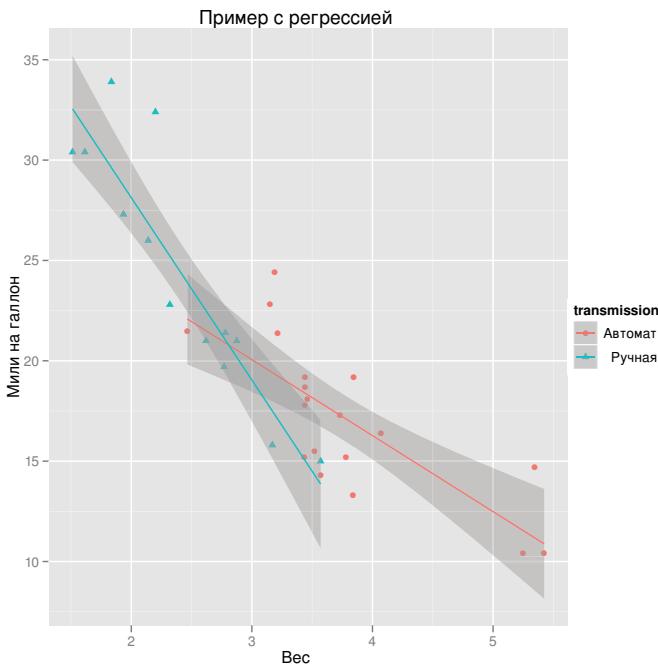


Рис. 16.9. Диаграмма рассеяния для расхода топлива и веса автомобиля с отдельными линиями регрессии и доверительными интервалами для машин с разным типом коробки передач

В третьем примере мы создадим категоризованную диаграмму. В каждой ее ячейке будет размещена отдельная диаграмма рассеяния для расхода топлива и веса машины. Строки определяются типом коробки передач, а столбцы — числом цилиндров. Размер символов соответствует классу мощности автомобиля.

```
library(ggplot2)
mtcars$cyl <- factor(mtcars$cyl, levels=c(4, 6, 8),
                      labels=c("4 цилиндра", "6 цилиндров", "8 цилиндров"))
mtcars$am <- factor(mtcars$am, levels=c(0, 1),
```

```
labels=c("Автомат", "Ручная"))
qplot(wt,mpg, data=mtcars, facets=am~cyl, size=hp)
```

Полученная диаграмма представлена на рис. 16.10. Обратите внимание, как просто создать эту пузырьковую диаграмму. Вы можете попробовать добавить параметры формы и цвета символов и посмотреть, как это повлияет на получившуюся диаграмму.

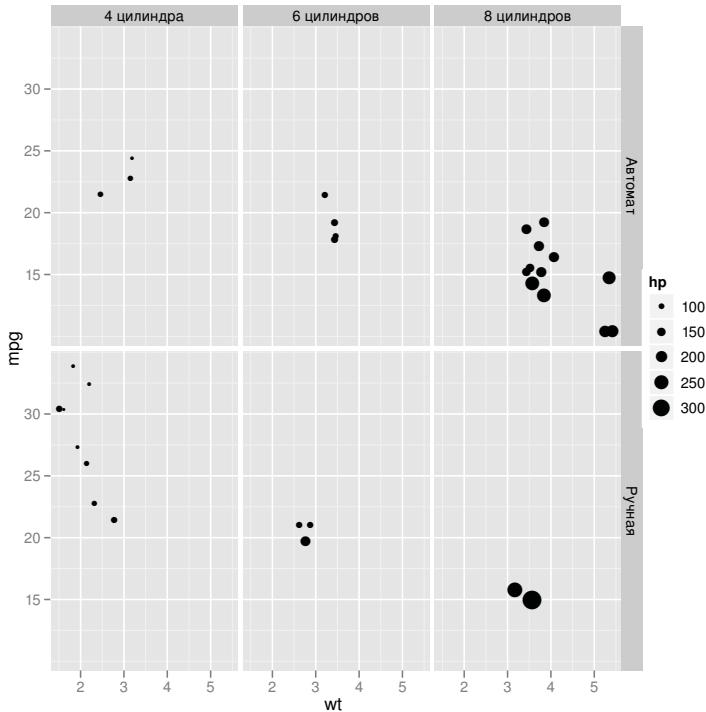


Рис. 16.10. Диаграмма рассеяния для расхода топлива и веса машин с разным типом коробки передач (ручная – manual и автоматическая – automatic) и числом цилиндров (4, 6 или 8). Размер символов соответствует мощности двигателя

В завершение этого раздела мы вернемся к набору данных `singer`, с которого мы начали эту главу. Приведенный ниже программный код позволяет построить диаграмму, отображенную на рис. 16.11.

```
library(ggplot2)
data(singer, package="lattice")
qplot(height, data=singer, geom=c("density"))
  facets=voice.part~, fill=voice.part)
```

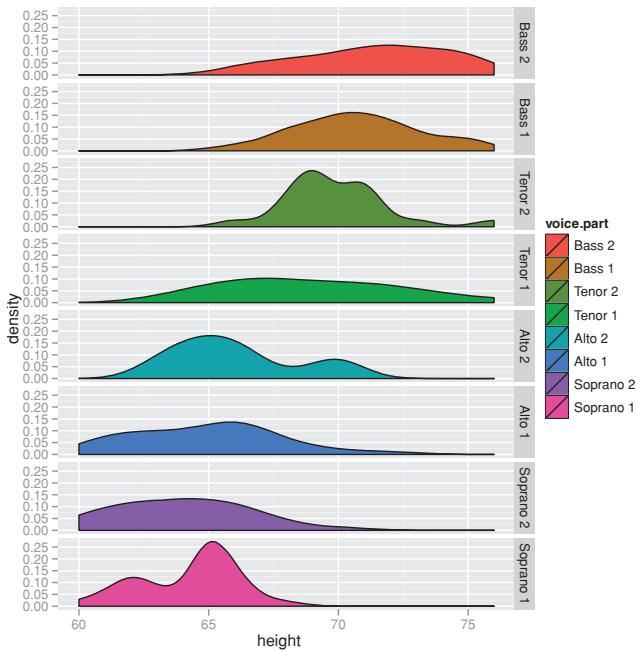


Рис. 16.11. Диаграммы плотности распределения значений роста исполнителей разных вокальных партий

Сравнивать распределения значений роста на этой диаграмме легче, чем в том виде, в каком эти данные были представлены на рис. 16.1.

Мы лишь поверхностно познакомились с этой мощной графической системой. Для получения дополнительной информации мы отсылаем заинтересованных читателей к публикации Wickham (2009) и к сайту, посвященному пакету `ggplot2` (<http://had.co.nz/ggplot2/>). Мы закончим эту главу обзором интерактивной графики и функций R, которые поддерживают ее.

16.4. Интерактивная графика

В базовой версии R реализовано мало возможностей интерактивного взаимодействия с диаграммами. Вы можете настраивать диаграммы при помощи дополнительных графических параметров, однако при помощи мыши почти невозможно изменить готовую диаграмму или считать с нее новую информацию. К счастью, существуют дополнительные пакеты, которые значительно расширяют возможности взаим-

модействия с создаваемыми вами диаграммами. В этом разделе мы сосредоточимся на функциях пакетов `playwith`, `latticeist`, `ipplots` и `rggobi`. Не забудьте установить их перед первым использованием.

16.4.1. Взаимодействие с диаграммами: идентификация точек

Перед тем, как перейти к специализированным пакетам, рассмотримвшую в базовую версию программы функцию `identify()`, при помощи которой на диаграммах рассеяния можно обозначать точки номером или названием их строки². Например, после введения команд

```
plot(mtcars$wt, mtcars$mpg)
identify(mtcars$wt, mtcars$mpg, labels=row.names(mtcars))
```

форма курсора изменяется со стрелки на крестик. Щелчок на точку диаграммы рассеяния будет добавлять к ней подпись, пока вы не выберете **Stop** в меню графического устройства или в контекстном меню, которое появляется после нажатия правой клавиши мыши³.

Многие графические функции из дополнительных пакетов (включая функции пакета `cag`, которые обсуждались в главе 8) используют этот метод обозначения точек. К сожалению, функцию `identify()` нельзя применить к диаграммам, построенным при помощи пакетов `lattice` или `ggplot2`.

16.4.2. Пакет `playwith`

В пакете `playwith` используется графический пользовательский интерфейс GTK+, в котором можно изменять диаграммы и взаимодействовать с ними в интерактивном режиме. Этот пакет можно установить на любой операционной системе при помощи команды `install.packages("playwith", depend=TRUE)`. В операционных системах Mac OS X и Linux лучше всего дополнительно установить графический пользовательский интерфейс JGR (см. приложение А) и загружать пакет `playwith` из него.

Функция `playwith()` позволяет обозначать и подписывать точки, просматривать значения всех переменных для данного наблюдения, изменять масштаб и кадрировать, добавлять подписи (текст, стрелки,

2 Можна также обозначать точки соответствующими значениями любой другой переменной. – Прим. пер.

3 Нажатие клавиши **Esc** также приведет к остановке интерактивного режима. – Прим. пер.

линии, прямоугольники, названия, подписи), изменять графические элементы (цвета, размер шрифта и т. д.), применять сохраненные ранее стили и сохранять полученные диаграммы в самых разных форматах. Это просто показать на примере. После введения этих команд

```
library(playwith)
library(lattice)
playwith(
  xyplot(mpg~wt|factor(cyl)*factor(am),
         data=mtcars, subscripts=TRUE,
         type=c("r", "p"))
)
```

на экране появляется окно, изображенное на рис. 16.12. Попробуйте по-нажимать кнопки слева и выбрать разные пункты меню. Этот графический пользовательский интерфейс во многом интуитивен. В отличие от функции `identify()`, команда `playwith()` применима к диаграммам, созданным в пакетах `lattice` и `ggplot2`, наряду с базовой графической системой. Некоторые опции тематического (Theme) меню правильно взаимодействуют только с базовой графической системой. Кроме того, некоторые функции работают с `ggplot2` графикой (например, аннотирование), а некоторые (такие как идентификация точек) – нет.

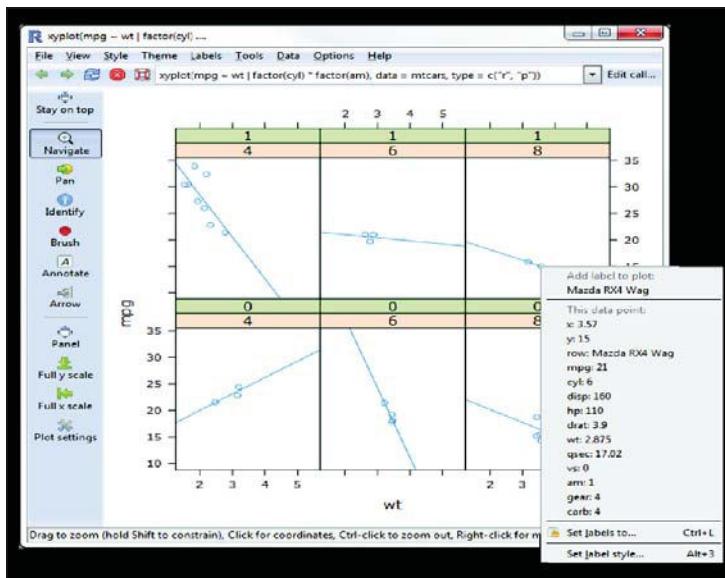


Рис. 16.12. Графический интерфейс GTK+ пакета `playwith`. В этом интерфейсе пользователь может изменять диаграмму при помощи мыши

Чтобы больше узнать о пакете `playwith`, зайдите на сайт этого проекта: <http://code.google.com/p/playwith/>.

16.4.3. Пакет *latticist*

При помощи пакета *latticist* можно работать с данными в формате категоризованных диаграмм. В этом пакете реализован графический пользовательский интерфейс для диаграмм, описанных в разделе 16.2, но также можно работать с диаграммами, полученными при помощи пакета *vcd* (см. раздел 11.4 из главы 11). При необходимости пакет *latticist* можно интегрировать в пакет *playwith*. К примеру, выполнение следующих команд

```
library(latticist)
mtcars$cyl <- factor(mtcars$cyl)
mtcars$gear <- factor(mtcars$gear)
latticist(mtcars, use.playwith=TRUE)
```

вызовет интерфейс, показанный на рис. 16.13.

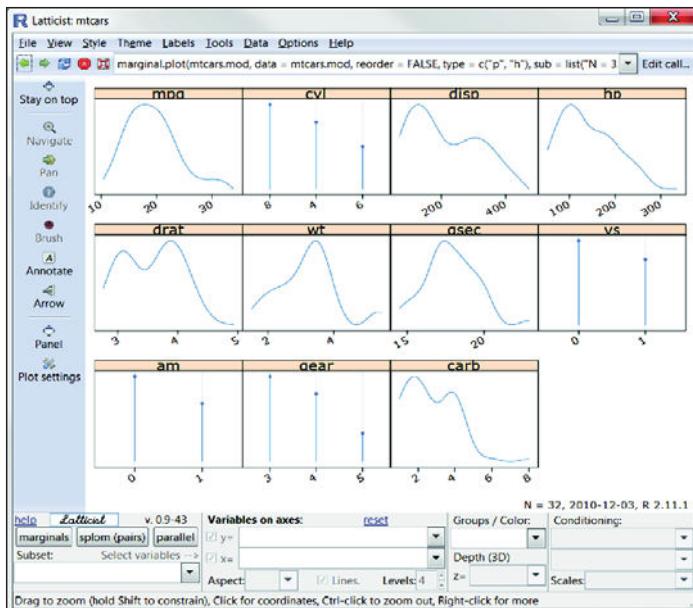


Рис. 16.13. Графический интерфейс пакета *playwith* с функциональными возможностями, реализованными в пакете *latticist*. Пользователи могут создавать диаграммы из пакетов *lattice* и *vcd* в интерактивном режиме

В дополнение к возможностям пакета (идентификация точек, аннотация, изменение масштаба, кадрирование, стили) теперь пользователь может создавать категоризованные диаграммы при помощи выпадающих меню и кнопок. Чтобы узнать больше о пакете *latticist*, зайдите на сайт <http://code.google.com/p/latticist/>.

Сходный графический пользовательский интерфейс под названием Deducer доступен и для пакета *ggplot2* при помощи вспомогательной программы Plot Builder (см. приложение А). Мы не обсуждаем его здесь, поскольку этот интерфейс нельзя запустить из консоли R. Если вас интересует эта тема, зайдите на сайт Deducer: www.deducer.org.

16.4.4. Создание интерактивной графики при помощи пакета *iplots*

В то время как пакеты *latticist* и *playwith* позволяют вам взаимодействовать в интерактивном режиме с одной диаграммой, пакет *iplots* трактует интерактивность иначе. В этом пакете создаются интерактивные мозаичные, столбчатые, параллельные диаграммы, диаграммы размахов, диаграммы рассеяния и гистограммы, которые можно объединять и раскрашивать. Это означает, что вы можете выбирать и обозначать наблюдения при помощи мыши; при этом выделение наблюдения на одной диаграмме автоматически приведет к выделению этого наблюдения на всех остальных открытых диаграммах. Вы также можете использовать мышь для получения информации о таких графических объектах, как точки, столбики, линии и диаграммы размахов.

Пакет *iplots* работает на основе языка Java, основные функции приведены в табл. 16.5.

Таблица 16.5. Функции пакета *iplots*

Функция	Описание
<code>iбар()</code>	Интерактивная столбчатая диаграмма
<code>ibox()</code>	Интерактивные диаграммы размахов
<code>ihist()</code>	Интерактивная гистограмма
<code>imap()</code>	Интерактивная карта
<code>imosaic()</code>	Интерактивная мозаичная диаграмма
<code>iprcp()</code>	Интерактивная диаграмма параллельных координат
<code>iplot()</code>	Интерактивная диаграмма рассеяния

Для того чтобы понять, как работает пакет *iplots*, запустите программный код 16.6.

Программный код 16.6. Демонстрация работы пакета *iplots*

```
library(iplots)
attach(mtcars)
cylinders <- factor(cyl)
gears <- factor(gear)
transmission <- factor(am)
ihist(mpg)
ibar(gears)
iplot(mpg, wt)
ibox(mtcars[c("mpg", "wt", "qsec", "disp", "hp")])
ipcp(mtcars[c("mpg", "wt", "qsec", "disp", "hp")])
imosaic(transmission, cylinders)
detach(mtcars)
```

Откроются шесть окон с диаграммами. Расположите их на рабочем пространстве так, чтобы каждая из них была видна (при необходимости можно изменять размер каждой диаграммы). Часть рабочего пространства показана на рис. 16.14.

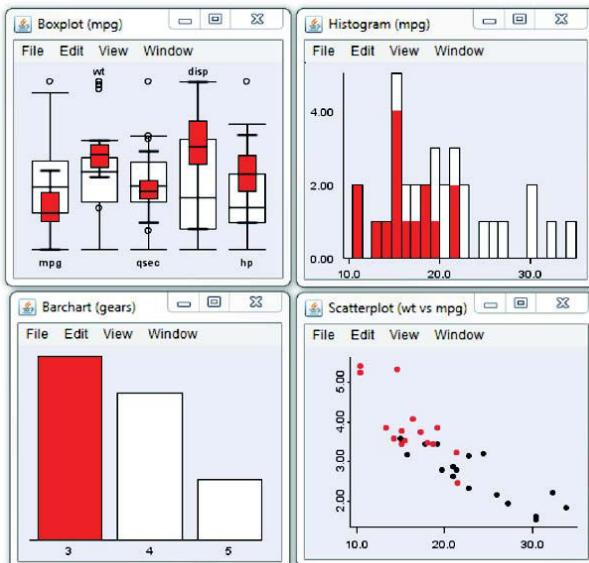


Рис. 16.14. Пример работы в пакете *iplots*, созданный при помощи программного кода 16.6. Для экономии места представлены лишь четыре из шести окон. На этих диаграммах пользователь щелкнул мышью по столбцу «три передачи» столбчатой диаграммы в левом нижнем окне

Теперь попробуйте сделать следующее:

- Щелкните мышью по столбцу, соответствующему машинам с тремя скоростями, в окне столбчатой диаграммы (Barchart). Столбец будет выделен красным цветом. В дополнение все элементы, соответствующие автомобилям с тремя скоростями на остальных диаграммах, тоже будут выделены.
- Выделите при помощи курсора прямоугольную область на диаграмме рассеяния (Scatterplot). Эти точки будут выделены цветом, и соответствующие наблюдения на других диаграммах тоже будут обозначены красным цветом.
- Подведите курсор к точке, столбцу, диаграмме размахов или линии на любой из диаграмм, нажав клавишу **Ctrl**. Информация о данном объекте появится во всплывающем окне.
- Нажмите правую кнопку мыши, подведя курсор к любому графическому объекту, и посмотрите, какие опции будут предложены в контекстном меню. Например, вы можете, щелкнув по диаграмме размахов, заменить ее на диаграмму параллельных координат.
- Вы можете выделить несколько объектов (точек, столбцов и т. д.), удерживая нажатой правую кнопку мыши или клавишу **Shift** (для объектов, расположенных не рядом). Попробуйте выбрать одновременно столбцы для трех и пяти передач на столбчатой диаграмме.

Функции пакета *iplots* позволяют нам исследовать распределения значений переменных и связи между переменными в подгруппах наблюдений, которые определяются в интерактивном режиме. При этом можно подметить такие закономерности, которые было бы сложно и трудоемко обнаружить при помощи других подходов. Более подробную информацию об этом пакете можно получить на сайте соответствующего проекта: <http://rosuda.org/iplots/>.

16.4.5. Пакет *rggobi*

Последний пример интерактивной графики выходит за пределы программы R – мы обратимся к программе GGobi (www.ggobi.org). Эта мощная программа для визуального и динамического исследования многомерных данных, доступная свободно для операционных систем Windows, Mac OS X и Linux. В ней реализовано множество привлекательных возможностей, включая связанные между собой интерактивные диаграммы: рассеяния, столбчатые, параллельных координат,

временных рядов, матрицы диаграмм рассеяния и вращение в трехмерном пространстве; раскрашивание и идентификацию объектов; методы многомерных преобразований и поддержку комплексных исследований, в том числе автоматизированные и управляемые в ручном режиме одно- и двухмерные проекции. К счастью, пакет `rggobi` создает интерфейс, который удачно объединяет GGobi и R.

Первый шаг в использовании GGobi – это скачать и установить подходящее для вашей операционной системы программное обеспечение (www.ggobi.org/downloads/). Затем установите в R пакет `rggobi` при помощи команды `install.packages("rggobi", depend=TRUE)`.

Как только вы все это установили, можете использовать функцию `ggobi()`, чтобы открыть GGobi из R. Это позволяет строить сложные интерактивные графики для данных, загруженных в программу R. Чтобы увидеть, как это работает, наберите в командной строке

```
library(rggobi)
g <- ggobi(mtcars)
```

Откроется интерфейс программы GGobi, и у вас появится возможность исследовать набор данных `mtcars` в высшей степени интерактивном способом. Чтобы узнать больше, прочтите введение, учебное руководство, справочник и посмотрите видеоруководства, доступные на сайте GGobi. Всеобъемлющий обзор также представлен в публикации Cook & Swayne (2008).

16.5. Резюме

В этой главе мы рассмотрели несколько пакетов, которые предоставляют доступ к продвинутым графическим методам. Мы начали с пакета `lattice`, который был разработан для создания категоризованных диаграмм. Затем мы познакомились с пакетом `ggplot2`, основанным на тщательно разработанной графической грамматике. Оба эти пакета предоставляют самодостаточную мощную альтернативу базовой графической системе R. В каждом из них реализованы методы создания привлекательных и продуманных способов визуализации данных, которые сложно реализовать другими средствами.

Затем мы исследовали несколько пакетов для интерактивного взаимодействия с диаграммами в режиме реального времени, таких как `playwith`, `latticist`, `iplots` и `rggobi`. В этих пакетах можно напрямую управлять данными на диаграммах, что позволяет лучше понять данные и помогает подметить важные закономерности.



Теперь вы должны хорошо представлять все многообразие способов визуального представления данных в R. Если рисунок стоит тысячи слов, а в R существует тысяча способов графической обработки данных, тогда R должен стоить миллиона слов (ну или чего-нибудь в этом роде). Эти возможности – результат тяжелого и бескорыстного труда исходной команды разработчиков программы R и тысяч часов работы авторов дополнительных пакетов.



ПОСЛЕСЛОВИЕ: В погоне за кроликом

В этой книге мы рассмотрели самые разнообразные темы, включая основные, такие как устройство среды R, управление данными, традиционные статистические модели и статистическая графика. Мы также познакомились с тайными сокровищами, такими как методы повторных выборок, восстановление пропущенных данных и интерактивные диаграммы. Замечательное (или, может быть, приводящее в бешенство) свойство R – в этой программе всегда есть чему научиться.

R – это обширная, мощная и постоянно развивающаяся статистическая среда и язык программирования. Как можно оставаться в курсе событий при таком большом числе новых пакетов, частых обновлениях и появлении новых направлений развития? К счастью, много сайтов содержат информацию об изменениях этой статистической среды и пакетов, новых методологиях и размещают пользовательские руководства. Ниже я перечислил некоторые из моих любимых сайтов¹.

Проект R (*The R Project*, <http://www.r-project.org/>)

Официальный сайт программы и ваша первая остановка на пути выяснения любых вопросов о программе. На сайте размещена обширная документация, включая «Введение в R» (*An Introduction to R*), «Определение языка R» (*The R Language Definition*), «Создание дополнительных приложений в R» (*Writing R Extensions*), «Экспорт и импорт данных в R» (*R Data Import/Export*), «Установка и администрирование R» (*R Installation and Administration*) и «Часто задаваемые вопросы про R» (*The R FAQ*).

Журнал R (*The R Journal*, <http://journal.r-project.org/>)

Рецензируемый журнал свободного доступа, в котором публикуются статьи по R и дополнительным пакетам.

¹ Конечно, все они англоязычные. Русскоязычные интернет-сообщества расположены, например, по адресам <http://r-statistics.livejournal.com/> и <http://vk.com/club8142131>. – Прим. пер.

Блоги, посвященные R (R Bloggers, <http://www.r-bloggers.com/>)

Интернет-сообщество, посвященное R. Новые статьи появляются там ежедневно. Я постоянно посещаю этот ресурс.

Планета R (Planet R, <http://planetr.stderr.org>)

Еще один интернет-ресурс, на котором собирается информация с многих сайтов. Обновляется ежедневно.

БРусника (CRANberries, <http://dirk.eddelbuettel.com/cranberries/>)

Сайт, на котором собирается информация о новых и обновленных пакетах со ссылками для их скачивания на CRAN (всеобъемлющий сетевой архив R, Comprehensive R Archive Network).

Галерея графики R (R Graph Gallery, <http://addictedtor.free.fr/graphiques/>)

Коллекция новых диаграмм вместе с программными кодами для их построения.

Пользовательское руководство по графике R (R Graphics Manual, <http://bm2.genes.nig.ac.jp/>)²

Коллекция графики R, отсортированная по темам, пакетам и функциям. По моим последним подсчетам, там было более 35 000 изображений!

Журнал статистического программного обеспечения (Journal of Statistical Software, <http://www.jstatsoft.org/>)

Рецензируемый журнал свободного доступа, содержащий статьи, отзывы о книгах и фрагменты программного кода для статистических вычислений. Там часто публикуются статьи по R.

Revolutions (<http://blog.revolution-computing.com/>)

Популярный хорошо структурированный блог, посвященный новостям и информации про R.

Тематический указатель (CRAN Task Views, <http://cran.r-project.org/web/views/>)

Тематический указатель – это рекомендации по использованию R в разных областях прикладной и теоретической науки. Здесь размещены описания пакетов и методов, подходящих для определенных областей исследований. В настоящее время имеются рекомендации по 27 областям³:

2 Текущий адрес этого сайта: <http://rgm3.lab.nig.ac.jp/RGM/>. – Прим. пер.

3 Их число постоянно растет. Этот указатель можно установить при помощи команды `install.package()`. – Прим. пер.

- Байесовские методы;
- Хемометрия и вычислительная физика;
- Планирование, мониторинг и анализ клинических испытаний;
- Кластерный анализ и конечные смешанные модели;
- Вероятностные распределения;
- Эконометрика;
- Анализ экологических данных и состояния окружающей среды;
- Планирование экспериментов;
- Эмпирические финансовые исследования;
- Статистическая генетика;
- Графические построения и интерактивная графика;
- Графические модели в R;
- Высокопроизводительные и параллельные вычисления в R;
- Машинное обучение и статистическое обучение;
- Графический анализ медицинских данных;
- Многомерная статистика;
- Обработка лингвистической информации естественного языка;
- Официальная статистика и методология создания выборок;
- Оптимизация и математическое программирование;
- Анализ данных фармакокинетики;
- Филогенетика, в особенности сравнительные методы;
- Психометрические модели и методы;
- Устойчивые (робастные) статистические методы;
- Статистика в социологии;
- Анализ пространственных данных;
- Анализ выживания;
- Анализ временных рядов.

Основная рассылка помощи по R (R-Help Main R Mailing List,
<https://stat.ethz.ch/mailman/listinfo/r-help>)

Эта онлайн-рассылка – лучшее место, чтобы задать вопрос по R. Можно осуществлять поиск и по ее архивам. Перед тем как задать вопрос, убедитесь в том, что он отсутствует в списке часто задаваемых вопросов.

R по-быстрому (Quick-R, <http://www.statmethods.net>)

Это мой собственный сайт, посвященный R. Там размещено более 80 кратких руководств по разным областям R. Ложная скромность не позволяет мне сказать больше.



Сообщество R всегда готово помочь, полно энергии и вдохновляет на многое. Добро пожаловать в Страну чудес.



ПРИЛОЖЕНИЕ А. Графические пользовательские интерфейсы

Вы заглянули сюда первым делом, не правда ли? По умолчанию в R реализован простой интерфейс командной строки. Пользователи вводят команды в ответ на приглашение (> по умолчанию), эти команды выполняются по одной. Для многих людей, которые обрабатывают данные, интерфейс командной строки – это наиболее серьезное препятствие на пути овладения R.

Было предпринято множество попыток по созданию графических интерфейсов, начиная с редакторов кодов, взаимодействующих с R (таких как RStudio), и графических интерфейсов для определенных функций и пакетов (таких как BiplotGUI) до развитых графических интерфейсов, которые позволяют пользователям выполнять анализ при помощи работы с меню и диалоговыми окнами (таких как R Commander).

Наиболее полезные редакторы программного кода перечислены в табл. A.1.

Таблица А.1. Интегрированные среды разработки и синтаксические редакторы

Название	Ссылка
Eclipse со вспомогательной программой StatET	http://www.eclipse.org и http://www.walware.de/goto/statet
ESS	http://ess.r-project.org/
Komodo Edit со вспомогательной программой SciViews-K	http://www.activestate.com/komodo_edit/ http://www.sciviews.org/SciViews-K/
JGR	http://www.r-forge.net/JGR/

Название	Ссылка
RStudio	http://www.rstudio.org
Tinn-R (только для Windows)	http://www.sciviews.org/Tinn-R/
Notepad++ в сочетании с NppToR (только для Windows)	http://notepad-plus-plus.org/ http://sourceforge.net/projects/nppror/

Редакторы программного кода, перечисленные в табл. А.1, позволяют пользователям редактировать и выполнять команды R, представляя возможности цветового выделения синтаксиса, завершения команд, исследования объектов, организации проектов и помощи в онлайн-режиме. Пример рабочего пространства RStudio представлен на рис. А.1.

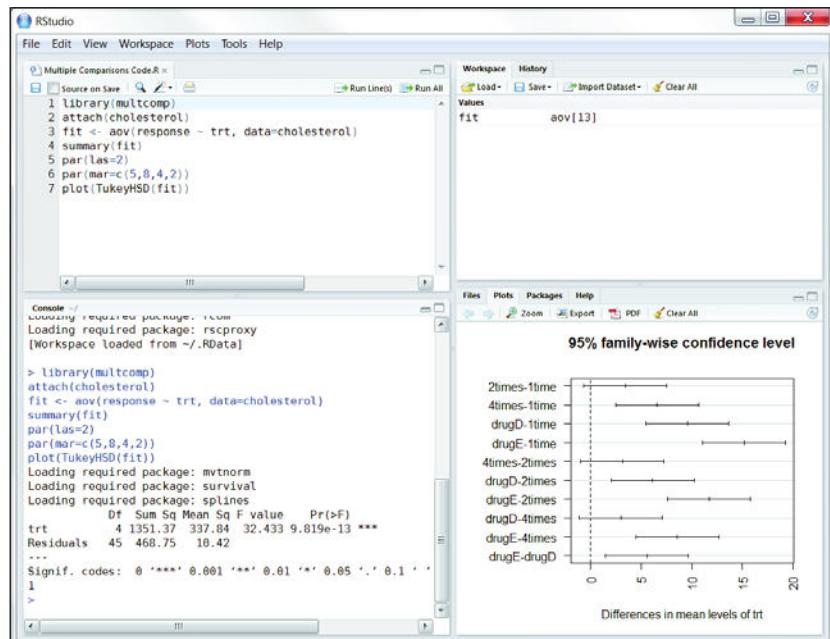


Рис. А.1. Графический интерфейс RStudio

Несколько развитых графических пользовательских интерфейсов для R перечислены в табл. А.2. Эти интерфейсы не так всеобъемлющи и хорошо разработаны, как интерфейсы для SAS или IBM SPSS, однако они быстро развиваются.

Таблица А.2. Развитые графические пользовательские интерфейсы для R

Название	Ссылка
JGR/ Deducer	Main.DeducerManual">http://ifellows.ucsd.edu/pmwiki/pmwiki.php?n>Main.DeducerManual
R AnalyticFlow	http://www.ef-prime.com/products/ranalyticflow_en/
Rattle (для исследования структуры данных)	http://rattle.togaware.com/
R Commander	http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/
Red R	http://www.red-r.org/
Rkward	http://rkward.sourceforge.net/

Мой любимый графический интерфейс для вводных курсов по статистике – это R Commander (показан на рис. А.2).

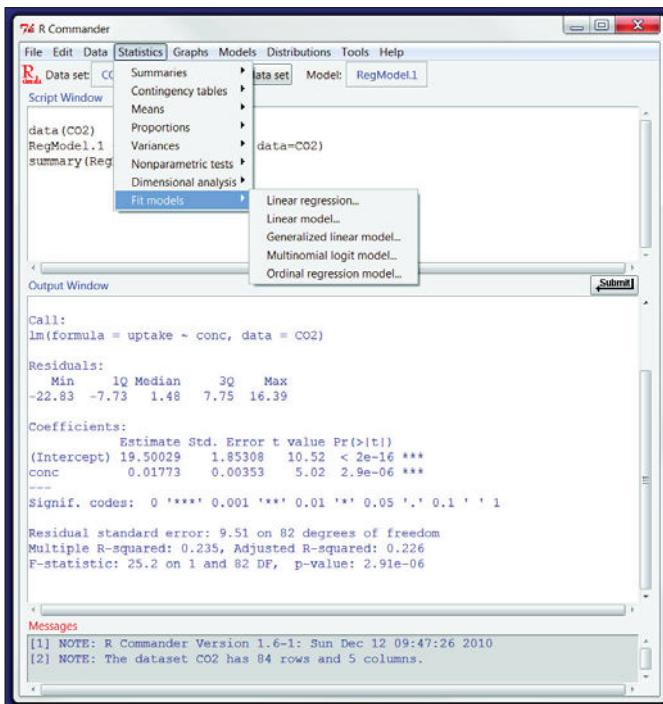


Рис. А.2. Графический интерфейс R Commander



Наконец, существует ряд приложений, которые позволяют создать системную оболочку в виде графического интерфейса для любой функции R (включая созданные пользователями). К таким приложениям относится генератор графических пользовательских интерфейсов R (R GUI Generator, RGG, <http://rgg.r-forge.r-project.org/>), а также пакеты `fgui` и `twiddler`, которые можно скачать из архива CRAN.

Проекты по созданию графических интерфейсов в R сейчас находятся в фазе активных изменений и развития. Для получения более подробной информации зайдите на страницу проектов графических пользовательских интерфейсов R: http://www.sciviews.org/_rgui/.



ПРИЛОЖЕНИЕ В. Настройка начальной конфигурации программы

Одна из вещей, которые в первую очередь любят делать программисты, – это настройка начальной конфигурации программы, чтобы она соответствовала их предпочтениям. Такая настройка позволяет задать параметры программы, указать рабочую директорию, загрузить часто используемые пакеты и пользовательские функции, установить сайт для скачивания CRAN и выполнить любое число служебных задач.

Конфигурацию программы можно настроить либо при помощи файла инициализации сайта (*Rprofile.site*), либо файла инициализации директории (*.Rprofile*). Это текстовые файлы, содержащие программный код R, который выполняется при запуске программы.

При запуске программа извлекает файл *Rprofile.site* из директории *R_HOME/...*, где *R_HOME* – это переменная среды. Затем выполняется поиск файла *.Rprofile* в текущей рабочей директории. Если R не находит этот файл, поиск будет продолжен в домашней директории пользователя. Вы можете использовать команды *Sys.getenv("R_HOME")*, *Sys.getenv("HOME")* и *getwd()*, для того чтобы выяснить местонахождение директорий *R_HOME*, *HOME* и текущей рабочей директории соответственно.

В упомянутых файлах можно разместить две специализированные функции. Функция *.First()* выполняется в начале каждой сессии R, а функция *.Last()* – в конце. Пример содержания файла *Rprofile.site* приведен в программном коде В.1.

Программный код В.1. Пример файла *Rprofile.site*

```
options(papersize="a4")
options(editor="notepad")
options(pager="internal")
options(tab.width = 2)
options(width = 130)
options(graphics.record=TRUE)
```

◀ Назначение общих
параметров

```
options(show.signif.stars=FALSE)

options(prompt="> ")
options(continue="+ ")

.libPaths("C:/my_R_library")           ↗ Настройка интерактивного
                                         приглашения к вводу команд

local({r <- getOption("repos")
       r["CRAN"] <- "http://cran.case.edu/"
       options(repos=r)})

.First <- function() {                ↗ Настройка начала
library(lattice)                      работы
library(Hmisc)
source("C:/mydir/myfunctions.R")
cat("\nWelcome at", date(), "\n")
}

.Last <- function() {                 ↗ Настройка завершения
cat("\nGoodbye at ", date(), "\n")
}
```

В этом файле нужно обратить внимание на несколько вещей:

- Задав значение переменной `.libPaths`, вы можете сформировать локальную библиотеку пакетов вне системы директорий R. Это может пригодиться для создания резервных копий при обновлении пакетов.
- Установка зеркала сайта CRAN избавит вас от необходимости выбирать его при установке каждого нового пакета.
- Функция `.First` – это чрезвычайно удачное место, для того чтобы загрузить часто используемые пакеты и пользовательские функции.
- Функция `.Last` создана для «наведения порядка» при окончании работы, включая сохранение истории команд, полученных результатов и файлов данных.

Существуют и другие способы настройки начальной конфигурации программы, включая параметры командной строки и переменные среды. Для получения дополнительной информации введите `help(Startup)` или прочтите приложение В в руководстве «Введение в R» (<http://cran.r-project.org/doc/manuals/R-intro.pdf>).



ПРИЛОЖЕНИЕ С.

Экспорт данных из R

Во второй главе мы рассмотрели широкий спектр методов для импортирования данных в R. Однако бывают случаи, когда вам нужно пойти другим путем – экспортировать данные из R – так, чтобы их можно было заархивировать или открыть в других программах. В этом приложении описано, как сохранить объект R в виде текстового файла с разделителями, таблицы Excel или файла статистической программы (такой как SPSS, SAS или Stata).

C.1. Текстовый файл с разделителями

Для сохранения объекта R в виде текстового файла с разделителями вы можете использовать функцию `write.table()`. Формат ее применения таков:

```
write.table(x, outfile, sep=разделитель, quote=TRUE, na="NA"),
```

где `x` – это объект R, а `outfile` – название итогового файла. Например, выражение

```
write.table(mydata, "mydata.txt", sep=",")
```

позволит сохранить набор данных `mydata` в виде файла `mydata.txt` с разделителями-запятыми в текущей рабочей директории. Укажите путь доступа (например, `"c:/myprojects/mydata.txt"`), чтобы сохранить итоговый файл в другом месте. Замена параметра `sep=","` на `sep="\t"` позволит сохранить данные в виде файла с разделителями-табуляторами. По умолчанию строки заключены в кавычки (""), а пропущенные значения записаны как NA.

C.2. Таблица Excel

Функция `write.xlsx()` из пакета `xlsx` позволяет сохранить таблицу данных R в виде рабочей книги Excel 2007. Формат ее применения

таков:

```
library(xlsx)
write.xlsx(x, outfile, col.NAMES=TRUE, row.names=TRUE,
sheetName="Sheet 1", append=FALSE)
```

Например, выражения

```
library(xlsx)
write.xlsx(mydata, "mydata.xlsx")
```

позволяют сохранить таблицу данных в виде листа (по умолчанию первого) рабочей книги Excel с названием `mydata.xlsx` в текущей рабочей директории. По умолчанию названия переменных в наборе данных используются для создания названий столбцов в рабочем листе, а имена строк размещаются в первом столбце. Если файл `mydata.xlsx` уже существует, то он заменяется на новый.

С.3. Другие статистические программы

Для экспорта таблиц данных в другие статистические программы можно использовать функцию `write.foreign()` из пакета `foreign`. При этом создаются два файла – файл свободного формата, содержащий данные, и файл с кодом, в котором записаны инструкции для прочтения данных другими статистическими программами. Формат применения функции таков:

```
write.foreign(dataframe, datafile, codefile, package=package)
```

Например, код

```
library(foreign)
write.foreign(mydata, "mydata.txt", "mycode.sps", package="SPSS")
```

сохраняет таблицу данных в виде текстового файла `mydata.txt` свободного формата в текущей рабочей директории и программы SPSS с названием `mycode.sps`, которую можно использовать для чтения текстового файла. Параметр `package` может также принимать значения "SAS" и "Stata".

Для того чтобы узнать больше об экспорте данных из R, прочтите документацию по импорту/экспорту данных R, размещенную по адресу <http://cran.r-project.org/doc/manuals/R-data.pdf>.



ПРИЛОЖЕНИЕ D.

Сохранение

результатов в пригодном для публикации качестве

Исследование не заканчивается с выполнением последнего статистического анализа или созданием последней диаграммы. Результаты нужно представить в виде сообщения, чтобы донести их должным образом до учителя, руководителя, клиента, государственного органа или редактора журнала. Хотя в R можно создавать современные диаграммы, текстовые результаты выводятся в прискорбно устаревшем виде – таблицы из монотипичного текста с разделенными пробелами столбцами.

Для сохранения результатов в пригодном для публикации качестве обычно используются два пакета – `Sweave` и `odfWeave`. Первый позволяет встраивать программный код и результаты в файлы формата LaTeX для получения отчетов типографского качества в форматах PDF, PostScript и DVI. Это изящный, точный и гибкий способ, однако для его применения нужно уметь работать в системе LaTeX.

Сходным образом пакет `odfWeave` позволяет встраивать программный код и результаты R в файлы формата ODF (Open Documents Format). Полученные отчеты можно совершенствовать в текстовом редакторе ODF, таком как OpenOffice Writer, а затем сохранять в формате ODF или Microsoft Word. Этот процесс не такой гибкий, как описанный выше, зато он не требует владения системой LaTeX. Мы рассмотрим данные способы по очереди.

D.1. Подготовка отчета типографского качества при помощи пакета **Sweave** (**R + LaTeX**)

LaTeX – это система подготовки документов типографского качества (<http://www.latex-project.org>), которая свободно доступна для операционных систем Windows, Mac и Linux. Пользователь создает текстовый документ, который содержит фрагменты кода, обеспечивающие форматирование (код разметки). Затем файл обрабатывается компилирующей программой, в результате чего получается окончательный документ в формате PDF, PostScript или DVI.

Пакет **Sweave** позволяет встраивать в файл программный код R и результаты его применения (включая диаграммы). Это многоступенчатый процесс:

1. В любом текстовом редакторе создается специальный файл (*noweb* file, обычно с расширением .Rnw). В этом файле хранятся все записи, код разметки LaTeX и фрагменты программного кода R. Каждый фрагмент кода R начинается с символов <<>>= и заканчивается символом @.
2. Этот файл обрабатывается функцией `Sweave()`, в результате чего получается файл LaTeX. На этом этапе выполняются фрагменты программного кода R и, в зависимости от заданных параметров, замещаются кодом в формате LaTeX и результатами действия этого кода. Этот этап запускается из программы R или из командной строки.

При запуске из R формат применения функции таков:

```
Sweave("infile.Rnw")
```

По умолчанию эта команда использует файл example.Rnw, находящийся в текущей рабочей директории, и сохраняет вывод в виде файла example.tex в той же директории. В качестве альтернативы можно использовать команду

```
Sweave("infile.Rnw", syntax="SweaveSyntaxNoweb")
```

Определение параметра `syntax` позволит избежать некоторых распространенных синтаксических ошибок, а также конфликтов с пакетом R2HTML.

Запуск функции из командной строки зависит от операционной системы. Например, в Linux он может выглядеть как

```
$ R CMD Sweave infile.Rnw
```

3. Затем файл формата LaTeX обрабатывается компилятором LaTeX, в результате чего получается файл в формате PDF, PostScript или DVI. К распространенным компиляторам LaTeX относятся TeX Live для Linux, MacTeX для Mac и proTeXt для Windows.

Схема всего процесса изображена на рис. D.1.



Рис. D.1. Процесс создания отчета типографского качества при помощи пакета Sweave

Как уже говорилось ранее, каждый фрагмент кода R помещен между символами <>>= и @. К каждому символу <>>= можно добавить определенные параметры для контроля выполнения соответствующего фрагмента программного кода. Например, выражение

```
<<echo=TRUE, results=HIDE>>=
summary(lm(Y~X, data=mydata))
@
```

выводит сам код, но не результаты, тогда как выражение

```
<<echo=FALSE, fig=TRUE>>=
plot(A)
@
```

позволит не печатать сам код, а вывести только диаграмму. Часто используемые параметры приведены в табл. D.1.

Таблица D.1. Распространенные параметры фрагментов программного кода R

Параметр	Описание
echo	Выводит код (<code>echo=TRUE</code> по умолчанию) или нет (<code>echo=FALSE</code>)
eval	Используйте параметр <code>eval=FALSE</code> , чтобы предотвратить выполнение кода. Значение по умолчанию – <code>TRUE</code>
fig	Используйте параметр <code>fig=TRUE</code> для вывода диаграммы. Значение по умолчанию – <code>FALSE</code>
results	Выводит код (<code>results=verbatim</code> по умолчанию), предотвращает вывод результатов (<code>results=hide</code>) или выводит результат, подразумевая, что он содержит разметку LaTeX (<code>results=tex</code>). Используйте параметр <code>results=tex</code> , когда результат создается при помощи функции <code>xtable()</code> из пакета <code>xtable</code> или при помощи функции <code>latex()</code> из пакета <code>Hmisc</code>

По умолчанию `Sweave` добавляет код разметки LaTeX, чтобы красиво отформатировать таблицы данных, матрицы и векторы. Кроме того, несколько объектов R могут быть размещены на одной строке при помощи выражения `\Sexpr{}`. Учтите, что для правильного отображения категоризованных диаграмм соответствующий код нужно поместить в скобки выражения `print()`.

Функция `xtable()` из пакета `xtable` используется для более точного форматирования таблиц данных и матриц. Кроме того, с ее помощью можно форматировать другие объекты R, включая те, которые создаются в результате действия функций `m()`, `glm()`, `aov()`, `table()`, `ts()` и `coxph()`. Введите команду `method(xtable)`, чтобы увидеть полный список функций. При форматировании результатов при помощи команды `xtable()` не забудьте добавить параметр `results=tex` в выражение, обозначающее начало кода.

Проще всего понять, как это работает, на примере. Изучите файл `.Rnw` в программном коде D.1. Там повторен однофакторный дисперсионный анализ из раздела 8.3. Код разметки LaTeX начинается с обратного слэша (`\`), за исключением выражения `\Sexpr{}`, которое создается функцией `Sweave`. Программный код R выделен полужирным курсивом.

Программный код D.1. Пример файла формата noweb (example.Rnw)¹

```
\documentclass[12pt]{article}
\usepackage[T2A]{fontenc}
\usepackage[utf8]{inputenc}
\usepackage[russian]{babel}
\usepackage{indentfirst}
\usepackage{noae}{Sweave}
\title{Пример отчета}
\author{Роберт И. Кабаков, доктор наук}
\date{}
\begin{document}
\maketitle

<<echo=false, results=hide>>=
library(multcomp)
library(xtable)
attach(cholesterol)
pdf.options(family="NimbusSan", encoding="CP1251.enc")
@

\section{Результаты}

Снижение уровня холестерина изучали, случайно разделив \
Sexpr{nrow(cholesterol)} пациентов на \Sexpr{length(unique(trt))} \
групп с разным типом лечения. Общие характеристики экспериментальных \
групп представлены в табл. \ref{table:descriptives}.

<<echo = false, results = tex>>=
descTable <- data.frame("Лечение" = sort(unique(trt)),
  "N"      = as.vector(table(trt)),
  "Среднее" = tapply(response, list(trt), mean, na.rm=TRUE),
  "SD"     = tapply(response, list(trt), sd, na.rm=TRUE)
)
print(xtable(descTable, caption = "Общие характеристики \
каждой экспериментальной группы", label = "table:descriptives"),
  caption.placement = "top", include.rownames = FALSE)
@

Результаты дисперсионного анализа представлены в табл. \ref{table:anova},
```

¹ Для правильного отображения кириллицы в итоговом файле формата PDF пришлось немного дополнить исходный файл формата noweb (эти дополнения выделены курсивом). Этот исходный файл обязательно нужно сохранять в кодировке UTF-8. Учтите, что название файла, в который встраиваются шрифты (`embedFonts("example-004.pdf")`), должно состоять из имени исходного файла формата noweb и порядкового номера фрагмента кода R, который «отвечает» за графику. Другой вариант решения этой задачи – задать название диаграммы в явном виде в заголовке фрагмента кода, «отвечающего» за построение диаграммы. – *Прим. пер.*

```
<<echo=false, results=tex>>=
fit <- aov(response ~ trt)
print(xtable(fit, caption = "Дисперсионный анализ",
             label = «table:anova»), caption.placement = «top»)
@

\noindent а межгрупповые различия показаны на рис. \ref{figure:tukey}.

\begin{figure}\label{figure:tukey}
\begin{center}

<<fig=TRUE, echo=FALSE>>=
par(mar=c(5,4,6,2))
tuk <- glht(fit, linfct=mcp(trt="Tukey"))
plot(cld(tuk, level=.05), col="lightgrey", xlab="Тип лечения",
     ylab="Эффект")
box("figure")
@

\caption{Эффект от разных способов лечения и их попарные сравнения.}
\end{center}
\end{figure}

<<embedFonts>>=
embedFonts ("example-004.pdf")
@

\end{document}
```

После обработки *noweb*-файла в R при помощи функции *Sweave()* из полученного TeX-файла в компиляторе LaTeX² создан файл формата PDF, представленный на рис. D.2 и D.3.

Чтобы больше узнать о пакете *Sweave*, посетите посвященную ему страницу в Интернете: (www.stat.uni-muenchen.de/~leisch/Sweave/). Прекрасное введение также написано Терезой Скотт (Theresa Scott, <http://biostat.mc.vanderbilt.edu/TheresaScott>). Для получения дополнительной информации о LaTeX проглядите статью «Не такое уж короткое введение в LaTeX 2e» (“The Not So Short Introduction to LaTeX 2e”), доступную на домашней странице LaTeX (www.latex-project.org).

² Для правильного отображения кириллицы нужно обработать файл программой pdflatex дважды. – *Прим. пер.*

Пример отчета

Роберт И. Кабаков, доктор наук

1 Результаты

Снижение уровня холестерина изучали, случайно разделив 50 пациентов на 5 групп с разным типом лечения. Общие характеристики экспериментальных групп представлены в табл. 1.

Таблица 1: Общие характеристики каждой экспериментальной группы

Лечение	N	Среднее	SD
1time	10	5.78	2.88
2times	10	9.22	3.48
4times	10	12.37	2.92
drugD	10	15.36	3.45
drugE	10	20.95	3.35

Результаты дисперсионного анализа представлены в табл. 2,

Таблица 2: Дисперсионный анализ

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
trt	4	1351.37	337.84	32.43	0.0000
Residuals	45	468.75	10.42		

а межгрупповые различия показаны на рис. 1.

Рис. D.2. Первая страница отчета, созданного при помощи файла example.Rnw, содержание которого представлено в программном коде D.1.

Этот файл был обработан в R при помощи функции `Sweave()`, а из полученного TeX-файла в компиляторе LaTeX создан файл формата PDF

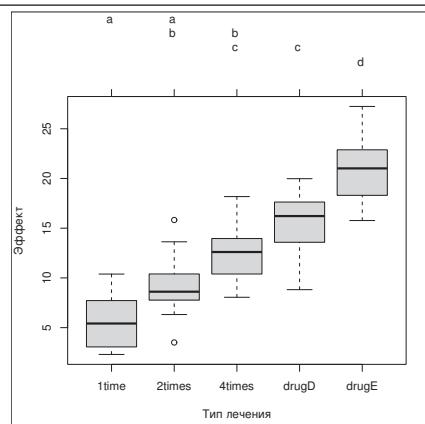


Рис. 1: Эффект от разных способов лечения и их попарные сравнения.

Рис. D.3. Вторая страница отчета, созданного при помощи noweb-файла, содержание которого представлено в программном коде D.1

D.2. Объединение сил с OpenOffice при помощи пакета odfWeave

Sweave позволяет встроить программный код R и результаты его применения в файл формата LaTeX, который затем может быть преобразован в файл PDF, PostScript или DVI. Хотя такие файлы красиво выглядят, их нельзя редактировать. Кроме того, многие люди хотят получить отчет в формате Word.

Пакет `odfWeave` позволяет встраивать программный код R и результаты его применения в файлы OpenOffice. Вместо того чтобы размещать фрагменты кода R в файле формата LaTeX, пользователь может вставить их в файл OpenOffice формата ODT (см. рис. D.3). Преимущество такого подхода заключается в том, что файл ODT можно создать в WYSIWYG³-редакторе, таком как OpenOffice Writer (www.OpenOffice.org); для этого не нужно овладевать языком разметки.

Как только вы создали специальный файл формата ODT, его можно обработать при помощи функции `odfWeave()` из пакета `odfWeave`. В отличие от пакета `Sweave`, `odfWeave` нужно скачать и установить перед первым использованием: `install.packages("odfWeave")`, а затем загружать в начале каждой сессии. К примеру, команды

```
library(odfWeave)
infile <- "example.odt"
outfile <- "example-out.odt"
odfWeave(infile, outfile)
```

на основе файла `example.odt`, содержание которого показано на рис. D.4, создадут файл `example-out.odt` (рис. D.5). Добавление команды `options(SweaveSyntax="SweaveSyntaxNoweb")` перед командой `odfWeave()` может предотвратить синтаксические ошибки в некоторых операционных системах.

Между пакетами `Sweave` и `odfWeave` существует несколько различий:

- в пакете `odfWeave` не действует функция `xtable()`. По умолчанию функция `odfWeave` представляет таблицы данных, матрицы и векторы в привлекательном виде. Если необходим

3 What You See Is What You Get – что видите, то и получаете. Это значит, что внешний вид распечатки в точности соответствует тому, что пользователь видит на экране. – *Прим. пер.*

больший контроль над форматом этих объектов, можно использовать функцию `odfTable()`;

- документы ODF используют разметку XML, а не LaTeX. Поэтому во фрагментах программного кода для функции `odfTable()` нужно использовать параметр `result=xml`, а не `result=tex`;
- файлы ввода и вывода должны иметь разные имена. В отличие от `Sweave`, команда `odfWeave("example.odt")` заменит исходный файл окончательным отчетом.

Пример моего отчета

Роберт И. Кабаков, доктор наук

```
<<echo=false, results=hide>>=
library(multcomp)
library(xtable)
attach(cholesterol)
@
```

1 Результаты

Снижение уровня холестерина изучали, случайно разделив `\Sexpr{trow(cholesterol)}` пациентов на `\Sexpr{length(unique(trt))}` групп с разным типом лечения. Общие характеристики экспериментальных групп представлены в табл. 1.

Таблица 1. Общие характеристики для каждой экспериментальной группы

```
<<echo = false, results = xml>>=
descTable <- data.frame("Лечение" = sort(unique(trt)),
  «N» = as.vector(table(trt)),
  «Среднее» = tapply(response, list(trt), mean, na.rm=TRUE),
  «SD» = tapply(response, list(trt), sd, na.rm=TRUE)
)
odfTable(descTable)
@
```

Результаты дисперсионного анализа представлены в табл. 2,

Таблица 2. Результаты дисперсионного анализа

```
<<echo=false>>=
fit <- aov(response ~ trt)
summary(fit)
@
```

а межгрупповые различия представлены на рис. 1.

```
<<fig=TRUE, echo=FALSE>>=
par(mar=c(5,4,6,2))
tuk <- glht(fit, linfct=mcp(trt="Tukey"))
plot(cld(tuk, level=.05), col="lightgrey", xlab="Лечение", ylab="Эффект")
box("figure")
@
```

Рисунок 1. Эффект от разных способов лечения и их попарные сравнения

Рис. D.4. Исходный `noweb`-файл (`example.odt`), который предстоит обработать в пакете `odfWeave`

Пример моего отчета
Роберт И. Кабаков, доктор наук

1 Результаты

Снижение уровня холестерина изучали, случайно разделив 50 пациентов на 5 групп с разным типом лечения. Общие характеристики экспериментальных групп представлены в табл. 1.

Таблица 1. Общие характеристики для каждой экспериментальной группы

	Лечение	N	Среднее	SD
1time	1time	10	5.782	2.878
2times	2times	10	9.225	3.483
4times	4times	10	12.375	2.923
drugD	drugD	10	15.361	3.455
drugE	drugE	10	20.948	3.345

Результаты дисперсионного анализа представлены в табл. 2,

Таблица 2. Результаты дисперсионного анализа

```
Df   Sum Sq Mean Sq F value    Pr(>F)
trt      4 1351.37 337.84 32.433 9.819e-13 ***
Residuals 45  468.75   10.42
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

а межгрупповые различия представлены на рис. 1.

Рис. D.5. Окончательный отчет в формате ODF (содержимое файла example-out.odt). Вторая страница этого отчета похожа на вторую страницу отчета, созданного в пакете Sweave (рис. D.2), и не показана здесь для экономии места

Если вы посмотрите на рис. D.5, то увидите, что таблица результатов дисперсионного анализа не так привлекательно оформлена, как в результате применения функции Sweave. Вместо этого она представлена стандартным моноширинным шрифтом, создаваемым R. Это происходит потому, что в пакете odfWeave нет функции для форматирования объектов, возвращаемых функциями lm(), glm() и т. д. Чтобы представить эти объекты в подходящем формате, нам бы потребовалось выделить отдельные компоненты интересующего нас объекта (fit в данном случае) и скомпоновать их в виде матрицы или таблицы данных.

Как только вы получили отчет в формате ODF, можете продолжать редактировать его, улучшая форматирование, а затем сохранить ре-

зультат в форматах ODT, HTML, DOC или DOCX. Для получения дальнейшей информации прочтите руководство к пакету odfWeave.

D.3. Комментарии

Использование пакетов Sweave и odfWeave выгодно по нескольким причинам. Встраивая нужный для проведения статистического анализа программный код непосредственно в отчет, вы аккуратно документируете процесс получения результатов. Спустя полгода вы можете легко понять, что было сделано. Вы также можете изменять статистический анализ или добавлять новые данные и тут же создавать новый отчет, затрачивая минимум усилий. Кроме того, вам не нужно вырезать и вставлять, а также переформатировать результаты.

К сожалению, вы получаете эти преимущества, затрачивая существенно больше труда на подготовительном этапе. Существуют и другие недостатки. В случае с LaTeX вам понадобится выучить новый язык верстки. В случае ODF вам нужно использовать непривычные программы типа OpenOffice.

Хорошо это или плохо, но в современном деловом мире принято создавать отчеты в Microsoft Word, а презентации – в PowerPoint. Для создания соответствующих документов со встроенными результатами из R можно использовать пакеты R2wd и R2PPT, однако пока они находятся в начальной стадии разработки. Я с нетерпением жду появления их окончательных версий.



ПРИЛОЖЕНИЕ Е.

Матричная алгебра в R

Многие из описанных в этой книге функций оперируют с матрицами. Манипуляции с матрицами – это неотъемлемая составляющая языка R. В табл. Е.1 описаны операторы и функции, которые особенно важны для решения задач линейной алгебры. В этой таблице A и B обозначают матрицы, x и b – векторы, a k – скаляр.

Таблица Е.1. Функции и операторы матричной алгебры в R

Оператор или функция	Описание
+ - * / ^	Поэлементное сложение, вычитание, умножение, деление и возведение в степень соответственно
A %*% B	Умножение матриц
A %o% B	Тензорное произведение, AB'
cbind(A, B, ...)	Совмещение векторов или матриц в горизонтальной плоскости
chol(A)	Разложение Холецкого для матрицы A. Если R <- chol(A), то chol(A) содержит фактор верхнетреугольной матрицы, такой что $R'R = A$
colMeans(A)	Возвращает вектор, который содержит средние значения для столбцов A
crossprod(A)	$A'A$
crossprod(A, B)	$A'B$
colSums(A)	Возвращает вектор, который содержит суммы значений столбцов A
diag(A)	Возвращает вектор, который содержит значения главной диагонали
diag(x)	Создает диагональную матрицу, главная диагональ которой представлена элементами x
diag(k)	Создает идентичную матрицу $k \times k$

Оператор или функция	Описание
eigen (A)	Собственные значения и собственные векторы A. Если <code>y <- eigen(A)</code> , то <code>y\$val</code> – это собственные значения A и <code>y\$vec</code> – собственные векторы A
ginv (A)	Квазиобращение Мура-Пенроуза матрицы A. (Требуется пакет MASS)
qr (A)	QR-разложение матрицы A. Если <code>y <- qr(A)</code> , то верхний треугольник матрицы <code>y\$qr</code> содержит само разложение, а нижний треугольник – информацию о разложении, <code>y\$rank</code> – это ранг A, <code>y\$qraux</code> – вектор, содержащий дополнительную информацию о Q, а <code>y\$pivot</code> содержит информацию об использованной стратегии разложения
rbind (A, B, ...)	Объединение матриц или векторов в вертикальной плоскости
rowMeans (A)	Возвращает вектор, который содержит средние значения для строк A
rowSums (A)	Возвращает вектор, который содержит суммы значений строк A
solve (A)	Инверсия A, где A – это квадратная матрица
solve (A, b)	Находит вектор x из уравнения $b = Ax$
svd (A)	Сингулярное разложение матрицы A. Если <code>y <- svd(A)</code> , то <code>y\$d</code> – вектор, содержащий сингулярные значения A, <code>y\$u</code> – матрица со столбцами, содержащими левые сингулярные векторы A, а <code>y\$v</code> матрица со столбцами, содержащими правые сингулярные векторы A
t (A)	Транспозиция матрицы A

Существует несколько дополнительных пакетов, которые особенно полезны при матричных преобразованиях. Пакет `matlab` содержит интерфейсные функции, которые с максимально возможной точностью воспроизводят обращения к функциям программы MATLAB. Они помогают перенести приложения и коды MATLAB в R. На сайте <http://mathesaurus.sourceforge.net/octave-r.html> размещена удобная памятка по преобразованию команд MATLAB в команды R.

Пакет `Matrix` содержит функции, которые расширяют возможности R, позволяя обрабатывать очень плотные (`dense`) или разреженные



(sparse) матрицы. Этот пакет предоставляет эффективный доступ к алгоритмам BLAS (Basic Linear Algebra Subroutines, подпрограммы для базовой линейной алгебры), Lapack (плотные матрицы), TAUCS (разреженные матрицы) и UMFPACK (разреженные матрицы).

Наконец, в пакете `matrixStats` реализованы методы работы со строками и столбцами матриц, включая функции, которые производят подсчет значений, рассчитывают суммы, произведения, меры центральной тенденции и разброса данных, и не только. Все эти функции оптимизированы для быстрого и эффективного использования памяти.



ПРИЛОЖЕНИЕ F.

Пакеты, упомянутые в этой книге

Широта охвата и мощность R в значительной мере обусловлены наличием дополнительных пакетов, созданных самоотверженными пользователями. В табл. F.1 перечислены все дополнительные пакеты, упомянутые в этой книге, вместе с номерами глав, в которых они обсуждаются.

Таблица F.1. Дополнительные пакеты, упомянутые в этой книге

Пакет	Авторы	Описание	Главы
AER	Christian Kleiber & Achim Zeileis	Функции, наборы данных, примеры, демонстрации и руководства из книги «Прикладная эконометрика в R» (<i>Applied Econometrics with R</i>), написанной Christian Kleiber и Achim Zeileis	13
Amelia	James Honaker, Gary King & Matthew Blackwell	Amelia II: программа для множественного восстановления пропущенных данных	15
arrayImpute	Eun-kyung Lee, Dankyu Yoon & Taesung Park	Восстановление пропущенных значений в данных, полученных при помощи технологии микрочипов	15
arrayMissPattern	Eun-kyung Lee, Dankyu Yoon & Taesung Park	Исследование структуры пропущенных значений в данных, полученных при помощи технологии микрочипов	15

Пакет	Авторы	Описание	Главы
boot	Оригинал на языке S – Angelo Canty. Версия для R – Brian Ripley	Функции для бутстреп-анализа	12
ca	Michael Greenacre & Oleg Nenadic	Простой, множественный и совместный анализ соответствий	7
car	John Fox & Sanford Weisberg	Дополнение к «Прикладной регрессии»	1, 8, 9, 10, 11
cat	Адаптировано для R – Ted Harding & Fernando Tusell. Оригинал – Joseph L. Schafer	Анализ категориальных переменных с пропущенными значениями	15
coin	Torsten Hothorn, Kurt Hornik, Mark A. van de Wiel & Achim Zeileis	Процедуры условных умозаключений в тестах с перестановками	12
corrgram	Kevin Wright	Создание коррелограммы	11
corrperm	Douglas M. Potter	Тесты с перестановками для корреляции с повторными измерениями	12
doBy	Søren Højsgaard с дополнениями от Kevin Wright & Alessandro A. Leidi.	Вычисления общих характеристик, общих линейных контрастов и прочих вспомогательных параметров для отдельных групп	7
effects	John Fox & Jangman Hong	Графическое отображение эффектов для линейных, обобщенных линейных, полиномиальных логит-моделей и логит-моделей отношений шансов	8, 9
FactoMineR	Francois Husson, Julie Josse, Sebastien Le & Jeremy Mazet	Многомерный первичный анализ и исследование структуры данных	14
FAIR	Ben Goodrich	Факторный анализ на основе генетического алгоритма	14
fCalendar	Diethelm Wuertz & Yohan Chalabi	Функции для временных и календарных объектов	4

Пакет	Авторы	Описание	Главы
foreign	Члены основной команды разработчиков R, Saikat DebRoy, Roger Bivand и др.	Читает данные из форматов Minitab, S, SAS, SPSS, Stata, Systat, dBase и других программ	2
gclus	Catherine Hurley	Графический кластерный анализ	1, 11
ggplot2	Hadley Wickam	Реализация идеи графической грамматики	16
glmPerm	Wiebke Werft & Douglas M. Potter	Проверка гипотез в обобщенных линейных моделях методом перестановок	12
gmodels	Gregory R. Warnes. Использован программный код R и/или документация, созданные Ben Bolker, Thomas Lumley, & Randall C. Johnson. Вклад последнего защищен авторским правом (2005) SAIC-Frederick, Inc.	Различные программные средства для подгонки моделей	7
gplots	Gregory R. Warnes. Использован программный код R и/или документация, созданные Ben Bolker, Lodewijk Bonebakker, Robert Gentleman, Wolfgang Huber Andy Liaw, Thomas Lumley, Martin Maechler, Arni Magnusson, Steffen Moeller, Marc Schwartz & Bill Venables	Различные команды R для визуализации данных	6, 9
grid	Paul Murrell	Переработанные возможности макетирования графики, дополнительно, а также некоторые возможности для интерактивности	16
gvmlm	Edsel A. Pena & Elizabeth H. Slate	Общая оценка допущений линейных моделей	8

Пакет	Авторы	Описание	Главы
hdf5	Marcus G. Daniels	Интерфейс библиотеки NCSA HDF5	2
hexbin	Dan Carr, адаптировано для R Nicholas Lewin-Koh & Martin Maechler	Программные средства для объединения данных в шестиугольные ячейки	11
HH	Richard M. Heiberger	Вспомогательная программа для «Статистического анализа и отображения данных» (Statistical Analysis and Data Display), написанного Heiberger & Holland	9
Hmisc	Frank E. Harrell мл., при участии многих других пользователей	Разные функции Харрелла для анализа данных, высокоровневой графики, вспомогательных операций и т. д.	2, 3, 7
kmi	Arthur Alligno	Множественное восстановление данных методом Каплана-Мейера для анализа функций накопленной частоты событий в условиях конкурирующих рисков	15
lattice	Deepayan Sarkar	Категоризованные графики	16
latticist	Felix Andrews	Графический пользовательский интерфейс для визуального исследования данных	16
lavaan	Yves Rosseel	Функции для моделей с латентными переменными, включая конfirmаторный факторный анализ, моделирование структурных уравнений и латентных моделей роста	14
lcda	Michael Buecker	Дискриминантный анализ для латентных классов	14

Пакет	Авторы	Описание	Главы
leaps	Thomas Lumley с использованием кода на Fortran, написанного Alan Miller	Выбор подмножества регрессии посредством полного перебора вариантов	8
lmPerm	Bob Wheeler	Тесты с перестановками для линейных моделей	12
logregperm	Douglas M. Potter	Проверка гипотез логистической регрессии при помощи тестов с перестановками	12
longitudinalData	Christophe Genolini	Средства для работы с данными, полученными в ходе продольного обследования (логнитюдными данными)	15
lsa	Fridolin Wild	Анализ скрытой семантики	14
ltm	Dimitris Rizopoulos	Модели скрытых характеристик в свете стохастической теории тестов	14
lubridate	Garrett Grolemund & Hadley Wickham	Функции для обнаружения и вставки данных в формате времени и даты, извлечения и изменения компонентов таких данных, аккуратных вычислений со временем и датами, а также работы со временными зонами и переходом на летнее время	4
MASS	Оригинал на языке S написан Venables & Ripley. Адаптировано для R Brian Ripley в соответствии с ранними разработками Kurt Hornik и Albrecht Gebhardt	Функции и наборы данных для книги Venables & Ripley's «Современная прикладная статистика при помощи S» (Modern Applied Statistics with S, 4-е издание)	4, 5, 7, 8, 9, 12
mlogit	Yves Croissant	Оценка полиномиальной логит-модели	13

Пакет	Авторы	Описание	Главы
multcomp	Torsten Hothorn, Frank Bretz Peter Westfall, Richard M. Heiberger & Andre Schuetzenmeister	Одновременное проведение тестов и вычисление довери- тельных интервалов для обобщенных линейных гипотез в параметри- ческих моделях, включая линейные, обобщенные линейные, со смешан- ными линейными эффектами, а также модели выживания	9, 12
mvnmle	Kevin Gross, с по- мощью Douglas Bates	Оценка пропущенных значений многомерных нормальных данных методом максимального правдоподобия	15
mvoutlier	Moritz Gschwandtner & Peter Filzmoser	Обнаружение выбросов в многомерных данных при помощи устойчивых методов	9
ncdf, ncdf4	David Pierce	Интерфейс для фай- лов данных в формате Unidata netCDF	2
nFactors	Gilles Raiche	Параллельный анализ и неграфическая вер- сия теста собственных значений	14
prmc	Joerg Helms & Ullrich Munzel	Непараметрические множественные срав- нения	7
OpenMx	Steven Boker, Michael Neale, Hermine Maes, Michael Wilde, Michael Spiegel, Timothy R. Brick, Jeffrey Spies, Ryne Estabrook, Sarah Kenny, Timothy Bates, Paras Mehta & John Fox	Продвинутое моделиро- вание на основе струк- турных уравнений	14
pastecs	Frederic Ibanez, Philippe Grosjean & Michele Etienne	Анализ рядов экологи- ческих данных в про- странстве и времени	7

Пакет	Авторы	Описание	Главы
riface	Russell Lenth, адаптирован для R Tobias Verbeke	Приложения на языке Java для вычисления мощности и размера выборки	10
playwith	Felix Andrews	Графический пользовательский интерфейс GTK+ для редактирования графиков R и взаимодействия с ними	16
poLCA	Drew Linzer & Jeffrey Lewis	Анализ скрытых классов для полиномиальных переменных	14
psych	William Revelle	Процедуры для психологических, психометрических и личностных исследований	7, 14
pwr	Stephane Champely	Базовые функции для анализа мощности	10
qcc	Luca Scrucca	Диаграммы контроля качества	13
randomLCA	Ken Beath	Анализ скрытых классов со случайными эффектами	14
Rcmdr	John Fox, с дополнениями Liviu Andronic, Michael Ash, Theophilus Boye, Stefano Calza, Andy Chang, Philippe Grosjean, Richard Heiberger, G. Jay Kerns, Renaud Lancelot, Matthieu Lesnoff, Uwe Ligges, Samir Messad, Martin Maechler, Robert Muenchen, Duncan Murdoch, Erich Neuwirth, Dan Putler, Brian Ripley, Miroslav Ristic & Peter Wolf	R Commander, независимый от операционной системы графический пользовательский интерфейс для R с базовыми статистическими функциями, основанный на пакете tcltk	11
reshape	Hadley Wickham	Гибкое переформатирование данных	4, 5, 7

Пакет	Авторы	Описание	Главы
rggobi	Duncan Temple Lang, Debby Swayne, Hadley Wickham & Michael Lawrence	Интерфейс между R и Ggobi	16
rgl	Daniel Adler & Duncan Murdoch	Система трехмерной визуализации (OpenGL)	11
RJDBC	Simon Urbanek	Предоставляет доступ к базам данным через интерфейс JDBC	2
rms	Frank E. Harrell, мл.	Стратегии регрессионного моделирования – около 225 функций, которые облегчают регрессионное моделирование, проверку гипотез, оценку, графическое представление, предсказание и набор текста	13
robust	Jiahui Wang, Ruben Zamar, Alfio Marazzi, Victor Yohai, Matias Salibian-Barrera, Ricardo Maronna, Eric Zivot, David Rocke, Doug Martin, Martin Maechler & Kjell Konis	Устойчивые методы	13
RODBC	Brian Ripley & Michael Lapsley	Доступ к базам данных ODBC	2
ROracle	David A. James & Jake Luciani	Интерфейс для баз данных Oracle	2
rrcov	Valentin Todorov	Устойчивые оценки положения и дисперсии, а также устойчивый многомерный анализ	9
sampling	Yves Tillé & Alina Matei	Функции для создания и калибровки выборок	4
scatterplot3d	Uwe Ligges	Изображение облака точек в трехмерном пространстве	11
sem	John Fox при участии Adam Kramer & Michael Friendly	Модели структурных уравнений	14

Пакет	Авторы	Описание	Главы
SeqKnn	Ki-Yeol Kim & Gwan-Su Yi, лаборатория CSBio, Университет исследования и передачи информации (Information and Communications University)	Последовательный KNN-метод замещения пропущенных данных	15
sm	Adrian Bowman & Adelchi Azzalini. Адаптировано для R B.D. Ripley вплоть до версии 2.0, версия 2.1 подготовлена Adrian Bowman & Adelchi Azzalini, версия 2.2 – Adrian Bowman	Методы сглаживания для непараметрической регрессии и оценки плотности	6, 9
vcd	David Meyer, Achim Zeileis & Kurt Hornik	Функции для визуализации категориальных данных	1, 6, 7, 11, 12
vegan	Jari Oksanen, F. Guillaume Blanchet, Roeland Kindt, Pierre Legendre, R. B. O'Hara, Gavin L. Simpson, Peter Solymos, M. Henry H. Stevens & Helene Wagner	Методы классификации, анализ разнообразия и другие функции для экологов, изучающих сообщества и растительность	9
VIM	Matthias Templ, Andreas Alfons & Alexander Kowarik	Визуализация и замещение пропущенных значений	15
xlsx	Adrian A. Dragulescu	Чтение, запись и формирование файлов Excel 2007 (xlsx)	2
XML	Duncan Temple Lang	Средства анализа и создания XML при помощи R и S-Plus	2



ПРИЛОЖЕНИЕ G. Работа с большими наборами данных

Все объекты R хранятся в виртуальной памяти. Для многих из нас эта особенность программы обернулась запоминающимся опытом интерактивной работы, однако для аналитиков, имеющих дело с большими объемами данных, это может привести к медленному выполнению программ и ошибкам памяти.

Ограничения памяти в основном зависят от сборки программы (32 или 64 бит), а для 32-битной сборки программы для Windows – от версии операционной системы. Сообщения об ошибке, начинающиеся со слов `cannot allocate vector of size1`, как правило, указывают на превышение лимита рабочей памяти, а сообщения об ошибке, которые начинаются со слов `cannot allocate vector of length2`, свидетельствуют о превышении допустимой длины вектора. При работе с большими наборами данных старайтесь по возможности использовать 64-битную версию программы. Однако во всех версиях число элементов в векторе не должно превышать 2 147 483 647 (введите `?Memory` для получения дальнейшей информации)³.

Есть три аспекта, которые нужно учитывать при работе с большими объемами данных: а). эффективное программирование для ускорения выполнения программ, б). внешнее хранение данных для ограничения обращений к памяти, с). использование специальных статистических методов, разработанных для эффективного анализа больших объемов данных. Мы кратко рассмотрим все эти вопросы один за другим.

1 Не могу разместить вектор данного размера. – *Прим. пер.*

2 Не могу разместить вектор данной длины. – *Прим. пер.*

3 Это ограничение снято, начиная с версии программы 3.0.0 (на 64-битных машинах). – *Прим. пер.*

G.1. Эффективное программирование

У программистов есть ряд приемов, которые помогут оптимизировать обработку больших наборов данных.

- По возможности векторизуйте вычисления. Используйте встроенные функции R для работы с векторами, матрицами и списками (например, `sapply`, `lapply` и `mapply`) и применяйте циклы (`for` и `while`) только там, где они действительно необходимы.
- Используйте матрицы вместо таблиц данных (это более производительно).
- При использовании функций из семейства `read.table()` для сохранения внешних данных в виде таблиц R указывайте параметры `colClasses` и `nrows` в явном виде, задайте параметр `comment.char = ""` и присвойте значение "NULL" ненужным столбцам. Это снизит интенсивность использования памяти и значительно ускорит обработку данных. При сохранении внешних данных в виде матрицы R вместо этого используйте функцию `scan()`.
- Проверяйте программы на небольшой части данных, чтобы оптимизировать программный код и удалить ошибки, перед тем как пытаться обработать весь большой набор данных целиком.
- Удаляйте временные объекты и объекты, которые больше не нужны. Команда `rm(list=ls())` позволяет избавиться от всех объектов в памяти. Отдельные объекты можно удалить при помощи команды `rm(объект)4`.
- Используйте функцию `.ls.objects()`, описанную в блоге Джероми Англима ([Jeromy Anglim, jeromyanglim.blogspot.com](http://jeromyanglim.blogspot.com)), в записи под названием «Управление памятью в R: несколько советов и уловок» ("Memory Management in R: A Few Tips and Tricks"). Эта функция создает список всех объектов рабочего пространства, отсортированных по размеру (Mb). Это позволит обнаружить «пожирающие память» объекты и разобраться с ними.
- Исследуйте свои программы, чтобы узнать, сколько времени затрачивается на выполнение каждой команды. Этую

⁴ А еще не забывайте освобождать память при помощи команды `gc()`. – Прим. пер.

задачу можно выполнить при помощи функций `Rprof()` и `summaryRprof()`. Функция `system.time()` тоже может пригодиться. Пакеты `profR` и `prooftools` содержат функции, которые можно использовать при анализе профиля программы.

- Если требуются более специализированные подпрограммы, можно использовать пакет `Rcpp` для преобразования объектов R в функции C++ и обратно.

Увеличение эффективности программного кода – это только полдела при работе с большими объемами данных. Борясь с ограничениями памяти, можно хранить данные вне программы и использовать специализированные методы анализа.

G.2. Хранение данных вне оперативной памяти

Существует несколько пакетов, предназначенных для хранения данных вне оперативной памяти R. Идея заключается в том, чтобы хранить данные во внешних базах данных или в неструктурированных двоичных файлах на диске, а затем загружать эти данные частями по мере необходимости. Несколько полезных пакетов описаны в табл. G.1.

Таблица G.1. Пакеты R, предназначенные для доступа к обширным наборам данных

Пакет	Описание
<code>ff</code>	Структурирует данные таким образом, что они хранятся на жестком диске, а «ведут себя» так, как будто находятся в оперативной памяти
<code>bigmemory</code>	Поддерживает создание, хранение обширных матриц, доступ к ним и манипуляции с ними. Эти матрицы размещены в разделяемой памяти и распределенных в памяти файлах
<code>filehash</code>	Реализует идею простых баз данных типа «ключ-значение», где ключи из символьных цепочек ассоциированы со значениями данных, хранящихся на диске
<code>ncdf</code> , <code>ncdf4</code>	Предоставляет интерфейс для файлов данных формата Unidata netCDF
<code>RODBC</code> , <code>RMySQL</code> , <code>ROracle</code> , <code>RPostgreSQL</code> , <code>RSQlite</code>	Каждый из этих пакетов дает возможность доступа к системам управления внешними реляционными базами данных

Перечисленные выше пакеты помогают преодолеть ограничения по хранению данных в памяти R. Однако при попытках анализа обширных наборов данных за разумное время требуются специализированные методы. Некоторые из наиболее полезных методов описаны ниже.

G.3. Аналитические пакеты для больших объемов данных

В R есть несколько пакетов, предназначенных для анализа больших объемов данных:

- Пакеты `biglm` и `speedglm` предназначены для подбора линейных или обобщенных линейных моделей для больших наборов данных с эффективным использованием памяти. В этих пакетах реализованы аналоги функций `lm()` и `glm()` для работы с большими объемами данных.
- В некоторых пакетах есть аналитические функции для работы с обширными матрицами, создаваемыми при помощи пакета `bigmemory`. Пакет `biganalytics` позволяет проводить кластерный анализ k -средних, вычислять статистики для переменных и создает программную оболочку для пакета `biglm`. Пакет `bigtabulate` содержит аналоги функций `table()`, `split()` и `tapply()`, а пакет `bigalgebra` позволяет применять усовершенствованные функции линейной алгебры.
- В пакете `biglars` в сочетании с пакетом `ff` представлены возможности регрессии наименьшего угла (least-angle), лассо и ступенчатой регрессии для наборов данных, которые слишком велики для хранения в памяти.
- Пакет `Broddingnag` может применяться для работы с большими числами (больше, чем 2^{1024}).

Обработка наборов данных объемом от гигабайта до терабайта затруднительна для любых программ. Для получения дальнейшей информации о методах обработки больших объемов данных в R обратитесь к соответствующему тематическому разделу архива CRAN «Высокопроизводительные и параллельные вычисления в R» (cran.r-project.org/web/views/: High-Performance and Parallel Computing with R).



ПРИЛОЖЕНИЕ Н. Обновление версии R

С позиции потребителей, мы воспринимаем как должное возможность обновления программы при помощи опции «Проверить наличие обновлений». В первой главе я отметил, что скачать и установить последнюю версию дополнительных пакетов можно при помощи функции `update.packages()`. К сожалению, для обновления самой программы подобной функции не существует. Если вы хотите заменить версию R с 4.1.0 на 5.1.1, придется проявить изобретательность. (На момент написания книги текущая версия программы имеет номер 2.13.0, но я хочу, чтобы эта книга казалась современной спустя годы.)

Скачать и установить последнюю версию R из архива CRAN (<http://cran.r-project.org/bin/>) относительно просто. Все осложняется тем, что настройки (в том числе и уже имеющиеся дополнительные пакеты) не будут включены в новую версию программы. У меня установлено 248 дополнительных пакетов, и я действительно не хочу выписывать их названия и загружать их вручную, когда придет время обновить R.

В Сети долго дискутировали о наиболее изящном и рациональном способе обновления R. Описанный ниже способ ни изящен, ни рационален, но зато я обнаружил, что он хорошо действует в разных операционных системах (Windows, Mac и Linux).

При этом подходе для сохранения списка всех установленных пакетов вне системы директорий R используется функция `installed.packages()`. Затем при помощи функции `install.packages()` последние версии этих пакетов скачиваются и устанавливаются в обновленной версии программы. Вот этапы этой процедуры:

1. Если вы изменяли файл `Rprofile.site` (см. приложение B), сохраните его копию за пределами системы директорий R.
2. Запустите текущую версию R и введите следующие команды:

```
oldip <- installed.packages() [,1]
save(oldip, file="path/installerInstalledPackages.Rdata"),
```

где *path* – это директория вне папки R.

3. Скачайте и установите новую версию R.
4. Если вы сохранили измененный файл *Rprofile.site* на этапе 1, скопируйте его в новую версию программы.
5. Запустите новую версию R и введите следующие команды:

```
load("path/installerInstalledPackages.Rdata")
newip <- installed.packages() [,1]
for(i in setdiff(oldip, newip))
  install.packages(i),
```

где *path* – это местонахождение файла, указанное на этапе 2.

6. Удалите старую версию программы (по желанию).

При помощи этого способа можно установить только те пакеты, которые находятся в архиве CRAN. Вам придется отдельно найти и установить пакеты, которые расположены в других хранилищах. К счастью, в конце описанной выше процедуры установки пакетов появится список пакетов, которые не могут быть установлены. Когда я последний раз обновлял программу, не были обнаружены пакеты *globaltest* и *BioBase*. Поскольку я получил их с сайта Bioconductor, эти пакеты можно установить при помощи следующего программного кода:

```
source(http://bioconductor.org/biocLite.R)
biocLite("globaltest")
biocLite("BioBase")
```

На шестом этапе вам было предложено при желании удалить старую версию программы. Под Windows одновременно может быть установлено более одной версии R. При необходимости старую версию можно удалить при помощи вкладки «Удаление программ» на панели управления. В операционных системах Mac и Linux новая версия R будет записана поверх старой. Для удаления следов от старой версии под Mac перейдите в директорию */Library/Frameworks/R.frameworks/versions/* и удалите папку, соответствующую старой версии. В Linux, наверное, лучше оставить все как есть.

Очевидно, что обновление текущей версии R требует больше усилий, чем это необходимо для такой совершенной программы. Я надеюсь, что когда-нибудь в этом приложении будет просто написано «Для обновления текущей версии R выберите опцию “Проверить наличие обновлений”».



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.
- Bandolos, 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](http://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.
- 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.
- 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 Three-Dimensional 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. Ferreira, 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](http://www.resample.com/content/text/index.shtml)
- 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.



УКАЗАТЕЛЬ ПАКЕТОВ И ФУНКЦИЙ

Символ

! оператор 118
!= оператор 118
символ 36
%a символ 124
%A символ 124
%B символ 124
%b символ 124
%d символ 124
%m символ 124
%Y символ 124
%y символ 124
* оператор 116, 246
** оператор 116
... операция 97, 100
. символ 246
/ оператор 116
:

символов 246

? функция 11

?? функция 11

^ оператор 116, 246, 250

_ символ 246

+ оператор 116, 246

< оператор 118

<<- оператор 62

<= оператор 118

== оператор 118

> оператор 118

>= оператор 118

-1 символ 246

A

abline() функция 99, 358
abs() функция 139

acos() функция 140
acosh() функция 140
AER пакет 561
aggr() функция 479
aggregate() функция 164, 327
AIC() функция 247, 286
Amelia пакет 490, 494, 561
analytic пакет, 573
ancova() функция 317
anova() функция 247, 285, 307,
428
Anova() функция, 307, 326
aov() функция 304
apply() функция 151
apropos() функция 40
aq.plot() функция 330
arrayImpute пакет 495, 561
arrayMissPattern пакет 495, 561
as.character() функция 126
as.тип_данных()функция 127
as.Date() функция 124
asin() функция 140
asinh() функция 140
asyropow пакет 353
atan() функция 140
atanh() функция 140
attach() функция 60, 133
avPlots() функция 265, 278
axis() функция 96

B

barplot() функция 172, 175–177



biganalytics пакет 573
 biglars пакет 573
 bigmemory пакет 572
 bigtabulate пакет 573
 bmp() функция 83
 boot пакет 409
 bootstrap пакет 293
 boxplot() функция 83, 325
 boxplot.stats() функция 189
 boxTidwell() функция 283
 Broddingnag пакет 573
 bzfile() функция 70

C

c() функция 37, 55, 79
 ca пакет 562
 car пакет 252, 254, 260, 269, 562
 cast() функция 166
 cat() функция 150, 162
 cat пакет 495, 562
 cbind() функция 79, 129,
 154–156, 327
 ceiling() функция 139
 class() функция 79
 cld() функция 311
 close() функция 76
 cm.colors() функция 90
 cmdscale() функция 470
 coefficients() функция 247, 423
 col.corrgram() функция 387
 colors() функция 90
 complete() функция 494
 confint() функция 247, 260
 contrasts() функция 332
 contr.helmert функция 332
 contr.poly функция 332
 contr.SAS функция 332
 contr.sum функция 332
 contr.treatment функция 332
 cooks.distance() функция 49

cor() функция 222
 corrrgram() функция 383, 387
 corrrgram пакет 383, 562
 corrgram пакет 562
 cos() функция 140
 cosh() функция 140
 cov() функция 222
 cov2cor() функция 460
 cpairs() функция 365
 crossval() функция 393
 crPlots() функция 264, 269
 cut() функция 119, 150, 507

D

data() функция 40
 data.frame() функция 59
 subset() функция 133–134
 date() функция 125
 demo() функция 39
 density() функция 186
 densityplot() функция 504
 detach() функция 60
 dev.new() функция 84
 dev.next() функция 84
 dev.off() функция 43, 84
 dev.prev() функция 84
 dev.set() функция 84
 diff() функция 141
 difftime() функция 126
 dim() функция 79
 dir.create() функция 42
 dmat.color() функция, 365
 doBy пакет 119, 205, 562
 dotchart() функция 194
 durbinWatsonTest() функция 264,
 269

E

edit() функция 68
 effect() функция 258, 315

effects пакет 315, 562
ES.w2() функция 347
example() функция 40
exp() функция 140
GPArotation пакет 468
nFactors пакет 468, 566
expression() функция 516

F

fa() функция 446, 462, 467
factanal() функция 446
FactoMineR пакет 468, 562
factor() функция 63, 78
factor.plot() функция 446, 466
fa.diagram() функция 446, 466
FAiR пакет 468, 562
fan.plot() функция 181
fa.parallel() функция 446, 449,
 460

fCalendar пакет 126, 562
ff пакет 572
fgui пакет 542
file() функция 70
filehash пакет 572
First() функция 543
fitted() функция 247
fix() функция 79
FlexMix пакет 469
floor() функция 139
foreign пакет 72, 546, 563
format() функция 125

G

gap пакет 354
gclus пакет 46, 365, 563
getwd() функция 41, 543
ggplot2 пакет 501, 520, 563
glht() функция 310
glm() функция 431
glmPerm пакет 563
gls() функция 326

gmodels пакет 563
GPArotation пакет 468
gplots пакет 177, 300, 320, 563
gray() функция 90
grep() функция 72, 149
grid пакет 499, 563
gsub() функция 72
gylma() функция 272
gylma пакет 266, 272, 563
gzfile() функция 70

H

hdf5 пакет 74, 564
head() функция 79
heat.colors() функция 90
help() или ? функция 40
help.search() или ?? функция 40
help.start() функция 40
hexbin() функция 369
hexbin пакет 369, 564
HH пакет 300, 313, 564
hist() функция 83
history() функция 41
Hmisc пакет 72, 99, 495, 564
hsv() функция 90

I

ibar() функция 530
ibox() функция 530
identify() функция 527
IDPmisc пакет 370
ihist() функция 530
imap() функция 530
imosaic() функция 530
influencePlot() функция 265, 279
installed.packages() функция 46,
 574
install.packages() функция 46,
 574
interaction2wt() функция 321
interaction.plot() функция 320

ipairs() функция 370
 ipcp() функция 530
 iplot() функция 369, 530
 iplots пакет 500, 527, 530
 is.tip_данных() функция 127
 is.infinite() функция 476
 is.na() функция 121, 476
 is.nan() функция 476
 isoMDS() функция 470

J

jpeg() функция 83

K

kmi пакет 495, 564

L

lapply() функция 152
 Last() функция 543
 lattice пакет 84, 499–500, 564
 latticist пакет 500, 529, 564
 lavaan пакет 469, 564
 layout() функция 105–108
 lcda пакет 469, 564
 lcm() функция 109
 lcmm пакет 469
 leaps пакет 289, 565
 legend() функция 100, 187
 length() функция 79, 150
 .libPaths() функция 45, 544
 library() функция 45–46
 line() функция 98
 lines() функция 98, 177, 379
 list() функция 65
 lm() функция 245, 254, 259
 lme4 пакет 326
 lmer() функция 326
 lmPerm пакет 565
 load() функция 41
 loadhistory() функция 41
 loess() функция 358

log() функция 140
 log10() функция 140
 logregperm пакет 565
 longitudinalData пакет 495, 565
 lowess() функция 358
 ls() функция 41, 79
 lsa пакет 469, 565
 ls.objects() функция 571
 ltm пакет 565
 lubridate пакет 126, 565

M

mad() функция 141
 manova() функция 328
 marginplot() функция 481
 MASS пакет 146, 228, 326, 565
 matrixplot() функция 479
 max() функция 141
 md.pattern() функция 478
 mean() функция 141, 151, 154, 477
 median() функция 141
 melt() функция 167
 mi пакет 490, 494
 mice() функция 490
 mice пакет 473, 478, 490, 498
 min() функция 141
 minor.tick() функция 99
 mix пакет 495
 mlogit пакет 565
 mode() функция 79
 mosaic() function, 388
 mosaicplot() функция 388
 mtext() функция 98, 102
 multcomp пакет 307, 314, 566
 mvnmle пакет 495, 566
 mvoutlier пакет 300, 566
 mvrnorm() функция 146

N

names() функция 59, 120
 na.omit() функция 123

ncdf пакет 74, 566, 572
ncdf4 пакет 74, 566, 572
nchar() функция 148
ncvTest() функция 264, 270
nFactors пакет 468, 566
nlme пакет 326
prmc пакет 566

О

odbcConnectExcel() функция 71
odbcConnect() функция 75
odfTable() функция 555
odfWeave пакет 546
openMx пакет 469, 566
options() функция 41, 333
order() функция 128, 156
order.clusters() функция 47
order.single() функция 365
outlierTest() функция 265, 275,
313

П

pairs() функция 361
par пакет 495
panel.abline() функция 511
panel.grid() функция 510
panel.lmline() функция 511
panel.loess() функция 511
panel.rug() функция 510
panel.xyplot() функция 510, 511
par() функция 86, 94, 104–106,
177, 187, 518
paste() функция 131, 149
pastecs пакет 566
pdf() функция 83
pedantics пакет 354
pie3D() функция 180
piface пакет 352, 567
playwith пакет 527, 567
plot() функция 61, 85, 247, 260,
519

plot3d() функция 373
plotmeans() функция 308, 320
plot.new() функция 112
plotrix пакет 180
png() функция 83
poLC пакет 469
poLCA пакет 567
polygon() функция 186
pool() функция 490
postscript() функция 83
powerpkg пакет 353
powerSurvEpi пакет 353
predict() функция 424, 430
pretty() функция 150
principal() функция 446, 451, 456
princomp() функция 446
proft пакет 572
prooftools пакет 572
psych пакет 446, 567
pwr.2p2n.test() функция 340
pwr.2p.test() функция 340, 345
pwr.anova.test() функция 340,
342, 349
pwr.chisq.test() функция 340, 346
pwr.f2.test() функция 340, 344
PwrGSD пакет 353
pwr.p.test() функция 340
pwr.r.test() функция 340, 343, 350
pwr.t2n.test() функция 340
pwr.t.test() функция 340

Q

q() функция 37, 41
qcc пакет 567
qplot() функция 521
qqPlot() функция 264, 266
quantile() функция 141, 155
quartzFonts()функция 92

Р

R2HTML пакет 548

rainbow() функция 90, 180
range() функция 141
rbind() функция 79, 130, 316
readLines() функция 72
read.spss() функция 72
read.ssdi() функция 72
read.table() функция 69, 571
read.xlsx() функция 71
recode() функция 119
recodevar() функция 119
regsubsets() функция 289
relaimpo пакет 295
relweights() функция 296
rename() функция 119
rep() функция 150
residplot() функция 267
residuals() функция 247, 424
reshape пакет 119, 165, 567
rggobi пакет 532, 568
rgl пакет 373, 568
RJDBC пакет 568
rm() функция 41, 79
rms пакет 568
RMySQL пакет 76, 572
robust пакет 433, 568
RODBC пакет 71, 568
ROracle пакет 76, 568, 572
round() функция 139
RPostgreSQL пакет 76, 572
Rprof() функция 572
rrcov пакет 300, 568
RSiteSearch() функция 40
RSQLite пакет 76, 572
runif() функция 145

S

sample() функция 134
sampling пакет 135, 568
sapply() функция 152, 200
sas.get() функция 73

save() функция 41
savehistory() функция 41
save.image() функция 41
scale() функция 142, 154, 295
scan() функция 571
scatter3d() функция 374
scatterplot() функция 252, 265, 359
scatterplot3d() функция 371, 568
scatterplotMatrix() функция 254, 265, 361
scree() функция 446
sd() функция 141
sem пакет 469, 568
seq() функция 150
SeqKnn пакет 495, 569
set.seed() функция 145
setwd() функция 41
show.settings() функция 518
shrinkage() функция 293
signif() функция 140
sin() функция 140
sinh() функция 140
sink() функция 43
sm packages 186, 569
sm.density.compare() функция 186
smoothScatter() функция 367, 370
source() функция 43
speedglm пакет 573
spine() функция 178
split() функция 573
spreadLevelPlot() функция 265, 270, 283
sqldf() функция 135
sqldf пакет 135
sqlDrop() функция 76
sqlFetch() функция 75
sqlQuery() функция 75
sqlSave() функция 75
sqrt() функция 139

ssize.fdr пакет 354
stepAIC() функция 287
stop() функция 162
str() функция 65, 79
strsplit() функция 149, 155
sub() функция 148
subset() функция 133
subsets() функция 289
substr() функция 148
sum() функция 123, 141, 477
summary() функция 65, 200, 247,
 259
summary.aov() функция 328
summaryRprof() функция 572
Sweave() функция 548, 552
symbols() функция 375
Sys.Date() функция 125
Sys.getenv() функция 543
system.time() функция 572

T

t() функция 163
table() функция 172, 210, 573
tail() функция 79
tan() функция 140
tanh() функция 140
tapply() функция 573
terrain.colors()функция 90
text() функция 102
tiff() функция 83
title() функция 95, 176, 187
tolower() функция 149
topo.colors()функция 90
toupper() функция 149
transform() функция 117
trellis.par.get() функция 518
trellis.par.set() функция 518
trunk() функция 139
twiddler пакет 542

U

unz() функция 70
update() функция 507
update.packages() функция 574
url() функция 70

V

var() функция 141
vcd пакет 49, 172, 388, 567
vcov() функция 247
vegan пакет 569
vif() функция 265, 274
vignette() функция 40
VIM пакет 473, 479, 569
vioplot() функция 193
vioplot пакет 193

W

warning() функция 162
which() функция 133
Wilks.test() функция, 330
windowsFont() функция 91
win.metafile() функция 83
with() функция 60, 119, 490
within() функция 119
write.foreign() функция 546
write.table() функция 545
write.xlsx() функция 545

X

xfig() функция 83
xlsx пакет 71, 545, 569
XML пакет 71, 569
xtable() функция 550, 554
xtable пакет 550
xyplot() функция 504, 508
xzfile() функция 70

Книги издательства «ДМК Пресс» можно заказать в торгово-издательском холдинге
«Планета Альянс» наложенным платежом, выслав открытку или письмо по почтовому адресу:
115487, г. Москва, 2-й Нагатинский пр-д, д. 6А

При оформлении заказа следует указать адрес (полностью), по которому
должны быть высланы книги; фамилию, имя и отчество получателя.

Желательно также указать свой телефон и электронный адрес.

Эти книги вы можете заказать и в интернет-магазине: www.aliants-kniga.ru.

Оптовые закупки: тел. (499) 782-38-89

Электронный адрес: books@aliants-kniga.ru.

Роберт И. Кабаков

R в действии

Анализ и визуализация данных в программе R

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com

Перевод с английского *Волкова П. А.*

Корректор *Синяева Г. И.*

Верстка *Паранская Н. В.*

Дизайн обложки *Мовчан А. Г.*

Гарнитура «Петербург». Печать офсетная.
Усл. печ. л. 36,75. Тираж 200 экз.

Web-сайт издательства: www.dmk.ru