

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/337632058>

Статистическа обработка на данни с R : Практическо ръководство

Book · November 2019

CITATIONS

0

READS

157

3 authors:



Todor Balabanov

Bulgarian Academy of Sciences

136 PUBLICATIONS 83 CITATIONS

[SEE PROFILE](#)



Zornitsa Atanassova

Institute of Information and Communication Technologies

4 PUBLICATIONS 3 CITATIONS

[SEE PROFILE](#)



Rumen Ketipov

Bulgarian Academy of Sciences

12 PUBLICATIONS 2 CITATIONS

[SEE PROFILE](#)

Some of the authors of this publication are also working on these related projects:



ESGI 113 - Sofia, Bulgaria [View project](#)



ESGI 120 - Sofia, Bulgaria [View project](#)

Статистическа обработка на данни с R

Практическо ръководство

Тодор Балабанов, Зорница Атанасова, Румен Кетипов

Издателство “Образование и Познание”
София • 2019

eISBN: 978-619-7515-17-6



Авторски права © 2019

Тодор Балабанов, Зорница Атанасова, Румен Кетипов

ИЗДАТЕЛСТВО “ОБРАЗОВАНИЕ И ПОЗНАНИЕ”

[HTTPS://WWW.OBRAZOVANIEBG.NET/](https://www.obrazovaniebg.net/)

Разпространява се под свободен лиценз:

Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International Public License

<https://creativecommons.org/licenses/by-nc-nd/4.0>

Първо издание, 2019

Теми

Списък на фигурите	vi
Списък на листингите	x
Списък на формулите	xi
Списък на таблиците	xi
Предговор	1
1 Инсталация и стартиране	3
2 Пакетна организация, променливи, основни математически операции в R и типове данни	18
3 Сложни структури от данни и извикване на функции	33
4 Въвеждане на данни и извеждане на графики	48
5 Оператори за контрол на изпълнението и потребителски функции	62
6 Групиране и обхождане на данни	74
7 Реорганизация на данните и обработка на символни низове	97
8 Разширени графични възможности и вероятностни разпределения	106
9 Статистическа обработка на данните	137
10 Приблизени пресмятания - подходи, методи, алгоритми	153
11 Оформление на резултатите за печатно и електронно представяне	168
Заклучение	173
Библиография	174

Съдържание

Списък на фигурите	vi
Списък на листингите	x
Списък на формулите	xi
Списък на таблиците	xi
Предговор	1
1 Инсталация и стартиране	3
1.1 Изтегляне на инсталационните файлове	3
1.2 Инсталация	8
1.3 Работа в режим на команди	14
2 Пакетна организация, променливи, основни математически операции в R и типове данни	18
2.1 Инсталиране на пакети	18
2.2 Зареждане на пакети	23
2.3 Основни математически операции	24
2.4 Типове и променливи	27
2.4.1 Създаване на променливи	27
2.4.2 Премахване на променливи	27
2.4.3 Дискретно представяне на информацията	28
2.4.4 Числени стойности	29
2.4.5 Символни низове	30
2.4.6 Астрономическо време	30
2.4.7 Логически стойности	31
3 Сложни структури от данни и извикване на функции	33
3.1 Извикване на функции	33
3.2 Вектори	34
3.3 Липсващи стойности	37
3.4 Рамкирани данни	38
3.5 Списъци	41
3.6 Матрици	44
3.7 Масиви	46
4 Въвеждане на данни и извеждане на графики	48
4.1 Въвеждане на данни от външни източници	48
4.1.1 CSV файлове	48
4.1.2 Excel файлове	50
4.1.3 SQL бази данни	50
4.1.4 Други статистически програми като източници на данни	53

4.1.5	Бинарни файлове на R	53
4.1.6	Данни достъпни директно от R	54
4.1.7	Четене на данни от JSON формат	55
4.2	Визуализация на данни от статистически анализ	55
4.2.1	Хистограма	56
4.2.2	Диаграма на разсейване	58
4.2.3	Графики тип кутия	60
5	Оператори за контрол на изпълнението и потребителски функции	62
5.1	Оператори за преход	66
5.1.1	Оператор за условен преход	67
5.1.2	Алтернатива при условен преход	67
5.1.3	Каскада от условни преходи	68
5.1.4	Оператор за многовариантен избор	68
5.2	Оператори за цикъл	69
5.2.1	Цикъл за обхождане	69
5.2.2	Цикъл с условие за край	69
5.2.3	Прекъсване на циклите	70
5.3	Потребителски функции	71
5.3.1	Аргументи на функция	71
5.3.2	Аргументи с подразбираща се стойност	72
5.3.3	Променлив брой аргументи	72
5.3.4	Върната стойност	72
5.3.5	Предаване на функция като аргумент	73
6	Групиране и обхождане на данни	74
6.1	Фамилията функции apply	74
6.1.1	apply	74
6.1.2	lapply и sapply	75
6.1.3	mapply	76
6.1.4	Агрегация	76
6.2	Пакетът plyr	78
6.2.1	ddply	78
6.2.2	llply	80
6.2.3	Помощни функции и бързодействие	81
6.3	Пакетът data.table	81
6.3.1	Ключове	84
6.3.2	Агрегация	86
6.4	Бързи операции с пакета dplyr	87
6.4.1	Потоци и таблици	88
6.4.2	Извличане по колони	88
6.4.3	Филтриране	89
6.4.4	Модификация, обобщение, групиране и подреждане	90
6.4.5	Специфични изчисления и връзка с база данни	92
6.5	Пакетът rpgrr	94
6.5.1	Фамилията функции map	94
7	Реорганизация на данните и обработка на символни низове	97
7.1	Обединяване на множества от данни	97
7.1.1	Функция merge	98
7.1.2	Функция join	99
7.1.3	Транспониране на данните	99
7.2	Сложни сливания на данни и трансформация на форматите	101
7.2.1	Обединение на редове и колони	101

7.2.2	Сложни сливания на данни	101
7.2.3	Реформатиране на данните	102
7.3	Работа със символни низове	103
7.3.1	Формиране на текст	103
7.3.2	Извличане на текст	104
8	Разширени графични възможности и вероятностни разпределения	106
8.1	Разширени графични възможности	106
8.1.1	Хистограми и плътности	106
8.1.2	Диаграми на разсейване	109
8.1.3	Графики тип кутия и цигулка	113
8.1.4	Линейни графики	117
8.1.5	Тематично оформление	119
8.2	Изследване на случайни величини	123
8.3	Вероятностни разпределения	128
8.3.1	Нормално разпределение	129
8.3.2	Биномно разпределение	131
8.3.3	Поасоново разпределение	134
8.3.4	Други разпределения	135
9	Статистическа обработка на данните	137
9.1	Описателна статистика	137
9.1.1	Средна стойност	137
9.1.2	Минимална стойност, максимална стойност, медиана и мода	138
9.1.3	Дисперсия и стандартно отклонение	139
9.1.4	Квантили и обобщение	139
9.2	Сравнителна статистика	139
9.2.1	Ковариация и корелация	139
9.2.2	Тест на Стюдънт	141
9.2.3	Дисперсионен анализ	147
9.3	Линейна регресия	149
10	Приближени пресмятания - подходи, методи, алгоритми	153
10.1	Монте-Карло методи	153
10.2	Генетични алгоритми	156
10.3	Изкуствени неверонни мрежи	161
11	Оформление на резултатите за печатно и електронно представяне	168
11.1	Работа с LaTeX	168
11.2	Работа с RMarkdown	170
11.2.1	Статични документи	171
11.2.2	Динамични документи	171
	Заклучение	173
	Библиография	174
	Азбучен указател	176

Списък на фигурите

1.1	Начална уеб страница на продукта	4
1.2	Списък със сървъри за изтегляне	5
1.3	Избор на подходяща инсталация за операционната система	6
1.4	Избор на версия за изтегляне	7
1.5	Запазване на инсталационния файл	8
1.6	Активиране на инсталатора	9
1.7	Подробности за инсталацията	10
1.8	Лицензно споразумение за ползване	11
1.9	Изрично съгласие	12
1.10	Информация за директорията и използваното дисково пространство	13
1.11	Успешно приключване на инсталацията	14
1.12	Основен прозорец на продукта	15
1.13	Изпълнение на команда за печат	16
2.1	Команда за инсталиране на пакета coefplot	19
2.2	Избор на сървър за изтегляне на пакета	20
2.3	Зависимости между пакетите	21
2.4	Резултат от инсталацията на пакета	22
2.5	Зареждане на пакета coefplot	23
2.6	Премахване на пакета coefplot от общата памет	24
2.7	Примерни аритметични операции	25
4.1	Хистограма на каратите	57
4.2	Хистограма на каратите при 100 групи	58
4.3	Диаграма на разсейване за камъните според отношението тегло към цена	59
4.4	Графика тип кутия за теглото на диамантите	60
5.1	Текстов редактор към продукта R	63
5.2	Стартиране на R скрипт	64
5.3	Резултат от изпълнението на R скрипт	65
5.4	Резултат от изпълнението на R скрипт в конзолата на операционната система	66
7.1	Разход на пари по програми и години	100
8.1	Хистограма при 30 групи	107
8.2	Хистограма при 100 групи	108
8.3	Плътносна функция	109
8.4	Диаграма на разпръскване с ggplot2	110
8.5	Диаграма на разпръскване по групи за цвят на диамантите	111
8.6	Визуализация с групиране по два признака	112
8.7	Визуализация на хистограми с групиране	113
8.8	Визуализация на характеристиката за дълбочина на диамантите	114
8.9	Дълбочина на диамантите в групи според сръза	115
8.10	Графика тип цигулки	116
8.11	Добавяне на декорация с точки	117

8.12	Нарастване на популацията във времето	118
8.13	Визуализиране на приръста по години	119
8.14	Тема Wall Street Journal	120
8.15	Тема Edward Tufte	121
8.16	Тема в стил Microsoft Excel	122
8.17	Тема Economist	123
8.18	Хистограма на хвърлянията за един зар	125
8.19	Хистограма на хвърлянията за два зара	126
8.20	Хистограма на хвърлянията за шест зара	127
8.21	Плътностна диаграма на хвърлянията за десет зара	128
8.22	Плътностна функция на нормално разпределение	130
8.23	Кумулативна функция на нормално разпределение	131
8.24	Хистограма на биномно разпределение	133
8.25	Хистограма на Поасоново разпределение	135
8.26	Списък с най-използваните вероятностни разпределения	136
9.1	Т-статистика за бакшиши	142
9.2	Разпределение на бакшишите според пола на сервитьора	145
9.3	Визуализация на средните при сдвоения тест	147
9.4	Линейна регресия бащи-синове	151
10.1	Пресмятане на числото π с Монте-Карло метод	154
10.2	Сходимост на процеса по оптимизация	161
10.3	Схема на биологична нервна клетка	163
10.4	Схема на изкуствен неврон	163
10.5	Функции за активация на изкуствен неврон	164
10.6	Трислойна изкуствена невронна мрежа	166

Списък на листингите

2.1	Събиране	25
2.2	Унарен минус	25
2.3	Аритметичен израз с две събирания	26
2.4	Израз за каскадно присвояване	26
2.5	Контекстна зависимост на операциите	26
2.6	Контекстна зависимост на операцията за делене	26
2.7	Приоритет на операциите	26
2.8	Смяна на приоритета	27
2.9	Операции за присвояване	27
2.10	Алтернативи за операцията присвояване	27
2.11	Премахване на променливи от глобалната памет	28
2.12	Грешка от препълване при събиране	28
2.13	Проверка за типа на променливата	29
2.14	Проверка за типа numeric	29
2.15	Използване на целочислен тип	29
2.16	Символни низове в R	30
2.17	Дължина на символен низ или числена стойност	30
2.18	Типове данни за време	30
2.19	Логически тип данни	31
2.20	Операции за сравнение	32
3.1	Извикване на функции	33
3.2	Документация за функциите	33
3.3	Документация за операции	34
3.4	Частично търсене	34
3.5	Вектор от числа и вектор от символни низове	34
3.6	Базови операции над вектори	34
3.7	Алтернативен синтаксис за създаване на вектори	35
3.8	Операции между вектори с еднаква дължина	35
3.9	Проверка дали някоя или всички стойности от вектора отговарят на определено условие	36
3.10	Достъп до отделни елементи на вектор	36
3.11	Имена на елементите на вектора	36
3.12	Трансформация на вектор във фактор	36
3.13	Вектори с латинските букви	37
3.14	Липсващи стойности	37
3.15	Липсващи стойности	38
3.16	Създаване на рамкирани данни	38
3.17	Създаване на рамкирани данни с имена на колоните	39
3.18	Атрибути на рамкираните данни	39
3.19	Фактори в рамковите данни	40
3.20	Вътрешно представяне на факторите	41
3.21	Създаване на списък	41
3.22	Вектор в списък	41
3.23	Списък с разнородни данни	42
3.24	Названия на елементите в списъка	42

3.25	Достъп до елементите на списъка	43
3.26	Вложено позоваване	43
3.27	Добавяне на елемент	44
3.28	Създаване на матрици	44
3.29	Операции с матрици	45
3.30	Матрично умножение	45
3.31	Имена на колоните и редовете	46
3.32	Работа с масиви	46
4.1	Зареждане на данни от CSV файл	48
4.2	Проверка на типовете които колоните имат	49
4.3	Зареждане на символни низове	49
4.4	Работна директория	50
4.5	Сваляне на файл с данни	51
4.6	Връзка към базата данни	51
4.7	Изследване на базата данни	51
4.8	Изследване на базата данни	52
4.9	Прекъсване на връзката към базата данни	52
4.10	Използване на множество от данни	53
4.11	Запис и четене в RData файл	53
4.12	Запис и четене на един обект	54
4.13	Зареждане на примерни данни	54
4.14	Четене на JSON данни	55
4.15	Четене на JSON данни	55
4.16	Генериране на хистограма за теглото на камъните	56
4.17	Хистограма с повече групи	57
4.18	Генериране на диаграма на разсейване	58
4.19	Алтернативна команда за диаграма на разсейване	59
4.20	Генериране на графика от тип кутия	60
5.1	Адрес на примерен R скрипт	62
5.2	Изпълнение на R скрипт от конзолата на операционната система	65
5.3	Оператор за условен преход if	67
5.4	Оператор за условен преход if-else	67
5.5	Каскада от if-else	68
5.6	Функцията ifelse	68
5.7	Конструкция за многовариантен избор switch	68
5.8	Оператор за цикъл for	69
5.9	Обхождане на вектор от символни низове	69
5.10	Цикъл с условие за край	69
5.11	Прекъсване на итерация	70
5.12	Прекъсване на цикъла	70
5.13	Примерна потребителска функция	71
5.14	Извикване на функция с аргумент	71
5.15	Извикване на функция с повече аргументи	71
5.16	Извикване на функция с подразбиращи се аргументи	72
5.17	Функция с променлив брой аргументи	72
5.18	Връщане на стойност от функция	72
5.19	Избор на функция за извикване по време на изпълнение	73
6.1	Сума по редове и колони	74
6.2	Сума на обекти в списък	75
6.3	Проверка за идентичност на елементите	76
6.4	Групиране на данни	76
6.5	Групиране по повече от една колона	77
6.6	Прилагане на агрегатна функция върху повече колони в едни и същи групи	77
6.7	Използване на повече колони от двете страни на формулата	77

6.8	Бейзболна статистика	78
6.9	Корекция на данните	79
6.10	Пресмятане на ОВР	79
6.11	Пресмятане на ОВР за цялата кариера на играча	80
6.12	Сума на всеки елемент в списък	80
6.13	Повече от една агрегатна функция	81
6.14	Бързодействие при използване на референции	81
6.15	Създаване на data.table	82
6.16	Зареждане на data.table от data.frame	82
6.17	Достъп до редовете	83
6.18	Достъп до колоните	83
6.19	Операции с таблици	84
6.20	Агрегатни функции	86
6.21	Поточни операции	88
6.22	Таблични данни в dplyr	88
6.23	Избор на редове	88
6.24	Търсене по частично съвпадение	89
6.25	Изключване на колони	89
6.26	Филтриране на редове	89
6.27	Избор по индекси	90
6.28	Модификация на колони	90
6.29	Отразяване на модификациите	91
6.30	Обобщаваща информация	91
6.31	Групиране при обобщение	91
6.32	Подредба на резултатите	92
6.33	Специфични изчисления	92
6.34	Работа с база данни	93
6.35	Пресмятания с данни в база данни	94
6.36	Прилагане на функцията map	94
6.37	Извиквания на map според типа на върнатата стойност	95
6.38	Обхождане на data.frame	95
6.39	Едновременно обхождане на два списъка	96
7.1	Обединяване на множества от данни	97
7.2	USAID множество от данни	97
7.3	Зареждане USAID данните в R	98
7.4	Сливане на данни с merge	98
7.5	Сливане на данни при data.table	98
7.6	Сливане на данни с join	99
7.7	От колони към редове	99
7.8	От редове към колони	100
7.9	Обединяване по колони и редове	101
7.10	Сложни сливания	101
7.11	Данни за реакциите	102
7.12	Свиване от колони в редове	102
7.13	Разпъване от редове в колони	103
7.14	Конкатенация на символни низове	103
7.15	Разпъване от редове в колони	104
7.16	Извличане на текст	104
7.17	Регулярни изрази	104
7.18	Сложни регулярни изрази	105
8.1	Хистограма и плътност	106
8.2	Диаграма на разсейване с ggplot2	109
8.3	Диаграма на разпръскване групирани по признак	110
8.4	Визуализация тип кутия	113

8.5	Линейни графики	117
8.6	Избор на теми за визуално представяне	119
8.7	Случайни величини със зарове	123
8.8	Нормално разпределение	129
8.9	Биномно разпределение	132
8.10	Разпределение на Поасон	134
9.1	Генериране на извадка от случайни числа	137
9.2	Средна стойност	137
9.3	Претеглена средна стойност	138
9.4	Минимум, максимум, медиана и мода	138
9.5	Дисперсия и стандартно отклонение	139
9.6	Квантили и обобщение	139
9.7	Ковариация и корелация	140
9.8	Тестово множество за бакшиши	141
9.9	Тест на единична извадка	141
9.10	Визуализация на t-разпределение	142
9.11	Едностранны t-статистика	143
9.12	Сравнение на две извадки	143
9.13	T-тест на сдвоени данни	145
9.14	ANOVA тест	147
9.15	Линейна регресия	149
10.1	Монте-Карло пресмятане	155
10.2	Оптимизация на задачата за раницата с генетични алгоритми	158
10.3	Анимирана визуализация на процеса по търсене на оптимално решение	159
10.4	Класифициране с изкуствена невронна мрежа	165
11.1	Инструкция за R фрагмент в LaTeX документ	169
11.2	Транслиране от Rnw до Tex	169
11.3	Транслиране от Tex до PDF	169
11.4	Адрес на примерени LaTeX документи	169
11.5	Линейна регресия на разходи спрямо спестявания	169
11.6	Адрес на примерни RMarkdown документ	171
11.7	Транслиране от RMarkdown в HTML и PDF	171
11.8	Адрес на примерни интерактивни документ	172
11.9	Създаване на интерактивни документи	172

Списък на таблиците

4.1	Функции за четене на данни	53
4.2	Характеристики на диамантите	56
6.1	Характеристики на бейзболните играчи	79
6.2	Фамилия функции map	95
10.1	Сравнение на средна, медиана и мода за експеримент със зарове	156
10.2	Предмети с определена ценност и тегло	157
10.3	Тренировъчно множество данни	164
10.4	Тестово множество данни	166

Предговор

Това учебно помагало е предвидено за студенти и докторанти, които в своите магистърски или докторски тези се сблъскват с необходимостта да извършат определени експерименти, а след това статистически да обработят получените резултати.

В съвременния живот нуждата от обработка на информация постоянно нараства. Налага се да бъдат вземани решения в множество ситуации от ежедневието ни. От своя страна, всяко решение е толкова по-успешно, колкото по-информирано е взето то. Статистическата обработка на събраната информация е една от основите за вземането на информирани решения. В областта на статистическата обработка съществуват множество софтуерни решения, като се започне от по-достъпните за хора без опит, като *Microsoft Excel* и се стигне до професионалните пакети, като *SPSS* [1], *Matlab* [2] и *Mathematica* [3].

Настоящото учебното помагало представя програмния продукт *R*, който първоначално се разработва от Robert Gentleman и Ross Ihaka в University of Auckland през 1993 година. *R* е замислен като алтернатива на програмния продукт *S*, създаден от John Chambers, служител в Bell Labs. Първоначалният замисъл за *R* е да представлява инструмент, който да се използва в интерактивен режим, през командния ред. В последствие тази идея прераства в самостоятелен програмен език. Основното предназначение на *R* е обработка на данни, което включва въвеждане, пресмятане, визуално представяне на графики и отчети.

Езикът получава значително по-голяма популярност след 2000 година, като излиза от рамките на академичните среди и навлиза във финансовите среди, маркетинга, фармацията, социологията, психологията и в много други области. Най-често потребителите на *R* са хора с опит в програмни езици, като *C/C++* [5], *Java* [6], *C#* [7] или пък преди това са използвали други статистически пакети, като *SAS* [4], *SPSS* или дори *Microsoft Excel*. Тези потребители дават значителен тласък в развитието на пакета *R*, добавяйки множество софтуерни приставки (add-ons).

В някои случаи *R* се оказва стряскащ и дори смущаващ, особено за начинаещите потребители, но с времето и с процеса по навлизане в материята овладяването му се улеснява и ускорява. Целта на това учебно помагало е да представи информацията по един достъпен и олекотен за възприемане начин. Изложени са предимно най-важните аспекти от използването на пакета *R*, което от своя страна дава стабилна основа за бъдещо самостоятелно развитие на читателя. Въпреки наличието на голям избор от учебни материали по *R* [26, 27, 28, 29, 30, 31, 32, 23, 24, 25], материалът в това учебно помагало е съобразен основно със съдържанието на курса „Анализ на данни с *R*“, провеждан в Център за обучение към Българска академия на науките. Учебното помагало е организирано в следните глави:

Глава 1 - Инсталация и стартиране: Представа процеса по изтегляне, инсталиране и стартиране на програмния продукт.

Глава 2 - Пакетна организация, променливи, основни математически операции в *R* и типове данни: Разяснява основните концепции за работата с програмния продукт *R*, като акцентите са върху пакетната организация на продукта и как най-ефективно да бъдат използвани възможностите му. Разглеждат се базовите типове данни и основните математически операции.

Глава 3 - Сложни структури от данни и извикване на функции: Демонстрира използването на функции в *R*. Разглеждат се също сложни типове данни, като вектори, фактори, извадки от данни, масиви и матрици.

Глава 4 - Въвеждане на данни и извеждане на графики: Представа възможностите на системата за въвеждане на данни от външни източници, чрез прочитане на файлове или ресурси в Глобалната мрежа. Демонстрират се основните възможности на програмния продукт за визуализиране на данни и получени резултати.

Глава 5 - Оператори за контрол на изпълнението и потребителски функции: Въвежда основни принципи, използвани в конвенционалното програмиране като оператори за преход – *if*, *else* и *switch*, цикли – *for* и *while*, както и структуриране на кода под формата на потребителски функции.

Глава 6 - Групиране и обхождане на данни: Представа възможностите за прилагане на функции върху данни. Показват се групиране на данните и прилагане на агрегатни функции, като *sum*, *min*, *max* и други. Демонстрират се използването на данни във фреймовете и таблици. Демонстрират се възможностите за директна работа с релационна база данни и за обхождане по редове.

Глава 7 - Реорганизация на данните и обработка на символни низове: Демонстрират се възможностите за сливане на данни, трансформация на колони в редове и обратното. Разглеждат се възможностите за обработка на символни низове, което включва конструиране на символни низове, търсене на информация в символни низове и как могат да се използват регулярни изрази.

Глава 8 - Разширени графични възможности и вероятностни разпределения: Представят се по-разширени възможности за графично оформление на информацията. Демонстрират се най-често използваните вероятностни разпределение. Обсъжда се значението на кумулативните и плътностните функции. Демонстрират се възможностите за анализ на случайна величина. Разглеждат се нормално разпределение, биномно разпределение и Пуасоново разпределение.

Глава 9 - Статистическа обработка на данните: Въвежда в статистическата обработка на информацията. Първоначално представа описателните статистики като средна, медиана и мода. Следва представяне на сравнителните статистики като F-тест, T-тест и ANOVA. Завършва с линеен регресионен анализ.

Глава 10 - Приблизени пресмятания - подходи, методи, алгоритми: Посветена е на най-популярните подходи, методи и алгоритми за приблизителни числени пресмятания. С помощта на Монте-Карло методите се извършват множество статистически оценки в различни сфери от икономиката. При глобална оптимизация в многомерни пространства генетичните алгоритми са един от най-ефективните инструменти за търсене на субоптимални решения, близки до глобалните оптимуми. При необходимост от обработка на информация, спрямо съществуващи примерни, но без ясно формулирани правила за пресмятане, изкуствените невронни мрежи показват удовлетворителни резултати.

Глава 11 - Оформление на резултатите за печатно и електронно представяне: Представени са възможностите за оформление на получените резултати като предпечатна подготовка и представяне с мултимедийна техника. В основата на визуалното оформление с програмния продукт R са заложили тагиращите езици *LaTeX* и *Markdown*. В резултат на финалната компилация продуктите могат да бъдат PDF файлове, HTML страници, слайдове за презентация и текстови документи.

Разгледаните теми в учебното помагало дават възможност на заинтересованите потребители да започнат като напълно начинаещи работа с програмния продукт R. Макар и полезни, предварителни знания по програмиране не са необходими, но е желателно да са налични предварителни знания по статистика. Изложеният материал превежда читателите от темите за начинаещи до потребители в средно ниво. Учебното помагало не засяга темите за напреднали, които включват сложните методи за статистическа обработка, като нелинейни регресионни модели и дървовидни структури на решенията. Също така, не се засягат темите за напреднали по отношение на самия програмен продукт R и неговите възможности като самостоятелен програмен език, какъвто пример е темата за програмирането и добавянето на нови пакети (софтуерни библиотеки).

Авторите изразяват своята надежда, че всеки заинтересован читател ще намери полезна информация в настоящото учебно помагало, която ще му позволи да разшири своите знания, умения, а и цялостен мироглед към света и живота. Авторите изказват най-искрени благодарности към колегите доц. д-р Вера Ангелова и Нина Керемедчиева, за безценното съдействие, което указаха в процеса по създаването на това учебно помагало.

Глава 1

Инсталация и стартиране

Тъй като програмният продукт R се разработва под формата на софтуер с отворен код, то употребата му с нетърговска цел не изисква заплащане. За работа с R е достатъчно да се инсталира основният пакет, въпреки това съществува и интегрирана среда за разработка наречена *R Studio* [16]. За нуждите на учебното помагало ще бъде използван само основният пакет. Всеки желаещ да разшири уменията си с използването на интегрираната среда за разработка, би могъл самостоятелно да разучи възможностите ѝ.

1.1 Изтегляне на инсталационните файлове



The R Project for Statistical Computing

[\[Home\]](#)

Download

[CRAN](#)

R Project

[About R](#)

[Logo](#)

[Contributors](#)

[What's New?](#)

[Reporting](#)

[Bugs](#)

[Development](#)

[Site](#)

[Conferences](#)

[Search](#)

R Foundation

[Foundation](#)

[Board](#)

[Members](#)

[Donors](#)

[Donate](#)

Help With R

[Getting Help](#)

Getting Started

R is a free software environment for statistical computing and graphics. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. To [download R](#), please choose your preferred [CRAN mirror](#).

If you have questions about R like how to download and install the software, or what the license terms are, please read our [answers to frequently asked questions](#) before you send an email.

News

- [R version 3.5.1 \(Feather Spray\)](#) has been released on 2018-07-02.
- The R Foundation has been awarded the Personality/Organization of the year 2018 award by the professional association of German market and social researchers.
- [R version 3.5.0 \(Joy in Playing\)](#) has been released on 2018-04-23.

News via Twitter

 The R Foundation Retweeted



useR! 2019
[@UseR2019_Conf](#)

Фигура 1.1: Начална уеб страница на продукта

Както множество софтуерни продукти и R е достъпен за изтегляне от уеб страницата на продукта в Интернет (Фиг. 1.1) с адрес: <http://www.r-project.org/>

CRAN Mirrors

The Comprehensive R Archive Network is available at the following URLs, please choose a location close to you. Some statistics on the status of the mirrors can be found here: [main page](#), [windows release](#), [windows old release](#).

If you want to host a new mirror at your institution, please have a look at the [CRAN Mirror HOWTO](#).

0-Cloud

<https://cloud.r-project.org/>

Automatic redirection to servers worldwide, currently sponsored by Rstudio

<http://cloud.r-project.org/>

Automatic redirection to servers worldwide, currently sponsored by Rstudio

Algeria

<https://cran.usthb.dz/>

University of Science and Technology Houari Boumediene

<http://cran.usthb.dz/>

University of Science and Technology Houari Boumediene

Argentina

<http://mirror.fcaglp.unlp.edu.ar/CRAN/>

Universidad Nacional de La Plata

Australia

<https://cran.csiro.au/>

CSIRO

<http://cran.csiro.au/>

CSIRO

<https://mirror.aarnet.edu.au/pub/CRAN/>

AARNET

<https://cran.ms.unimelb.edu.au/>

School of Mathematics and Statistics, University of Melbourne

<https://cran.curtin.edu.au/>

Curtin University of Technology

Austria

<https://cran.wu.ac.at/>

Wirtschaftsuniversität Wien

<http://cran.wu.ac.at/>

Wirtschaftsuniversität Wien

Belgium

<http://www.freeststatistics.org/cran/>

K.U.Leuven Association

<https://lib.ugent.be/CRAN/>

Ghent University Library

<http://lib.ugent.be/CRAN/>

Ghent University Library

Brazil

<https://cran.rstudio.com/mirrors/brazil/>

Center for Comp. Biol. at Universidade Estadual de Santa

Фигура 1.2: Списък със сървъри за изтегляне

В раздела за изтегляне са посочени множество активни връзки към различни географски локации (Фиг. 1.2). Обичайна практика, при софтуерните продукти с отворен код, е наличието на множество сървъри, разположени по цял свят, да предлагат изтегляне на нужните за инсталацията файлове. Тази практика се е наложила най-вече за ускоряване на изтеглянето, също така и за намаляване на натоварването, което сървърите понасят при множество заявки. Не на последно място, много често за разпространението на инсталационните файлове се разчита на доброволческа информационна инфраструктура, за която не се заплаща.



CRAN

[Mirrors](#)

[What's new?](#)

[Task Views](#)

[Search](#)

About R

[R Homepage](#)

[The R Journal](#)

Software

[R Sources](#)

[R Binaries](#)

[Packages](#)

[Other](#)

Documentation

[Manuals](#)

[FAQs](#)

[Contributed](#)

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2018-07-02, Feather Spray) [R-3.5.1.tar.gz](#), read [what's new](#) in the latest version.
- Sources of [R alpha and beta releases](#) (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are [available here](#). Please read about [new features and bug fixes](#) before filing corresponding feature requests or bug reports.
- Source code of older versions of R is [available here](#).
- Contributed extension [packages](#)

Questions About R

- If you have questions about R like how to download and install the

Фигура 1.3: Избор на подходяща инсталация за операционната система

Често срещана практика е продуктите с отворен код да се разпространяват за трите най-популярни операционни системи, този принцип е спазен и за продукта R (Фиг. 1.3). При комерсиалните софтуерни продукти много често се залага на една единствена операционна система, но при продуктите с отворен код идеологията е, че трябва да се достигне до възможно най-голям брой потребители и поради тази причина се полагат допълнителни усилия софтуерът да работи на възможно най-много платформи (платформа – комбинация между хардуер и операционна система). Тази стратегия за дълготрайно развитие залага и на следващата особеност в развитието на продуктите с отворен код, а именно, че една част от потребителите с времето се превръщат в хора добавящи програмен код към продуктите. Освен вида на операционната система, от значение е и размерът на машинната дума, която процесорът поддържа. Към настоящия момент, най-разпространени са изчислителните машини с 64 битова машинна дума, но тъй като все още има много техника, която работи на 32 битова машинна дума, продуктът R е достъпен и за двата варианта.



CRAN

[Mirrors](#)

[What's new?](#)

[Task Views](#)

[Search](#)

About R

[R Homepage](#)

[The R Journal](#)

Software

[R Sources](#)

[R Binaries](#)

[Packages](#)

[Other](#)

Documentation

[Manuals](#)

[FAQs](#)

[Contributed](#)

R for Mac OS X

This directory contains binaries for a base distribution and packages to run on Mac OS X (release 10.6 and above). Mac OS 8.6 to 9.2 (and Mac OS X 10.1) are no longer supported but you can find the last supported release of R for these systems (which is R 1.7.1) [here](#). Releases for old Mac OS X systems (through Mac OS X 10.5) and PowerPC Macs can be found in the [old](#) directory.

Note: CRAN does not have Mac OS X systems and cannot check these binaries for viruses. Although we take precautions when assembling binaries, please use the normal precautions with downloaded executables.

As of 2016/03/01 package binaries for R versions older than 2.12.0 are only available from the [CRAN archive](#) so users of such versions should adjust the CRAN mirror setting accordingly.

R 3.5.1 "Feather Spray" released on 2018/07/05

Important: since R 3.4.0 release we are now providing binaries for OS X 10.11 (El Capitan) and higher using non-Apple toolkit to provide support for OpenMP and C++17 standard features. To compile packages you may have to download tools from the [tools](#) directory and read the corresponding note below.

Please check the MD5 checksum of the downloaded image to ensure that it has not been tampered with or corrupted during the mirroring process. For example type

```
md5 R-3.5.1.pkg
```

in the *Terminal* application to print the MD5 checksum for the R-3.5.1.pkg image. On Mac OS X 10.7 and later you can also validate the signature using

```
pkgutil --check-signature R-3.5.1.pkg
```

Lastest release:

[R-3.5.1.pkg](#)

MD5-hash: 58eaff65fbd024f267ef1e521e17e7f8
SHA1-
hash: 76c01bfa62a6896d5f4a4511e25d17276d149621
(ca. 74MB)

R 3.5.1 binary for OS X 10.11 (El Capitan) and higher, signed package. Contains R 3.5.1 framework, R.app GUI 1.70 in 64-bit for Intel Macs, Tcl/Tk 8.6.6 X11 libraries and Texinfo 5.2. The latter two components are optional and can be omitted when choosing "custom install", they are only needed if you want to use the `tcltk` R package or build package documentation from

Фигура 1.4: Избор на версия за изтегляне

Добра практика е при работата със софтуерни продукти, които се разпространяват като отворен код, винаги да се използва най-новата стабилна версия. В случая, за операционната система Mac OS X, това е версията 3.5.1, която е налична под формата на инсталационен файл R-3.5.1.pkg (Фиг. 1.4).



CRAN

[Mirrors](#)

[What's new?](#)

[Task Views](#)

[Search](#)

About R

[R Homepage](#)

[The R Journal](#)

Software

[R Sources](#)

[R Binaries](#)

[Packages](#)

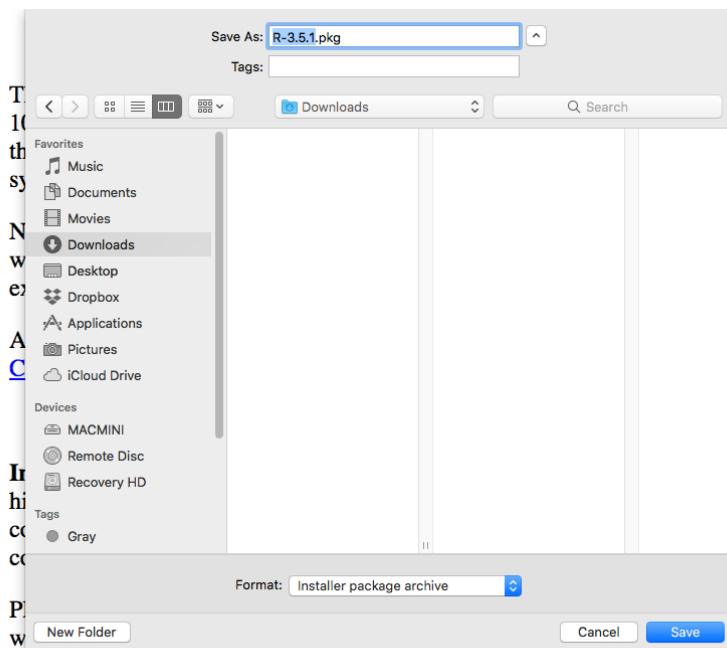
[Other](#)

Documentation

[Manuals](#)

[FAQs](#)

[Contributed](#)



md5 R-3.5.1.pkg

in the *Terminal* application to print the MD5 checksum for the R-3.5.1.pkg image. On Mac OS X 10.7 and later you can also validate the signature using

```
pkgutil --check-signature R-3.5.1.pkg
```

Latest release:

[R-3.5.1.pkg](#)

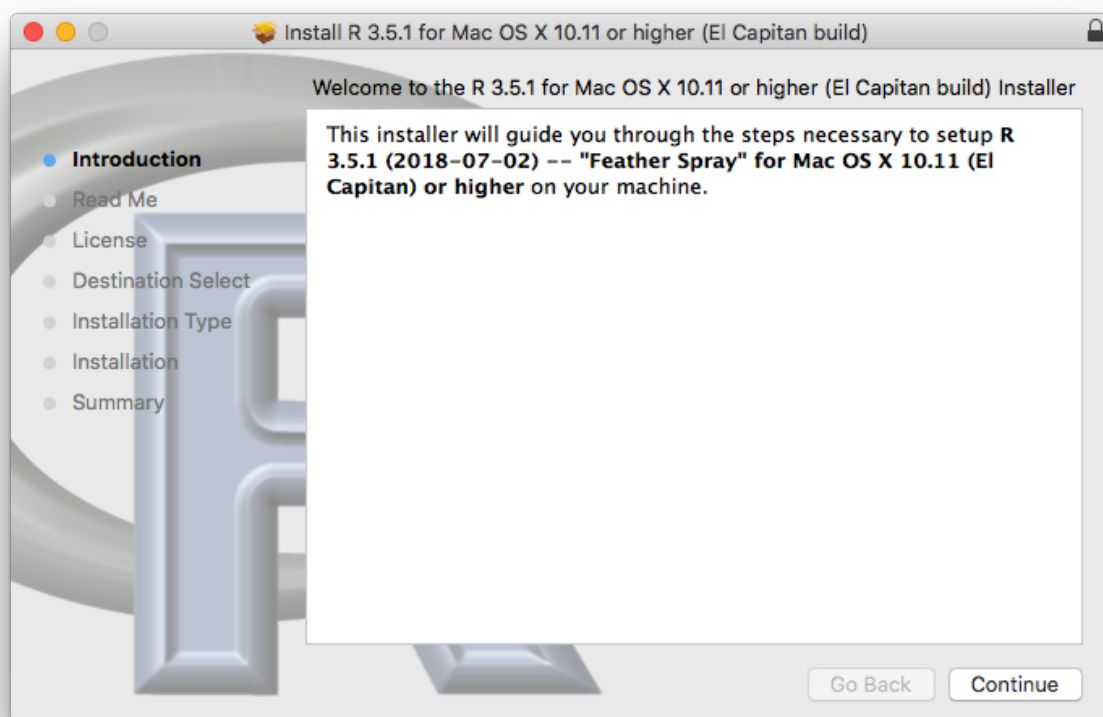
MD5-hash: 58eaff65fbd024f267ef1e521e17e7f8
SHA1-
hash: 76c01bfa62a6896d5f4a4511e25d17276d149621
(ca. 74MB)

R 3.5.1 binary for OS X 10.11 (El Capitan) and higher, signed package. Contains R 3.5.1 framework, R.app GUI 1.70 in 64-bit for Intel Macs, Tcl/Tk 8.6.6 X11 libraries and Texinfo 5.2. The latter two components are optional and can be omitted when choosing "custom install", they are only needed if you want to use the `tcltk` R package or build package documentation from

Фигура 1.5: Запазване на инсталационния файл

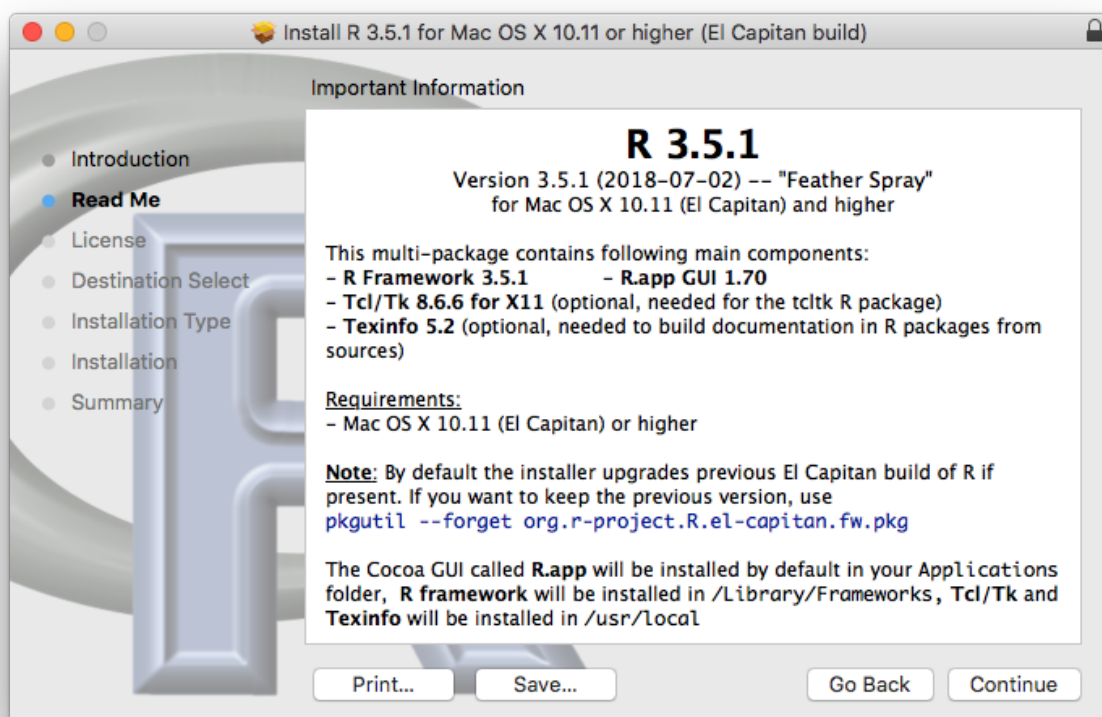
1.2 Инсталация

За всяка от операционните системи е достатъчно потребителят да следва инструкциите и инсталацията протича безпроблемно по указаните стъпки.



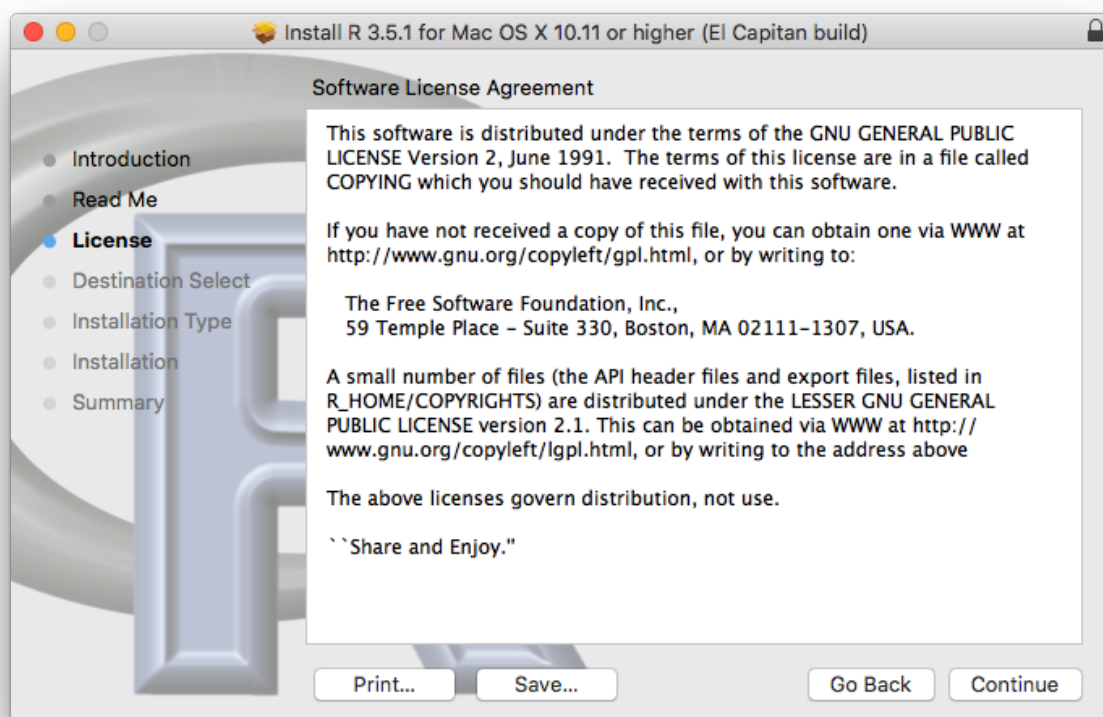
Фигура 1.6: Активиране на инсталатора

С двойно кликване на мишката се активира инсталаторът (Фиг. 1.6). След което следва екран с подробности за самата инсталация (Фиг. 1.7).



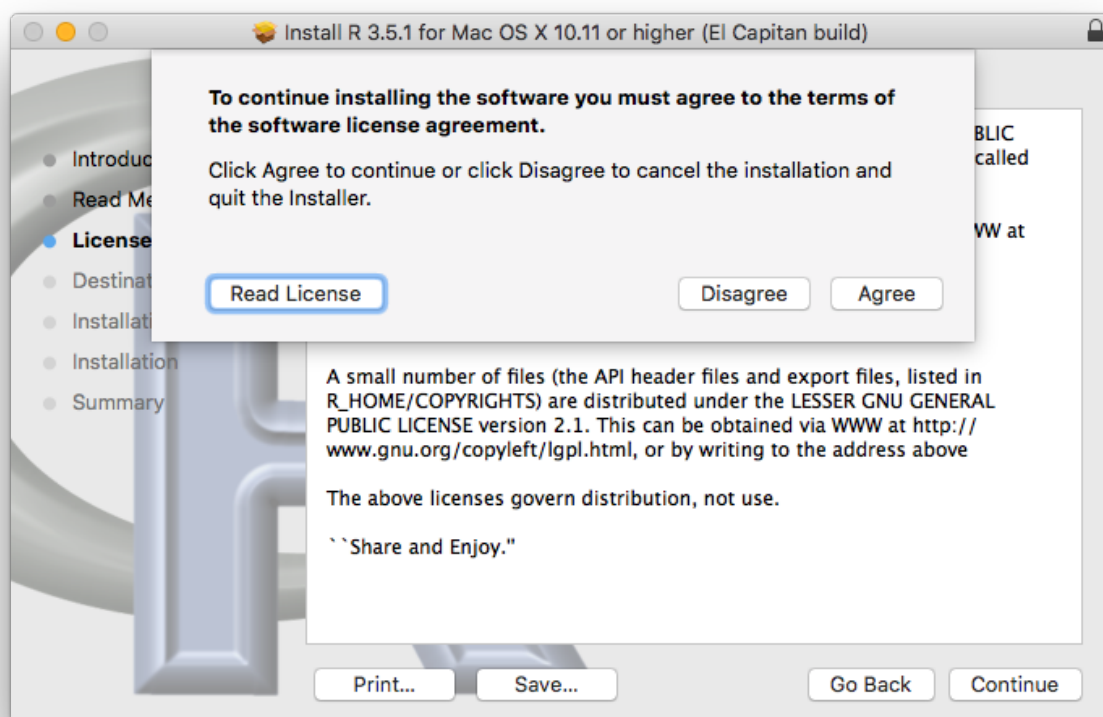
Фигура 1.7: Подробности за инсталацията

От съществено значение е потребителите на продукти с отворен код да са добре запознати с условията, при които те получават продуктите, особено когато е без заплащане. Поради тази причина, потребителят трябва изрично да се съгласи с условията на лиценза, под който се разпространява продуктът R (Фиг. 1.8, 1.9).



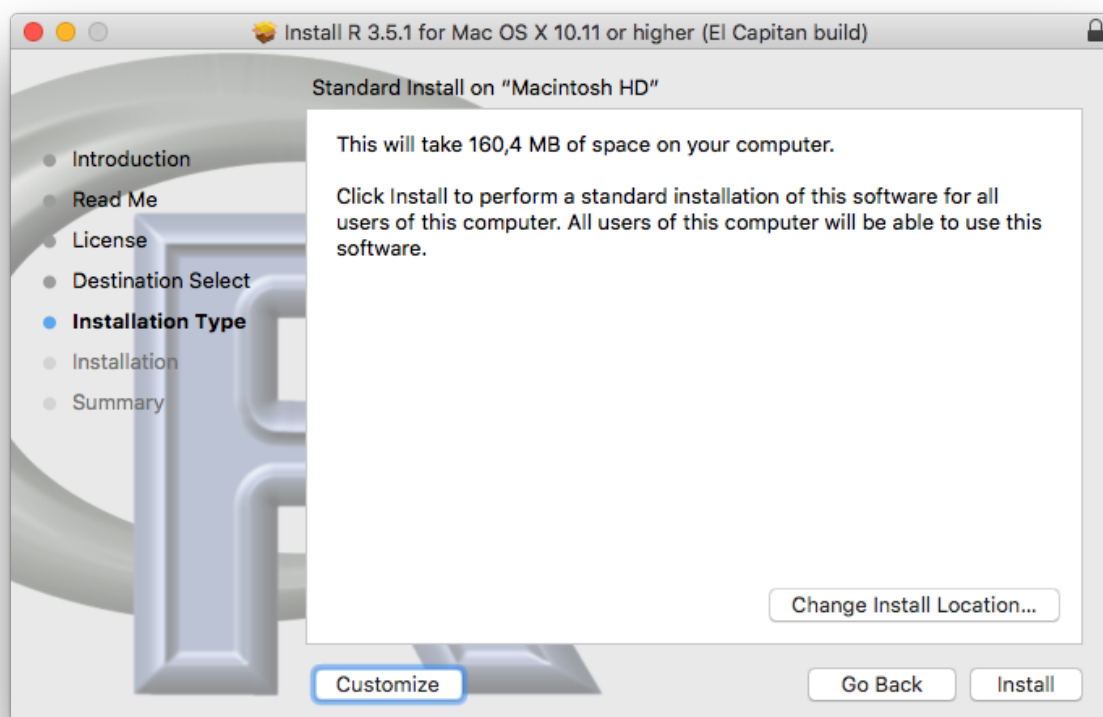
Фигура 1.8: Лицензно споразумение за ползване

Софтуерният продукт R се разпространява под отворен лиценз GPL2, който в най-общи линии очертава рамките на условията, при които потребителите получават софтуера. Най-важните клаузи в лиценза са свързани с липса на гаранция и с изричното съгласие на потребителя, че производителят не носи никаква юридическа отговорност, произтекла от употребата на софтуерния продукт. Пълният текст на лиценза е достъпен в уеб страницата на фондацията, която го поддържа [33].



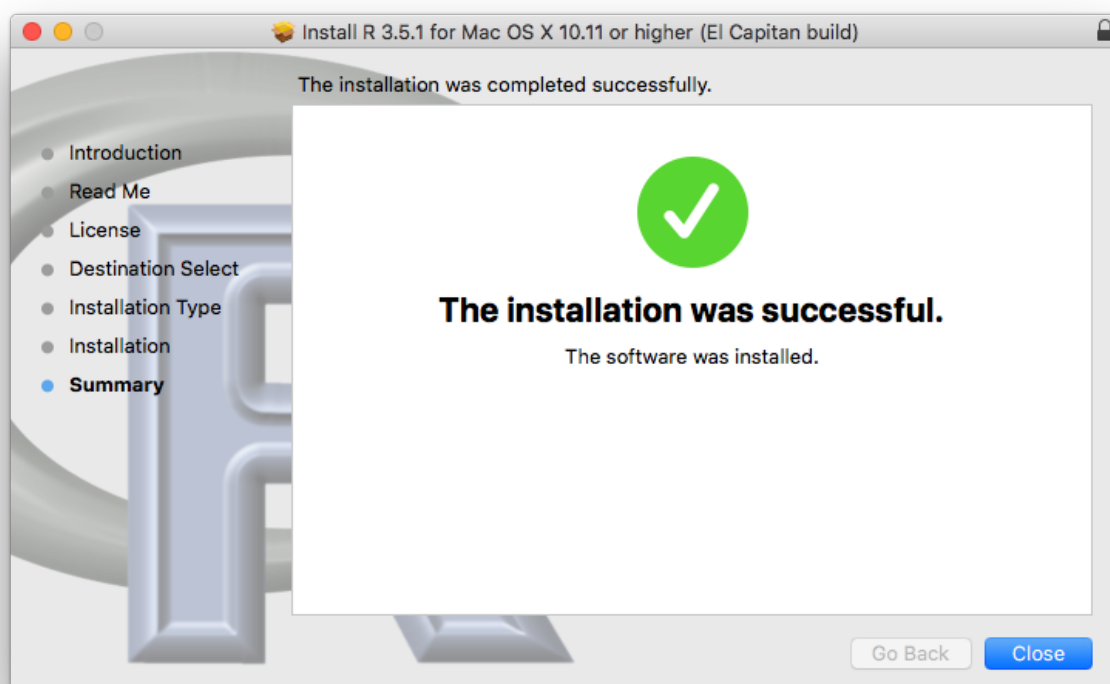
Фигура 1.9: Изрично съгласие

Съвременните операционни системи са от отворен тип и инсталациите на допълнителни софтуерни продукти се явяват своеобразно разширение на операционната система. Поради тази причина, всеки създател на операционна система е избрал правила и начини за добавяне на софтуерни продукти. Една от основните характеристики е определяне на директория във файловата система на операционната система, където новодобавеният софтуер да бъде поставен. Някои операционни системи (например Linux базираните дистрибуции) използват специално организиран мениджър на пакетите, който има грижата за консистентността на добавяните софтуерни продукти. При Mac OS X също е налична възможността за автоматично управление на инсталациите, но е дадена и възможност потребителят да избира мястото на инсталация. В операционната система Microsoft Windows, потребителят решава къде да помести новоинсталирания софтуер.



Фигура 1.10: Информация за директорията и използваното дисково пространство

При желание е възможно да бъде подменена инсталационната директория (Фиг. 1.10).

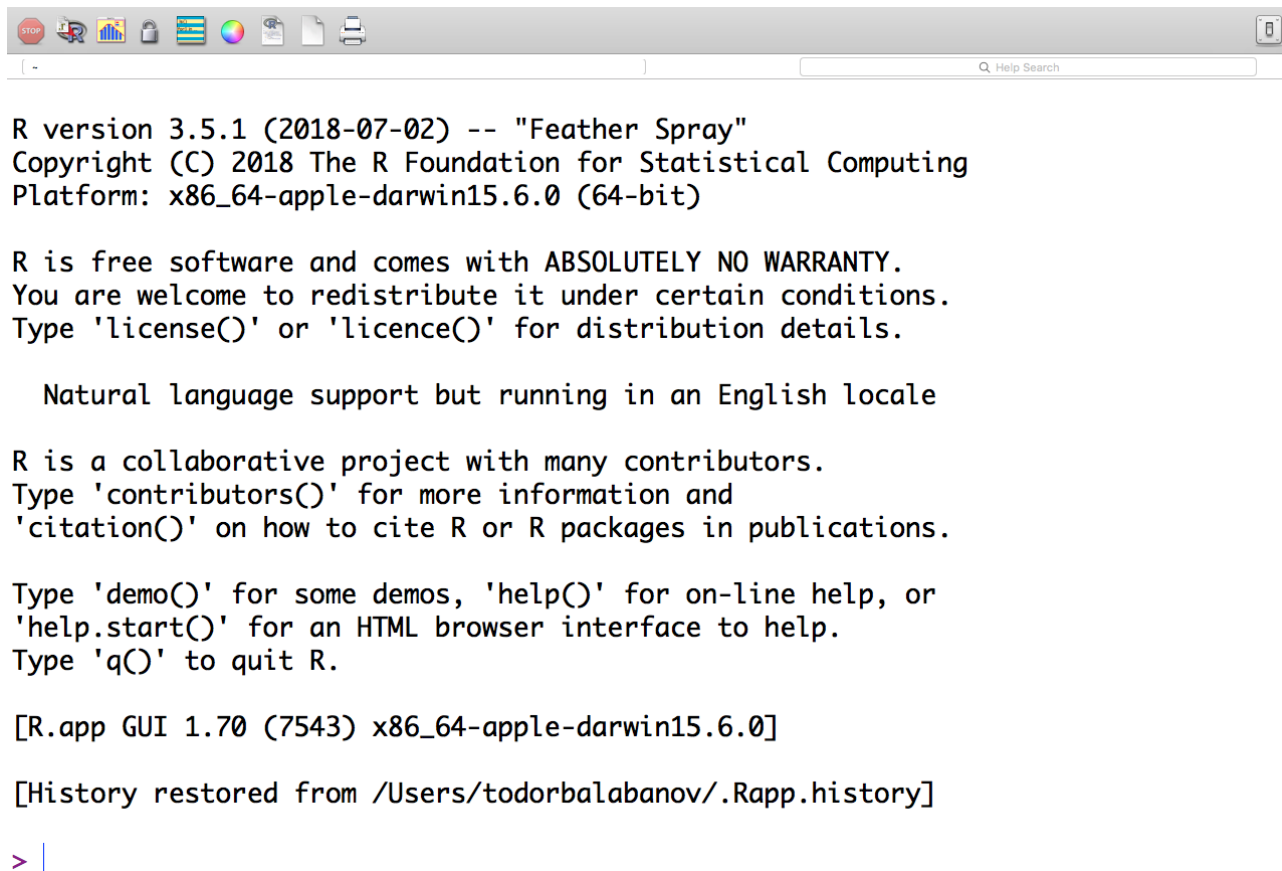


Фигура 1.11: Успешно приключване на инсталацията

Инсталацията приключва със съобщение за успешно изпълнение (Фиг. 1.11).

1.3 Работа в режим на команди

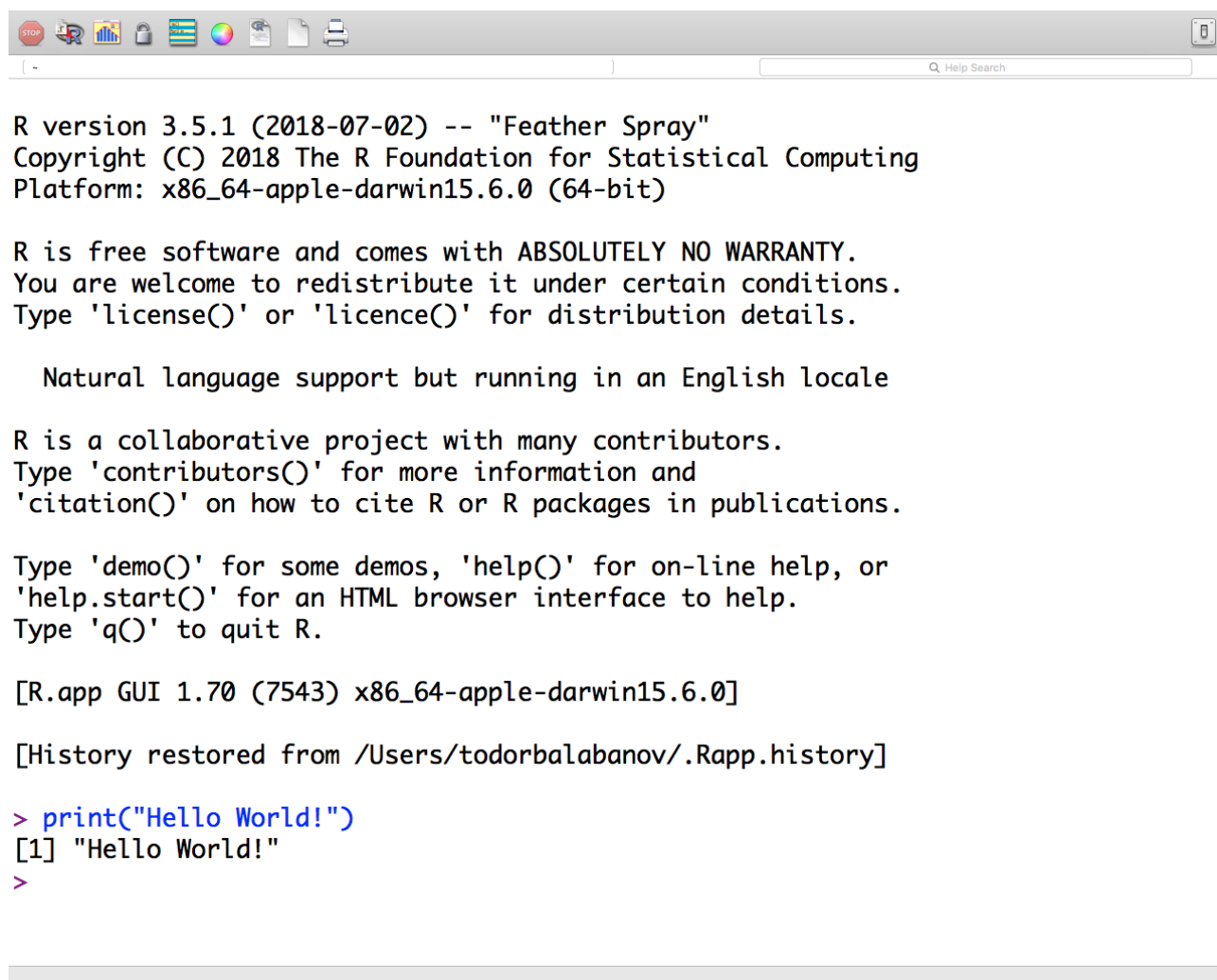
Инсталаторът създава икона за стартиране на R командния интерпретатор.



Фигура 1.12: Основен прозорец на продукта

При успешна инсталация и стартиране, потребителят получава достъп до прозорец, служещ като команден интерпретатор (Фиг. 1.12). Първоначалният замисъл за продукта R е бил това да представлява интерпретатор на команди. В интерактивен режим потребителят въвежда команда и наблюдава полученния резултат. Макар и това да е основният начин за работа, R позволява последователността от команди да бъде записана във файл и да се изпълняват като единен скрипт. За много потребители, особено такива с предишен опит в софтуерни продукти със силно застъпен графичен потребителски интерфейс (например статистически анализ на данни в Microsoft Excel), използването на терминал с команди първоначално е трудно и дори дразнещо, но с напредване на времето потребителите оценяват гъвкавостта, която този начин на работа позволява. Използването на поредица от команди често се оказва значително по-бърз начин за работа, в сравнение с подготовката на експеримент в софтуер с графичен интерфейс. Също така, наличието на серията команди дава възможност значително по-бързо експериментът да бъде преповторен. Често при по-голям обем данни софтуерните пакети с графичен потребителски интерфейс забавят работата си неприемливо много. И не на последно място, наличието на статистическия модел като скрипт позволява използването на системи за контрол на версиите (каквато примерно е Git и облачната услуга GitHub [13]), нещо което търговските бинарни файлови формати (например XLSX, в Microsoft Excel) не позволяват. При работата с командния интерпретатор на R, клавишът „стрелка нагоре“ повтаря последната използвана команда. Списъкът с вече изпълнени команди може да бъде преминат със стрелките нагоре и надолу. Тъй като интерпретацията на всяка команда става в момента на нейното повикване, е

възможно да бъде стартиран код, който да отнема твърде дълго време или да изпадне в безкраен цикъл. При тази ситуация, натискането на клавиша Esc или клавишната комбинация Ctrl+C прекъсва текущо изпълняваната команда.



```
R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

[R.app GUI 1.70 (7543) x86_64-apple-darwin15.6.0]

[History restored from /Users/todorbalebanov/.Rapp.history]

> print("Hello World!")
[1] "Hello World!"
>
```

Фигура 1.13: Изпълнение на команда за печат

При правилно работеща инсталация изписването на командата за печат би дала резултата показан на Фиг. 1.13. Както показва първата примерна команда, има фундаментална разлика в начина, по който работят компилаторите и интерпретаторите. Езици като C/C++ изискват компилация на програмния код до машинни инструкции, които процесорът на компютъра изпълнява. При интерпретативните езици, като PHP [10], Python [11], JavaScript [12] и разбира се R, всяка стъпка от програмата се интерпретира в момента на повикването ѝ. По-напредналите потребители на R са добре запознати с начина, по който са изградени библиотеките на езика и знаят добре, че съществува възможност код писан на C/C++ да бъде изпълняван в средата на R. Такава междуезикова връзка най-често се налага при голям обем данни за пресмятане и относително бавни алгоритми, които извършват пресмятането.

Заклучение

Употребата на всеки програмен продукт преминава през фазите за придобиване на продукта, инсталация и стартиране. Макар и напълно интуитивен, процесът по сваляне, инсталиране и стартиране има своите специфики.

Глава 2

Пакетна организация, променливи, основни математически операции в R и типове данни

Най-голямата сила на продукта R се дължи на хилядите пакети (софтуерни приставки), създадени от безброй потребители на продукта. Наличните пакети покриват цялата област на статистиката и статистическата обработка на данни.

Под пакет се разбира софтуерна библиотека от предварително написан програмен код, който има за цел да реши определена задача или група от задачи. Тъй като продуктът R е една отворена система, е важно да се има предвид, че не всички пакети са с еднакво качество. Една част от пакетите са изключително професионално написани, устойчиви са на некоректно използване и имат добра база от поддържащи ги потребители. В същото време друга част от пакетите са създадени с голяма доза добри намерения, но работят бавно, дават дефекти или просто не вършат това, за което са създадени. Голяма част от пакетите са написани от статистици за статистици и това може да доведе до някои странни въпроси при част от потребителите, особено при хора, идващи от индустрията за производство на софтуер.

Настоящото учебно помагало представя само най-основните пакети, достатъчни да бъде изложен материалът свързан с базовите познания по R. Опит да бъдат разгледани всички пакети е непосилен за едно издание, най-вече защото броят и видът на пакетите постоянно се променя.

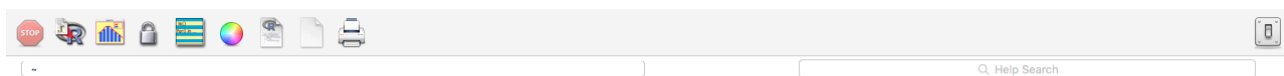
2.1 Инсталиране на пакети

Съществуват различни начини за инсталиране на пакети в R като най-използваният от тях е чрез команда в конзолата на пакета R.



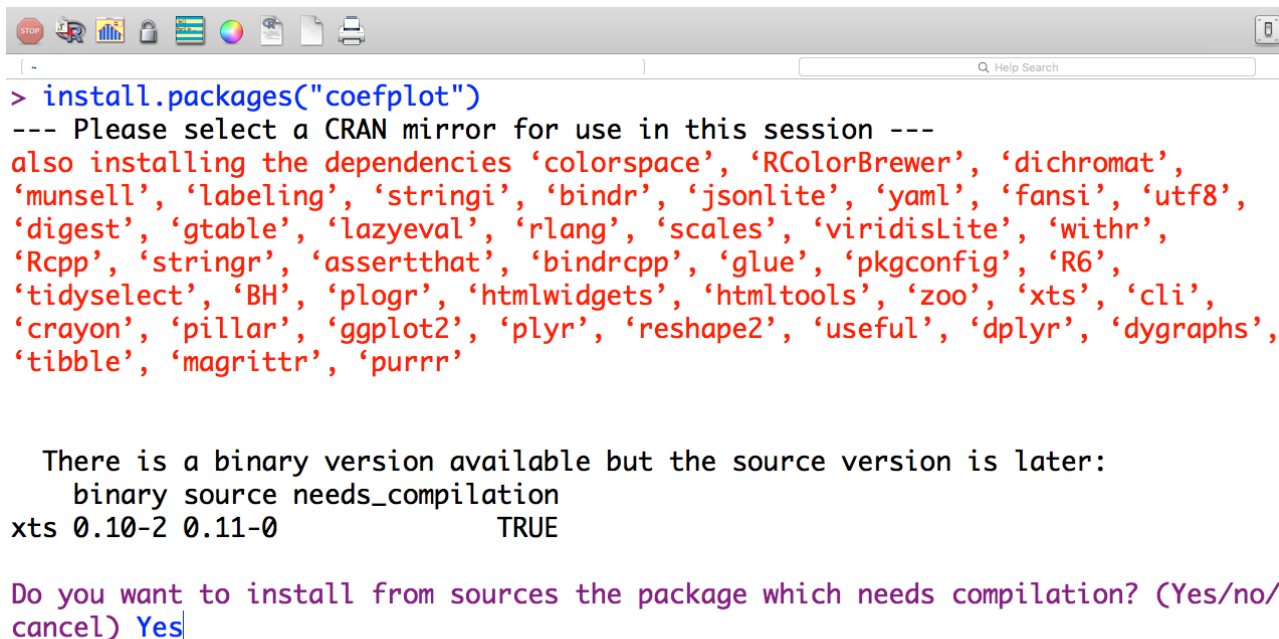
Фигура 2.1: Команда за инсталиране на пакета coefplot

За да започне инсталирането на пакет (в случая coefplot) е достатъчно да се изпише командата от Фиг. 2.1, със съответното име на пакета като неин параметър.



Фигура 2.2: Избор на сървър за изтегляне на пакета

Следва избор на сървър за изтегляне на пакета (Фиг. 2.2). Разумна стратегия е да се избират сървъри, които териториално се намират в близост до мястото, от което се работи. Това би осигурило малко по-голяма скорост на връзката в Глобалната мрежа.



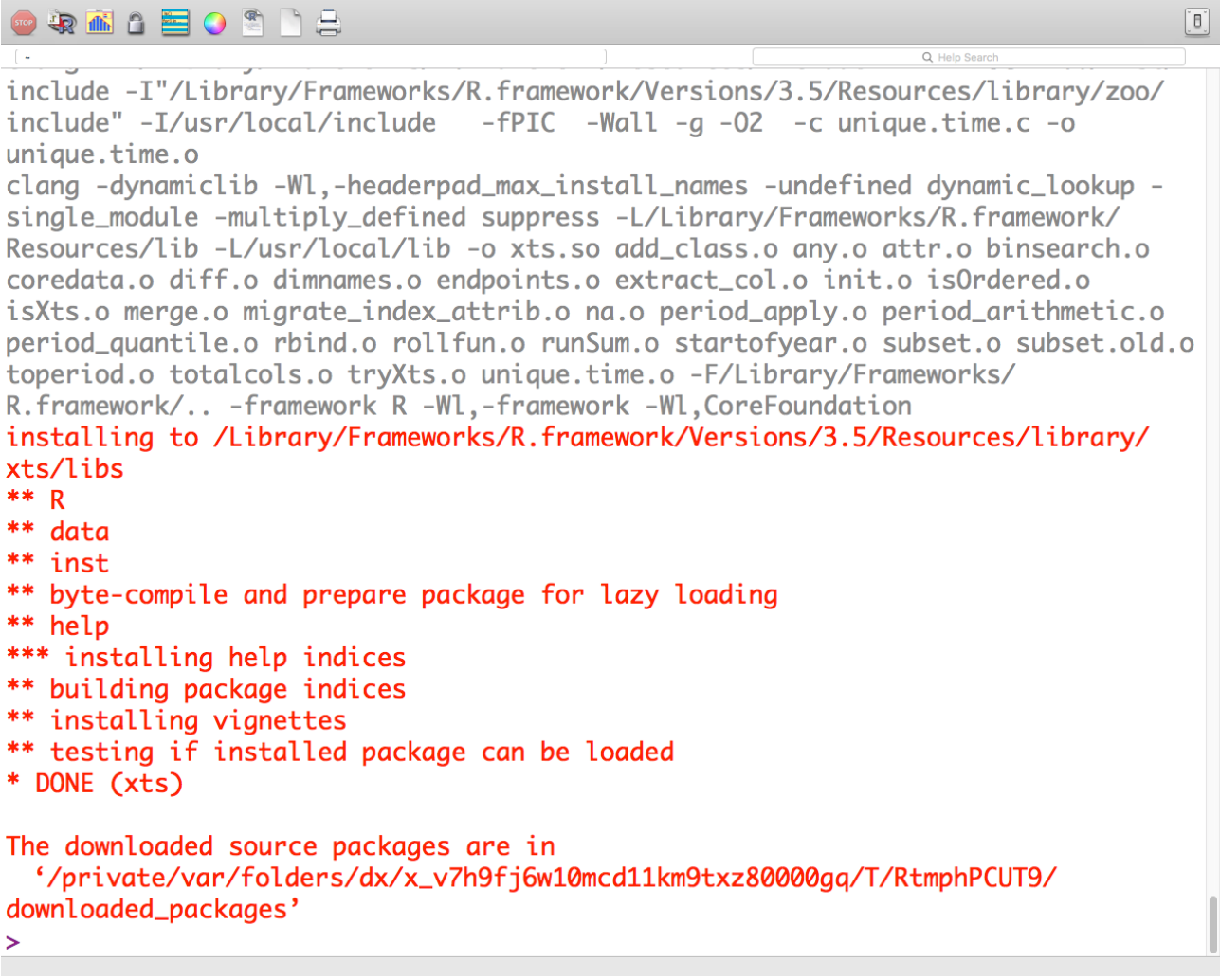
```
> install.packages("coefplot")
--- Please select a CRAN mirror for use in this session ---
also installing the dependencies 'colorspace', 'RColorBrewer', 'dichromat',
'munsell', 'labeling', 'stringi', 'bindr', 'jsonlite', 'yaml', 'fansi', 'utf8',
'digest', 'gtable', 'lazyeval', 'rlang', 'scales', 'viridisLite', 'withr',
'Rcpp', 'stringr', 'assertthat', 'bindrcpp', 'glue', 'pkgconfig', 'R6',
'tidyselect', 'BH', 'plogr', 'htmlwidgets', 'htmltools', 'zoo', 'xts', 'cli',
'crayon', 'pillar', 'ggplot2', 'plyr', 'reshape2', 'useful', 'dplyr', 'dygraphs',
'tibble', 'magrittr', 'purrr'

There is a binary version available but the source version is later:
  binary source needs_compilation
xts 0.10-2 0.11-0 TRUE

Do you want to install from sources the package which needs compilation? (Yes/no/
cancel) Yes
```

Фигура 2.3: Зависимости между пакетите

Често срещан случай е един пакет да има функционална зависимост от други пакети (Фиг. 2.3). В такава ситуация е необходимо всички нужни пакети също да бъдат инсталирани. Стратегията при раз-
работка на пакети е те да бъдат предлагани в компилиран (бинарен) вид, но понякога най-новите версии
са под формата на програмен код и тогава потребителят има възможност да избере между бинарната
версия или версията с програмен код.



```
include -I"/Library/Frameworks/R.framework/Versions/3.5/Resources/library/zoo/  
include" -I/usr/local/include -fPIC -Wall -g -O2 -c unique.time.c -o  
unique.time.o  
clang -dynamiclib -Wl,-headerpad_max_install_names -undefined dynamic_lookup -  
single_module -multiply_defined suppress -L/Library/Frameworks/R.framework/  
Resources/lib -L/usr/local/lib -o xts.so add_class.o any.o attr.o binsearch.o  
coredata.o diff.o dimnames.o endpoints.o extract_col.o init.o isOrdered.o  
isXts.o merge.o migrate_index_attr.o na.o period_apply.o period_arithmetic.o  
period_quantile.o rbind.o rollfun.o runSum.o startofyear.o subset.o subset.old.o  
toperiod.o totalcols.o tryXts.o unique.time.o -F/Library/Frameworks/  
R.framework/.. -framework R -Wl,-framework -Wl,CoreFoundation  
installing to /Library/Frameworks/R.framework/Versions/3.5/Resources/library/  
xts/libs  
** R  
** data  
** inst  
** byte-compile and prepare package for lazy loading  
** help  
*** installing help indices  
** building package indices  
** installing vignettes  
** testing if installed package can be loaded  
* DONE (xts)  
  
The downloaded source packages are in  
  ‘/private/var/folders/dx/x_v7h9fj6w10mcd11km9txz80000gq/T/RtmphPCUT9/  
downloaded_packages’  
>
```

Фигура 2.4: Резултат от инсталацията на пакета

Инсталацията на пакета приключва с подробен списък, съдържащ описание на извършените операции (Фиг. 2.4).



Фигура 2.5: Зареждане на пакета coefplot

Дали пакетът е надлежно инсталиран може да се провери с командата `require` (Фиг. 2.5), която зарежда пакета в паметта.

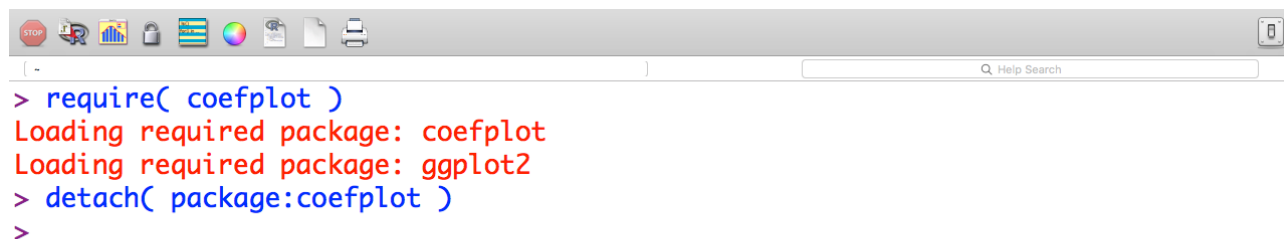
Съществува възможност пакетите да се инсталират под формата на програмен код, директно от хранилищата за програмен код, но за тази цел са нужни подходящите компилатори (най-често C/C++ и Fortran), както и по-задълбочени умения по програмиране. В редки случаи се налага инсталиране на пакета от ZIP файл. При такава ситуация е важно предварително да бъдат инсталирани всички пакети, от които инсталирания пакет зависи.

Премахване на инсталирани пакети става с помощта на командата `remove.packages`, на която се подава вектор с имената на пакетите, които трябва да бъдат премахнати.

2.2 Зареждане на пакети

За да бъдат използвани пакетите не е достатъчно те да бъдат инсталирани, необходимо е с команда да бъдат включени в текущата сесия от изчисления. R предлага две команди за зареждане на пакети – `library` и `require`. И двете изпълняват едно и също нещо – зареждат пакета в общата памет. Разликата

е, че `require` връща `TRUE`, ако зареждането е било успешно и `FALSE` при неуспех. Тази възможност е полезна в редките случаи, когато пакетът се зарежда от програмния текст на функция. Подобна практика не е препоръчителна, но R дава такава възможност. И двете функции получават като параметър името на пакета, със или без кавички. Пакетите се зареждат еднократно и остават налични през цялата сесия от изчисления или докато изрично не бъдат премахнати от общата памет.



Фигура 2.6: Премахване на пакета `coefplot` от общата памет

Премахването на пакет от общата памет става с командата `detach` (Фиг. 2.6). Съществено при тази команда е, че преди името на пакета се записва думата `package`.

Тъй като пакетите се разработват основно на доброволни начала, рядко се случва в различни пакети да има едноименни функции. При подобна колизия на имената решението е да се приложи операцията за принадлежност – двойно двоеточие (`::`). Когато бъде използвана операцията за принадлежност, може да не се зарежда пакетът, към който принадлежи функцията.

2.3 Основни математически операции

R позволява да се извършват сложни математически пресмятания, но също така може да се използва и за базови математически изчисления.

A screenshot of the R console interface. The top bar shows various icons (stop, undo, redo, etc.) and a search bar labeled 'Help Search'. The console area displays the following commands and their outputs:

```
> 1 + 1
[1] 2
> 2 + 2 * 2
[1] 6
> 5 / 3
[1] 1.666667
>
```

Фигура 2.7: Примерни аритметични операции

Най-базовите математически операции са събирането, изваждането, умножението и делението. Тези операции се изпълняват в R както е показано на Фиг. 2.7. Пресмятанията в R се състоят от операции и операнди. Когато няколко операции бъдат обединени, чрез операндите си, се получава математически израз.

Листинг 2.1: Събиране

```
1 + 1
```

В листинг 2.1 е демонстрирана операцията за събиране, която има два операнда. Когато става въпрос за математически операции, те имат серия свойства. Като най-съществена характеристика може да се отбележи броят на операндите. Събирането е класически пример за бинарна операция, тъй като има два операнда (ляв и десен).

Листинг 2.2: Унарн минус

```
-5
```

В гимназиалния курс по математика не се споменава наличието на унарн плюс, макар и да се учи за унарн минус (Листинг 2.2). Унарните плюс и минус променят значението на операнда. В примера от

листинг 2.2, унарният минус променя значението на числото пет от положително към отрицателно. Тъй като унарният плюс не променя значението на операнда си, масова практика е знакът на унарния плюс да не се записва, това е нещо, което не е възможно с унарния минус. Освен унарни и бинарни операции в някои езици (например C/C++, Java, C#, PHP и други) съществува една единствена тернарна операция (?:), която има смисъла на условния оператор за преход if.

Както бе споменато по-горе, комбинацията от няколко операции и техните операнди водят до съставянето на математически израз (Листинг 2.3).

Листинг 2.3: Аритметичен израз с две събирания

```
2 + 2 + 2
```

Тъй като съвременните изчислителни машини са организирани по такъв начин, че процесорът да извършва само една математическа операция на един такт от пресмятането, става актуален въпросът коя от операциите ще бъде изпълнена първа и коя втора, при положение, че операциите са с еднакъв приоритет. Тъй като в англосаксонската писмена система е прието да се пише и чете от ляво на дясно, то множество математически операции се изпълняват от ляво на дясно. Това се нарича лява асоциативност и събирането е точно от тази група операции.

Листинг 2.4: Израз за каскадно присвояване

```
a = b = 2
```

В гимназиалната математика символът равно се използва за проверка на идентичността между двата операнда, но в компютърните езици символът за равенство има смисъл на операция за присвояване. Това означава, че десният операнд бива присвоен като стойност на левия операнд. При съставянето на математически израз с каскада от присвоявания няма друг вариант, освен първо най-десният операнд да бъде изпълнен и едва накрая най-левият. Това се нарича дясна асоциативност и се използва при значително малък брой от математическите операции.

Листинг 2.5: Контекстна зависимост на операциите

```
"abc" + "def"  
2 + 2
```

Следващата важна характеристика на математическите операции е тяхната контекстна зависимост. В множество езици събирането на символни низове води до конкатенация (не и в R), докато събирането на числа води до резултат от числено събиране (Листинг 2.5).

Листинг 2.6: Контекстна зависимост на операцията за делене

```
5 / 3  
5.0 / 3.0
```

В множество програмни езици (с изключение на R) операцията за деление е контекстно зависима (Листинг 2.6). Когато и двата операнда са цели числа, резултатът е целочислено деление, а когато поне един от операндите е дробно число, то резултатът е дробно число.

Листинг 2.7: Приоритет на операциите

```
2 + 2 * 2
```

Когато в един математически израз участва повече от една операция с еднаква асоциативност, от значение става приоритетът на всяка от тях. Най-често даваният пример е събирането и умножението (Листинг 2.7). В случая първо се извършва умножението, тъй като е по-високо приоритетно, а едва след това събирането.

Листинг 2.8: Смяна на приоритета

```
(2 + 2) * 2
```

В компютърните езици кръглите скоби имат смисъла на операция за промяна на реда, по който ще се извърши пресмятането с цел смяна на приоритета (Листинг 2.8).

Последният съществен признак на операциите е в коя група попадат – аритметични, логически, побитови, за сравнение, за присвояване и други.

2.4 Типове и променливи

Повечето съвременни програмни езици организират работата с информация в групи от променливи. За разлика от строго типизираните езици, в R не се задава тип на променливата. Типът на променливата неявно се определя от стойността, която е присвоена към нея. Това позволява да се присвояват дори обекти или функции и означава, че една и съща променлива може да съдържа данни от различни типове в различни моменти от времето.

2.4.1 Създаване на променливи

Променливата се появява в общата памет веднага след първата операция за присвояване, за което съществува цяла група операции за присвояване (Листинг 2.9). Променливите в R могат да съдържат в имената си латинските букви и арабските цифри, също символа точка (.) и подчертавка (_). Имената на променливите не могат да започват с цифра или с подчертавка и са чувствителни към регистъра на буквите (малки/големи).

Листинг 2.9: Операции за присвояване

```
a = 1
b <- 2
c = d = 3
e <- f <- 4
assign("g", 5)
a += 6
b -= 7
```

Стрелка на ляво (<-) служи за присвояване в R, но в повечето конвенционални програмни езици не присъства.

Листинг 2.10: Алтернативи за операцията присвояване

```
median(x = 1:10)
median(x <- 1:10)
```

Разликата между двете операции си проличава най-ясно при извикването на функции с аргументи (Листинг 2.10). В първия случай променливата x не остава в глобалната памет, а изчезва, докато при втория случай променливата x остава в глобалната памет след извикването.

Добра практика е за имена на променливите да се избират съществителни имена, а не еднобуквени имена или съкращения.

2.4.2 Премахване на променливи

Листинг 2.11: Премахване на променливи от глобалната памет

```
rm( a )
rm( list=ls ( ) )
```

Премахването на променлива от общата памет става с командата `rm` (Листинг 2.11). За да се почисти цялата глобална памет се дава списък с всички променливи, налични в глобалната памет. Въпреки че `R` извиква `Garbage Collector-a`, на определени интервали от време с командата `gc()` може да бъде отправена пряка заявка за освобождаване на ненужно заетата памет.

2.4.3 Дискретно представяне на информацията

Информацията в съвременните изчислителни машини се представя само с две нива 0 (няма сигнал) и 1 (има сигнал). Това е свързано с факта, че съвременните изчислителни машини използват електричество. Наличието само на две различни нива позволява единствено бинарна логика. Една клетка, която може да съдържа 0 или 1 носи един бит информация. Това количество е крайно недостатъчно за изчисленията, които хората имат нужда да правят. За да се разширят възможностите на бинарната логика се използва принципът на супер позицията. Отделните битове се подреждат един до друг и колкото по-вяло е един бит, толкова по-голяма тежест има. В зората на изчислителната техника, макар и да е имало коментари за 4 битови изчислителни машини, такава никога не е създадена за реална употреба. Изборът пада върху 8 битовите машини, които позволяват кодирането на 256 стойности в една машинна дума. Групата от 8 бита, хората наричат байт. Байт е единицата, която масово се използва в ежедневието, също и кратните ѝ форми за МВ (мегабайт), GB (гигабайт) или ТВ (терабайт). Дължината на машинната дума има пряка връзка с размерността на някои от първите типове данни. При 8 битовите машини, в `C` компилатора, типът `int` е 8 бита, при 16 битовите е 16 бита, а при 32 битовите е 32 бита. Подредбата на битовете в байтове позволява да се обозначат положителните числа и в езика `C` това е типът `unsigned int`. Тъй като в света, който познаваме отрицателни неща няма, то въвеждането на абстракцията за отрицателни числа е нещо неестествено за изчислителните машини и трябва да се избере някаква семантика за различаване на отрицателните от положителните числа. При числата със знак е прието най-старшият разред да бъде използван за знак. По този начин се получава феноменът `+0` и `-0` нещо, което не се разглежда в гимназиалния курс по математика. Ако най-старшият разряд е високо ниво, а останалите са в ниско се получава отрицателна нула. Ако всички разряди са ниско ниво, то се получава положителна нула. Ограничеността на машинната дума води до серия ограничения при работата с числа. Най-често срещаният проблем е препълването на разрядната решетка. Това е проблем, който и в наши дни се среща в програмния език `Java` (Листинг 2.12).

Листинг 2.12: Грешка от препълване при събиране

```
System.out.println( 2 + 2 );
4
System.out.println( 2_000_000_000 + 2_000_000_000 );
-294967296
```

Когато се налага да се извършват пресмятания с много големи числа е важно да не се ползват конвенционалните програмни езици, а софтуерни продукти като `Matlab`, `Mathematica` или `R`.

Светът, който познаваме не съдържа концепцията за дробност. Дори елементарните частици са изградени от по-малки елементарни частици. Причината за това е, че живеем в дискретен, а не в непрекъснат свят. Все пак, човекът е въвел концепцията за дробните числа, така че да извършва пресмятания, които да му позволяват по-ясно разбиране на света. Поради чисто физическата си природа разрядната решетка не може да съдържа дробни стойности. Представянето на цели числа е възможно чрез принципа на супер позицията, но той не е приложим при дробните числа. Най-лесният начин да се представи едно дробно число е чрез използването на две цели числа – едно за цялата част и едно за дробната част. Такова представяне е чисто схематично, на практика дробните числа в изчислителните машини се представят по `IEEE` стандарт. Както може да се препълни цялата част, така може да се препълни и дробната част, което води до проблем познат като `underflow`. Причината е в това, че компютърът не може да отрази понятието

за безкрайност, а между две цели числа (например между 0 и 1) има безкрайно много дробни числа. В изчислителната математика е възприето правилото, че отговорни пресмятания никога не се правят в дробни числа, а се търси вариант за пресмятане с цели числа или чрез пакети за символно пресмятане.

Тъй като в изчислителната техника съществуват само сигнали от ниско и високо ниво, то няма как да бъде кодирана информация за букви или текст. Естественят начин за комуникация между човека и машината са точно текстовете. Поради тази причина в изчислителната техника буквите от естествените езици са номерирани по стандартизирани кодови таблици. Две от най-популярните кодови таблици са ASCII и UTF-8. ASCII стандартът е силно ограничен и не позволява кодирането на повечето естествени езици, поради тази причина е създаден стандартът UTF-8 (един от най-разпространените в наше време), който позволява достатъчно икономично да бъдат представени символите от азбуките на познатите естествени езици.

2.4.4 Числени стойности

Условно R спада към групата на безтиповите езици, но всяка променлива вътрешно съхранява информация, която определя типа на данните. Възможни са различни типове данни, но най-използваните са `numeric`, `character` (включително символни низове), `Date/POSIXct` (астрономическо време) и `logical` (`TRUE/FALSE`).

Листинг 2.13: Проверка за типа на променливата

```
a = 5
class( a )
[1] "numeric"
```

С помощта на функцията `class` (Листинг 2.13) може да се провери типът на променливата. Тъй като R основно се използва за изчисления, то най-употребяваният тип данни са числовите данни. В този аспект `numeric` съответства на типовете `float` и `double` в другите конвенционални езици за програмиране. Този тип данни се използва за цели и дробни числа, както и за нулата. Това е типът по подразбиране, който се задава на променливата при числено присвояване.

Листинг 2.14: Проверка за типа `numeric`

```
is.numeric( a )
[1] TRUE
```

Проверката за `numeric` типа се извършва както е показано в листинг 2.14.

Листинг 2.15: Използване на целочислен тип

```
b <- 6L

class( b )
[1] "integer"

is.numeric( b )
[1] TRUE

is.integer( b )
[1] TRUE

is.integer( a )
[1] FALSE
```

R допуска използването само на цели числа, чрез типа `integer`. За да бъде указан типът `integer` към числото се добавя буквата `L` (Листинг 2.15).

2.4.5 Символни низове

Листинг 2.16: Символни низове в R

```
s1 <- "Simple_text."
s1
[1] "Simple_text."

class( s1 )
[1] "character"

s2 <- factor( "Another_simple_text." )
s2
[1] Another simple text.
Levels: Another simple text.

class( s2 )
[1] "factor"
```

За символни низове има два възможни варианта, които са показани на листинг 2.16. Символните низове в R са чувствителни към малки и големи букви. Фактор е специален символен низ, който намира приложение при векторите.

Листинг 2.17: Дължина на символен низ или числена стойност

```
nchar( s1 )
[1] 12

nchar( s2 )
Error in nchar(s2) : 'nchar()' requires a character vector

nchar( a )
[1] 1
```

Дължината на символен низ или числена стойност може да се определи с функцията `nchar`, която обаче не е приложима за променливите от тип фактор (Листинг 2.17).

2.4.6 Астрономическо време

При статистическата обработка на данни много често измерванията се извършват в точно определен момент от времето. Това налага наличието на възможности за обработка на астрономическо време. Работата с астрономическо време носи своите трудности поради множество фактори. От една страна има множество часови зони. Също така различните месеци в годината имат различен брой дни. Някои години се различават по брой дни. В една част от държавите се минава към лятно часово време и зимно часово време, докато в другите това не се прави.

Листинг 2.18: Типове данни за време

```
d1 <- as.Date("1979-04-21")
d1
[1] "1979-04-21"

class( d1 )
[1] "Date"
```

```

as.numeric( d1 )
[1] 3397

d2 <- as.POSIXct("1980-02-12_05:25")
d2
[1] "1980-02-12_05:25:00_EET"

class( d2 )
[1] "POSIXct" "POSIXt"

as.numeric( d2 )
[1] 319173900

```

В R най-често използваните типове за астрономическо време са Date и POSIXct (Листинг 2.18). Типът Date съхранява само информацията за дата, докато типът POSIXct съхранява информация за датата и за часа. Вътрешното представяне на Date е брой дни от 1 януари 1970 година, а вътрешното представяне на POSIXct е брой секунди от 00:00:00 на 1 януари 1970 година. Тъй като боравенето с информация за астрономическо време може да бъде относително сложно, то за тази цел са налични два помощни пакета - lubridate и chron.

2.4.7 Логически стойности

Логическият тип данни е също често използван и е в групата на простите типове данни. Променливите съдържат стойности FALSE или TRUE. На предефинираните константи са присвоени числени стойности 0 за FALSE и 1 за TRUE.

Листинг 2.19: Логически тип данни

```

x1 <- TRUE
x1
[1] TRUE

class( x1 )
[1] "logical"

is.logical( x1 )
[1] TRUE

as.numeric( x1 )
[1] 1

x2 <- F
x2
[1] FALSE

class( x2 )
[1] "logical"

is.logical( x2 )
[1] TRUE

as.numeric( x2 )
[1] 0

```

Освен пълното изписване на булевите стойности, може да се използват променливите Т и F, но те не са препоръчителни, тъй като могат да бъдат предефинирани и това да доведе до сериозни логически грешки (Листинг 2.19).

Листинг 2.20: Операции за сравнение

```
2 == 3  
[1] FALSE
```

```
5 != 6  
[1] TRUE
```

```
"Peter" == "Ivan"  
[1] FALSE
```

Логическият тип данни най-често се получава в следствие на операциите за сравнение (Листинг 2.20).

Заклучение

В тази глава са разгледани основните принципи за работа с пакети, работата с променливи, най-съществените операции с данни и някои от базовите типове данни.

Глава 3

Сложни структури от данни и извикване на функции

3.1 Извикване на функции

Функциите са последователност от инструкции, обособени като едно цяло, така че да са подходящи за многократно извикване. Функциите приемат входящи параметри, могат да имат върната стойност, а символът диез (#) се използва в началото на ред за коментар. Организацията на програмния текст във функции позволява лесна четимост и по-лесно откриване на програмни дефекти (бъгове). По своята същност командите в конзолата на R са функции, които потребителят извиква. Поради този факт е важно да се знаят възможностите за работа с функции. За разлика от масово наложените езици за обектно-ориентирано програмиране, в R функциите са по-съществени от обектите.

Листинг 3.1: Извикване на функции

```
x <- c(1, 2, 3, 5, 6, 7, 8, 9)
x
[1] 1 2 3 5 6 7 8 9

mean( x )
[1] 5.125

median( x )
[1] 5.5

sd( x )
[1] 2.900123
```

Листинг 3.1 демонстрира извикването на три функции, които получават като единствен аргумент вектор от числени стойности. По-сложните функции може да имат повече аргументи и те да се подават по различен начин.

Всяка функция, която е достъпна в R, има съпровождаща документация, но качеството на тази документация може да варира според уменията на автора ѝ. Най-бързият начин за достъп до информацията е чрез поставяне на въпросителен (?) пред името на функцията (Листинг 3.2).

Листинг 3.2: Документация за функциите

```
? mean
?? mean
```



```
? median
?? median
? sd
?? sd
```

Един въпросителен знак отваря информацията в локален прозорец, а два въпросителни знака отварят уеб страницата, съдържаща документацията на функцията.

Листинг 3.3: Документация за операции

```
? '+'
? '-'
? '*'
? '/'
```

За голяма част от операциите също може да се получи информация по сходен начин (Листинг 3.3), но операцията трябва да бъде оградена със символа апостроф (').

Листинг 3.4: Частично търсене

```
apropos( "med" )
[1] "elNamed"          "elNamed<-"      "median"          "median.default"
[5] "medpolish"        "runmed"
```

Често потребителите имат идея каква функция търсят, но не се досещат за точното изписване на името ѝ. В такива ситуации е полезна възможността за частично търсене, която предоставя функцията `apropos` (Листинг 3.4).

3.2 Вектори

Векторът е колекция от елементи, които са от един и същи тип (Листинг 3.5).

Листинг 3.5: Вектор от числа и вектор от символни низове

```
v1 <- c(1, 3, 2, 1, 5)
v2 <- c("Peter", "Ivan", "Geroge")
```

Векторите в R имат значителна роля за езика, тъй като R е векторизиран език, което го прави различен от конвенционалните програмни езици, като C/C++, C# или Java. Това означава, че всяка математическа операция се изпълнява върху целия вектор и не е нужно да се обикалят отделните елементи един по един (Листинг 3.6). За разлика от математическата концепция, в R векторите не се делят на вектор-стълб или вектор-ред. При нужда от вектор-ред или вектор-стълб може да се използват матрици с единична стойност на един от размерите.

Листинг 3.6: Базови операции над вектори

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

x * 5
[1] 5 10 15 20 25 30 35 40 45 50

x + 3
[1] 4 5 6 7 8 9 10 11 12 13
```

```

x ~ 4
[1] -3 -2 -1  0  1  2  3  4  5  6

x / 5
[1] 0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8 2.0

x ^ 3
[1]      1      8     27     64    125    216    343    512    729   1000

sqrt( x )
[1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
[9] 3.000000 3.162278

```

Основният начин за създаване на вектор е чрез функцията `c`, като названието ѝ идва от `combine` (комбиниране на елементи), но също е възможно да се използва и алтернативен запис (Листинг 3.7).

Листинг 3.7: Алтернативен синтаксис за създаване на вектори

```

1:10
[1] 1 2 3 4 5 6 7 8 9 10

10:1
[1] 10 9 8 7 6 5 4 3 2 1

-2:3
[1] -2 -1 0 1 2 3

5:-7
[1] 5 4 3 2 1 0 -1 -2 -3 -4 -5 -6 -7

```

Когато двата операнда на операцията са вектори с еднакви дължини, то операцията се прилага на всичките елементи по двойки (Листинг 3.8).

Листинг 3.8: Операции между вектори с еднаква дължина

```

x <- 1:10
y <- -10:-1

nchar( x )
[1] 1 1 1 1 1 1 1 1 1 2

nchar( y )
[1] 3 2 2 2 2 2 2 2 2 2

x + y
[1] -9 -7 -5 -3 -1  1  3  5  7  9

x - y
[1] 11 11 11 11 11 11 11 11 11 11

x * y
[1] -10 -18 -24 -28 -30 -30 -28 -24 -18 -10

x / y
[1] -0.1000000 -0.2222222 -0.3750000 -0.5714286 -0.8333333 -1.2000000

```

```
[7] -1.7500000 -2.6666667 -4.5000000 -10.0000000

x ^ y
[1] 1.0000000e+00 1.953125e-03 1.524158e-04 6.103516e-05 6.4000000e-05
[6] 1.286008e-04 4.164931e-04 1.953125e-03 1.234568e-02 1.0000000e-01

x > y
[1] TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE TRUE
```

Когато векторите са с различна дължина, по-късият вектор се превърта и се започва от началото му. Функцията `pchar` показва колко символа са необходими за изписването на всеки от елементите.

Листинг 3.9: Проверка дали някоя или всички стойности от вектора отговарят на определено условие

```
any(x+y < 0)
[1] TRUE

all(x+y < 0)
[1] FALSE
```

При група от проверки може да се установи дали всички елементи на вектора изпълняват определено условие или поне някои елементи го изпълняват (Листинг 3.9).

Листинг 3.10: Достъп до отделни елементи на вектор

```
x[ 1 ]
[1] 1

x[ 2:3 ]
[1] 2 3

x[ c(2,5,7) ]
[1] 2 5 7
```

Достъп до отделни елементи на вектор може да се осъществи по индекс или множество от индекси (Листинг 3.10).

Листинг 3.11: Имена на елементите на вектора

```
z <- c(One=1, Two=2, Three=3)
z
  One   Two Three
   1     2     3

names( z )
[1] "One"  "Two"  "Three"
```

R позволява на елементите на вектора да се поставят имена (Листинг 3.11).

Листинг 3.12: Трансформация на вектор във фактор

```
e <- c("High_School", "College", "Masters", "Doctorate")
e

[1] "High_School" "College"      "Masters"      "Doctorate"
fl <- as.factor( e )
fl
```

```
[1] High School College      Masters      Doctorate
Levels: College Doctorate High School Masters

as.numeric( f1 )
[1] 3 1 4 2

f2 <- factor(c("High_School", "College", "Masters", "Doctorate"),
             levels=c("High_School", "College", "Masters", "Doctorate"),
             ordered=TRUE)

f2
[1] High School College      Masters      Doctorate
Levels: High School < College < Masters < Doctorate

as.numeric( f2 )
[1] 1 2 3 4
```

Вектор може да бъде трансформиран във фактор с помощта на функции за трансформация (Листинг 3.12). Факторът е тип данни, при който всяка стойност се среща само по един път. Някои множества са неподредени (както е `f1`) и при тях няма значение редът на елементите, докато при подредените множества (както е `f2`) редът на елементите има значение. Функцията `factor` позволява изрично да се зададе какъв е редът на елементите. Факторът е значително по-икономичен на памет от вектора, тъй като се запазват единствено числените стойности на отделните елементи, но пък използването му може да доведе до трудни за откриване логически грешки.

Листинг 3.13: Вектори с латинските букви

```
letters
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r"
[19] "s" "t" "u" "v" "w" "x" "y" "z"

LETTERS
[1] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" "L" "M" "N" "O" "P" "Q" "R"
[19] "S" "T" "U" "V" "W" "X" "Y" "Z"
```

В R има два специално предефинирани вектора, които съдържат буквите от латинската азбука (Листинг 3.13).

3.3 Липсващи стойности

Липсващи стойности в данните е ежедневен проблем за хората обработващи статистическа информация. Причините за липсващите данни могат да произлизат от различни обстоятелства, например пропуснато измерване или дефектирал датчик. R дава две възможности за обозначаване на липсващи стойности в данните (NA и NULL). Макар да имат сходно значение, тези две стойности водят до различни резултати при различните пресмятания.

Когато има липсващи стойности в данните съществуват множество начини този факт да бъде отразен. В някои комплекти данни се записва недопустима числена стойност или се използва някаква символна комбинация. В езика R е възприето липсващите стойности да се обозначават с NA (Листинг 3.14).

Листинг 3.14: Липсващи стойности

```
x <- c(1, NA, 3, NA, 5)
x
[1] 1 NA 3 NA 5
```

```
is.na( x )  
[1] FALSE TRUE FALSE TRUE FALSE  
  
mean( x )  
[1] NA  
  
median( x )  
[1] NA  
  
sd( x )  
[1] NA
```

Значението на NULL е липса, а не изпусната стойност, поради тази причина векторът се редуцира с толкова елементи, колкото NULL стойности има в него (Листинг 3.15).

Листинг 3.15: Липсващи стойности

```
y <- c(1, NULL, 3, NULL, 5)  
y  
[1] 1 3 5  
  
is.na( y )  
[1] FALSE FALSE FALSE  
  
mean( y )  
[1] 3  
  
median( y )  
[1] 3  
  
sd( y )  
[1] 2  
  
is.null( y )  
[1] FALSE
```

Тъй като на практика векторът се редуцира с броя на NULL стойностите си, то функцията `is.null` не е векторизирана, а се отнася за целия обект.

3.4 Рамкирани данни

Рамкираните данни (`data.frame`) са една от най-полезните структури от данни в езика R. Най-интуитивната аналогия за рамкирани данни е един лист (`data sheet`) в Microsoft Excel, състоящ се от колони и редове. В термините на статистиката, всяка колона е наблюдавана променлива, а всеки ред е едно конкретно наблюдение (измерване). В термините на R, всяка колона е вектор, а дължината на всичките вектори е една и съща. По този начин всяка колона може да съдържа различни типове данни. Също така, в рамките на една колона всички елементи са от един и същи тип.

Съществуват множество начини да се създадат рамкирани данни, но най-лесният е с функцията `data.frame`.

Листинг 3.16: Създаване на рамкирани данни

```
x <- sample(1:5)
x
[1] 4 1 3 2 5

y <- sample(-2:2)
y
[1] 0 -2 1 -1 2

q <- c("Football", "Basketball", "Volleyball", "Handball", "Rugby")
q
[1] "Football" "Basketball" "Volleyball" "Handball" "Rugby"

df1 <- data.frame(x, y, q)
df1
  x y      q
1 4 0 Football
2 1 -2 Basketball
3 3 1 Volleyball
4 2 -1 Handball
5 5 2 Rugby
```

Листинг 3.16 демонстрира създаването на рамкирани данни от два вектора с числа (`sample` служи за разбъркване на стойностите по случаен начин) и един вектор със символни низове. Така получената структура е с размери 5x3 и се състои от три вектора. Имената на колоните се вземат служебно, но е възможно те да бъдат определени при създаването на самата структура (Листинг 3.17).

Листинг 3.17: Създаване на рамкирани данни с имена на колоните

```
df2 <- data.frame(First=x, Second=y, Sport=q)
rownames( df2 ) <- c("One", "Two", "Three", "Four", "Five")
df2
      First Second      Sport
One         4      0 Football
Two         1     -2 Basketball
Three        3      1 Volleyball
Four         2     -1 Handball
Five         5      2 Rugby
```

Рамкираните данни имат множество атрибути, като най-съществените са броят редове и броят колони (Листинг 3.18). Атрибутите имат съществено значение при прилагането на различните алгоритми за статистически анализ.

Листинг 3.18: Атрибути на рамкираните данни

```
nrow( df1 )
[1] 5

ncol( df1 )
[1] 3

dim( df1 )
[1] 5 3

names( df2 )
[1] "First" "Second" "Sport"
```

```
rownames( df2 )
[1] "One"    "Two"    "Three"  "Four"   "Five"

head(df1 , n=3)
  x  y      q
1 4  0 Football
2 1 -2 Basketball
3 3  1 Volleyball

tail(df1 , n=3)
  x  y      q
3 3  1 Volleyball
4 2 -1 Handball
5 5  2 Rugby

class( df1 )
[1] "data.frame"
```

Също така, може да се проверят имената на колоните и имената на редовете, с функцията `head` за първите няколко реда, а с функцията `tail` за последните няколко реда.

Листинг 3.19: Фактори в рамковите данни

```
df2[1, 2]
[1] 0

df2[3, 2:3]
      Second      Sport
Three      1 Volleyball

df2$Sport
[1] Football Basketball Volleyball Handball Rugby
Levels: Basketball Football Handball Rugby Volleyball

class( df2$Sport )
[1] "factor"

df2$Sport[1:2]
[1] Football Basketball
Levels: Basketball Football Handball Rugby Volleyball

df2[3, ]
      First Second      Sport
Three      3      1 Volleyball

df2[, c("First", "Sport")]
      First      Sport
One      4 Football
Two      1 Basketball
Three    3 Volleyball
Four     2 Handball
Five     5 Rugby
```

Рамкираните данни позволяват достъп до елементите като индекси на двумерен масив или директно с адресиране на конкретна колона (Листинг 3.19). Достъпът до цял ред става без указване на колона. За

достъп до колоните по име се съставя вектор с имената на колоните.

Листинг 3.20: Вътрешно представяне на факторите

```
f1 <- factor( c("Sofia", "Plovdiv", "Varna", "Burgas", "Ruse") )
f1
[1] Sofia    Plovdiv  Varna    Burgas    Ruse
Levels: Burgas Plovdiv Ruse Sofia Varna

model.matrix(~f1 - 1)
  f1Burgas f1Plovdiv f1Ruse f1Sofia f1Varna
1         0         0         0         1         0
2         0         1         0         0         0
3         0         0         0         0         1
4         1         0         0         0         0
5         0         0         1         0         0
attr(,"assign")
[1] 1 1 1 1 1
attr(,"contrasts")
attr(,"contrasts")$f1
[1] "contr.treatment"
```

Факторите са малко по-различни от векторите, за да се проследи вътрешното им представяне в рамкираните данни може да се приложи функцията `model.matrix` (Листинг 3.20).

3.5 Списъци

В някои ситуации е нужно да се ползва контейнер, който да съдържа обекти от различни типове. В R това се постига със списъчните структури. Този тип структури могат да съдържат голям брой и различни по тип елементи. Списъците се създават с функцията `list` (Листинг 3.21).

Листинг 3.21: Създаване на списък

```
l1 <- list(1, 2, 3, 4, 5)
l1
[[1]]
[1] 1

[[2]]
[1] 2

[[3]]
[1] 3

[[4]]
[1] 4

[[5]]
[1] 5
```

Всеки елемент в списъка е самостоятелен (Листинг 3.21), но е възможно да има и списък с единствен елемент, който е вектор (Листинг 3.22).

Листинг 3.22: Вектор в списък


```
l2 <- list( c(1, 2, 3, 4, 5) )
l2
[[1]]
[1] 1 2 3 4 5
```

Разнородни елементи на списък са показани в Листинг 3.23.

Листинг 3.23: Списък с разнородни данни

```
l3 <- list( df2, 1:5, l1 )
l3
[[1]]
      First Second      Sport
One       4      0  Football
Two       1     -2  Basketball
Three     3      1  Volleyball
Four      2     -1   Handball
Five      5      2    Rugby

[[2]]
[1] 1 2 3 4 5

[[3]]
[[3]][[1]]
[1] 1

[[3]][[2]]
[1] 2

[[3]][[3]]
[1] 3

[[3]][[4]]
[1] 4

[[3]][[5]]
[1] 5
```

По подобие на рамкираните данни, списъците също могат да съдържат наименования на елементите си (Листинг 3.24).

Листинг 3.24: Названия на елементите в списъка

```
names( l3 ) <- c("Frame", "Vector", "Element")
l3
$Frame
      First Second      Sport
One       4      0  Football
Two       1     -2  Basketball
Three     3      1  Volleyball
Four      2     -1   Handball
Five      5      2    Rugby

$Vector
[1] 1 2 3 4 5
```

```

$Element
$Element [[1]]
[1] 1

$Element [[2]]
[1] 2

$Element [[3]]
[1] 3

$Element [[4]]
[1] 4

$Element [[5]]
[1] 5

```

Достъпът до елементите на списъка може да стане по индекс или по название на елемента (Листинг 3.25).

Листинг 3.25: Достъп до елементите на списъка

```

13 [ 1 ]
$Frame
      First Second      Sport
One       4      0  Football
Two       1     -2 Basketball
Three     3      1 Volleyball
Four      2     -1  Handball
Five      5      2    Rugby

13 [ "Frame" ]
$Frame
      First Second      Sport
One       4      0  Football
Two       1     -2 Basketball
Three     3      1 Volleyball
Four      2     -1  Handball
Five      5      2    Rugby

```

Чрез вложено позоваване може да се достъпи конкретен елемент (Листинг 3.26). Тъй като сложните структури от данни могат да съдържат на свой ред сложни структури от данни, вложеното позоваване може да добие твърде неприветлив вид.

Листинг 3.26: Вложено позоваване

```

13 [[1]]
      First Second      Sport
One       4      0  Football
Two       1     -2 Basketball
Three     3      1 Volleyball
Four      2     -1  Handball
Five      5      2    Rugby

13 [["Frame"]][Sport]
[1] Football Basketball Volleyball Handball Rugby
Levels: Basketball Football Handball Rugby Volleyball

```

Добавянето на елемент към списъка става с директно позоваване към елемента, на който индекс трябва да попадне новият елемент, дори и това място да не е предварително предвидено (Листинг 3.27).

Листинг 3.27: Добавяне на елемент

```
l3 [ 4 ] <- "Games"
l3
$Frame
      First Second      Sport
One       4      0   Football
Two       1     -2  Basketball
Three     3      1  Volleyball
Four      2     -1   Handball
Five      5      2     Rugby

$Vector
[1] 1 2 3 4 5

$Element
$Element [[1]]
[1] 1

$Element [[2]]
[1] 2

$Element [[3]]
[1] 3

$Element [[4]]
[1] 4

$Element [[5]]
[1] 5

[[4]]
[1] "Games"

length( l3 )
[1] 4
```

3.6 Матрици

Една от най-важните структури в математиката и статистиката е матрицата. Матриците в R много наподобяват рамкираните данни, тъй като се състоят от колони и редове с разликата, че всички елементи на матрицата са еднотипни. По аналогия с векторите, матриците също се обработват с матрична аритметика, а не с обикаляне на елементите един по един.

Листинг 3.28: Създаване на матрици

```
m1 <- matrix(1:6, nrow=3)
m1
      [,1] [,2]
[1,]    1    4
[2,]    2    5
```

```
[3,]      3      6

m2 <- matrix(7:12, nrow=3)
m2
      [,1] [,2]
[1,]      7     10
[2,]      8     11
[3,]      9     12

m3 <- matrix(7:18, nrow=2)
m3
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,]      7      9     11     13     15     17
[2,]      8     10     12     14     16     18
```

Създаването на матрици става с функцията `matrix` (Листинг 3.28). От съществено значение е размерът на матрицата, както и попълването на елементите, което се случва колона по колона.

Листинг 3.29: Операции с матрици

```
nrow( m1 )
[1] 3

ncol( m1 )
[1] 2

dim( m1 )
[1] 3 2

m1 + m2
      [,1] [,2]
[1,]      8     14
[2,]     10     16
[3,]     12     18

m1 * m2
      [,1] [,2]
[1,]      7     40
[2,]     16     55
[3,]     27     72

m1 == m2
      [,1] [,2]
[1,] FALSE FALSE
[2,] FALSE FALSE
[3,] FALSE FALSE
```

Повечето матрични операции се изпълняват елемент за елемент (Листинг 3.29), но матричното умножение е малко по-особено тъй като изисква съчетаване на размерите по колони и редове (Листинг 3.30).

Листинг 3.30: Матрично умножение

```
m1 %*% t(m2)
      [,1] [,2] [,3]
[1,]     47     52     57
```

[2,]	64	71	78
[3,]	81	90	99

Както при рамкираните данни, така и при матриците може да има имена на колоните и редовете (Листинг 3.31). Тази възможност значително подобрява визуализирането на данните след извършването на математическите пресмятания.

Листинг 3.31: Имена на колоните и редовете

```
colnames( m1 ) <- c("First", "Second")
rownames( m1 ) <- c("One", "Two", "Three")
m1
      First Second
One       1      4
Two       2      5
Three     3      6

colnames( m2 ) <- c("Left", "Right")
rownames( m2 ) <- c("1st", "2nd", "3rd")
m2
      Left Right
1st     7     10
2nd     8     11
3rd     9     12
```

Функцията `t` служи за транспониране на матрица. Транспонирането най-често се налага при матричното умножение (Листинг 3.30). За да бъде транспонирана една матрица, елементите ѝ се разменят симетрично, спрямо главния диагонал. Не е нужно матрицата да бъде квадратна за да бъде транспонирана. Транспонирането е валидно и за правоъгълни матрици.

3.7 Масиви

Масивът по своята същност е многомерен вектор. Елементите на масива са еднотипни и достъпът до тях също се осъществява по индекс с квадратни скоби. Първият индекс е за ред, а вторият за колона и така нататък за по-високите размерности.

Листинг 3.32: Работа с масиви

```
a1 <- array(1:12, dim = c(2, 3, 2))
a1
, , 1
      [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2
      [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

a1[1, , ]
      [,1] [,2]
```

```

[1,]      1      7
[2,]      3      9
[3,]      5     11

a1[1, , 1]
[1] 1 3 5

a1[ , , 1]
[ ,1] [ ,2] [ ,3]
[1,]      1      3      5
[2,]      2      4      6

```

Основната разлика между матриците и масивите е, че матриците са ограничени до две размерности, докато масивите могат да имат много измерения.

Заклучение

В настоящата глава са представени възможностите за извикване на функции в R. Разгледани са начините за извикване на документация за функциите. Представени са начини за работа с липсващи данни и са демонстрирани някои от по-сложните типове данни.

Глава 4

Въвеждане на данни и извеждане на графики

Статистическата обработка на данни в R започва с въвеждането на събраната информация и завършва с визуализация на резултатите от анализа. Тези две фази от етапа на статистическата обработка имат своята важност, тъй като входните данни силно определят надеждността на извършвания анализ, а правилно визуализираните резултати определят степента на разбиране, която ще постигне аудиторията, пред която анализът се представя.

4.1 Въвеждане на данни от външни източници

Данните в примерите до тази глава бяха фиксирани и се въвеждаха ръчно от конзолата, в интерактивен режим. Този начин на работа не е най-рационалният, когато се правят модели и с данните за модела се провеждат многократни експерименти. Обичайната практика е командите за съставянето на модела да бъдат написани в обикновен текстов файл, с разширение „.r“, а данните да бъдат зареждани от външен файл. Този начин на работа позволява да бъдат създадени множество модели, които често се различават по нещо дребно и също да бъдат зареждани различни входни данни, например за различни периоди на измерване.

4.1.1 CSV файлове

Продуктът R позволява множество различни начини за въвеждане на данни в системата, но най-достъпният начин е през CSV (Comma Separated Values) файлове. CSV файловият формат е текстов файлов формат, който позволява таблично представяне на данни (колони и редове). CSV може да бъде четен и редактиран с обикновен текстов редактор, като Notepad под Microsoft Windows, TextEdit под Mac OS X или Nano под Linux. CSV комфортно се визуализира и обработва от продуктите Microsoft Excel, OpenOffice Calc и Libre Calc.

Листинг 4.1: Зареждане на данни от CSV файл

```
df <- read.table(file="http://raw.githubusercontent.com/TodorBalabanov/
  Statistical-Data-Processing-with-R/master/data/tomato.csv", header=TRUE, sep=
  ",")

head(df)
  Round      Tomato Price      Source Sweet Acid Color Texture Overall
1     1      Simpson SM   3.99 Whole Foods    2.8  2.8   3.7    3.4    3.4
```

2	1	Tuttorosso (blue)	2.99	Pioneer	3.3	2.8	3.4	3.0	2.9
3	1	Tuttorosso (green)	0.99	Pioneer	2.8	2.6	3.3	2.8	2.9
4	1	La Fede SM DOP	3.99	Shop Rite	2.6	2.8	3.0	2.3	2.8
5	2	Cento SM DOP	5.49	D Agostino	3.3	3.1	2.9	2.8	3.1
6	2	Cento Organic	4.99	D Agostino	3.2	2.9	2.9	3.1	2.9
Avg. of. Totals Total. of. Avg									
1		16.1	16.1						
2		15.3	15.3						
3		14.3	14.3						
4		13.4	13.4						
5		14.4	15.2						
6		15.5	15.1						
tail(df)									
	Round	Tomato	Price	Source	Sweet	Acid	Color	Texture	
11	3	Scotts Backyard SM	0.00	Home Grown	1.6	2.9	3.1	2.4	
12	3	Di Casa Barone (organic)	12.80	Eataly	1.7	3.6	3.8	2.3	
13	4	Trader Joes Plum	1.49	Trader Joes	3.4	3.3	4.0	3.6	
14	4	365 Whole Foods	1.49	Whole Foods	2.8	2.7	3.4	3.1	
15	4	Muir Glen Organic	3.19	Whole Foods	2.9	2.8	2.7	3.2	
16	4	Bionature Organic	3.39	Whole Foods	2.4	3.3	3.4	3.2	
Overall Avg. of. Totals Total. of. Avg									
11	1.9	11.9	11.9						
12	1.4	12.7	12.7						
13	3.9	17.8	18.2						
14	3.1	14.8	15.2						
15	3.1	14.8	14.7						
16	2.8	15.1	15.2						

Зареждането на CSV в R най-ефективно се постига с функцията `read.table` (Листинг 4.1). Резултатът от четенето е обект от тип рамкирани данни. При извикването на функцията, параметрите се подават с явно изписване на имената им. Точният адрес на файла се подава в кавички, а когато първият ред от данните е заглавен ред, се подава флаг за заглавен ред. Третият аргумент е за указване на разделителя в редовете, тъй като не винаги този разделител е запетая. Често софтуерните продукти за електронни таблици поставят символа за табулация като разделител между данните на един ред. Символът табулация попада в групата на белите символи (white spaces), тъй като не се изобразява видимо, а с празно пространство. Когато трябва да бъде подаден като аргумент, за разделител се използва комбинацията от обратна наклонена черта и буквата `t` (`\t`).

Листинг 4.2: Проверка на типовете които колоните имат

sapply(df, class)				
Round	Tomato	Price	Source	Sweet
"integer"	"factor"	"numeric"	"factor"	"numeric"
Acid	Color	Texture	Overall	Avg. of. Totals
"numeric"	"numeric"	"numeric"	"numeric"	"numeric"
Total. of. Avg				
"numeric"				

При зареждане на данни от CSV файл по подразбиране текстовите колони се зареждат като фактори, а не като вектори от символни низове (Листинг 4.2). Тъй като използването на фактори има своите особености, понякога се налага зареждането да става в символни низове. За тези случаи функцията `read.table` има параметър `stringsAsFactors`, който може да се установи на `FALSE` и това ще предотврати зареждането на фактори (Листинг 4.3).

Листинг 4.3: Зареждане на символни низове

```

supply(read.table(file="http://raw.githubusercontent.com/TodorBalabanov/
  Statistical-Data-Processing-with-R/master/data/tomato.csv", header=TRUE, sep=
  ",", stringsAsFactors=FALSE), class)
  Round      Tomato      Price      Source      Sweet
  "integer" "character" "numeric" "character" "numeric"
  Acid      Color      Texture      Overall Avg. of . Totals
  "numeric" "numeric" "numeric" "numeric" "numeric"
  Total. of . Avg
  "numeric"

```

Същият аргумент може да се използва и във функцията `data.frame`, когато от вектори се създават рамкирани данни.

Възможно е да се появят проблеми с прочитането на CSV файловете. Причини за тези проблеми може да са лошо форматиране или наличие на разделители за стойностите в редовете. В такива ситуации могат да се пробват алтернативните функции за четене, каквито са `read.csv2` и `read.delim2`.

4.1.2 Excel файлове

Microsoft Excel е може би най-популярният инструмент за извършване на статистически анализи и въпреки това четенето на Excel файлове в R не е толкова лесно. Основните трудности идват от това, че Microsoft Excel е комерсиален софтуер и бинарните му файлови формати не са с отворен лиценз. Най-лесният начин за четене на данни от Excel е файловете да бъдат съхранени като CSV файлове.

Общността разработчици полага някои усилия да осигури четене на Excel файлове директно в R, но наличните пакети, като `gdata`, `XLConnect`, `xlsReadWrite`, далеч не са достатъчно надеждни и често изискват допълнителни софтуерни модули, като Java, Perl или 32 битова версия на R. В пакета `RODBC` съществува функция `odbcConnectExcel2007`, която чете Excel файлове, но тя изисква DSN (Data Source Name) връзка (най-често връзка към база данни). По своята структура, файловете след Excel 2007 са в XML формат, което би трябвало да улеснява разчитането им, но за момента R не предлага такава възможност.

4.1.3 SQL бази данни

Голямо количество от данните, събирани до наши дни, се съхраняват в бази данни. Повечето от тези системи за управление на бази от данни са релационни и разчитат на езика SQL [19] за манипулация на структурата или самите данни. За най-популярните релационни бази данни в R са достъпни пакети като `Rpostgresql` или `Rmysql`. За други релационни бази данни, които не са съпроводжани с конкретен R пакет може да се използва пакетът `RODBC`. Връзката към база данни може да е съпроводена с много трудности и поради тази причина е създаден пакетът `DBI`. Този пакет позволява унифициран начин на работа с различните бази данни.

Боравенето с релационна база данни е извън обхвата на настоящото изложение и вследствие на тази причина примерите са реализирани на `SQLite` като една от най-достъпните и лесни за използване системи за управление на бази от данни.

Командният интерпретатор на R се изпълнява с конкретна работна директория. С функцията `getwd` тази директория може да бъде проверена, а с функцията `setwd` директорията може да бъде променена (Листинг 4.4).

Листинг 4.4: Работна директория

```

getwd()
[1] "/Users/todorbalabanov"

```

```
setwd("~/Desktop")
getwd()
[1] "/Users/todorbalabanov/Desktop"
```

За улеснение по време на работа, работната директория се установява да бъде директорията на работния плот, където ще се разположи и файлът с данните. Файлът може да бъде свален и със средствата на операционната система, но R предоставя команда за тази операция (Листинг 4.5).

Листинг 4.5: Сваляне на файл с данни

```
download.file("https://github.com/TodorBalabanov/Statistical-Data-Processing-
  with-R/blob/master/data/diamonds.db?raw=true", destfile="./diamonds.db", mode
  ="wb")

trying URL 'https://github.com/TodorBalabanov/Statistical-Data-Processing-with-R
  /blob/master/data/diamonds.db?raw=true'

Content type 'application/octet-stream' length 5909504 bytes (5.6 MB)

downloaded 5.6 MB
```

Едно от многото предимства на SQLite е, че базата данни се помещава в един единствен файл. Тъй като за SQLite е разработен конкретен R пакет, то той се използва за връзка с данните (Листинг 4.6). При липса на конкретен пакет остава алтернативата за използване на RODBC.

Листинг 4.6: Връзка към базата данни

```
library(RSQLite)

driver <- dbDriver("SQLite")
class(driver)
[1] "SQLiteDriver"
attr(,"package")
[1] "RSQLite"

connection <- dbConnect(driver, "./diamonds.db")
class(connection)
[1] "SQLiteConnection"
attr(,"package")
[1] "RSQLite"
```

След зареждането на пакета за работа с базата данни се зарежда драйверът. Променливата, която съдържа драйвера се подава като аргумент на функцията за осъществяване на връзка към базата данни. Командата за осъществяване на връзка към базата данни може да се различава за различните операционни системи, така че е съществено да се провери документацията на R за конкретната операционна система.

След като бъде изградена връзка към базата данни, може да се изгълнят команди за изследване на структурата и данните (Листинг 4.7).

Листинг 4.7: Изследване на базата данни

```
dbListTables(connection)
[1] "DiamondColors" "diamonds"      "sqlite_stat1"

dbListFields(connection, name="diamonds")
```

```
[1] "carat" "cut" "color" "clarity" "depth" "table" "price"
[8] "x" "y" "z"

dbListFields(connection, name="DiamondColors")
[1] "Color" "Description" "Details"
```

Когато структурата на базата данни е известна, над нея могат да се изпълняват всички валидни SQL заявки. Тази цел се постига с функцията `dbGetQuery`, която връща `data.frame` структура в резултат (Листинг 4.8).

Листинг 4.8: Изследване на базата данни

```
# Simple select query.
diamondsTable <- dbGetQuery(connection, "SELECT_*_FROM_diamonds",
  stringsAsFactors = FALSE)
colorTable <- dbGetQuery(connection, "SELECT_*_FROM_DiamondColors",
  stringsAsFactors = FALSE)

# Join between the two tables.
diamondsJoin <- dbGetQuery(connection, "SELECT_*_FROM_diamonds, _DiamondColors_
  WHERE_diamonds.color_=_DiamondColors.Color", stringsAsFactors = FALSE)

head(diamondsTable, n=3)
  carat      cut color clarity depth table price      x      y      z
1  0.23   Ideal     E    SI2   61.5    55   326  3.95  3.98  2.43
2  0.21 Premium     E    SI1   59.8    61   326  3.89  3.84  2.31
3  0.23    Good     E    VS1   56.9    65   327  4.05  4.07  2.31

head(colorTable, n=3)
  Color      Description      Details
1    D Absolutely Colorless      No color
2    E              Colorless Minute traces of color
3    F              Colorless Minute traces of color

head(diamondsJoin, n=3)
  carat      cut color clarity depth table price      x      y      z Color
1  0.23   Ideal     E    SI2   61.5    55   326  3.95  3.98  2.43     E
2  0.21 Premium     E    SI1   59.8    61   326  3.89  3.84  2.31     E
3  0.23    Good     E    VS1   56.9    65   327  4.05  4.07  2.31     E
  Description      Details
1 Colorless Minute traces of color
2 Colorless Minute traces of color
3 Colorless Minute traces of color
```

При затварянето на R сесията, връзката към базата данни ще бъде прекратена, но добрата работна практика изисква всички ненужни повече ресурси да се освобождават веднага, след като работата с тях приключи. За тази цел в R има команда за прекратяване на връзката към базата данни (Листинг 4.9).

Листинг 4.9: Прекъсване на връзката към базата данни

```
dbDisconnect( connection )
```

Важно е също да се вземе предвид, че R може да поддържа само една връзка към база данни в конкретен период от време. При нужда да се работи с повече от една база данни, връзките трябва да се редуват.

4.1.4 Други статистически програми като източници на данни

Тъй като R е само една от алтернативите за статистическа обработка на данни, в реалната практика се използват множество други софтуерни решения, част от които разчитат на затворени (търговски) файлови формати (например SPSS, SAS или Octave). За достъп до тези файлове, пакетът `foreign` предлага множество функции, работещи по сходен начин на функцията `read.table`. Част от функциите са изброени в Таблица 4.1.

Функция	Файлов формат
<code>read.spss</code>	SPSS
<code>read.ssd</code>	SAS
<code>read.ocatave</code>	Octave
<code>read.dta</code>	Stata
<code>read.systat</code>	Systat
<code>read.mtp</code>	Minitab

Таблица 4.1: Функции за четене на данни

Параметрите на групата функции са подобни на параметрите подавани към `read.table`. В общия случай, функциите връщат резултат под формата на `data.frame`. За някои файлови формати (например SAS) може да се изисква валиден софтуерен лиценз.

4.1.5 Бинарни файлове на R

При работа между различни R потребители е удачно информацията да се разменя в бинарния файлов формат поддържан от R (`Rdata`). Този файлов формат е бинарен и поддържа различните обекти, които са достъпни в процеса на работа с R. Голямо предимство на този файлов формат е, че е съвместим с различните операционни системи и може да се предават данни между различни инсталации на програмния продукт.

Листинг 4.10: Използване на множество от данни

```
tomato <- read.table(file="http://raw.githubusercontent.com/TodorBalabanov/
  Statistical-Data-Processing-with-R/master/data/tomato.csv", header=TRUE, sep=
  ",")

head(tomato, n=3)
  Round      Tomato Price Source Sweet Acid Color Texture Overall
1     1      Simpson SM  3.99 Whole Foods  2.8  2.8  3.7    3.4    3.4
2     1  Tutturosso (blue)  2.99 Pioneer  3.3  2.8  3.4    3.0    2.9
3     1  Tutturosso (green)  0.99 Pioneer  2.8  2.6  3.3    2.8    2.9
  Avg. of. Totals Total. of. Avg
1          16.1          16.1
2          15.3          15.3
3          14.3          14.3
```

При наличен `data.frame` в общата памет (Листинг 4.10), следва да се изпълнят команди за съхраняване, изтриване на общата памет и прочитане на съхранените данни (Листинг 4.11).

Листинг 4.11: Запис и четене в `RData` файл

```
save(tomato, file="./tomato.rdata")

rm( tomato )
```

```
head( tomato )
Error in head(tomato) : object 'tomato' not found

load("tomato.rdata")

head(tomato, n=3)
```

	Round	Tomato	Price	Source	Sweet	Acid	Color	Texture	Overall
1	1	Simpson SM	3.99	Whole Foods	2.8	2.8	3.7	3.4	3.4
2	1	Tuttorosso (blue)	2.99	Pioneer	3.3	2.8	3.4	3.0	2.9
3	1	Tuttorosso (green)	0.99	Pioneer	2.8	2.6	3.3	2.8	2.9
	Avg. of . Totals		Total. of . Avg						
1		16.1		16.1					
2		15.3		15.3					
3		14.3		14.3					

Всички обекти, които трябва да се съхранят в RData файла се изброяват преди името на самия файл. При възстановяването им от диска в общата памет те придобиват имената, които са имали по време на съхраняването.

Листинг 4.12: Запис и четене на един обект

```
saveRDS(c(21,04,1979), "object.rds")

readRDS("object.rds")
[1] 21 4 1979
```

Съществува и втора възможност за запазване на данни, под формата на RDS файл. Запазването става с функцията `saveRDS`, а прочитането с функцията `readRDS`. Разликата с предходните функции е, че в този случай се съхранява само един обект, без да се запазва неговото име и при прочитането резултатът трябва да се присвои изрично на променлива (Листинг 4.12).

4.1.6 Данни достъпни директно от R

Софтуерният продукт R и някои от неговите пакети идват с предварително заложените множества от данни. Този вид примерни данни основно се използват за демонстриране на възможностите, които R има или възможностите, които съответният пакет предоставя. За да бъдат ползвани тези данни е достатъчно да се знае в кой пакет се намират.

Листинг 4.13: Зареждане на примерни данни

```
data(diamonds, package="ggplot2")

head(diamonds, n=3)
```

	carat	cut	color	clarity	depth	table	price	x	y	z
1	0.23	Ideal	E	SI2	61.5	55	326	3.95	3.98	2.43
2	0.21	Premium	E	SI1	59.8	61	326	3.89	3.84	2.31
3	0.23	Good	E	VS1	56.9	65	327	4.05	4.07	2.31

Зареждането става с функцията `data` и името на пакета. Като например, множеството данни за диамантите е налично в пакета `ggplot2` (Листинг 4.13). Списък на всички налични комплекти от данни може да се получи, ако функцията `data` се извика без аргументи.

4.1.7 Четене на данни от JSON формат

Един от най-популярните съвременни формати на данни е JSON [14]. Първоначално е замислен като формат за обмен на JavaScript обекти, но след това намира широко приложение в множество софтуерни решения. JSON се използва за съхраняване и предаване на структурирана информация в чист текстов вид (plain text). Два са основните пакета за четене на JSON в R (rjson и jsonlite).

Листинг 4.14: Четене на JSON данни

```
library( jsonlite )

pizza <- fromJSON("https://raw.githubusercontent.com/TodorBalabanov/Statistical-Data-Processing-with-R/master/data/pizza.json")

head(pizza, n=3)
      Name                               Details
1 Di Fara Pizza      1424 Avenue J, Brooklyn, NY, 11230
2 Fiore's Pizza 165_Bleecker_St, New_York, NY, 10012
3 Julia's Pizza 19 Old Fulton St, Brooklyn, NY, 11201

class( pizza )
[1] "data.frame"

class( pizza$Name )
[1] "character"

class( pizza$Details )
[1] "list"

class( pizza$Details[[1]] )
[1] "data.frame"
```

Примерният файл съдържа описание на ресторанти за пица, което включва: име и подробности (адрес, град, пощенски код и телефон). Функцията `fromJSON` (Листинг 4.14) се опитва да представи резултата в `data.frame` структура. Такова представяне не винаги е възможно, тъй като JSON данните могат да бъдат организирани в сложна и/или непълна йерархия. Резултатът от четенето на примерните данни за пица-ресторантите е `data.frame` в две колони, но по своята същност всеки елемент в `Details` е също `data.frame`.

4.2 Визуализация на данни от статистически анализ

Една от най-трудните и отговорни фази в статистическата обработка на данни е визуализацията на получените резултати. Доброто графично оформление на получените резултати може значително да подобри разбирането на информацията от аудиторията, пред която тя се представя. За визуално оформление на резултатите, R предоставя достатъчно възможности, както в основната инсталация, така и в допълнителните пакети, като `ggplot2` и `lattice`.

За да се демонстрират графичните възможности на R е необходимо зареждането на подходящо множество от данни. Такова множество от данни е `diamonds` в пакета `ggplot2` (Листинг 4.15).

Листинг 4.15: Четене на JSON данни

```
library( ggplot2 )

data( diamonds )
```

```
head(diamonds, n=3)
# A tibble: 3 x 10
  carat cut      color clarity depth table price      x      y      z
  <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.23 Ideal   E      SI2     61.5   55   326   3.95   3.98   2.43
2  0.21 Premium E      SI1     59.8   61   326   3.89   3.84   2.31
3  0.23 Good    E      VS1     56.9   65   327   4.05   4.07   2.31
```

Данните за диамантите са организирани в 10 колони и отчитат 10 различни характеристики на скъпоценните камъни (Таблица 4.2).

Характеристика	Значение
carat	Тегло на камъка (дробно число)
cut	Качество на среза (изброимо множество)
color	Цвят на камъка (изброимо множество)
clarity	Чистота на камъка (изброимо множество)
depth	Дълбочина на камъка (проценти)
table	Ширина на горната част, спрямо най-широката част (дробно число)
price	Цена (щатски долари)
x	Дължина (милиметри)
y	Ширина (милиметри)
z	Дълбочина (милиметри)

Таблица 4.2: Характеристики на диамантите

Наличието на достатъчно и разнообразни характеристики в данните за диамантите, дава възможност за демонстриране на множество графични възможности в програмния продукт R.

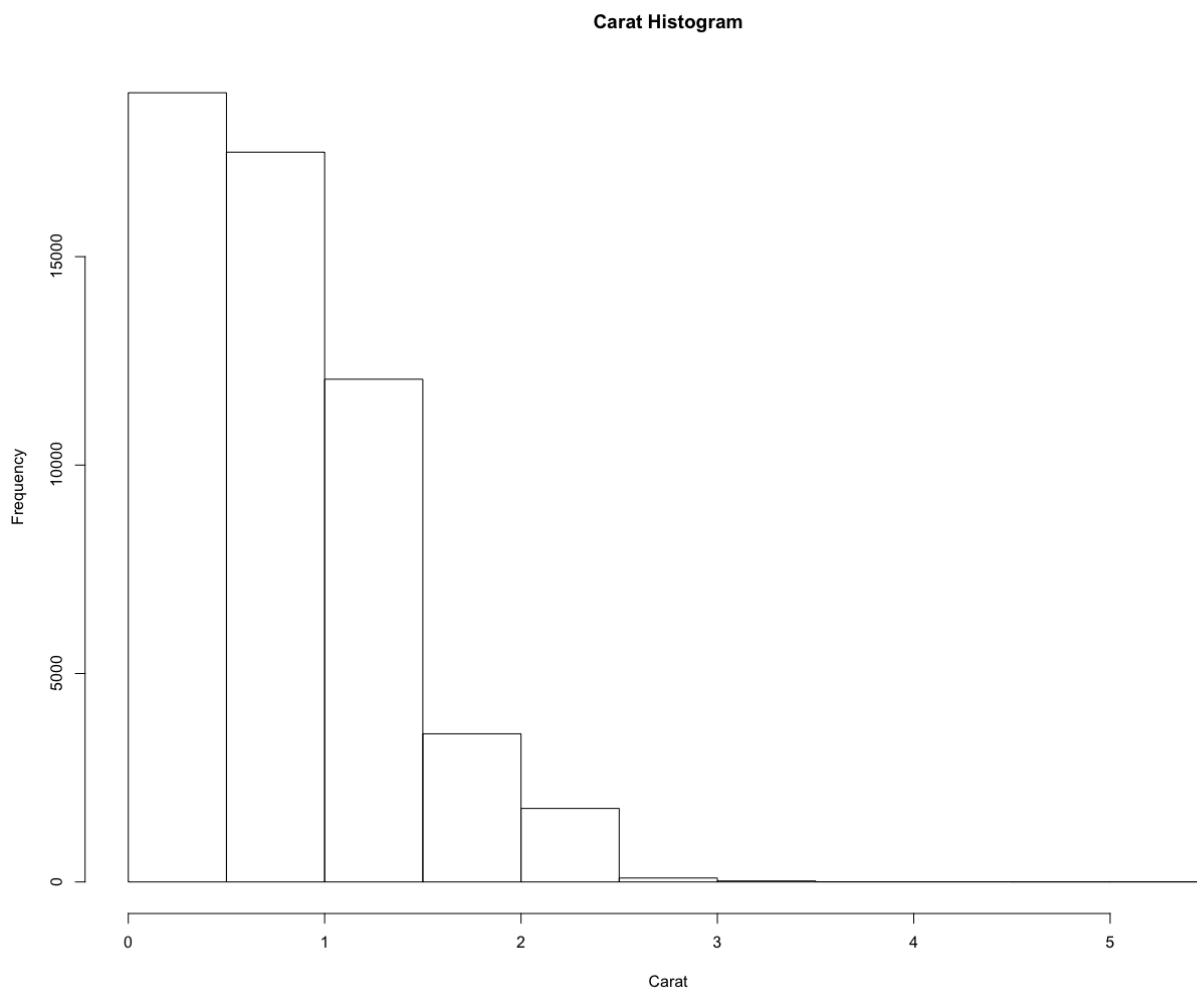
4.2.1 Хистограма

Най-базовата графика за онагледяване на една случайна променлива е хистограмата. Хистограмата показва разпределението на стойностите, които променливата може да приема.

Листинг 4.16: Генериране на хистограма за теглото на камъните

```
hist(diamonds$carat, main="_Carat_Histogram", xlab="_Carat")
```

В множеството за диамантите, теглото на камъка е идеален вариант да се демонстрират възможностите за изчертаване на хистограма (Листинг 4.16). Обичайно камъните не са пренебрежимо малки, но също така камъните имат и ограничен горен размер, който най-вече се налага от процеса за шлифоването им (Фиг. 4.1).



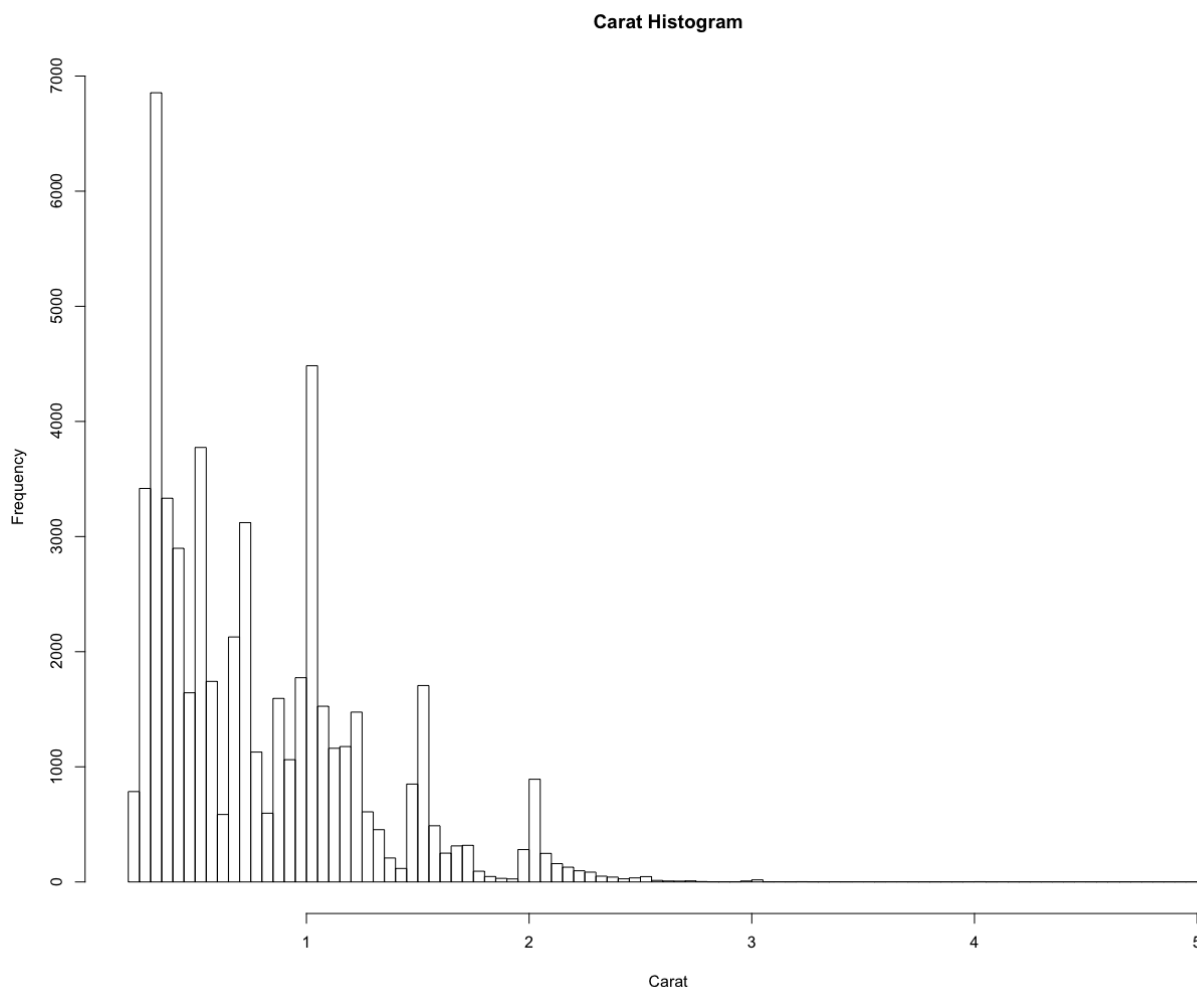
Фигура 4.1: Хистограма на каратите

Хистограмата се изчислява като предварително се избират броят групи, в които да се класифицират стойностите (Листинг 4.17), а след това се преброяват колко от стойностите попадат в съответната група.

Листинг 4.17: Хистограма с повече групи

```
hist(diamonds$carat , main="Carat_Histogram" , xlab="Carat" , nclass=100)
```

Функцията `hist` самостоятелно избира броят групи (`nclass`), но когато е нужна по-дребна гранулярност тя може да бъде изрично зададена (Фиг. 4.2).



Фигура 4.2: Хистограма на каратите при 100 групи

При изследването на непозната вероятностна променлива, хистограмата е първият инструмент, който може да послужи за ориентир какъв статистически анализ да се приложи. Колко групи да се използват за генериране на хистограмата, най-често се определя с експерименти, докато се достигне желаната експресивност.

4.2.2 Диаграма на разсейване

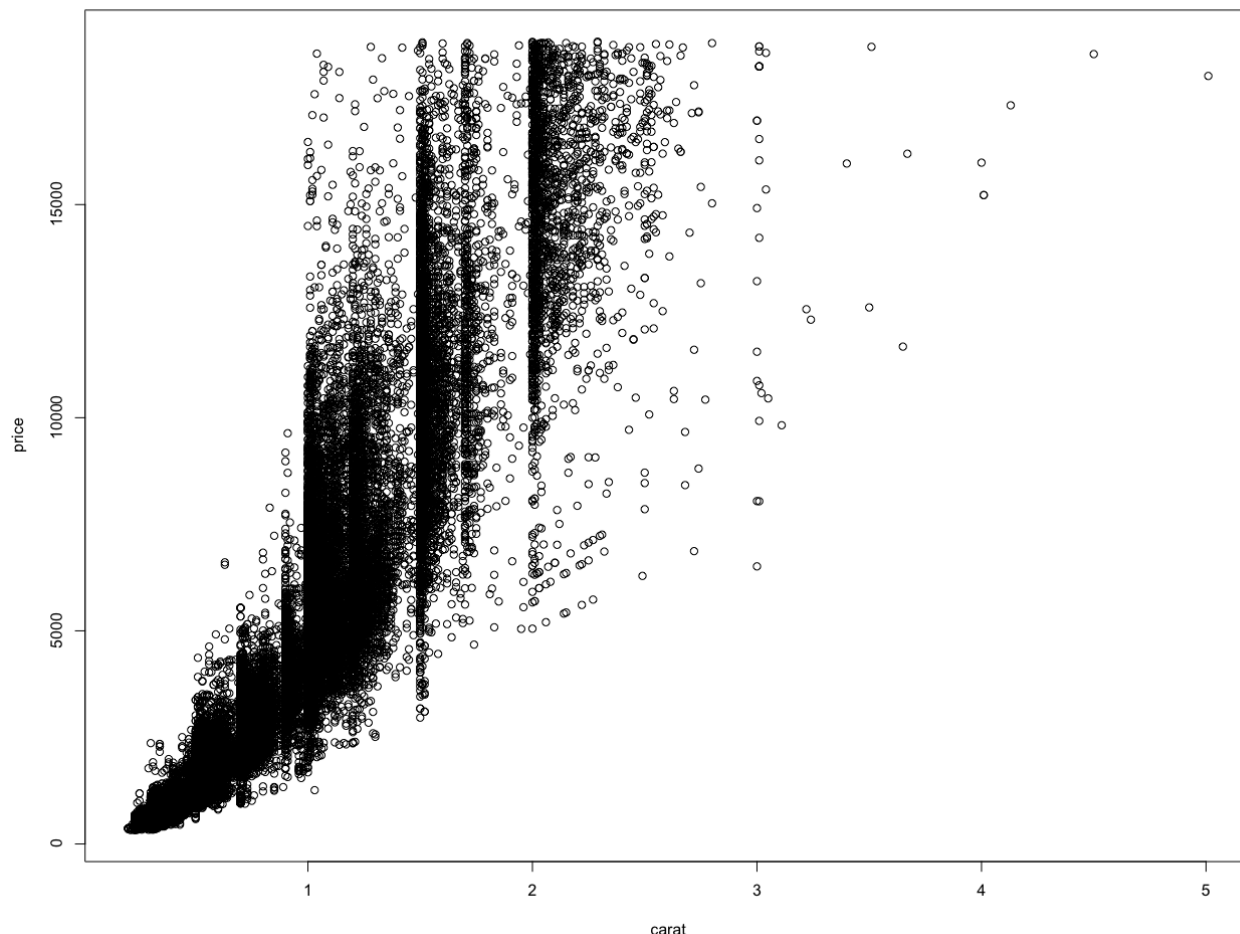
В статистическата обработка на данни много често е от значение как две случайни променливи си влияят една на друга. В такава ситуация изключително полезен инструмент се явява диаграмата на разсейване (scatter plot). Всяка точка на диаграмата отразява едно наблюдение на двете променливи. Едната променлива се отчита по оста x , а другата променлива се отчита по оста y .

Листинг 4.18: Генериране на диаграма на разсейване

```
plot(price~carat, data=diamonds)
```

Чисто интуитивно знаем, че колкото по-голям е един скъпоценен камък, толкова по-висока е цената му. В същото време, цената на камъка се определя и от други фактори. Диаграмата на разсейването

(Листинг 4.18) може да ни покаже каква е връзката между тегло и цена при скъпоценните камъни като не отчита другите фактори за определяне на цената.



Фигура 4.3: Диаграма на разсейване за камъните според отношението тегло към цена

За разчертаването на диаграмата се използва нотацията за формули. При тази нотация, символът тилда (\sim) показва, че се търси зависимостта на цената от теглото. Поради тази причина цената се изобразява на ординатната ос, а теглото на камъка на абсцисната ос. На Фиг. 4.3 ясно се вижда, че с нарастване на размера, нараства и цената на скъпоценния камък. В допълнение се забелязват и някои по-особени зони, които нарушават равномерното разпределение на точките. Този вид аномалии подсказват за наличието и на други фактори при формирането на цената. Тъй като информацията е недостатъчна, диаграмата на разсейването не може да даде идея кои фактори допълнително оказват влияние за образуването на цената.

Листинг 4.19: Алтернативна команда за диаграма на разсейване

```
plot(diamonds$carat , diamonds$price)
```

Не е нужно вероятностните променливи да се намират в един data.frame за да се генерира диаграма на разсейване. Достатъчно е да се използва алтернативния запис за извикване на функцията plot (Листинг 4.19).

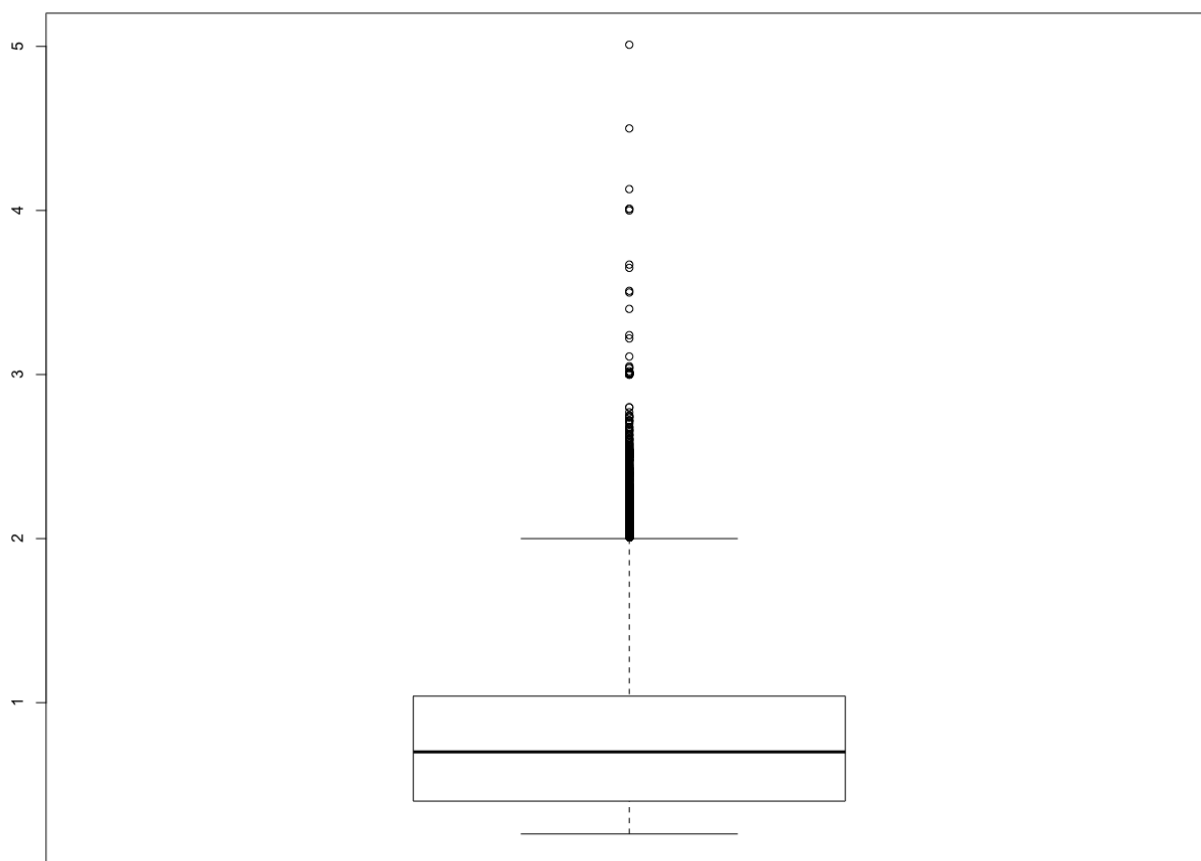
4.2.3 Графики тип кутия

Графиките тип кутия (box plot) са инструмент за изобразяване на статистическа информация, който не среща голямо одобрение в общността на статистиците. Въпреки мнението в професионалните общности, графиките тип кутия са често използвани от студентите.

Листинг 4.20: Генериране на графика от тип кутия

```
boxplot( diamonds$carat )
```

Информацията от хистограмата за каратите на скъпоценните камъни може да се представи в значително по-стилизиран вид, чрез графики от тип кутия (Листинг 4.20).



Фигура 4.4: Графика тип кутия за теглото на диамантите

При графиката от тип кутия (Фиг. 4.4) линията, която преминава през средата на кутията показва къде е медианата на множеството от стойностите. Медианата се изчислява като се сортират всички елементи и се вземе стойността на елемента, който стои в средата. Ако броят елементи е четно число, медианата е средна стойност между двете стойности, стоящи в средата на множеството. Медианата разделя множеството на две равни части. Горният и долният ръб на кутията показват първия квантил и третия квантил. На практика, тези три черти разделят множеството от стойности на четири равни части. При графиките от тип кутия понякога се получава нещо наречено опашка и се изобразява с точки от единични

измервания. Тези стойности се смятат твърде необичайни и е прието да се изпускат в определянето на параметрите за кутията. В най-долния и най-горния край са отчетени най-малкият карат и най-големият карат в примерното множество от данни.

Графиките от тип кутия често се използват при изследвания с Монте Карло симулации, където трябва да се демонстрира сходимостта на стохастичния процес. От практиката е наложено да се правят 30 измервания по оста на времето и обобщената статистика за всяко измерване да се визуализира под формата на графика тип кутия.

Заклучение

Коректното въвеждане на данните в системата е от изключителна важност за осъществяването на коректен анализ и постигането на приемливи статистически резултати. В другия край на процеса е самото визуализиране на получените резултати и максималната експресивност, която може да се постигне за представянето пред широка аудитория. Тези две стъпки от процеса по статистически анализ са свързани с въвеждането на информацията и графичното визуализиране на получените резултати.

Глава 5

Оператори за контрол на изпълнението и потребителски функции

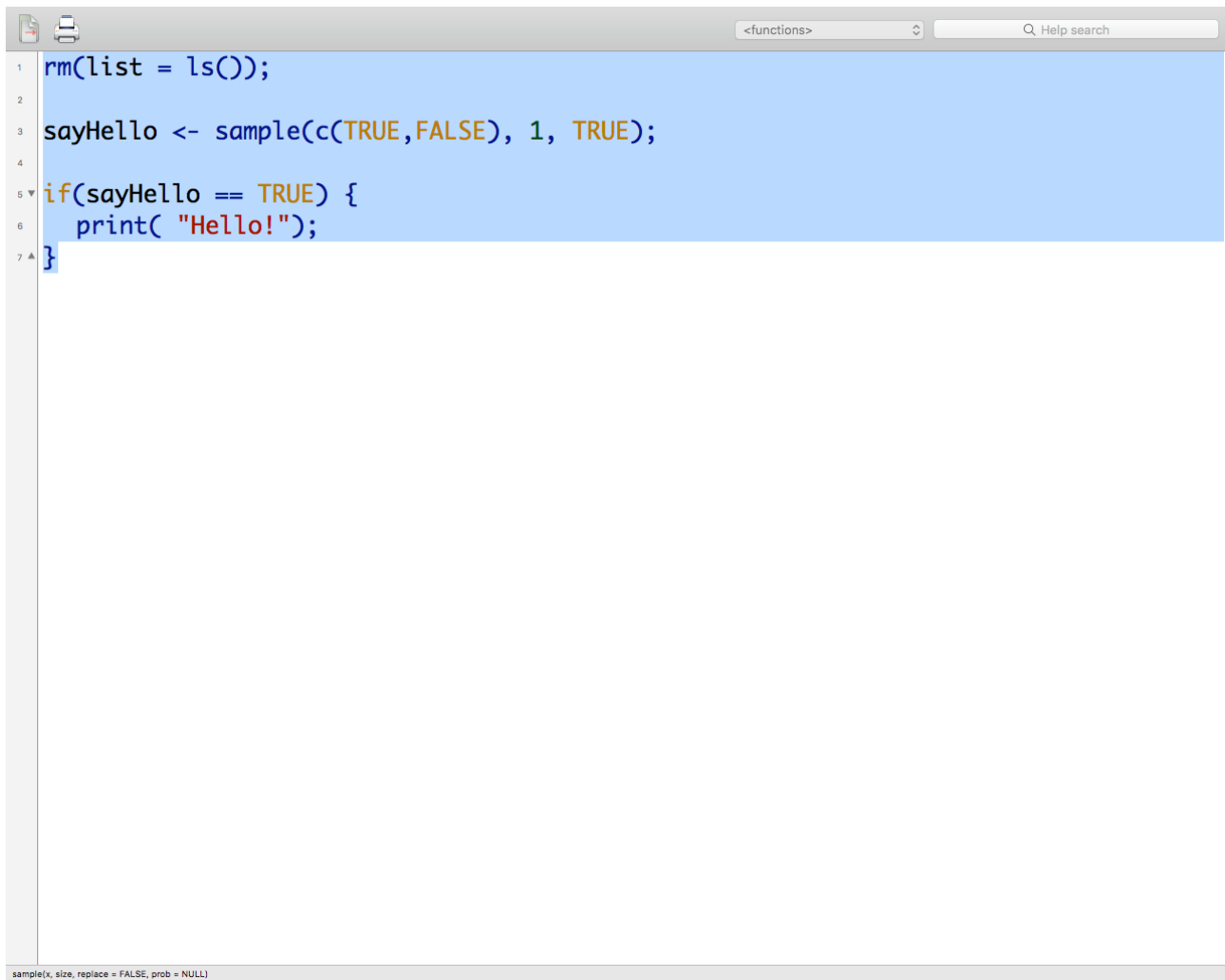
С процеса на усвояване на софтуерния продукт R всеки потребител установява, че някои команди биват повтаряни многократно. Това навежда на мисълта, че тези команди може да бъдат групирани и съхранени като програмен скрипт (модел на експеримента). Основното предимство на такава организация е възможността един и същи модел да бъде използван за множество експерименти. Второто предимство е, че моделът може да бъде разменян между различни потребители, които работят над същата задача или сходни задачи. Езикът R е в групата на интерпретативните програмни езици, където се намира и програмният език JavaScript. За потребителите познаващи JavaScript, някои конструкции в R биха били изключително познати.

Скриптовете на R се записват в обикновени текстови файлове с разширение *.r*, както примерният файл на адрес:

Листинг 5.1: Адрес на примерен R скрипт

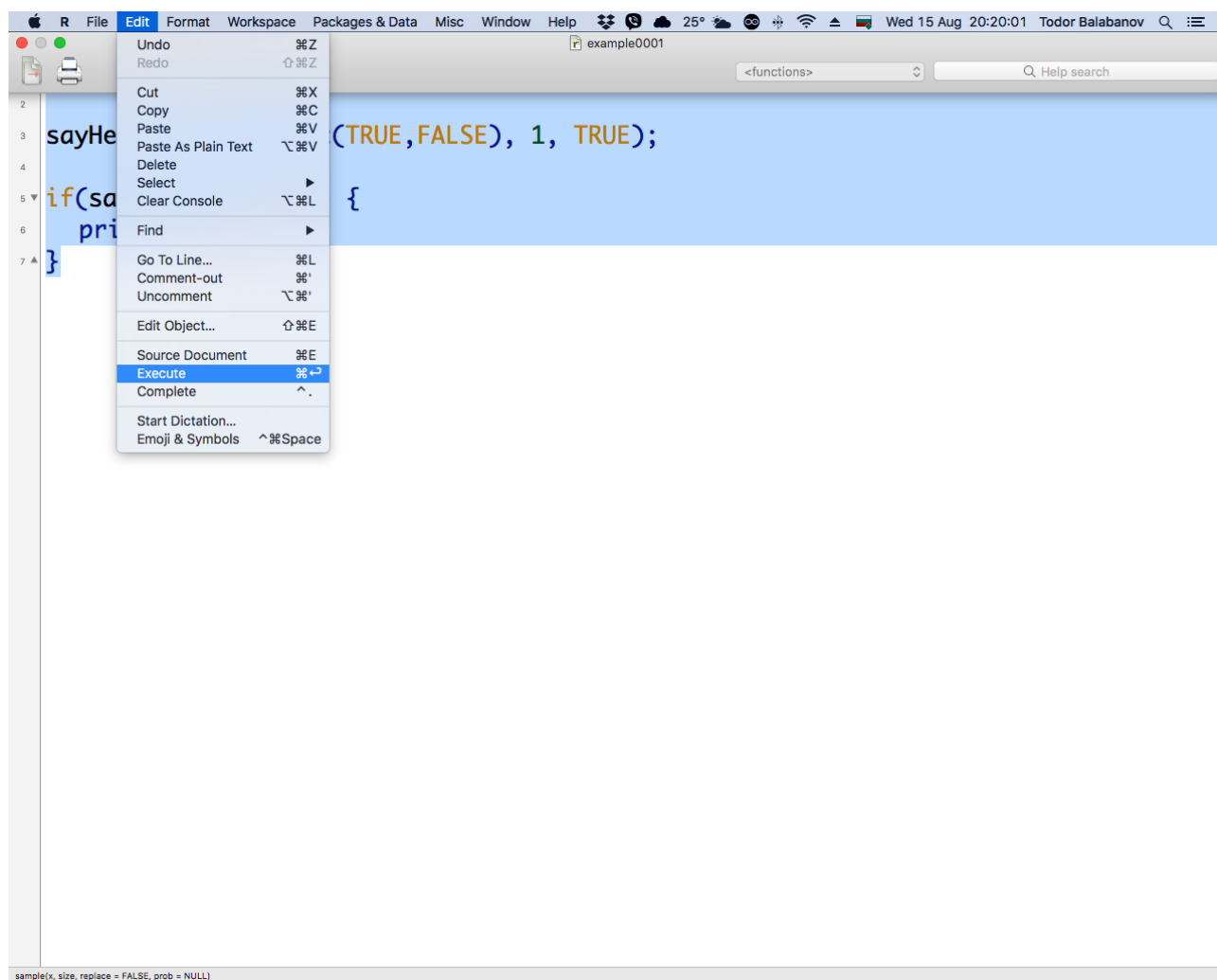
```
https://raw.githubusercontent.com/TodorBalabanov/Statistical-Data-Processing-  
with-R/master/code/example0001.r
```

Може да се пише с текстов редактор по избор на потребителя, но може да се използва и текстовият редактор към продукта (Фиг. 5.1).



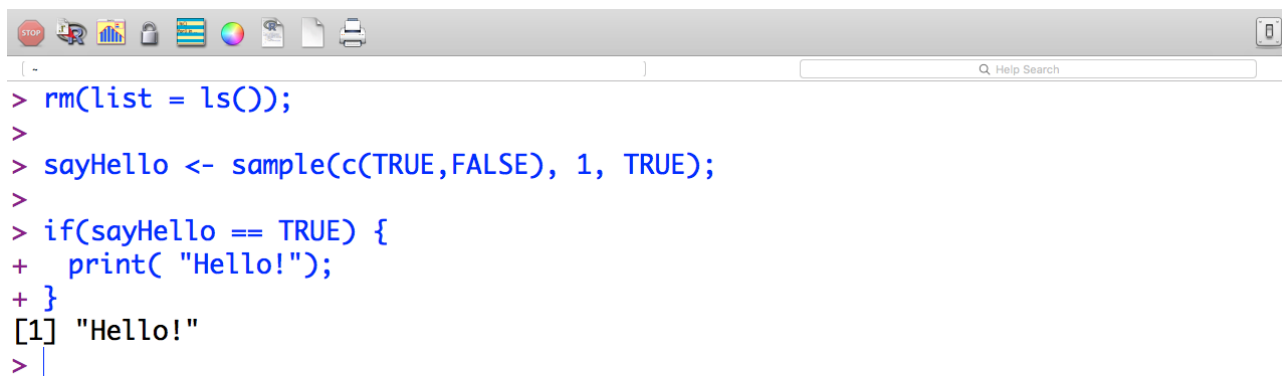
Фигура 5.1: Текстов редактор към продукта R

Най-бързият начин за стартиране на скрипта е чрез менюто на вградения текстов редактор Edit->Execute (Фиг. 5.2).



Фигура 5.2: Стартиране на R скрипт

Съществено е целият текст на скрипта да бъде маркиран, тъй като командният интерпретатор е оптимизиран в режим за изпълнение на команда по команда. Когато целият скрипт е маркиран се изпълняват, една след друга, всички команди.



```
> rm(list = ls());
>
> sayHello <- sample(c(TRUE,FALSE), 1, TRUE);
>
> if(sayHello == TRUE) {
+   print( "Hello!");
+ }
[1] "Hello!"
> |
```

Фигура 5.3: Резултат от изпълнението на R скрипт

Резултатът от изпълнението на R скрипта се наблюдава в командния интерпретатор на продукта (Фиг. 5.3) и изглежда точно както би трябвало командите да се въведат, на ръка, ако не бяха заредени от скриптов файл.

Алтернативна възможност за стартиране на R скриптове е конзолата на операционната система. При този вариант в конзолата на операционната система се извиква приложението R, а като параметри на приложението се подава файлът, съдържащ скрипта и параметър дали сесията от изпълнението на скрипта да бъде съхранена (Листинг 5.2).

Листинг 5.2: Изпълнение на R скрипт от конзолата на операционната система

```
r < ./Statistical-Data-Processing-with-R/code/example0001.r --no-save
```

При такова изпълнение се спестява зареждането на целия програмен продукт R за постоянно в оперативната памет (Фиг. 5.4).


```
MACMINI:Desktop todorbalabanov$ r < ./Statistical-Data-Processing-with-R/code/example0001.r --no-save

R version 3.5.1 (2018-07-02) -- "Feather Spray"
Copyright (C) 2018 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin15.6.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> rm(list = ls());
>
> sayHello <- sample(c(TRUE,FALSE), 1, TRUE);
>
> if(sayHello == TRUE) {
+   print( "Hello!" );
+ }
>
> print( "Bye!" );
[1] "Bye!"
>
MACMINI:Desktop todorbalabanov$
```

Фигура 5.4: Резултат от изпълнението на R скрипт в конзолата на операционната система

Програмните скриптове се състоят от последователни инструкции, но с подходящи оператори за преход или повторение изпълнението на командите може да протече в различен от линейния ред. Тази група оператори се нарича оператори за контрол на изпълнението. Общата конструкция на операторите е заглавна част (ключова дума и условие) и тяло.

5.1 Оператори за преход

Операторите за условен преход променят изпълнението на скрипта в зависимост от логическо или числено условие. От там идва и названието им. В тази група оператори попадат `if`, `else` и `switch`.

В заглавните части на операторите за преход може да се проверява само едно условие или да се проверяват цяла група от условия (логически изрази). За тази цел в R има логически операции като „И“ (операции `&` и `&&`) и „ИЛИ“ (операции `|` и `||`). При двойната форма на операциите се сравнява само по една стойност от двете страни (не са векторизирани), докато при единичната форма се сравняват елемент по елемент множествата от елементи от двете страни. Поради тази причина двойната форма е полезна при `if` оператора, а единичната форма се изисква при `ifelse` конструкцията. Друга много важна разлика между единичната и двойната форма е в начинът, по който се изчисляват изразите от двете страни на

операцията. При единичната форма задължително се изчисляват и двата операнда, независимо дали това действително е нужно. При двойната форма се изчисляват само тези операнди, които са достатъчни за да определят финалния резултат от пресмятането на логическия израз. Тази разлика в използването на логическите операции може да се окаже изключително съществена, когато операндите са функции, връщащи логическа стойност. В случаите когато е ненужно някой от операндите да се изчисли, тъй като другите вече са определили резултата, то част от функциите няма да бъдат извикани. С помощта на логическите операции могат да се построят много сложни логически изрази, при които важат общите правила за приоритет на операциите, както и възможността приоритетът да се променя, чрез подходящо поставяне на скоби.

5.1.1 Оператор за условен преход

Дори чисто исторически в програмните езици един от първите оператори за преход е операторът за условен преход (if оператор).

Листинг 5.3: Оператор за условен преход if

```
sayHello <- sample(c(TRUE,FALSE), 1, TRUE);

if(sayHello == TRUE) {
    print( "Hello!" );
}

print( "Bye!" );
```

Операторът за условен преход използва ключовата дума if (Листинг 5.3), а в заглавната му част се записва израз, пресмятан до логическа стойност TRUE или FALSE. Смисълът на оператора if е, че тялото му бива изпълнено единствено, ако изчислението на израза в заглавната част доведе до стойност TRUE. Ако изразът в заглавната част бъде изчислено до стойност FALSE, тялото на оператора се пропуска и изпълнението на програмата продължава след него.

В примера от листинг 5.3 променливата sayHello получава една случайна логическа стойност (TRUE или FALSE), като двете възможности са равно вероятни. На следващия ред операторът if изписва "Hello!" или го пропуска и изписва "Bye!". Скриптът трябва да се стартира няколко пъти, за да се наблюдава ефектът от случайния избор на стойност за променливата sayHello.

5.1.2 Алтернатива при условен преход

В множество ситуации, освен основна алтернатива за оператора if, е необходимо да има и допълнителна алтернатива, която да се изпълни при резултат от логическия израз FALSE. За тази цел конструкцията на оператора if може да се разшири с добавяне на else конструкция (Листинг 5.4).

Листинг 5.4: Оператор за условен преход if-else

```
sayHello <- sample(c(TRUE,FALSE), 1, TRUE);

if(sayHello == TRUE) {
    print( "Hello!" );
} else {
    print( "Hi!" );
}

print( "Bye!" );
```

Конструкцията `else` е контекстно зависима и поради тази причина може да се използва единствено в комбинация с конструкцията на оператора `if`. В примерния код от листинг 5.4 в половината от случаите на конзолата ще се изпише "Hello!" а в другата половина "Hi!".

5.1.3 Каскада от условни преходи

Условният преход ограничава до две възможности, но практиката понякога налага да се избират повече алтернативи. В такава ситуация може да се използва каскада от `if-else` конструкции (Листинг 5.5).

Листинг 5.5: Каскада от `if-else`

```
sayHello <- sample(c(0,1,2), 1, TRUE);

if(sayHello == 0) {
  print("Hello!");
} else if(sayHello == 1) {
  print("Hi!");
} else if(sayHello == 2) {
  print("Yoo!");
} else {
  print("Error!");
}

print("Bye!");
```

Недостатък на каскадните проверки е, че всяко условие трябва да бъде проверявано по отделно. Каскадата може да завършва с `else` конструкция, но тя не е задължителна.

R предлага `ifelse` оператор, който много прилича на `if` конструкцията в Microsoft Excel (Листинг 5.6) и сериозно се различава от `if-else` оператора. Една от най-силните страни на `ifelse` конструкцията е, че тя е векторизирана и може да се прилага над група елементи едновременно.

Листинг 5.6: Функцията `ifelse`

```
ifelse(sample(c(FALSE,TRUE), 1, TRUE), "Yes", "No")

ifelse(c(1,1,0,1,0,1)==1, "Yes", "No")
```

5.1.4 Оператор за многовариантен избор

Писането на каскадни конструкции от типа `if-else` може да бъде твърде неудобно и поради тази причина съществува `switch` конструкцията (Листинг 5.7).

Листинг 5.7: Конструкция за многовариантен избор `switch`

```
switch(sample(c("a","b","c","d","e"),1,TRUE), "a"="one", "b"="two", "c"="three",
        "d"="four", "other")
```

Първият аргумент на `switch` е стойността която ще се проверява, а след това са изброени алтернативните възможности. Последната стойност, ако не ѝ е зададена стойност за проверка, служи за отговор, когато нито една от алтернативите не е била определена. В примерния код на случаен принцип се избира една буква от пет възможни (конструкцията `sample`), след което се проверяват четири алтернативи и последна опция за `else` условие.

5.2 Оператори за цикъл

В конвенционалните програмни езици е обичайна практика елементите на масивите и контейнерите за данни (списъци, стекове, опашки и други) да бъдат обхождани един по един с помощта на цикли. В R целта е да бъдат прилагани векторизирани операции и максимално да се избягва използването на цикли за обхождане на елементи в контейнер за данни. Въпреки това, в някои ситуации се налага използването на цикли и поради тази причина R поддържа циклите `for` и `while`.

5.2.1 Цикъл за обхождане

Операторът `for` в R представлява цикъл за обхождане на елементи във вектор, като елементът от текущата итерация е достъпен за използване в тялото на цикъла (Листинг 5.8).

Листинг 5.8: Оператор за цикъл `for`

```
for(number in 1:10) { print(number); }
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

Заглавната част се състои от променлива (в случая `number`), ключовата дума `in` и вектор от възможни стойности (в случая числата от едно до десет). Векторите могат да бъдат с различен тип на елементите, примерно символни низове (Листинг 5.9).

Листинг 5.9: Обхождане на вектор от символни низове

```
for(f in c("orange", "lemon", "kiwi", "cherry")) { print(f); }
[1] "orange"
[1] "lemon"
[1] "kiwi"
[1] "cherry"
```

5.2.2 Цикъл с условие за край

Когато множество от елементи няма да бъде обхождано, е по-удачно да се използва цикълът `while`, който в заглавната си част съдържа логически израз, определящ условието за край на цикъла. Цикълът се върти, докато логическият израз в заглавната му част се пресмята до стойност `TRUE` (Листинг 5.10).

Листинг 5.10: Цикъл с условие за край

```
counter <- 1;
while(counter <= 5) {
  print(counter);
  counter <- counter + 1;
}
[1] 1
[1] 2
```

```
[1] 3
[1] 4
[1] 5
```

При цикъла с условие за край променливата, която определя условията за приключване на итерациите трябва да се определи преди началото на цикъла. За да не бъде цикълът безкраен, е нужно тази променлива да бъде променена в тялото на цикъла, по такъв начин, че той да приключи изпълнението си.

5.2.3 Прекъсване на циклите

Понякога се налага определена итерация на цикъла да бъде прекъсната. За тази цел R предлага ключовата дума `next`, която прекъсва текущата итерация и преминава към следващата (Листинг 5.11).

Листинг 5.11: Прекъсване на итерация

```
for (number in 1:10) {
  if (number == 7) {
    next;
  }

  print (number);
}
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 8
[1] 9
[1] 10
```

В този случай пропуснатото число е седем, тъй като при седмата итерация е било изпълнено условието на оператора `if` и неговото тяло съдържа ключовата дума `next`.

В други ситуации се налага цикълът да бъде спрян изцяло и тогава се използва ключовата дума `break` (Листинг 5.12).

Листинг 5.12: Прекъсване на цикъла

```
for (number in 1:10) {
  if (number == 3) {
    break;
  }

  print (number);
}
[1] 1
[1] 2
```

Двете ключови думи (`next` и `break`) са контекстно зависими конструкции и могат да се използват единствено в телата на циклите `for` и `while`.

5.3 Потребителски функции

При писането на програмен код, в съвременните програмни езици, е изключително добра практика, кодът да се групира в серия инструкции и те да се оформят като самостоятелна единица. За тази цел, R предлага възможността кодът да се оформя в потребителски написани функции.

Оформянето на скриптовете в добре организирани и малки по размер функции е основен похват в съвременното програмиране. Такъв стил на работа позволява по-лесна поддръжка, по-лесна проверка на резултатите, създавани от функцията и по-лесна преизползваемост на кода.

Писането на функции в R малко се различава от повечето конвенционални програмни езици, но много прилича на функциите, писани в JavaScript. Самата функция представлява обект, който бива присвоен на идентификатор (Листинг 5.13).

Листинг 5.13: Примерна потребителска функция

```
say.hello <- function() {
  print("Hello ,_World!");
}

say.hello();
[1] "Hello ,_World!"
```

Важно е да се отбележи, че символът точка (.) е валиден символ за съставяне на име на функцията. Това е фундаментална разлика спрямо повечето конвенционални програмни езици. Въпреки това, имената не бива да започват само с точка, тъй като обекти именувани по този начин имат по-специфична употреба. Функцията се създава с ключовата дума function, а тялото на функцията се огражда в къдрави скоби. Много добра практика е да се спазват стриктни правила за подравняване на различните конструкции, използвани като команди в тялото на функцията. Такъв стил на писане подобрява възможностите за четене на програмния текст и възможностите за откриване на евентуални грешки в него.

5.3.1 Аргументи на функция

Тъй като функциите са самостоятелно обособени групи от инструкции, то понякога е нужно към групата да бъдат подавани параметри, под формата на аргументи на функцията (Листинг 5.14).

Листинг 5.14: Извикване на функция с аргумент

```
hello.person <- function( name ) {
  print( sprintf("Hello ,_%s!",name) );
}

hello.person("Dessislava");
[1] "Hello ,_Dessislava!"
```

Променливата name е достъпна само в тялото на функцията и не може да бъде използвана извън него.

Листинг 5.15: Извикване на функция с повече аргументи

```
hello.person <- function(first , last) {
  print( sprintf("Hello ,_%s_%s!",first , last) );
}

hello.person("Dessislava" , "Gruncharova");
[1] "Hello ,_Dessislava_Gruncharova!"
```

```
hello.person(last="Mladevnova", first="Vyara");  
[1] "Hello, _Vyara_Mladevnova!"
```

Функциите в R позволяват извикване с изброяване на параметрите по позиции или чрез явно указване кой аргумент каква стойност да получи (Листинг 5.15).

5.3.2 Аргументи с подразбираща се стойност

В някои ситуации е удачно някои от аргументите да имат зададена стойност по подразбиране. В много от съвременните езици това се постига с едноименни функции (overloading), но в R този ефект е възможен, чрез задаване на подразбираща се стойност (Листинг 5.16).

Листинг 5.16: Извикване на функция с подразбиращи се аргументи

```
hello.person <- function(first, last, title="") {  
  print( sprintf("Hello, %s_%s_%s!", title, first, last) );  
}  
  
hello.person("Zornitsa", "Radeva", "Miss");  
[1] "Hello, _Miss_Zornitsa_Radeva!"  
  
hello.person("Todor", "Balabanov");  
[1] "Hello, _Todor_Balabanov!"
```

5.3.3 Променлив брой аргументи

По аналогия с програмния език C, в R са възможни функции с променлив брой аргументи, което се постига с операцията триеточие (...) в заглавната част на функцията (Листинг 5.17). Механизмът с променлив брой аргументи е особено полезен, но той също така води и до много сериозни програмни грешки.

Листинг 5.17: Функция с променлив брой аргументи

```
sum.up <- function(a, b, ...) {  
  print( a+b );  
}  
  
sum.up(1, 2, 3, 4);
```

5.3.4 Върната стойност

Потребителските функции трябва да се пишат по такъв начин, че да функционират на принципа на черната кутия – аргументи на входа с резултат на изхода и капсулирано изчисление в тялото на функцията. За да бъде постигната тази цел, функциите в R могат да връщат стойност. По аналогия с JavaScript, в R може да се връща стойност от различен тип с ключовата дума return (Листинг 5.18).

Листинг 5.18: Връщане на стойност от функция

```
sum.up <- function(a, b, ...) {  
  return(a + b);  
}  
  
print( sum.up(1,2,3,4) );
```

5.3.5 Предаване на функция като аргумент

В относително редки случаи се налага върху входните данни да бъдат извикани различни функции. В такава ситуация е важно към самата функция да бъде подаден обект от тип функция (Листинг 5.19).

Листинг 5.19: Избор на функция за извикване по време на изпълнение

```
do.stat <- function(values, calculation) {  
  do.call(calculation, args=list(values));  
}  
  
print( do.stat(1:10,mean) );  
[1] 5.5  
  
print( do.stat(1:10,median) );  
[1] 5.5  
  
print( do.stat(1:10,sd) );  
[1] 3.02765
```

В други програмни езици този ефект се постига с функционални указатели (езика C) или функционални обекти (езика Java).

Заклучение

Операторите за контрол на изпълнението позволяват създаването на по-сложни експериментални модели и извършването на по-задълбочени статистически изследвания. За ефективна и надеждна работа, езикът R позволява командите и операторите за контрол на изпълнението да бъдат организирани в отделни потребителски функции и файлове, съдържащи програмните скриптове.

Глава 6

Групиране и обхождане на данни

Практиката показва, че около 80% от статистическия анализ е обработка на данните. Това често налага повтарящи се операции върху различни участъци от данните. Данните се разделят на отделни фрагменти, след това върху определени фрагменти се прилагат определени операции и накрая фрагментите се обединяват в едно цяло.

6.1 Фамилията функции `apply`

Фамилията функции `apply` служи за групово манипулиране на данни. Тъй като има различни входно-изходни комбинации на данните, то в R е представена цяла фамилия функции, а не една единствена.

6.1.1 `apply`

Функцията `apply` е първата, която потребителите научават и тя идва с най-много ограничения. Функцията се прилага върху матрици, което означава, че всичките елементи в матрицата трябва да са от един и същи тип. Ако функцията бъде приложена върху друг тип обект, то първо данните ще бъдат преобразувани до матрица. Първият аргумент при извикването е обектът, който ще бъде обработван. Вторият аргумент определя дали да се работи по редове (стойност 1) или по колони (стойност 2). Третият аргумент е функцията, която трябва да се приложи. Аргументите след третия могат да са променлив брой и се предават на функцията, която е посочена в третия аргумент.

Листинг 6.1: Сума по редове и колони

```
m1 <- matrix(11:19, nrow=3)
m1
  [,1] [,2] [,3]
[1,]  11  14  17
[2,]  12  15  18
[3,]  13  16  19

apply(m1, 1, sum)
[1] 42 45 48

apply(m1, 2, sum)
[1] 36 45 54

m1[2,2] <- NA
```

```

m1
      [,1] [,2] [,3]
[1,]   11  14  17
[2,]   12  NA  18
[3,]   13  16  19

apply(m1, 1, sum)
[1] 42 NA 48

apply(m1, 2, sum)
[1] 36 NA 54

apply(m1, 1, sum, na.rm=TRUE)
[1] 42 30 48

apply(m1, 2, sum, na.rm=TRUE)
[1] 36 30 54

```

Едно от най-лесните пресмятания, за илюстрация на функцията `apply`, е сумата на елементите по редове и по колони в една матрица (Листинг 6.1). Когато в матрицата има неопределени стойности (NA), подадената функция изчислява стойността до NA. Това поведение може да бъде променено, чрез игнориране на липсващите стойности, с подаване на параметъра `na.rm=TRUE`.

6.1.2 lapply и sapply

Функцията `lapply` получава като аргумент списък и връща като резултат списък (Листинг 6.2).

Листинг 6.2: Сума на обекти в списък

```

l1 <- list(m2=matrix(1:9,3), l2=1:5, m3=matrix(1:4,2), n1=2)
l1
$m2
      [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

$l2
[1] 1 2 3 4 5

$m3
      [,1] [,2]
[1,]    1    3
[2,]    2    4

$n1
[1] 2

lapply(l1, sum)
$m2
[1] 45

$l2
[1] 15

```

```

$m3
[1] 10

$n1
[1] 2

sapply(l1, sum)
m2 l2 m3 n1
45 15 10 2

```

Ако върнатата стойност трябва да бъде вектор, то вместо `lapply` се използва `sapply`, която във всяко друго отношение работи като `lapply`.

6.1.3 mapply

Функцията `mapply` се ползва за прилагане на функция върху всеки елемент от множество списъци (Листинг 6.3).

Листинг 6.3: Проверка за идентичност на елементите

```

l3 <- list(m4=matrix(1:25,5), m5=matrix(1:16,2), l4=1:5)
l5 <- list(m6=matrix(1:25,5), m7=matrix(1:16,8), l6=15:1)

mapply(identical, l3, l5)
      m4      m5      l4
TRUE FALSE FALSE

mapply(f1<-function(x,y){NCOLNCOL(x)-NCOL(y)}, l3, l5)
m4 m5 l4
10 10 20

```

Също така, `mapply` позволява и потребителски дефинирани функции.

6.1.4 Агрегация

При употребата на SQL е много популярно данните да се групират по признак/признаци и върху тях да бъдат изпълнявани агрегатни функции. Един от начините да се постигне агрегация в R е, чрез функцията `aggregate` и синтаксиса за запис на формула. Формулите се записват с лява част и дясна част, отделени със символа тилда (`~`). От лявата страна стои променливата, по която ще се извършва пресмятането, а от дясната страна стои променлива (или група променливи), по която ще се извършва групирането.

Листинг 6.4: Групиране на данни

```

data(diamonds, package="ggplot2")
head(diamonds, n=3)
  carat    cut color clarity depth table price     x     y     z
1  0.23  Ideal     E    SI2   61.5    55   326  3.95  3.98  2.43
2  0.21 Premium     E    SI1   59.8    61   326  3.89  3.84  2.31
3  0.23   Good     E    VS1   56.9    65   327  4.05  4.07  2.31

aggregate(price~cut, diamonds, mean)
      cut      price
1   Fair 4358.758

```

2	Good	3928.864
3	Very Good	3981.760
4	Premium	4584.258
5	Ideal	3457.542

Като пример за групиране и използване на агрегатна функция може да се изпълни пресмятането на средна цена за диамант, организирано в групи според вида на среза (Листинг 6.4). Първият аргумент на функцията е формулата за групиране, вторият аргумент е обектът с данните, а третият аргумент е функцията, която се прилага върху отделните групи. Тъй като вторият аргумент показва кое множество данни се използва, то във формулата не е нужно да се указва множеството, а може да се използват директно имената на колоните. След третия аргумент могат да се добавят и други аргументи, като например na.rm=TRUE.

Листинг 6.5: Групиране по повече от една колона

```
aggregate(price~cut+color, diamonds, mean)
      cut color      price
1    Fair    D 4291.061
2    Good    D 3405.382
3 Very Good    D 3470.467
4 Premium    D 3631.293
5   Ideal    D 2629.095
6    Fair    E 3682.312
7    Good    E 3423.644
8 Very Good    E 3214.652
9 Premium    E 3538.914
10   Ideal    E 2597.550
...
```

Както в SQL, така и при групирането в R е възможно групирането да се извърши по повече от една колона (Листинг 6.5).

Листинг 6.6: Прилагане на агрегатна функция върху повече колони в едни и същи групи

```
aggregate(cbind(price,carat)~cut, diamonds, mean)
      cut price      carat
1    Fair 4358.758 1.0461366
2    Good 3928.864 0.8491847
3 Very Good 3981.760 0.8063814
4 Premium 4584.258 0.8919549
5   Ideal 3457.542 0.7028370
```

Възможно е агрегатната функция да бъде приложена върху повече от една колона, като това се постига с функцията cbind (Листинг 6.6). Особеността при това извикване е, че само една агрегатна функция може да се приложи и всички избрани колони ще бъдат пресметнати спрямо нея (в случая се изчислява средна стойност).

Листинг 6.7: Използване на повече колони от двете страни на формулата

```
aggregate(cbind(price,carat)~cut+color, diamonds, mean)
      cut color      price      carat
1    Fair    D 4291.061 0.9201227
2    Good    D 3405.382 0.7445166
3 Very Good    D 3470.467 0.6964243
4 Premium    D 3631.293 0.7215471
5   Ideal    D 2629.095 0.5657657
```

6	Fair	E	3682.312	0.8566071
7	Good	E	3423.644	0.7451340
8	Very Good	E	3214.652	0.6763167
9	Premium	E	3538.914	0.7177450
10	Ideal	E	2597.550	0.5784012
...				

Изпълнението на агрегатни функции позволява да има повече от една колона и от двете страни на формулата (Листинг 6.7). При използването на функцията за агрегация трябва да се има предвид, че тя може да бъде много бавна по отношение на изпълнението, особено при данни с голям обем.

6.2 Пакетът plyr

Пакетът plyr дава допълнителни възможности по схемата за обработка на данни разделяне-манипулиране-обединение. Ядрото на пакета се състои от функциите dply, lply и ldply. При тези функции е използвана конвенция, която подсказва какъв е типът на входните данни и типът на изходните данни (в по-широката ѝ употреба това е унгарската нотация [34]). Първата буква определя входния тип данни, а втората буква определя изходния тип данни. Буквата d се използва за data.frame, буквата l за списък, буквата a за масив, а символът за подчертаване () се използва при липса на върната стойност.

6.2.1 ddply

За илюстрация на функцията ddply е удачно да се използват данните за бейзболни резултати от пакета plyr (Листинг 6.8).

Листинг 6.8: Бейзболна статистика

```
library( plyr )

head( baseball ,n=3)
      id year stint team lg   g   ab   r   h X2b X3b hr  rbi sb  cs  bb  so  ibb
4  ansonca01 1871     1  RC1   25 120 29 39   11   3  0  16  6  2  2  1  NA
44 forceda01 1871     1  WS3   32 162 45 45    9   4  0  29  8  0  4  0  NA
68 mathebo01 1871     1  FW1   19  89 15 24    3   1  0  10  2  1  2  0  NA
      hbp sh  sf  gidp
4    NA NA NA    NA
44   NA NA NA    NA
68   NA NA NA    NA
```

В това множество данни са отчетени 21699 записа в бейзболната статистика за 1228 играча, в диапазона от годините 1871 до 2007. Включени са само играчи с 15 или повече сезона игра. В множеството се отчитат 22 признака (колони), със следното значение:

Основна статистика в бейзбола е OBP (On Base Percentage), която се изчислява по формула 6.1.

$$OBP = \frac{H + BB + HBP}{AB + BB + HBP + SF} \quad (6.1)$$

Където: H – брой удари, BB – пробягвания, HBP – брой попадения от питчъра, AB – удари на батата, SF – пропуснати изстрели.

Характеристика	Значение
id	Идентификатор на играча (символен низ)
year	Година на събраните данни (цяло число)
stint	Ограничения (цяло число)
team	Отбор, в който играе (символен низ)
lg	Лига, в която играе (символен низ)
g	Брой игри (цяло число)
ab	Брой батирувания (цяло число)
r	Брой пробиявания (цяло число)
h	Брой реализирани удари (цяло число)
X2b	Брой успешни достигания до втора база (цяло число)
X3b	Брой успешни достигания до трета база (цяло число)
hr	Брой успешни хоум ръна (цяло число)
rbi	Брой тичания, в които е удрял (цяло число)
sb	Брой откраднати бази (цяло число)
cs	Брой хванати отнемания (цяло число)
bb	Брой пробиявания (цяло число)
so	Брой изхвърлени изстрели (цяло число)
ibb	Брой международни пробиявания (цяло число)
hbp	Брой попадения от питчъра (цяло число)
sh	Брой пожертвани удари (цяло число)
sf	Брой пожертвани изстрела (цяло число)
gidp	Брой приземявания при двойна игра (цяло число)

Таблица 6.1: Характеристики на бейзболните играчи

Преди 1954 година, за SF стойностите са 0, поради различния начин за отчитане на статистиката. В данните има множество NA стойности за HBP. Липсващите стойности трябва да се заменят с 0. От множеството данни се изключват и всички редове, за които AB е по-малко от 50 (Листинг 6.9).

Листинг 6.9: Корекция на данните

```
any( is.na(baseball$sf) )
[1] TRUE

baseball$sf[baseball$year < 1954] <- 0

any( is.na(baseball$sf) )
[1] FALSE

any( is.na(baseball$hbp) )
[1] TRUE

baseball$hbp[ is.na(baseball$hbp) ] <- 0

any( is.na(baseball$hbp) )
[1] FALSE

baseball <- baseball[ baseball$ab >= 50, ]
```

Пресмятането на коефициента ОВР за всеки играч, за конкретна година е изключително лесно, поради възможността да се изпълни векторно пресмятане (Листинг 6.10).

Листинг 6.10: Пресмятане на OBP

```
baseball$OBP <- with(baseball, (h+bb+hbp)/(ab+bb+hbp+sf))

head(cbind(baseball$id, baseball$OBP), n=3)
  [,1]      [,2]
[1,] "ansonca01" "0.336065573770492"
[2,] "forceda01" "0.295180722891566"
[3,] "mathebo01" "0.285714285714286"
```

Функцията `with` позволява да се упомене обектът с данните само един път, а в следващите аргументи да се използват само имената на колоните, без да се уточнява на кое множество данни принадлежат.

OBP коефициентът за цялата кариера на играча не може да се пресметне чрез просто усредняване на сезонния OBP коефициент. Трябва да се пресметне и сумира числителят и след това да се раздели на сумата в делителя. Това може да се постигне с функцията `ddply` (Листинг 6.11).

Листинг 6.11: Пресмятане на OBP за цялата кариера на играча

```
career <- ddply(baseball, .variables="id", .fun=function(data){c(OBP=with(data,
  sum(h+bb+hbp)/sum(ab+bb+hbp+sf))})})

career <- career[order(career$OBP, decreasing=TRUE), ]

head(career, n=3)
      id      OBP
1089 willite01 0.4816861
875  ruthba01 0.4742209
658  mcgrajo01 0.4657478
```

Вградената потребителска функция извършва пресмятането, а след това `ddply` изпълнява пресмятането върху всеки играч. Полученият резултат е сортиран в низходящ ред, според кариерния OBP коефициент.

6.2.2 lply

Функцията `lply` може да бъде използвана по аналогичен начин, както се използва функцията `lapply` (Листинг 6.12).

Листинг 6.12: Сума на всеки елемент в списък

```
l1 <- list(m2=matrix(1:9,3), l2=1:5, m3=matrix(1:4,2), n1=2)

lply(l1, sum)
$m2
[1] 45

$m3
[1] 15

$m3
[1] 10

$n1
[1] 2
```

```
identical(lapply(l1,sum), llply(l1,sum))
[1] TRUE

lapply(l1, sum)
[1] 45 15 10 2
```

Функцията `lapply` извършва същото пресмятане, но връща резултата под формата на вектор.

6.2.3 Помощни функции и бързодействие

В пакета `plyr` са добавени група помощни функции като функцията `each`, която позволява да се изпращат повече от една агрегатна функция на функцията `aggregate` (Листинг 6.13). Недостатък на `each` е, че тя отнема възможността за изпращане на допълнителни параметри.

Листинг 6.13: Повече от една агрегатна функция

```
library(ggplot2)

aggregate(price~cut, diamonds, each(mean, median))
      cut price.mean price.median
1   Fair   4358.758    3282.000
2   Good   3928.864    3050.500
3 Very Good  3981.760    2648.000
4 Premium  4584.258    3185.000
5   Ideal   3457.542    1810.000
```

Друга полезна функция е `idata.frame`, която позволява създаването на референция към `data.frame`, така че формирането на подмножества да става по-бързо и при значително по-ниска консумация на оперативна памет (Листинг 6.14).

Листинг 6.14: Бързодействие при използване на референции

```
system.time(dplyr(baseball, "id", nrow))
  user  system elapsed 
0.159   0.009   0.169 

reference <- idata.frame( baseball )

system.time(dplyr(reference, "id", nrow))
  user  system elapsed 
0.178   0.005   0.183
```

Ускорението, което ще се постигне, много зависи от размера на данните и от вида на пресмятането, което се извършва над тях. Пакетът `plyr` често води до компромис с бързодействието в замяна на по-голямо удобство при пресмятането. Повечето функции в пакета може да се изпълнят и с функции от базовата инсталация, `plyr` просто дава по-удобен начин за пресмятане.

6.3 Пакетът `data.table`

Този пакет е създаден с идеята да увеличи възможностите за използване на `data.frame` структурата от данни. Пакетът изисква малко по-различен синтаксис за работа спрямо този, който вече е наложен при използването на `data.frame`. Бързодействието при `data.table` основно се дължи на това, че вътрешната реализация е аналогична на индексите при базите данни. Това позволява по-бърз достъп до данните, по-бързи операции за групиране и по-бързи операции за сливане (`join`).

Листинг 6.15: Създаване на data.table

```
library(data.table)

df <- data.frame(x1=10:1, x2=letters[11:20], x3=LETTERS[1:10], x4=rep(c("One", "Two", "Three"), length.out=10))

df
  x1 x2 x3  x4
1 10 k  A  One
2  9 l  B  Two
3  8 m  C Three
4  7 n  D  One
5  6 o  E  Two
6  5 p  F Three
7  4 q  G  One
8  3 r  H  Two
9  2 s  I Three
10 1 t  J  One

class(df$x2)
[1] "factor"

dt <- data.table(x1=10:1, x2=letters[11:20], x3=LETTERS[1:10], x4=rep(c("One", "Two", "Three"), length.out=10))

dt
  x1 x2 x3  x4
1: 10 k  A  One
2:  9 l  B  Two
3:  8 m  C Three
4:  7 n  D  One
5:  6 o  E  Two
6:  5 p  F Three
7:  4 q  G  One
8:  3 r  H  Two
9:  2 s  I Three
10: 1 t  J  One

class(dt$x2)
[1] "character"
```

Създаването на data.table не се различава особено от създаването на data.frame (Листинг 6.15). Зареждането на данни е идентично с тази разлика, че data.frame преобразува символните низове до фактори, докато data.table ги запазва като символни низове.

Листинг 6.16: Зареждане на data.table от data.frame

```
diamonds <- data.table(diamonds)

diamonds
  carat      cut color clarity depth table price     x     y     z
1:  0.23    Ideal     E    SI2   61.5    55   326  3.95  3.98  2.43
2:  0.21 Premium     E    SI1   59.8    61   326  3.89  3.84  2.31
3:  0.23     Good     E    VS1   56.9    65   327  4.05  4.07  2.31
4:  0.29 Premium     I    VS2   62.4    58   334  4.20  4.23  2.63
```

	5:	0.31	Good	J	SI2	63.3	58	335	4.34	4.35	2.75
53936:	0.72		Ideal	D	SI1	60.8	57	2757	5.75	5.76	3.50
53937:	0.72		Good	D	SI1	63.1	55	2757	5.69	5.75	3.61
53938:	0.70	Very	Good	D	SI1	62.8	60	2757	5.66	5.68	3.56
53939:	0.86	Premium		H	SI2	61.0	58	2757	6.15	6.12	3.74
53940:	0.75		Ideal	D	SI2	62.2	55	2757	5.83	5.87	3.64

Създаването на `data.table` от `data.frame` става, чрез просто извикване на функцията (Листинг 6.16). При разпечатването на резултата има разлика в това, че се показват първите пет и последните пет реда от данните.

Листинг 6.17: Достъп до редовете

```
dt[1:2, ]
  x1 x2 x3 x4
1: 10 k  A One
2:  9 l  B Two

dt[dt$x1>=7, ]
  x1 x2 x3 x4
1: 10 k  A One
2:  9 l  B Two
3:  8 m  C Three
4:  7 n  D One

dt[x1>=7, ]
  x1 x2 x3 x4
1: 10 k  A One
2:  9 l  B Two
3:  8 m  C Three
4:  7 n  D One
```

Достъпът до редовете се осъществява по аналогичен начин, както е при `data.frame` (Листинг 6.17). Третото позоваване е възможно, тъй като `data.table` знае как да открие нужната колона и без да е указана точната `data.table` променлива.

Листинг 6.18: Достъп до колоните

```
dt[, list(x3, x4)]
  x3 x4
1:  A One
2:  B Two
3:  C Three
4:  D One
5:  E Two
6:  F Three
7:  G One
8:  H Two
9:  I Three
10: J One

dt[, x1]
[1] 10 9 8 7 6 5 4 3 2 1

dt[, list(x2)]
```

```

      x2
1:  k
2:  l
3:  m
4:  n
5:  o
6:  p
7:  q
8:  r
9:  s
10: t

dt[, "x4", with=FALSE]
      x4
1:  One
2:  Two
3: Three
4:  One
5:  Two
6: Three
7:  One
8:  Two
9: Three
10: One

td[, c("x2", "x3"), with=FALSE]
      x2 x3
1:  k  A
2:  l  B
3:  m  C
4:  n  D
5:  o  E
6:  p  F
7:  q  G
8:  r  H
9:  s  I
10: t  J

```

Достъпът до колоните се осъществява малко по-различно спрямо `data.frame` (Листинг 6.18). За да се позоват колоните като символни низове (примерно получени след някакво пресмятане), трябва да се подаде стойност `FALSE` на параметъра `with`.

6.3.1 Ключове

При наличие на няколко таблици в паметта, с тях може да се изпълнят серия операции. Като начало, списък с наличните таблици може да се получи чрез функцията `tables` (Листинг 6.19).

Листинг 6.19: Операции с таблици

```

tables()
      NAME  NROW NCOL MB          COLS KEY
1: diamonds 53,940   10  3 carat , cut , color , clarity , depth , table , ...
2:      dt      10    4  0              x1 , x2 , x3 , x4
Total: 3MB

```

```
setkey(dt, x4)
```

```
dt
```

	x1	x2	x3	x4
1:	10	k	A	One
2:	7	n	D	One
3:	4	q	G	One
4:	1	t	J	One
5:	8	m	C	Three
6:	5	p	F	Three
7:	2	s	I	Three
8:	9	l	B	Two
9:	6	o	E	Two
10:	3	r	H	Two

```
tables()
```

	NAME	NROW	NCOL	MB	COLS	KEY
1:	diamonds	53,940	10	3	carat, cut, color, clarity, depth, table, ...	
2:	dt	10	4	0	x1, x2, x3, x4	x4
Total:		3MB				

```
key(dt)
```

```
[1] "x4"
```

```
dt["One", ]
```

	x1	x2	x3	x4
1:	10	k	A	One
2:	7	n	D	One
3:	4	q	G	One
4:	1	t	J	One

```
dt[c("One", "Two"), ]
```

	x1	x2	x3	x4
1:	10	k	A	One
2:	7	n	D	One
3:	4	q	G	One
4:	1	t	J	One
5:	9	l	B	Two
6:	6	o	E	Two
7:	3	r	H	Two

```
setkey(diamonds, cut, color)
```

```
diamonds
```

	carat	cut	color	clarity	depth	table	price	x	y	z
1:	0.75	Fair	D	SI2	64.6	57	2848	5.74	5.72	3.70
2:	0.71	Fair	D	VS2	56.9	65	2858	5.89	5.84	3.34
3:	0.90	Fair	D	SI2	66.9	57	2885	6.02	5.90	3.99
4:	1.00	Fair	D	SI2	69.3	58	2974	5.96	5.87	4.10
5:	1.01	Fair	D	SI2	64.6	56	3003	6.31	6.24	4.05
<hr/>										
53936:	0.71	Ideal	J	SI1	60.6	57	2700	5.78	5.83	3.52
53937:	0.81	Ideal	J	VS2	62.1	56	2708	5.92	5.97	3.69

```
53938: 0.84 Ideal J VS2 61.1 57 2709 6.09 6.12 3.73
53939: 0.82 Ideal J VS2 61.6 56 2741 6.00 6.04 3.71
53940: 0.83 Ideal J VS2 62.3 55 2742 6.01 6.03 3.75
```

```
diamonds[J("Fair", "J"), ]
      carat cut color clarity depth table price x y z
1:  1.05 Fair  J   SI2  65.8   59  2789 6.41 6.27 4.18
2:  1.00 Fair  J   VS2  65.7   59  2811 6.14 6.07 4.01
3:  0.99 Fair  J   SI1  55.0   61  2812 6.72 6.67 3.68
4:  0.90 Fair  J   VS2  65.0   56  2815 6.08 6.04 3.94
5:  0.91 Fair  J   VS2  64.4   62  2854 6.06 6.03 3.89

115: 0.90 Fair  J   SI1  64.6   61  2438 5.92 5.87 3.81
116: 0.96 Fair  J   SI1  67.3   56  2517 6.06 6.01 4.06
117: 0.90 Fair  J   SI2  66.6   54  2536 6.05 5.99 4.01
118: 0.97 Fair  J   SI2  60.8   67  2538 6.41 6.32 3.87
119: 1.01 Fair  J   SI2  66.9   58  2683 6.13 6.07 4.08
```

В примера на нито една от таблиците не е присвоен ключ. Ключът е индексирана колона, която дава допълнително бързодействие при извършването на някои операции. При наличие на ключово поле е възможно изборът на редове да става и чрез подаване на стойности от ключовото поле. Ключът може да бъде и съставен, ако съдържа повече от едно поле. Функцията `J` служи за достъп до редовете според стойности на съставния ключ.

6.3.2 Агрегация

Основното предимство на индексираните данни е бързодействието при изпълнението на агрегатни функции. Въпреки че `aggregate` и фамилията функции `d*ply` работят с `data.table` обекти (Листинг 6.20), то използването им се характеризира с бавно изпълнение.

Листинг 6.20: Агрегатни функции

```
aggregate(price~cut, diamonds, mean)
      cut price
1 Fair 4358.758
2 Good 3928.864
3 Very Good 3981.760
4 Premium 4584.258
5 Ideal 3457.542

diamonds[, list(price=mean(price)), by=cut]
      cut price
1: Fair 4358.758
2: Good 3928.864
3: Very Good 3981.760
4: Premium 4584.258
5: Ideal 3457.542

diamonds[, list(price=mean(price)), by=list(cut, color)]
      cut color price
1: Fair D 4291.061
2: Fair E 3682.312
3: Fair F 3827.003
4: Fair G 4239.255
```

```

5:      Fair      H 5135.683
6:      Fair      I 4685.446
7:      Fair      J 4975.655
...
29:     Ideal      D 2629.095
30:     Ideal      E 2597.550
31:     Ideal      F 3374.939
32:     Ideal      G 3720.706
33:     Ideal      H 3889.335
34:     Ideal      I 4451.970
35:     Ideal      J 4918.186
      cut color      price
diamonds[, list(price=mean(price),carat=sum(carat)), by=list(cut,color)]
      cut color      price      carat
1:      Fair      D 4291.061    149.98
2:      Fair      E 3682.312    191.88
3:      Fair      F 3827.003    282.27
4:      Fair      G 4239.255    321.48
5:      Fair      H 5135.683    369.41
6:      Fair      I 4685.446    209.66
7:      Fair      J 4975.655    159.60
...
29:     Ideal      D 2629.095   1603.38
30:     Ideal      E 2597.550   2257.50
31:     Ideal      F 3374.939   2509.20
32:     Ideal      G 3720.706   3422.29
33:     Ideal      H 3889.335   2490.52
34:     Ideal      I 4451.970   1910.97
35:     Ideal      J 4918.186    952.98
      cut color      price      carat

```

По-добрият вариант е да се използват вградените в `data.table` агрегатни функции. За да бъде агрегацията по повече от една колона, нужните колони се подават като списък. За разлика от предишната агрегация, тук могат да се подават различни агрегатни функции (средна и сума, в показания пример), както и да се ползва списък от колони за групиране и за агрегиране.

6.4 Бързи операции с пакета `dplyr`

Пакетът `dplyr` е създаден основно с идеята за ускорение на изпълнението, спрямо бързодействието постигнато в пакета `plyr`. Пакетът `dplyr` основно борави с обекти от тип `data.frame`, а за списъци и вектори е предназначен пакетът `purrr`. Характерно за `dplyr` е, че използва названия на някои от функциите, които са добре познати в езика SQL.

Когато се работи едновременно с пакетите `plyr` и `dplyr` е важен редът на зареждането им, тъй като последно зареденият пакет в R е с приоритет, при наличие на едноименни функции. При колизия на имената конфликтът се разрешава чрез операцията за принадлежност към пакет (`::`), като от лявата страна е името на пакета, а от дясната страна името на функцията. Пример за такава колизия е функцията за обобщение на данните в двата пакета - `plyr::summarize` и `dplyr::summarize`.

6.4.1 Потоци и таблици

Концепцията за потоците (pipes) е развита още с първите масово използвани операционни системи, а в езика R тя е възможна благодарение на пакета `magrittr`. При класическата работа с функции, резултатът от извикването на първата функция се записва в междинна променлива, тази променлива се подава като аргумент на втората функция, която на свой ред записва във втора междинна променлива и този процес може да се повтаря многократно. Потоците позволяват да се избегне използването на междинни променливи, а използването на функциите да се организира като една верига от извиквания (Листинг 6.21). Синтаксисът за поточна операция в R е два символа за процент и знакът за по-голямо между тях (`%>%`).

Листинг 6.21: Поточни операции

```
library( ggplot2 )
library( magrittr )

dim( head(diamonds,n=4) )
[1] 4 10

diamonds %>% head(4) %>% dim
[1] 4 10
```

С поточната операция се подава обектът като първи аргумент на съответната функция. Поточните операции могат да се навързват във верига и обектът в резултат от изпълнението на една функция да се предава към следваща.

По аналогия с обектите от тип `data.table`, пакетът `dplyr` предлага разширение на `data.frame` под формата на `tbl` обекти, които се доразвиват и в пакета `tibble`. Съществено за `tbl` е, че при разпечатване се показват само подмножество на редовете, но също и подмножество на колоните, докато бъде изпълнен екранът. Третата разлика е, че под имената на колоните се визуализира информация за типа данни в колоната. При по-новите реализации на пакета `ggplot2`, данните за диамантите са представени и като `tbl` обект (Листинг 6.22).

Листинг 6.22: Таблични данни в `dplyr`

```
class( diamonds )
[1] "tbl_df"      "tbl"        "data.frame"

head(diamonds, n=3)
# A tibble: 3 x 10
  carat cut      color clarity depth table price      x      y      z
  <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.23 Ideal   E     SI2     61.5   55   326   3.95   3.98   2.43
2  0.21 Premium E     SI1     59.8   61   326   3.89   3.84   2.31
3  0.23 Good    E     VS1     56.9   65   327   4.05   4.07   2.31
```

6.4.2 Извличане по колонии

По аналогия с SQL, функцията `select` извършва подбор на редове от зададено множество данни (Листинг 6.23). Първият аргумент е `data.frame` (или `tbl` обект), а следващите аргументи изброяват желаните колонии. Функцията може да се използва самостоятелно или с потоци.

Листинг 6.23: Избор на редове

```
library( dplyr )
```

```

select(diamonds, carat, price)
diamonds %>% select(carat, price)
diamonds %>% select(c(carat, price))

diamonds %>% select_( 'carat', 'price' )
names <- c('carat', 'price')
diamonds %>% select_(.dots=names)

diamonds %>% select( one_of('carat', 'price') )
names <- c('carat', 'price')
diamonds %>% select( one_of(names) )

select(diamonds, 1, 7)
diamonds %>% select(1, 7)

```

Функцията `select` позволява извикване по множество различни начини, които могат да доведат до един и същи краен резултат от изпълнението. Като аргументи, `select` получава директно имената на колоните, без те да са обградени в кавички (тоест не са символни низове). Когато е нужно имената на колоните да се подават като символни низове се използва функцията `select_`, която се различава в името си с една подчертавка в края. Ако имената на колоните са запазени в променлива, то те се подават на `.dots` параметъра. От версия 0.6.0 на пакета `dplyr` не се препоръчва използването на функцията `select_`, а заместването ѝ с функцията `select` и аргумент върнат като стойност от функцията `one_of`. На функцията `select` освен имена на колони, може да се подават и техните индекси.

Листинг 6.24: Търсене по частично съвпадение

```

diamonds %>% select( starts_with('c') )
diamonds %>% select( ends_with('e') )
diamonds %>% select( contains('l') )

diamonds %>% select( matches('r.+t') )

```

Понякога в практиката се налага подбор на колони в множеството от данни по критерии за частично съвпадение (Листинг 6.24). При такава ситуация за начало се ползва функцията `starts_with`, за край функцията `ends_with`, а за съдържание функцията `contains`. При частичния избор е възможно да се използват и регулярни изрази.

Листинг 6.25: Изключване на колони

```

diamonds %>% select(-carat, -price)
diamonds %>% select( -c(carat, price) )
diamonds %>% select(-1, -7)
diamonds %>% select( -c(1,7) )
diamonds %>% select_(.dots=c('-carat', '-price'))
diamonds %>% select( -one_of('carat', 'price') )

```

В някои ситуации от множеството колони трябва да се изключат една част. Това се постига със записване на знак минус пред името/имената на колоната/колониите (Листинг 6.25).

6.4.3 Филтриране

Филтрирането се състои в избиране на редове от данните по зададен критерии (логически израз).

Листинг 6.26: Филтриране на редове


```
library( ggplot2 )
library( magrittr )
library( dplyr )

diamonds %>% filter( cut == 'Ideal' )
diamonds[ diamonds$cut == 'Ideal', ]

diamonds %>% filter( cut %in% c( 'Ideal', 'Good' ) )

diamonds %>% filter( carat > 2, price < 14000 )
diamonds %>% filter( carat > 2 & price < 14000 )

diamonds %>% filter( carat < 1 | carat > 5 )

diamonds %>% filter_( "cut == 'Ideal'" )
diamonds %>% filter_( ~cut == 'Ideal' )
cut <- 'Ideal'
diamonds %>% filter_( ~cut == cut )

col <- 'cut'
cut <- 'Ideal'
diamonds %>% filter_( sprintf( "%s == %s", col, cut ) )
```

Възможно е да се направи филтриране с поточните операции, но също така и с базовия синтаксис на езика R (Листинг 6.26). За да се извърши филтриране по повече от една стойност се използва операцията `%in%`. При филтрирането може да се използват всички стандартни операции за сравнение. При отделяне на повече от един израз със запетая (,) или амперсанд (&) се изпълнява филтриране, за което и всички условия трябва да се изчислят до стойност TRUE. Филтри с логическо ИЛИ се конструират с операцията вертикална черта (|).

Ако имената на колоните трябва да се зададат със символни низове, то вместо функцията `filter` се използва функцията `filter_`. Пред израза се поставя тилда (~). Едновременното представяне като символен низ на желаната стойност за филтриране и името на колоната, по която ще се филтрира, изисква малко повече усилия и може да се постигне с функцията за обработка на символни низове `sprintf`.

Листинг 6.27: Избор по индекси

```
diamonds %>% slice( 1:5 )
diamonds %>% slice( c( 1:5, 8, 15:20 ) )
diamonds %>% slice( -1 )
```

Освен филтриране по логически израз е възможно избирането на редове по зададени индекси. За тази цел се ползва функцията `slice` (Листинг 6.27). Индексите могат да се подадат под формата на вектор. Когато индексите са отрицателни, това означава, че тези редове няма да бъдат част от върнатия резултат.

6.4.4 Модификация, обобщение, групиране и подреждане

Създаването или модифицирането на колони в множеството от данни се извършва с функцията `mutate` (Листинг 6.28).

Листинг 6.28: Модификация на колони

```
diamonds %>% mutate( price / carat )
diamonds %>% select( carat, price ) %>% mutate( price / carat )
diamonds %>% select( carat, price ) %>% mutate( ratio = price / carat )
```

```
diamonds %>% select(carat, price) %>% mutate(ratio=price/carat, square=ratio*
ratio)
```

Добавянето на съотношение цена/карати се извършва с подаване на израза, като параметър на функцията `mutate`. На новосъздадената колона може да се сложи и име, което стои от лявата страна на знака за присвояване, когато се подава като аргумент на функцията `mutate`. Новосъздадените колони могат да се използват веднага, още в самото извикване на функцията `mutate`.

Добавянето на колона не модифицира оригиналното множество данни. За да се отразят промените, то новосъздаденото множество трябва да бъде присвоено на съответната променлива. Пакетът `magrittr` предоставя двупосочна поточна операция (`%<>%`). С двупосочната поточна операция промените направени от дясната страна на операцията се отразяват на обекта от лявата страна на операцията (Листинг 6.29).

Листинг 6.29: Отразяване на модификациите

```
diamonds2 <- diamonds

diamonds2 %>% select(carat, price) %>% mutate(ratio=price/carat, square=ratio*
ratio)

head(diamonds2, n=3)
# A tibble: 3 x 4
  carat price ratio square
<dbl> <int> <dbl>   <dbl>
1  0.23   326 1417. 2008998.
2  0.21   326 1552. 2409887.
3  0.23   327 1422. 2021342.
```

Докато функцията `mutate` изпълнява определена операция върху стойностите в една колона, то функцията `summarize` (Листинг 6.30) връща един обект, който обобщава резултатите от една функция (например `mean`, `max`, `min`, `median` и други).

Листинг 6.30: Обобщаваща информация

```
summarize(diamonds, sd(price))
diamonds %>% summarize(sd(price))
# A tibble: 1 x 1
  'sd(price)'
  <dbl>
1    3989.

diamonds %>% summarize(AveragePrice=mean(price), MedianPrice=median(price),
  AverageCarat=mean(carat))
# A tibble: 1 x 3
  AveragePrice MedianPrice AverageCarat
  <dbl>         <dbl>         <dbl>
1    3933.         2401         0.798
```

Удобното при функцията `summarize` е, че тя позволява изчисление по няколко различни агрегатни функции и също така позволява да се зададат имена на колоните.

Листинг 6.31: Групиране при обобщение

```
diamonds %>% group_by(cut, color) %>% summarize(AveragePrice=mean(price),
  SumCarat=sum(carat))
```

```
# A tibble: 35 x 4
# Groups:   cut [?]
  cut      color AveragePrice SumCarat
<ord> <ord>      <dbl>      <dbl>
1 Fair    D          4291.        150.
2 Fair    E          3682.        192.
3 Fair    F          3827.        282.
4 Fair    G          4239.        321.
5 Fair    H          5136.        369.
6 Fair    I          4685.        210.
7 Fair    J          4976.        160.
8 Good    D          3405.        493.
9 Good    E          3424.        695.
10 Good   F          3496.        705.
# ... with 25 more rows
```

Обобщението на информацията по колони е полезно само по себе си, но значително повече информация носи, когато обобщението е извършено по групи (Листинг 6.31). При групирането може да се задават една или повече колони за групиране, а при обобщаването една или повече агрегатни функции.

Листинг 6.32: Подредба на резултатите

```
diamonds %>% group_by(cut) %>% summarize(AveragePrice=mean(price), SumCarat=sum(
  carat)) %>% arrange(desc(SumCarat))
# A tibble: 5 x 3
  cut      AveragePrice SumCarat
<ord>      <dbl>      <dbl>
1 Ideal      3458.      15147.
2 Premium    4584.      12301.
3 Very Good  3982.       9743.
4 Good       3929.       4166.
5 Fair       4359.      1684.
```

Подредбата на резултатите, след групиране и обобщение, се постига чрез функцията `arrange` (Листинг 6.32). Ако бъде използвана функцията `desc`, сортирането е в низходящ ред. Ако не бъде използвана, по подразбиране, сортирането е във възходящ ред.

6.4.5 Специфични изчисления и връзка с база данни

За по-специфични изчисления, които не са предварително реализирани в пакета `dplyr`, съществува възможността за използване на функцията `do` (Листинг 6.33).

Листинг 6.33: Специфични изчисления

```
bottom <- function(x, N=5){ x %>% arrange(carat) %>% head(N) }

diamonds %>% group_by(cut) %>% do(bottom(., N=3))
# A tibble: 15 x 10
# Groups:   cut [5]
  carat cut      color clarity depth table price      x      y      z
<dbl> <ord> <ord> <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.22 Fair    E      VS2    65.1    61    337   3.87   3.78   2.49
2  0.23 Fair    G      VVS2    61.4    66    369   3.87   3.91   2.39
3  0.25 Fair    D      VS1    61.2    55    563   4.09   4.11   2.51
4  0.23 Good    E      VS1    56.9    65    327   4.05   4.07   2.31
```

5	0.23	Good	F	VS1	58.2	59	402	4.06	4.08	2.37
6	0.23	Good	E	VS1	64.1	59	402	3.83	3.85	2.46
7	0.2	Very Good	E	VS2	63.4	59	367	3.74	3.71	2.36
8	0.21	Very Good	E	VS2	63.2	54	386	3.82	3.78	2.4
9	0.23	Very Good	H	VS1	59.4	61	338	4	4.05	2.39
10	0.2	Premium	E	SI2	60.2	62	345	3.79	3.75	2.27
11	0.2	Premium	E	VS2	59.8	62	367	3.79	3.77	2.26
12	0.2	Premium	E	VS2	59	60	367	3.81	3.78	2.24
13	0.2	Ideal	E	VS2	59.7	55	367	3.86	3.84	2.3
14	0.2	Ideal	D	VS2	61.5	57	367	3.81	3.77	2.33
15	0.2	Ideal	E	VS2	62.2	57	367	3.76	3.73	2.33

Функцията позволява да се изпълняват потребителски функции върху данните. Чрез комбинацията на `do` и `group_by` се връщат най-долните `N` реда, сортирани по карати, за всяко подмножество от вид срязване на диамантите.

При използването на поточни операции, левият операнд на потока се явява първи аргумент на функцията от дясно. Тъй като в примера (Листинг 6.33) от лявата страна на потока са данни, а не функция, то чрез операцията точка (`.`) се оказва мястото, на което данните трябва да бъдат използвани.

Голямо предимство на пакета `dplyr` е възможността му да обработва данни от база данни, по начин много подобен на работата с `data.frame` обекти. Пакетът работи с повечето популярни системи за управление на бази от данни, като PostgreSQL, MySQL, SQLite и други. Трябва да се има предвид, че операциите за работа с база данни са относително бавни и бързодействието на пакета е значително по-малко в сравнение с бързодействието постигано при `data.frame` обектите. Положителното на директната работа с база данни е, че при много големи обеми от данни би било неприемливо те да се зареждат в оперативната памет (или дори невъзможно).

Листинг 6.34: Работа с база данни

```
setwd( "~/Desktop" )

download.file("https://github.com/TodorBalabanov/Statistical-Data-Processing-
  with-R/blob/master/data/diamonds.db?raw=true", destfile="./diamonds.db", mode
  ="wb")

source <- src_sqlite("diamonds.db")
source
src:  sqlite 3.22.0 [/Users/todorbalabanov/Desktop/diamonds.db]
tbls: DiamondColors, diamonds, sqlite_stat1

table <- tbl(source, "diamonds")
table %>% head(3)
# Source:   lazy query [?? x 10]
# Database:  sqlite 3.22.0 [/Users/todorbalabanov/Desktop/diamonds.db]
  carat cut      color clarity depth table price      x      y      z
  <dbl> <chr> <chr> <chr> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
1  0.23 Ideal    E      SI2     61.5   55   326   3.95   3.98   2.43
2  0.21 Premium E      SI1     59.8   61   326   3.89   3.84   2.31
3  0.23 Good     E      VS1     56.9   65   327   4.05   4.07   2.31
```

За илюстриране на работата с бази данни, SQLite е изключително удобна възможност. За да може пакетът `dplyr` да работи е необходимо да бъде инсталиран и пакетът `dbplyr`, дори и да не е зареден в паметта. Като начало се определя работната директория, в която ще се помести файлът, съдържащ базата данни (Листинг 6.34). Следва команда за изтеглянето на самия файл. Към изтегления файл се насочва променлива, която ще бъде използвана като източник на данните. След установяване връзка към

базата е необходимо да се насочи променлива и към конкретна таблица. Макар и данните да изглеждат като `data.frame`, то те са таблица в база данни и повечето `dplyr` операции се извършват директно върху данните в базата данни.

Листинг 6.35: Пресмятания с данни в база данни

```
table %>% group_by(cut) %>% dplyr::summarize(AveragePrice=mean(price),
  AverageCarat=mean(carat))
# Source:   lazy query [?? x 3]
# Database: sqlite 3.22.0 [/Users/todorbalabanov/Desktop/diamonds.db]
  cut      AveragePrice AverageCarat
<chr>      <dbl>         <dbl>
1 Fair      4359.           1.05
2 Good      3929.           0.849
3 Ideal     3458.           0.703
4 Premium   4584.           0.892
5 Very Good 3982.           0.806
Warning messages:
1: Missing values are always removed in SQL.
Use 'AVG(x, na.rm = TRUE)' to silence this warning
2: Missing values are always removed in SQL.
Use 'AVG(x, na.rm = TRUE)' to silence this warning
```

Важно е да се има предвид, че липсващите данни винаги се игнорират при работа с SQL данни (Листинг 6.35).

6.5 Пакетът purrr

Пакетът `purrr` е създаден с цел изготвянето на стандартизиран начин за обхождане на списъци и вектори. В пакета са заимствани похвати от функционалното програмиране. В широк смисъл това означава, че функциите не зависят от нищо друго, освен от входящите си аргументи.

6.5.1 Фамилията функции `map`

В основата на пакета `purrr` е функцията `map`. Целта ѝ е да приложи определена функция върху всеки елемент на списък (независимо елемент от елемент) и като резултат да върне списък със същата дължина. Поведението прилича на вече описаната функция `lapply`.

Листинг 6.36: Прилагане на функцията `map`

```
library(purrr)

l1 <- list(m2=matrix(1:9,3), l2=1:5, m3=matrix(1:4,2), n1=2)

l1 %>% map(sum)
l1 %>% map(function(x) sum(x, na.rm=TRUE))
l1 %>% map(sum, na.rm=TRUE)
```

При липсващи стойности (NA), за да се извърши правилно пресмятането, на сумиращата функция трябва да се подаде `na.rm=TRUE`.

Функцията `map` винаги връща списък, като резултат, но не винаги това е желаният краен ефект. Поради тази причина, в пакета `purrr` са реализирани цяла група `map_*` функции (на мястото на `*` е върнатият тип). Функциите връщат стойност или, ако това не е възможно - грешка (Таб. 6.2).

Функция	Тип на върнатата стойност
map	list
map_int	integer
map_dbl	numeric
map_chr	character
map_lgl	logical
map_df	data.frame

Таблица 6.2: Фамилия функции map

Всяка от изброените функции очаква да получи, като входен аргумент, вектор с дължина единица за всеки елемент. Ако това условие не е спазено, функцията завършва изпълнението си с грешка (Листинг 6.37).

Листинг 6.37: Извиквания на map според типа на върнатата стойност

```

11 %>% map_int(NROW)
m2 l2 m3 n1
 3  5  2  1

11 %>% map_dbl(mean)
m2 l2 m3 n1
5.0 3.0 2.5 2.0

11 %>% map_chr(class)
m2 l2 m3 n1
"matrix" "integer" "matrix" "numeric"

11 %>% map_lgl(function(x) NROW(x) < 3)
m2 l2 m3 n1
FALSE FALSE TRUE TRUE

list(3,4,1,5) %>% map( function(x){ data.frame(A=1:x,B=x:1) } )

list(3,4,1,5) %>% map_df( function(x){ data.frame(A=1:x,B=x:1) } )

11 %>% map_if(is.matrix, function(x) x*2)

11 %>% map_if(is.matrix, ~ .x*2)

```

За изграждането на data.frame списък, чрез помощна функция, може да се използва анонимна функция, а броят на редовете да се подаде през поточната операция. За да бъде резултатът оформен като един data.frame, се използва функцията map_df. При функцията map_if модификация търпят само елементите, които отговарят на предварително поставеното условие. Вместо анонимна функция е възможно да се използва формула при извикването на map_if.

Листинг 6.38: Обхождане на data.frame

```

data(diamonds, package='ggplot2')

diamonds %>% map_dbl(mean)
  carat      cut      color      clarity      depth      table
0.7979397      NA      NA      NA      61.7494049      57.4571839
  price      x      y      z
3932.7997219  5.7311572  5.7345260  3.5387338

```

Warning messages:

```
1: In mean.default(.x[[i]], ...) :  
  argument is not numeric or logical: returning NA  
2: In mean.default(.x[[i]], ...) :  
  argument is not numeric or logical: returning NA  
3: In mean.default(.x[[i]], ...) :  
  argument is not numeric or logical: returning NA
```

Обхождането на `data.frame` обект е изключително лесно, тъй като по същество той е съставен от списъци (Листинг 6.38). При не числените колони резултатът е `NA`, тъй като за тях не може да се пресметне средна стойност.

Листинг 6.39: Едновременно обхождане на два списъка

```
l3 <- list(m4=matrix(1:25,5), m5=matrix(1:16,2), l4=1:5)  
l5 <- list(m6=matrix(1:25,5), m7=matrix(1:16,8), l6=15:1)  
  
map2(l3, l5, function(x, y){NROW(x) + NROW(y)})  
map2_int(l3, l5, function(x, y){NROW(x) + NROW(y)})  
pmap(list(l3, l5), function(x, y){NROW(x) + NROW(y)})
```

За паралелното обхождане на два списъка, в пакета `rply` са предложени функциите `map2` и `pmap` (Листинг 6.39).

Заклучение

Подходящата предварителна обработка на данните за извършване на статистически анализ е ключова стъпка от процеса. В практиката това често налага групиране по признаци, филтриране по зададен критерии или итеративно обхождане и манипулация по зададени условия. Доброто разбиране и ефективната предварителна обработка на данните води до ясни и разбираеми резултати в следствие на статистическия анализ.

Глава 7

Реорганизация на данните и обработка на символни низове

Освен манипулацията на данните, често се налага реорганизиране на начина, по който те са структурирани. Понякога се налага транспониране или пък обединяване на няколко множества данни в едно общо. При обединяване на данни от различни източници не винаги данните имат една и съща структура, което допълнително усложнява задачата по преструктурирането им.

7.1 Обединяване на множества от данни

Най-елементарният случай на обединение е при наличието на две множества от данни, които имат идентични колони или колони, които съвпадат по брой и имена.

Листинг 7.1: Обединяване на множества от данни

```
ds1 <- cbind(TV=c("BNT", "bTV", "Nova"), Channel=c(1, 2, 3), Rating=c(0.1, 0.3, 0.2))

ds2 <- data.frame(TV=c("HBO", "VH1", "MTV"), Channel=c(4, 5, 6), Rating=c(0.4, 0.5, 0.6),
  stringsAsFactors=FALSE)

ds <- rbind(ds1, ds2)
```

В такава ситуация се използват функциите `cbind` и `rbind` (Листинг 7.1). С функцията `cbind` (свързване на колони) се формира матрица, което изисква броят редове в съставляващите я списъци да е еднакъв. Функцията `rbind` (свързване на редове) обединява две множества, при които броят редове може да се различава.

В реалната практика, данните рядко биват събирани в подходяща за обединяване структура. В такива случаи често се налага използването на сливане по ключ, което е добре познато на хората, работещи с езика SQL. За демонстрация на възможните сливания е използвано множеството данни предоставено от USAID Open Government инициативата (Листинг 7.2).

Листинг 7.2: USAID множество от данни

```
setwd("~/Desktop")

download.file(url="https://github.com/TodorBalabanov/Statistical-Data-Processing",
  destfile="aid.zip", withR=TRUE)
```



```
unzip("aid.zip", exdir="./")
```

След като бъде свален архивният файл, той трябва да бъде разархивиран.

Листинг 7.3: Зареждане USAID данните в R

```
library(stringr)

for(file in dir("./", pattern="\\.csv")) {
  name <- str_sub(string=file, start=12, end=18)

  data <- read.table(file=file.path(".", file), header=TRUE, sep="," ,
    stringsAsFactors=FALSE)

  assign(x=name, value=data)
}
```

Тъй като данните са разпръснати в множество CVS файлове, чиито имена са съставени по определен шаблон, то е удачно зареждането на информацията да бъде автоматизирано, като се прегледа цялата директория и бъдат прочетени всички налични в нея CSV файлове (Листинг 7.3). Информацията от всеки прочетен файл трябва да се присвои на променлива и затова е важно да се подберат подходящи имена за променливите. R е език, в който малките и големите букви имат значение и поради тази причина трябва да се внимава с изписването на променливите. Един от вариантите за избор на имена е частична информация от основното име на файла. Чрез подходящо съкращаване на символите (преди 12 и след 18) от името на файла, се формира достатъчно разпознаваемо име за променлива. Следва прочитане на информацията и присвояването ѝ на съответната променлива.

7.1.1 Функция merge

Листинг 7.4: Сливане на данни с merge

```
head(merge(x=Aid_90s, y=Aid_00s, by.x=c("Country.Name", "Program.Name"), by.y=c(
  "Country.Name", "Program.Name")))
```

За сливане на данни в две data.frame структури може да се използва функцията merge (Листинг 7.4). Чрез by.x се определя ключът в левия data.frame, а чрез by.y се определя ключът в десния data.frame. Определянето на различни колони, като ключ е най-значимата възможност на функцията merge. Трябва да се има предвид, че функцията merge е в базовия пакет на R и съответно може да бъде относително бавна при изпълнението си, в сравнение с други алтернативни функции.

Листинг 7.5: Сливане на данни при data.table

```
library(data.table)

dt90 <- data.table(Aid_90s, key=c("Country.Name", "Program.Name"))
dt00 <- data.table(Aid_00s, key=c("Country.Name", "Program.Name"))

dt0090 <- dt90[dt00]
```

Сливането на данни в пакета data.table използва малко по-различен синтаксис от този при функцията merge (Листинг 7.5).

7.1.2 Функция join

Функцията `join`, от пакета `plyr`, работи по аналогичен начин, както функцията `merge`, но е с по-добро бързодействие. Недостатък на тази функция е, че всички колони, участващи в ключа, трябва да имат идентични имена (Листинг 7.6).

Листинг 7.6: Сливане на данни с `join`

```
library(plyr)

head(join(x=Aid_90s, y=Aid_00s, by=c("Country.Name", "Program.Name")))
```

Функцията `join` има аргумент, с който може да се укажат различните видове сливане (ляво, дясно, вътрешно и външно). За да се получи едно общо множество, чрез функцията `Reduce` може да се изпълнят множество сливания по двойки.

7.1.3 Транспониране на данните

В практиката често се налага размяна на редовете с колоните и обратното. Въпреки че програмни пакети, като `Microsoft Excel`, предлагат такава функционалност, понякога техните ограничения могат да създадат значителни затруднения. Например в `Microsoft Excel` е възприето, че редовете по брой значително превъзхождат възможностите за брой колони. При транспониране на големи обеми от данни е напълно възможно броят колони да не достигнат и това да доведе до грешка при трансформацията.

Ако се погледнат данните в `Aid_00s` се забелязва, че информацията е събирана по години, като за всяка от десетте години има отделна колона. Всяка колона определя сумата на средствата (в щатски долари), отделени по всяка от програмите. Такава организация на данните е удобна за преглеждане на информацията от човек, но далеч не е толкова удобна при автоматизирана обработка (например при изчертаване на графики).

Листинг 7.7: От колони към редове

```
library(reshape2)

melt00 <- melt(Aid_00s, id.vars=c("Country.Name", "Program.Name"), variable.name
  ="Year", value.name="Dollars")
head(melt00, n=3)

library(scales)

melt00$Year <- as.numeric(str_sub(melt00$Year, start=3, 6))

melt00$Program.Name <- str_sub(melt00$Program.Name, start=1, end=10)

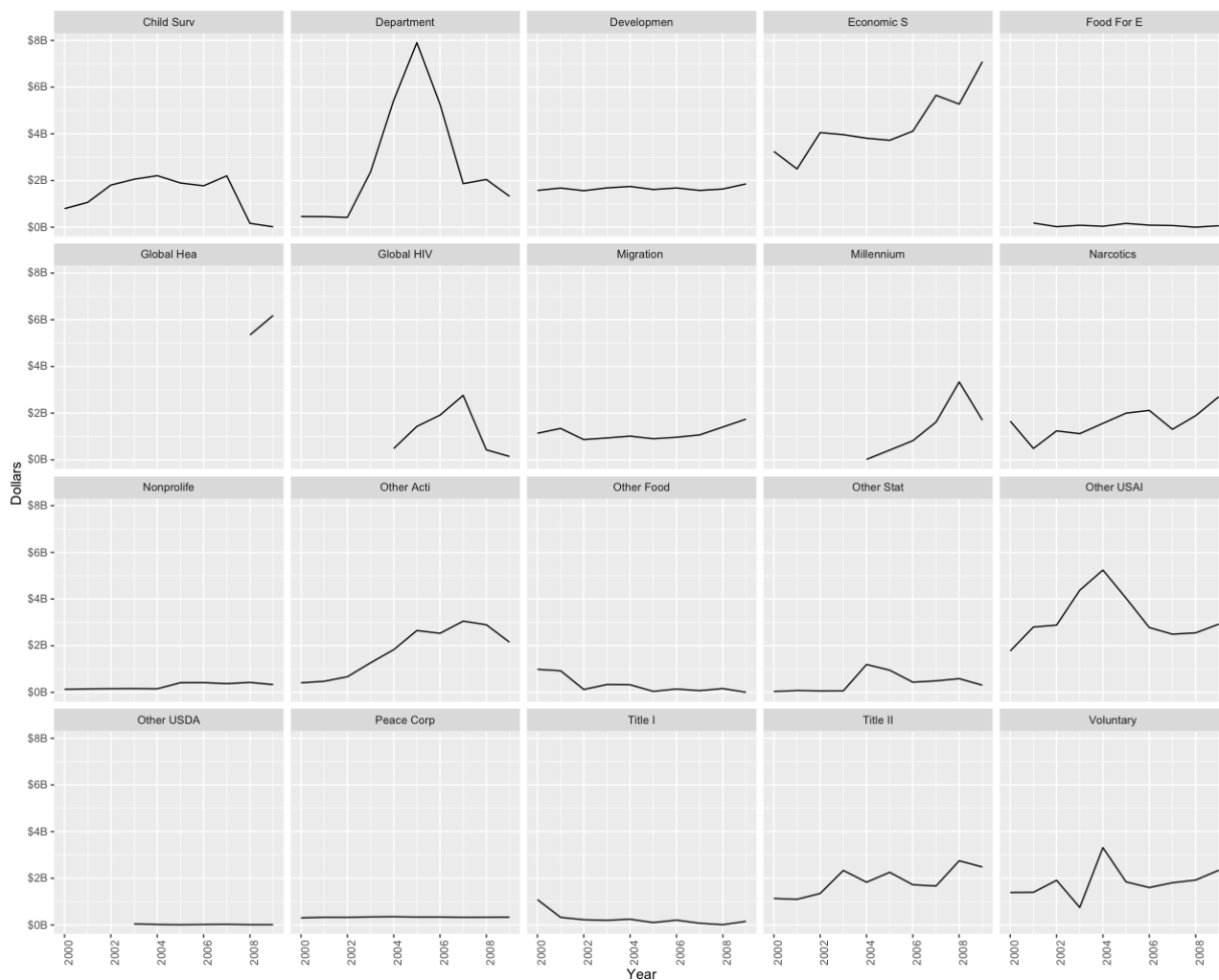
melt00 <- aggregate(Dollars ~ Program.Name + Year, data=melt00, sum, na.rm=TRUE)

library(ggplot2)
library(useful)

ggplot(melt00, aes(x=Year, y=Dollars)) + geom_line(aes(group=Program.Name)) +
  facet_wrap(~ Program.Name) + scale_x_continuous(breaks=seq(from=2000, to
    =2009, by=2)) + theme(axis.text.x=element_text(angle=90, vjust=1, hjust=0)) +
  scale_y_continuous(labels=multiple_format(extra=dollar, multiple="B"))
```

За преминаване от колонно-базирана организация на данните, към редово-базирана организация, в R може да се ползва функцията `melt` (Листинг 7.7). Аргументът `id.vars` определя имената на колоните,

които уникално идентифицират конкретен ред. За да се получи прилежно изглеждаща графика първоначално трябва да се премахнат буквите FY от полето за годината. След това имената на програмите се ограничават до 10 символа, тъй като някои от тези имена са твърде дълги и биха довели до несиметрично изписване при изчертаването на графиките. Следва агрегатна функция за сумиране по години. Така подготвени данните се подават за изчертаване от функцията ggplot (Фиг. 7.1).



Фигура 7.1: Разход на пари по програми и години

Организацията на данните е по редове. За да бъдат трансформирани от редове в колони се използва функцията dcast (Листинг 7.8).

Листинг 7.8: От редове към колони

```
melt00 <- melt(Aid_00s, id.vars=c("Country.Name", "Program.Name"), variable.name
              ="Year", value.name="Dollars")

cast00 <- dcast(melt00, Country.Name + Program.Name ~ Year, value.var="Dollars")
```

Първият аргумент на функцията dcast са множеството данни, които да бъдат използвани. Вторият аргумент е формула, на която лявата страна са колоните, които трябва да останат колони. От дясната страна са имената, които трябва да станат колони. Третият аргумент е колоната, която трябва да се използва за попълване на данните в новопоявилите се колони.

7.2 Сложни сливания на данни и трансформация на форматите

Към вече описаните функции за реорганизация на данните се добавят и функциите от пакетите `dplyr` и `tidyr`, които позволяват използването на поточни операции. В някои случаи тези функции имат по-добро въздействие от предходните.

7.2.1 Обединение на редове и колони

В пакета `dplyr` обединението на редове и колони се осъществява с функциите `bind_rows` и `bind_cols` (Листинг 7.9). Тези функции са проектирани да работят с `data.frame` и `tibble` обекти. Те не могат да работят с вектори и матрици.

Листинг 7.9: Обединяване по колони и редове

```
library(dplyr)
library(tibble)

ds1 <- bind_cols(tibble(TV=c("BNT", "bTV", "Nova"), Channel=c("1", "2", "3")), tibble(
  Rating=c("0.1", "0.3", "0.2")))

ds2 <- tribble(~TV, ~Channel, ~Rating, "HBO", "4", "0.4", "VH1", "5", "0.5", "MTV",
  "6", "0.6")

bind_rows(ds1, ds2)
```

И двете функции могат да работят с множество от обекти, едновременно.

7.2.2 Сложни сливания на данни

Пакетът `dplyr` предлага група от функции за извършване на сливания (`joins`). В множеството данни за диамантите цветът се отбелязва само с една латинска буква, без да има допълнителна информация за значението на самата маркировка. Допълнителна информация за цветовете на диамантите е налична в друго множество от данни, което е много подходящо за демонстрация на сливания между множества данни (Листинг 7.10).

Листинг 7.10: Сложни сливания

```
library(ggplot2)
library(readr)
library(dplyr)

colors <- as_tibble(read_table("https://raw.githubusercontent.com/TodorBalabanov/Statistical-Data-Processing-with-R/master/data/colors.csv", header=TRUE, sep=","))

left_join(diamonds, colors, by=c('color'='Color')) %>% select(carat, color, price, Description, Details)

tail(right_join(diamonds, colors, by=c('color'='Color'))))

inner_join(diamonds, colors, by=c('color'='Color'))

semi_join(colors, diamonds, by=c('Color'='color'))
```

```
anti_join(colors, diamonds, by=c('Color'='color'))
```

При лявото сливане (left join) данните за диамантите са отляво, а данните за цвета са от дясно. Тъй като колоната използвана за ключ се различава в главно и малко С, то ключът се задава с аргумента by. Данните се подават като вектор, където имената на елементите са колоните в лявата таблица, а стойностите са колоните от дясната таблица. При разминаване на типовете за колоните използвани като ключови (фактори със символни низове), автоматично се прави преобразуване. При лявото сливане участват всички редове от лявата таблица и само редовете, които отговарят по ключ от дясната таблица.

При десните сливания (right join) участват всички редове от дясната таблица, дори и когато няма съответствие по ключа в лявата таблица. При използвания пример, това води до поява на редове, съдържащи липсващи стойности (NA), там където няма съответствие.

При вътрешното сливане (inner join) участват само редове, които имат съответствие по ключа и в двете таблици.

При външното сливане (outer join) участват всички редове и от двете таблици, без значение дали имат съответствие по ключа.

В релационните бази данни няма полусливане (semi join). В езика R има реализация на полусливане. При полусливането на всеки ред от лявата таблица се връща първото срещане на ред в дясната таблица. Резултатът от полусливането е филтриране по редове. Връщат се само редове, които отговарят на съответствие по ключ.

Противоположното действие на полусливането в R се постига с anti join. Смисълът е, че се връщат тези редове от лявата таблица, които нямат съответствие по ключ в дясната таблица.

7.2.3 Реформатиране на данните

Размяната на колони с редове и обратното бе демонстрирана с функциите melt и dcast. Подобна реорганизация по редове и колони може да се постигне с функциите gather и spread от пакета tidyr.

Листинг 7.11: Данни за реакциите

```
library(readr)

emotions <- read_tsv("https://raw.githubusercontent.com/TodorBalabanov/
  Statistical-Data-Processing-with-R/master/data/reaction.txt")
Parsed with column specification:
cols(
  ID = col_integer(),
  Test = col_integer(),
  Age = col_double(),
  Gender = col_character(),
  BMI = col_double(),
  React = col_double(),
  Regulate = col_double()
```

За демонстрация на трансформацията от колони към редове е използвано множеството данни от Columbia University, което изследва емоционалните реакции и задръжки (Листинг 7.11). Тъй като данните са в обикновен текстов файл и като разделител се използва символът за табулация, за тяхното прочитане се използва функцията read_tsv (Tab Separated Values). Данните се зареждат в tibble обект, като за всяка колона се избира подходящ тип данни.

Листинг 7.12: Свиване от колони в редове

```
library(tidyr)
```

```
emotions %>% gather(key=Type, value=Measurement, Age, BMI, React, Regulate)
gather(emotions, key=Type, value=Measurement, -ID, -Test, -Gender)
```

Данните за емоциите са организирани на колони (wide format). С функцията `gather` част от колоните са трансформирани в редове (long format). При събирането на данните, за всяко наблюдавано лице, измерванията са записани в следните четири колони `Age`, `BMI`, `React` и `Regulate`. Така оформени данните са неподходящи за някои видове пресмятания и поради тази причина е разумно четирите вида измервания да бъдат в една колона (`Measurement`), а втора колона (`Type`) да показва за кой тип измерване е стойността (Листинг 7.12). С функцията `gather` се получава своеобразно събиране на стойностите от няколко колони в една. Първият аргумент е множеството данни, вторият аргумент ключът за новосъздадената колона, следва колоната със стойностите и след това се изброяват колоните, които формират общата колона със стойности. В оригиналните данни идентификаторът на изследвания обект присъства на два реда (два проведени експеримента), а в трансформираните данни идентификаторът на изследвания обект присъства на осем реда, тъй като за всеки експеримент са добавени по четири реда за различните видове измервания. Възможно е да се направи същата трансформация с изрично изброяване на колоните, които не трябва да бъдат събирани. Тогава се използва символът тилда пред имената на колоните.

Листинг 7.13: Разпъване от редове в колони

```
library(dplyr)

emotions %>% gather(key=Type, value=Measurement, Age, BMI, React, Regulate) %>%
  arrange(ID)

emotions %>% spread(key=Type, value=Measurement)
```

Обратното действие на събирането е разпъване в колони и в пакета `tidyr` то се постига с функцията `spread` (Листинг 7.13). Аргументът `key` определя имената на колоните които ще бъдат създадени. Аргументът `value` определя кои стойности ще бъдат използвани за попълване на новосъздадените колони.

7.3 Работа със символни низове

Работата със символни низове е много често срещана при обработката на статистически данни. Не рядко информацията е представена в чист текст (plain text). В R символни низове могат да се конструират или да бъдат изследвани, примерно с регулярни изрази. Пакетът `stringr` дава набор от функции за работа със символни низове.

7.3.1 Формиране на текст

Слепването (конкатенация) на символни низове е една от най-често срещаните операции и в R тя може да се постигне с функцията `paste` (Листинг 7.14). Ако не бъде зададен разделител, по подразбиране се използва символът интервал. Функцията `paste` е векторизирана и към нея могат да се подават аргументи под формата на вектори от символни низове. Важното е векторите да са с еднакви размери, така че елементите им да бъдат групирани по индекс. Функцията може да се използва и за слепване на вектор от символни низове, чрез разделител зададен в параметъра `collapse`.

Листинг 7.14: Конкатенация на символни низове

```
paste("Hello", "world", "!")

paste(c("Hello", "world", "!"), collapse="_")
```

Функцията `paset` е полезна за формирането на по-кратки текстове, но когато се работи със сложен или дълъг текст, в който трябва да се появят стойности на различни променливи, то функцията `sprintf` дава много повече възможности (Листинг 7.15).

Листинг 7.15: Разпъване от редове в колони

```
name = "Todor"
sprintf("Hello , %s!", name)
```

Функцията `sprintf` дава изключително много възможности за форматирането на текст и е позната още от библиотеките на програмния език C. Както много други функции в R, `sprintf` е векторизирана.

7.3.2 Извличане на текст

В практиката информацията често се получава в неструктуриран вид (предимно текстов формат) и в такива случаи се налага извличане на нужните данни, чрез анализ на неструктурирания поток (Листинг 7.16).

Листинг 7.16: Извличане на текст

```
library(XML)

presidents <- readHTMLTable("http://www.loc.gov/rr/print/list/057_chron.html",
  which=3, as.data.frame=TRUE, skip.rows=1, header=TRUE, stringsAsFactors=FALSE
)

library(stringr)

years <- str_split(string=presidents$YEAR, pattern="-")

ranges <- data.frame(Reduce(rbind, years))
names(ranges) <- c("Start", "Stop")

presidents <- cbind(presidents, ranges)
presidents$Start <- as.numeric(as.character(presidents$Start))
presidents$Stop <- as.numeric(as.character(presidents$Stop))

presidents[str_sub(string=presidents$Start, start=4, end=4) == 1, c("YEAR", "
PRESIDENT", "Start", "Stop")]
```

При четенето на неструктурирани данни, често в данните има нежелана информация и се налага допълнително коригиране. Първо трябва да се разделят годините, които са отделни с тире (-). Следва добавяне на две колони за начало и край. С помощта на функцията `str_sub` може да се използва част от символен низ.

Листинг 7.17: Регулярни изрази

```
presidents[str_detect(string=presidents$PRESIDENT, pattern="John"), c("YEAR", "
PRESIDENT", "Start", "Stop")]

sum( str_detect(presidents$PRESIDENT, "john") )
sum( str_detect(presidents$PRESIDENT, ignore.case("John")) )
```

При търсенето на по-сложен шаблон в текст, едно от най-мощните средства са регулярните изрази (Листинг 7.17). Регулярните изрази са реализация на краен автомат по дефиниран шаблон. Езикът на

регулярните изрази е твърде сложен за да бъде детайлно представен в настоящото изложение, поради тази причина се представят само няколко илюстративни примера.

Листинг 7.18: Сложни регулярни изрази

```
connction <- "https://github.com/TodorBalabanov/Statistical-Data-Processing-  
with-R/raw/master/data/war.rdata"  
  
load(connction)  
close(connection)  
  
wars[str_detect(string=wars, pattern="-")]  
str_split(string=wars, pattern="(ACAEA)|-", n=2)
```

Зареждането на бинарни файлове е малко по-различно от CSV файлове и това налага използване на функцията `load` (Листинг 7.18). Определянето на началото на всяка война става с функцията `str_detect`. Разделянето на символен низ, по зададен сепаратор, се извършва с функцията `str_split`. Функцията `str_trim` служи за премахване на „белите“ полета пред или след символния низ. За премахване на конкретна дума се използва функцията `str_extract`. Заместването по шаблон при първо срещане става с функцията `str_replace`, а на всяко срещане `str_replace_all`.

Заклучение

Реорганизирането на данните в подходяща форма е от ключово значение в процеса по осъществяването на статистически анализ. Този процес често включва трансформирането на колони в редове и обратното или сливане на различни множества от данни. Често в практиката данните са представени в текстов вид и това налага определени трансформации, така че данните да бъдат използвани при съответните статистически пресмятания.

Глава 8

Разширени графични възможности и вероятностни разпределения

8.1 Разширени графични възможности

Към базовите графични възможности на R, пакетите *ggplot2* и *lattice* добавят множество допълнителни функционалности. Синтаксисът на извикванията леко се различава от този на функциите в базовите възможности, но това създава затруднения само в началния етап от употребата на двата пакета.

За визуализация, чрез *ggplot2* като основа се използва функцията *ggplot*. В общия случай тази функция получава, като входен параметър данните за визуализиране и понякога някои допълнителни параметри. Резултатът от изпълнението на *ggplot* е обект, който в последствие може да бъде допълнително променян, чрез добавяне на възможности с помощта на операцията събиране (+).

8.1.1 Хистограми и плътности

Хистограмата служи за групиране на стойностите и изброяване на това колко стойности попадат по определените групи. Броят елементи във всяка група (bin) определя площта на стълба.

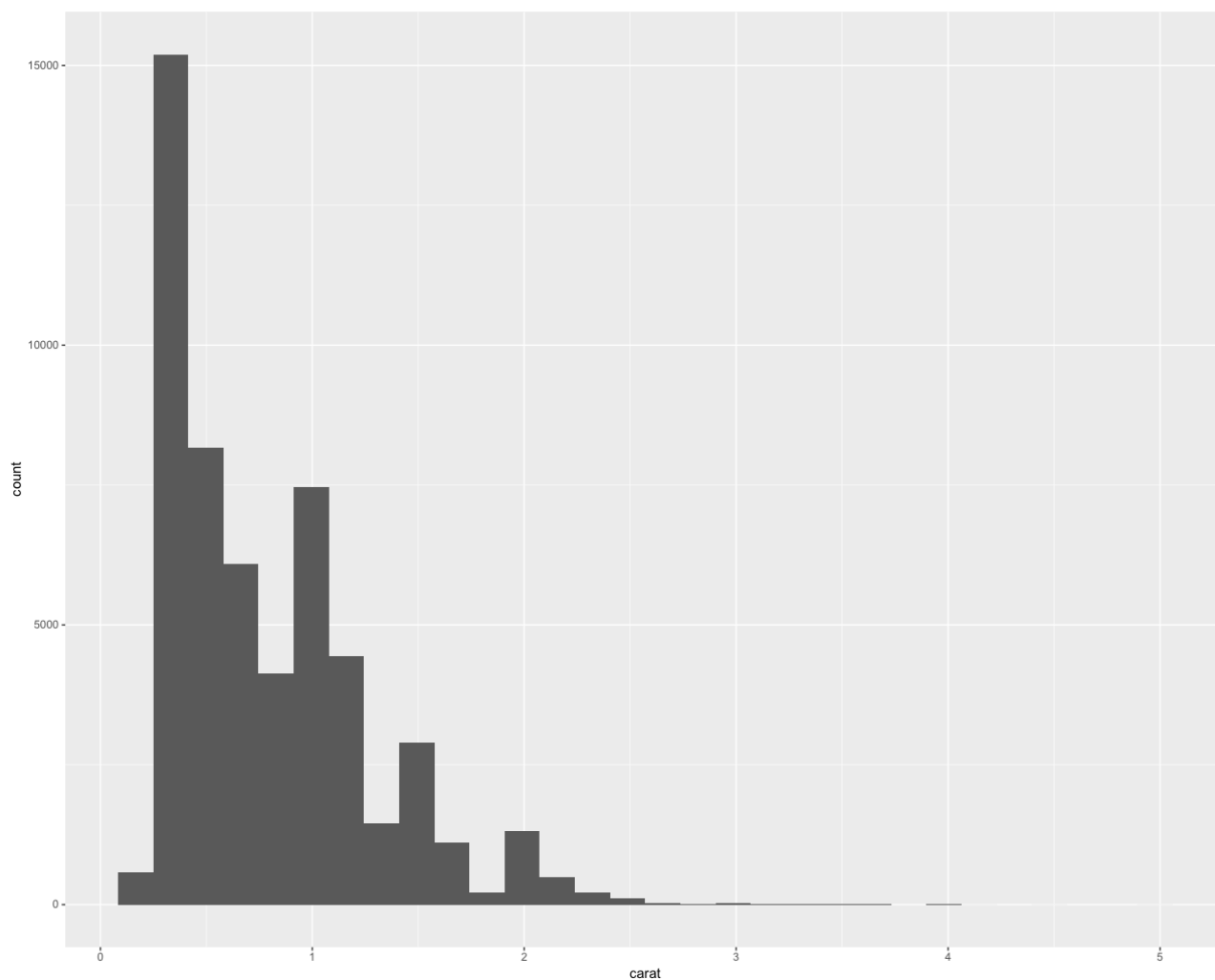
Листинг 8.1: Хистограма и плътност

```
library(ggplot2)

ggplot(data=diamonds) + geom_histogram(aes(x=carat))

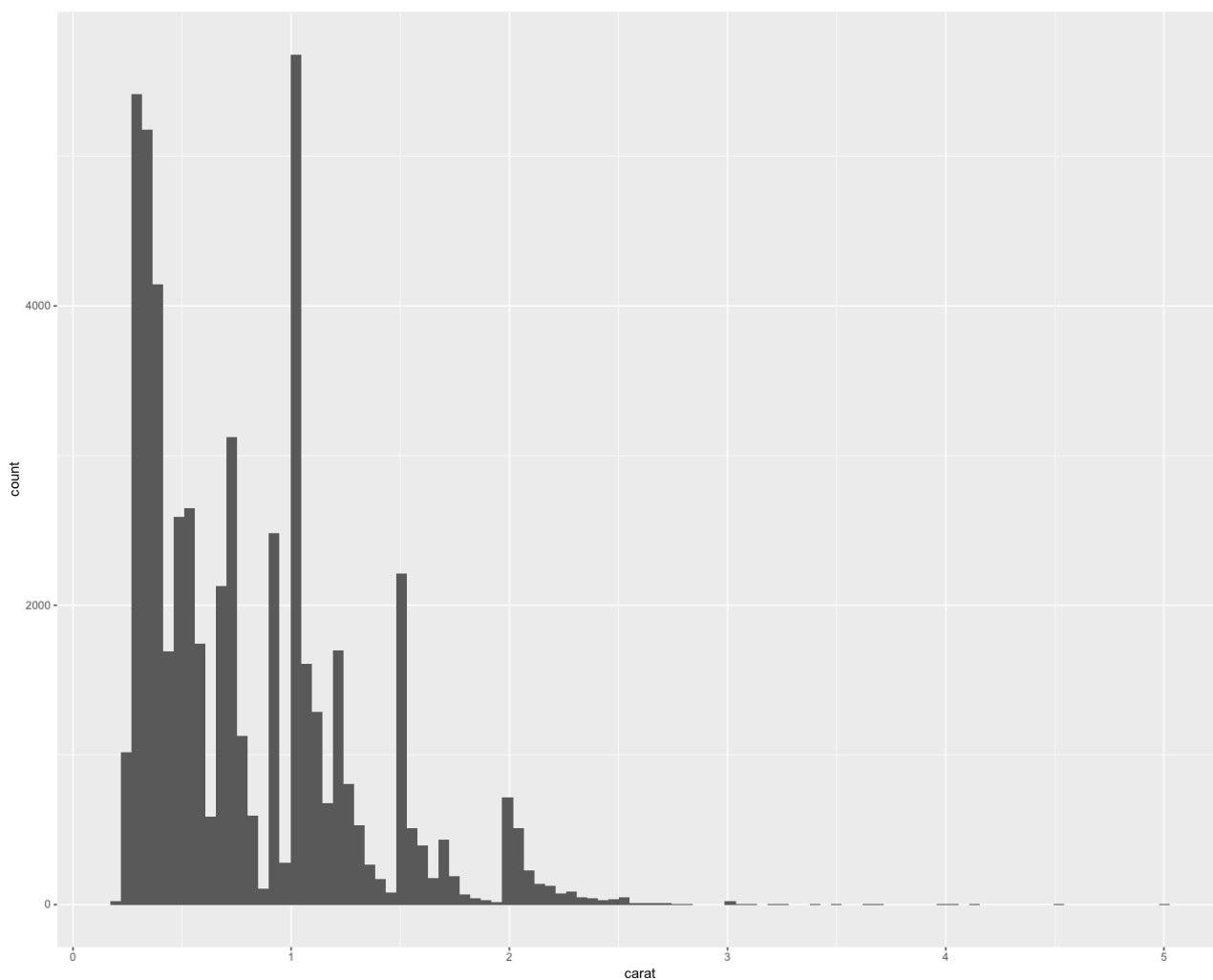
ggplot(data=diamonds) + geom_density(aes(x=carat), fill="grey50")
```

Както и в предходния пример за хистограма (Листинг 4.16), тук също е представено разпределението на диамантите по карати (Листинг 8.1).



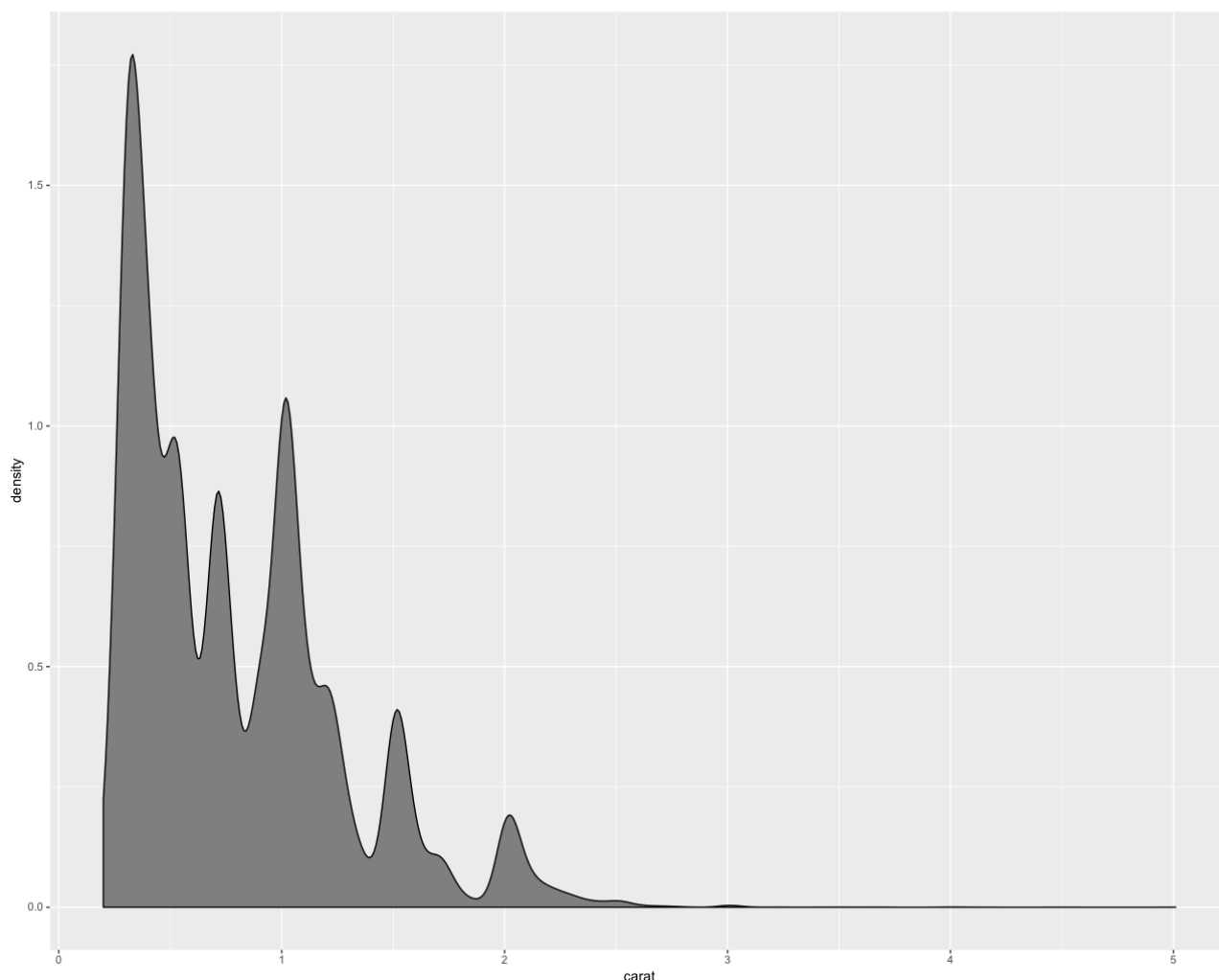
Фигура 8.1: Хистограма при 30 групи

Функцията *aes* определя кои данни да бъдат използвани за разполагане по осите. В примера с диамантите, това е характеристиката за тегло (карат).



Фигура 8.2: Хистограма при 100 групи

Размерът на групите в хистограмата може да варира (Фиг. 8.1,8.2). За да се изчертае плътностна функция е достатъчно графичният обект, генериран от *ggplot*, да бъде декориран с функцията *geom_density* (Фиг. 8.3), вместо с функцията *geom_histogram*.



Фигура 8.3: Плътностна функция

Хистограмата показва броене по групи, докато плътностната функция задава вероятността определен камък да попадне в предварително определен интервал. Макар и много да си приличат, хистограмата и плътностната функция са подходящи в два различни случая. Хистограмите са полезни при дискретни случайни величини, докато плътностните функции намират повече употреба в непрекъснатите случайни величини.

8.1.2 Диаграми на разсейване

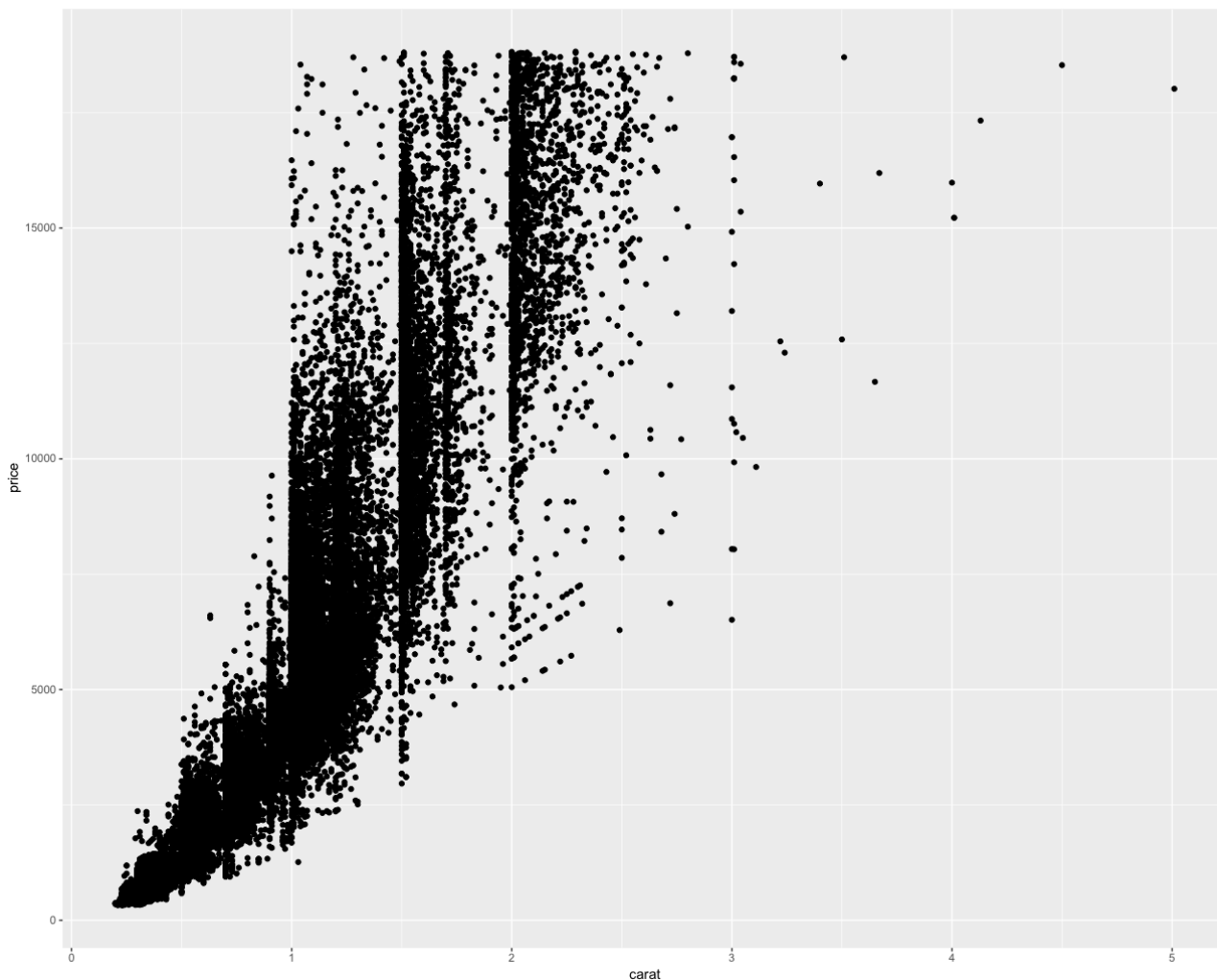
Пакетът *ggplot2* разширява възможностите за визуализация на диаграми на разпръскване, които базовата функционалност на R предлага (Листинг 8.2).

Листинг 8.2: Диаграма на разсейване с *ggplot2*

```
library(ggplot2)
ggplot(diamonds, aes(x=carat, y=price)) + geom_point()
```

Функцията *aes* определя кои колони от множеството данни да се използват при визуализацията (Фиг. 8.4). Пакетът *ggplot2* дава и друго много съществено предимство, графиката която ще се изчертава да

бъде съхранена в отделен обект, на който обект след това да се добавят различни визуални декорации (обектът *c2p* от примера).



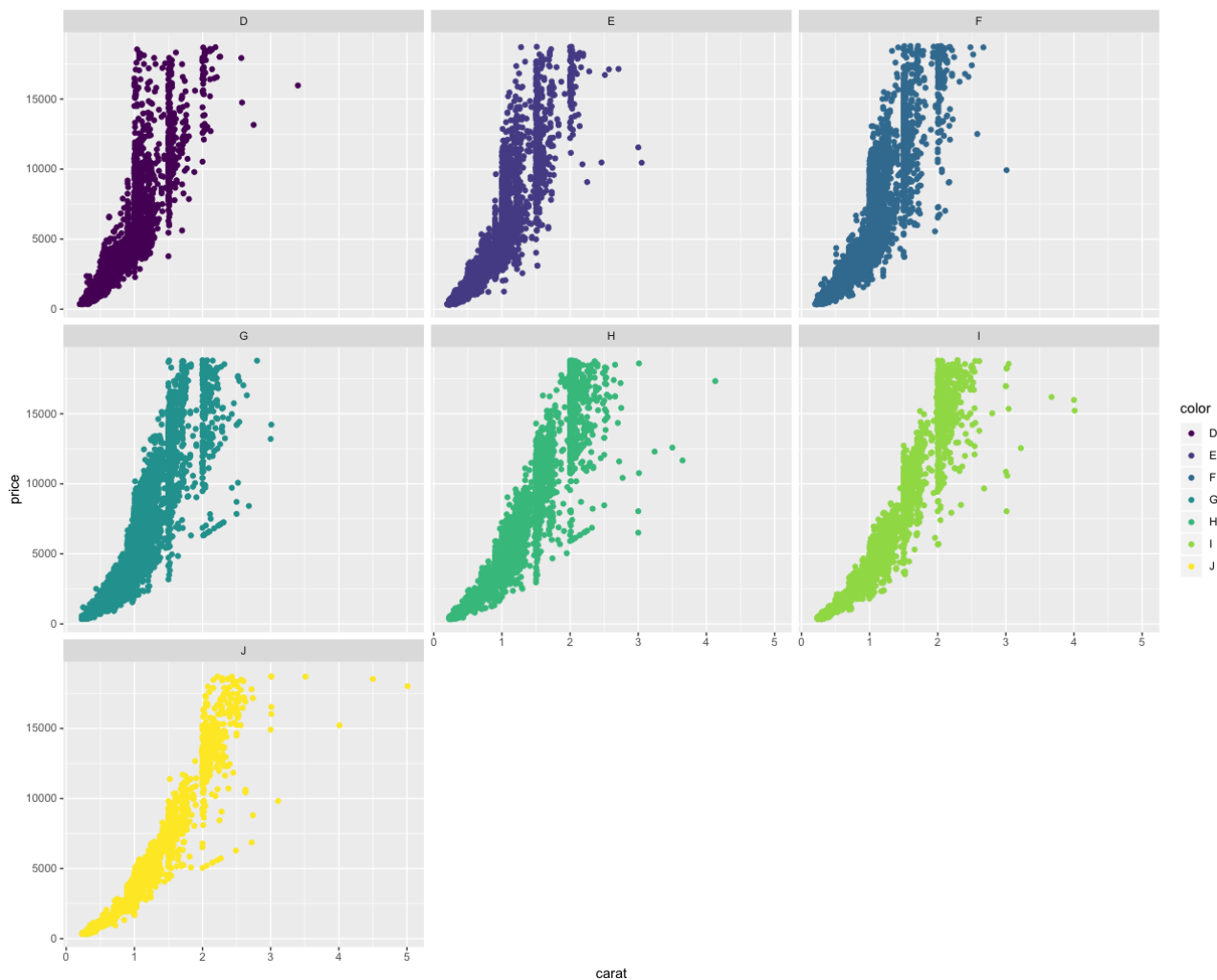
Фигура 8.4: Диаграма на разпръскване с ggplot2

Пакетът дава възможности и за подреждането на група от диаграми на разсейването. Това се постига с някои от функциите *facet_wrap* или *facet_grid* (Листинг 8.3).

Листинг 8.3: Диаграма на разпръскване групирани по признак

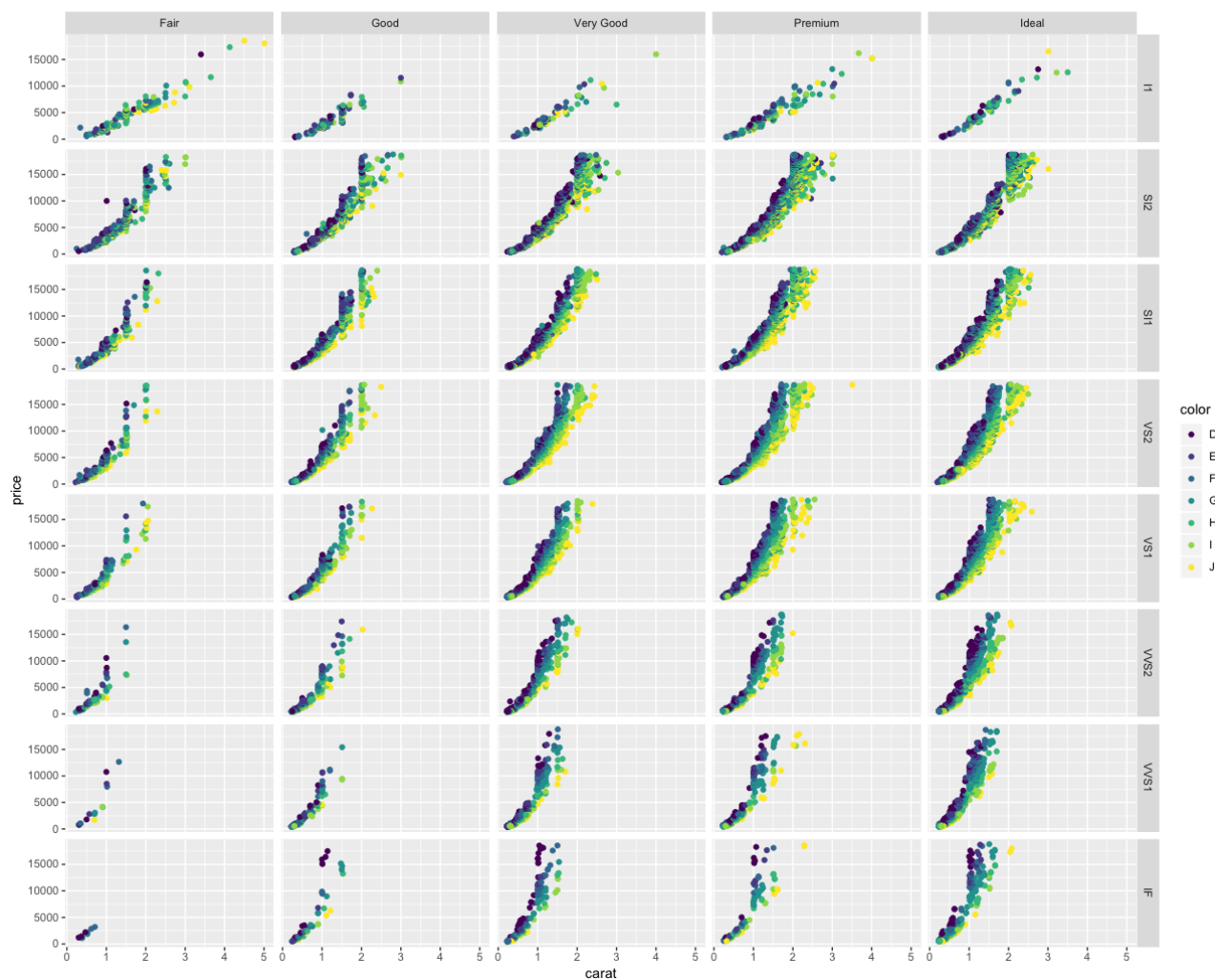
```
c2p <- ggplot(diamonds, aes(x=carat, y=price))
c2p + geom_point(aes(color=color)) + facet_wrap(~color)
c2p + geom_point(aes(color=color)) + facet_grid(clarity~cut)
```

Функцията *facet_wrap* разделя множеството от данните на групи, според зададения признак (в примера това е цветът на диамантите) и след това формира диаграма на разпръскване за всяка от групите (Фиг. 8.5).



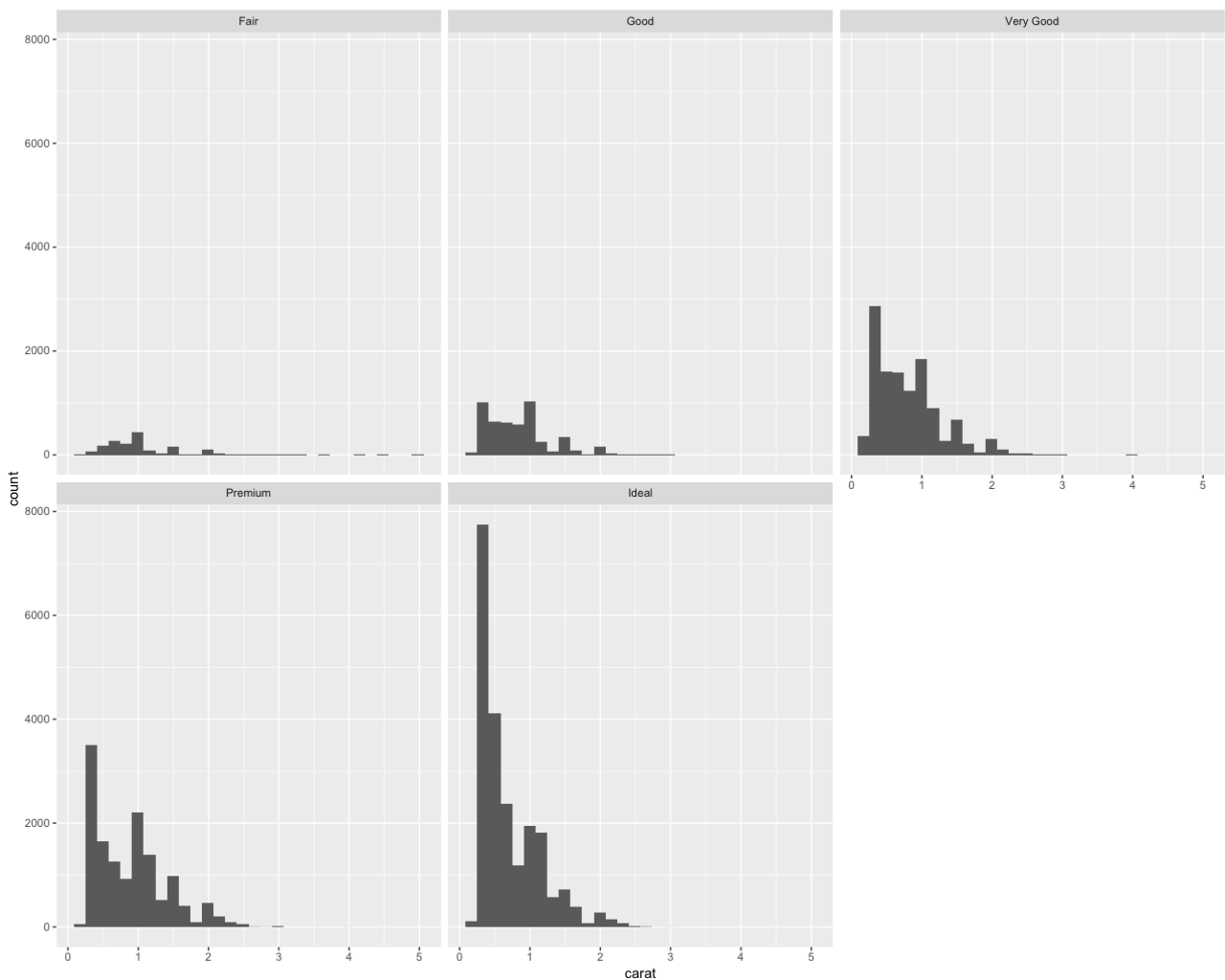
Фигура 8.5: Диаграма на разпръскване по групи за цвят на диамантите

Функцията *facet_grid* действа по сходен начин, но всички стойности на признака за групиране се отразяват на осите за всяка от графиките (Фиг. 8.6). Важно е да се отбележи начинът, по който са ориентирани осите на всяка от подграфиките. Ориентацията пряко зависи дали признакът за чистота е от ляво, а признакът за качество от дясно (*clarity~cut*) или обратното (*cut~clarity*).



Фигура 8.6: Визуализация с групиране по два признака

Организацията на графики по групи е възможна с различни графични представяния, като пример е представянето на хистограма в групи по качество на сряз (Фиг. 8.7).



Фигура 8.7: Визуализация на хистограми с групиране

8.1.3 Графики тип кутия и цигулка

Пакетът *ggplot2* дава възможност за визуализация на графики тип кутия (Листинг 8.4).

Листинг 8.4: Визуализация тип кутия

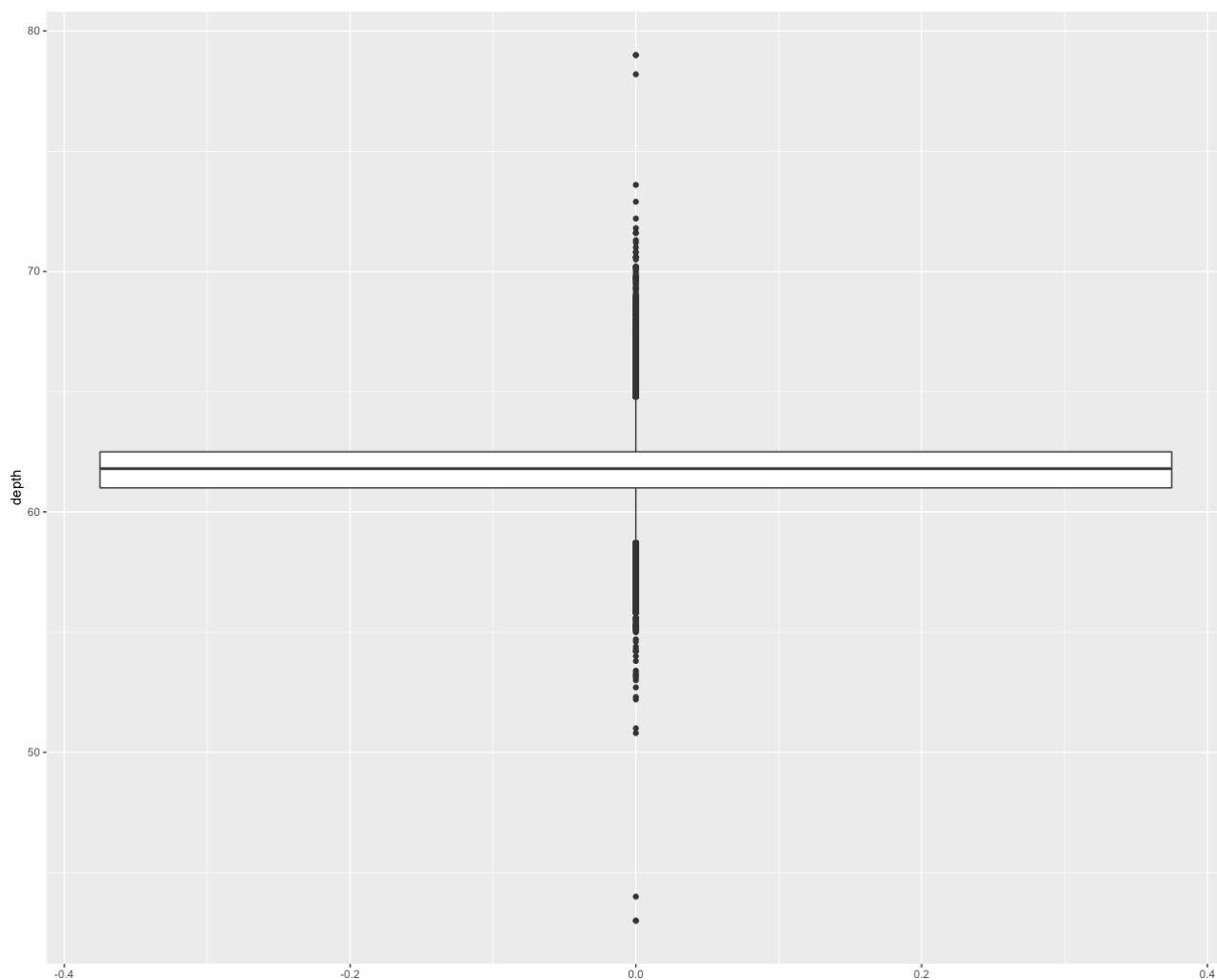
```
ggplot(diamonds, aes(y=depth)) + geom_boxplot()

ggplot(diamonds, aes(y=depth, x=cut)) + geom_boxplot()

ggplot(diamonds, aes(y=depth, x=cut)) + geom_boxplot() + geom_violin()

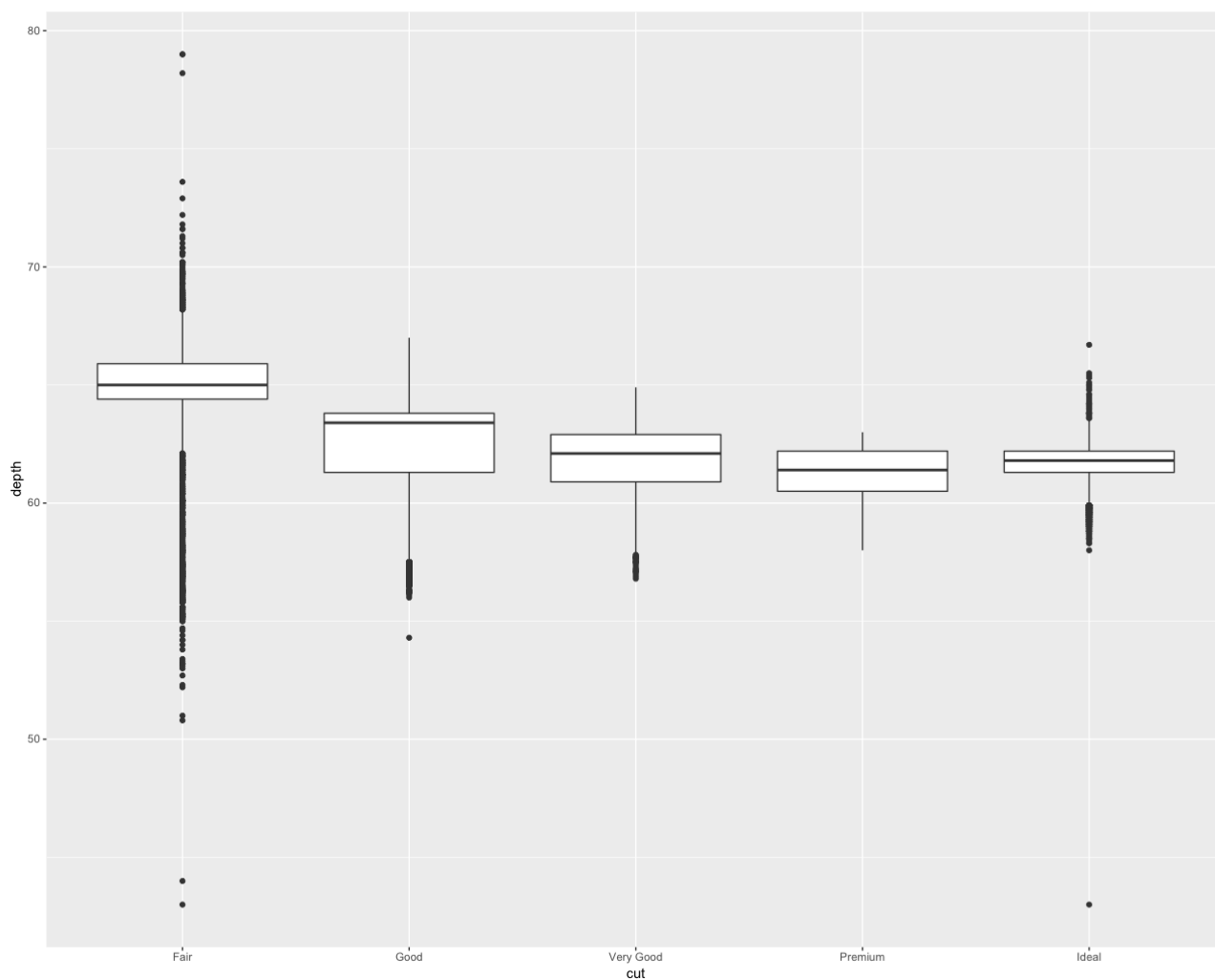
ggplot(diamonds, aes(y=depth, x=cut)) + geom_point() + geom_violin()
```

При обща визуализация на данните, без да се търси групиране по признак за абсцисната ос може да не се подава стойност (Фиг. 8.8).



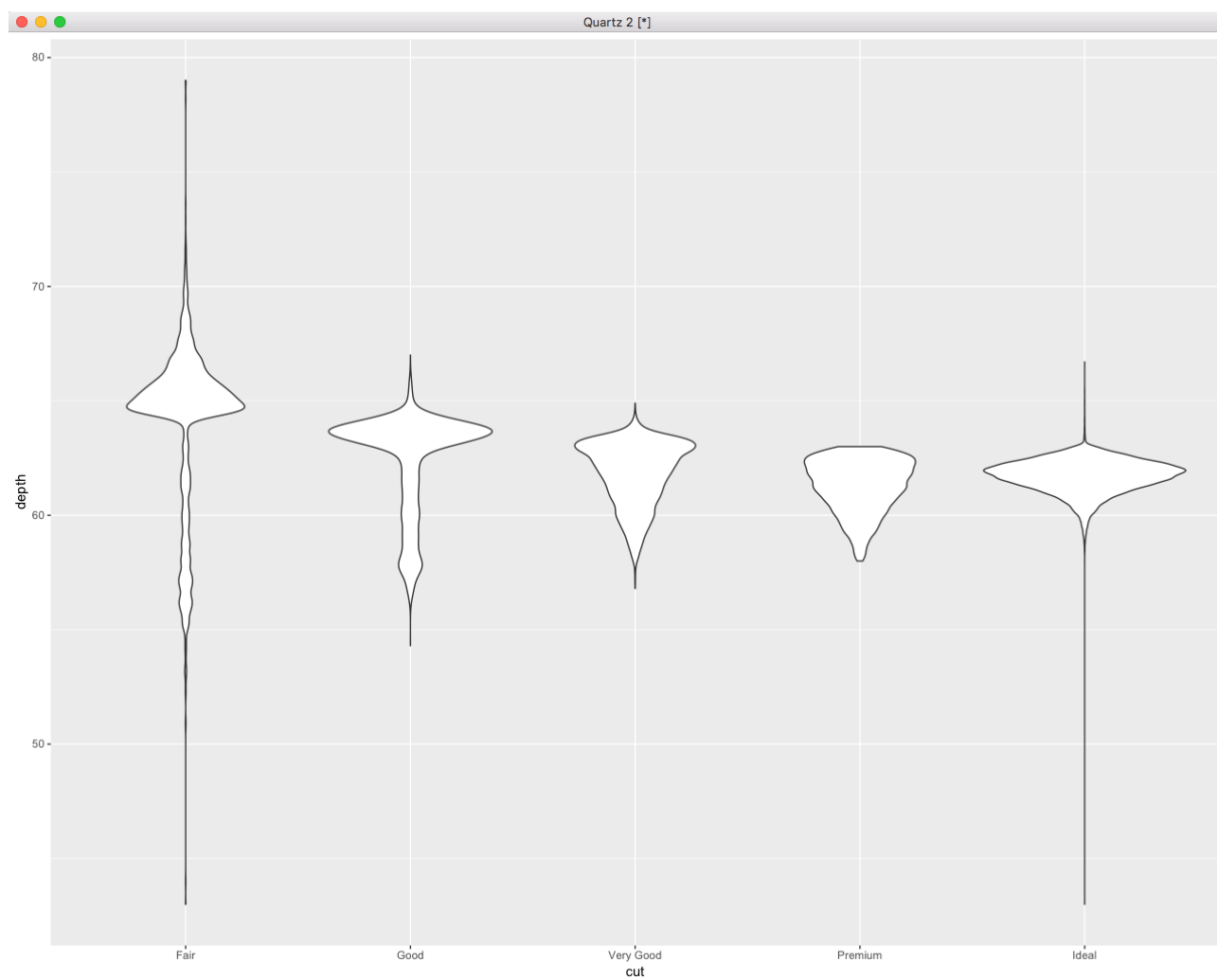
Фигура 8.8: Визуализация на характеристиката за дълбочина на диамантите

Визуализацията на графики от тип кутия, с групиране по признак се реализира чрез подаване на колоната, по която да се групира, като параметър за абсцисна ос, на функцията `aes` (Фиг. 8.9).



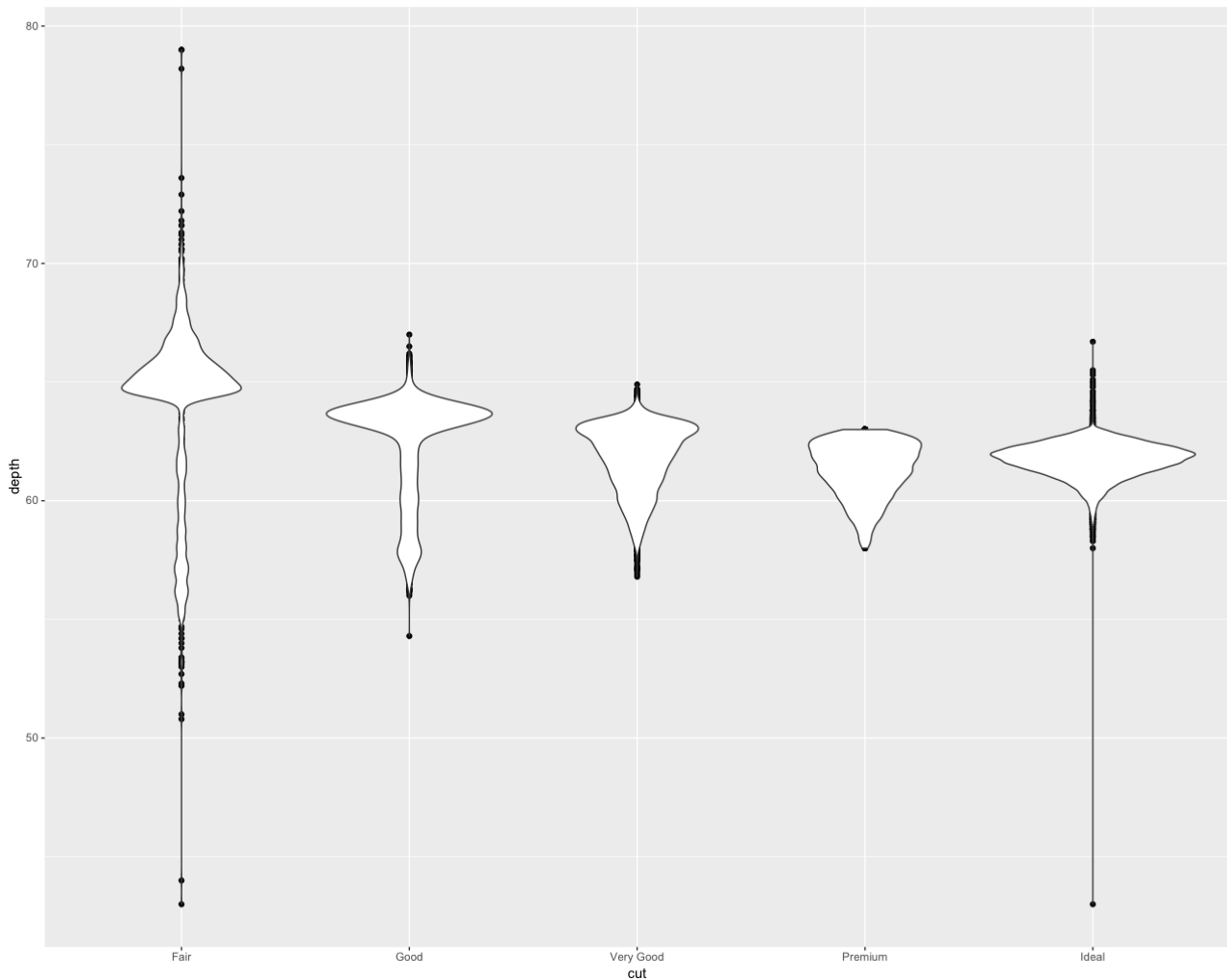
Фигура 8.9: Дълбочина на диамантите в групи според сряз

От графика тип кутии много лесно се преминава към графика от тип цигулки, чрез подмяна на декориращата функция (Фиг. 8.10).



Фигура 8.10: Графика тип цигулки

Графиките тип кутия и тип цигулка си приличат, като основната разлика е, че цигулките имат повече смисъла на плътностна функция и носят повече информация, отколкото правите ръбове на кутиите.



Фигура 8.11: Добавяне на декорация с точки

Декорациите за визуализация на данните може да се наслагват една върху друга, като от съществено значение е редът на изчертаването им (Фиг. 8.11). Ако декорацията с точките бъде добавена след декорацията с цигулките, точки ще се появят и върху самите цигулки.

8.1.4 Линеини графики

В определени случаи най-удачно е визуалното представяне на информацията да бъде извършено с линейни графики. Линейната графика е удачна примерно при представянето на тренд (Листинг 8.5).

Листинг 8.5: Линеини графики

```
library(lubridate)

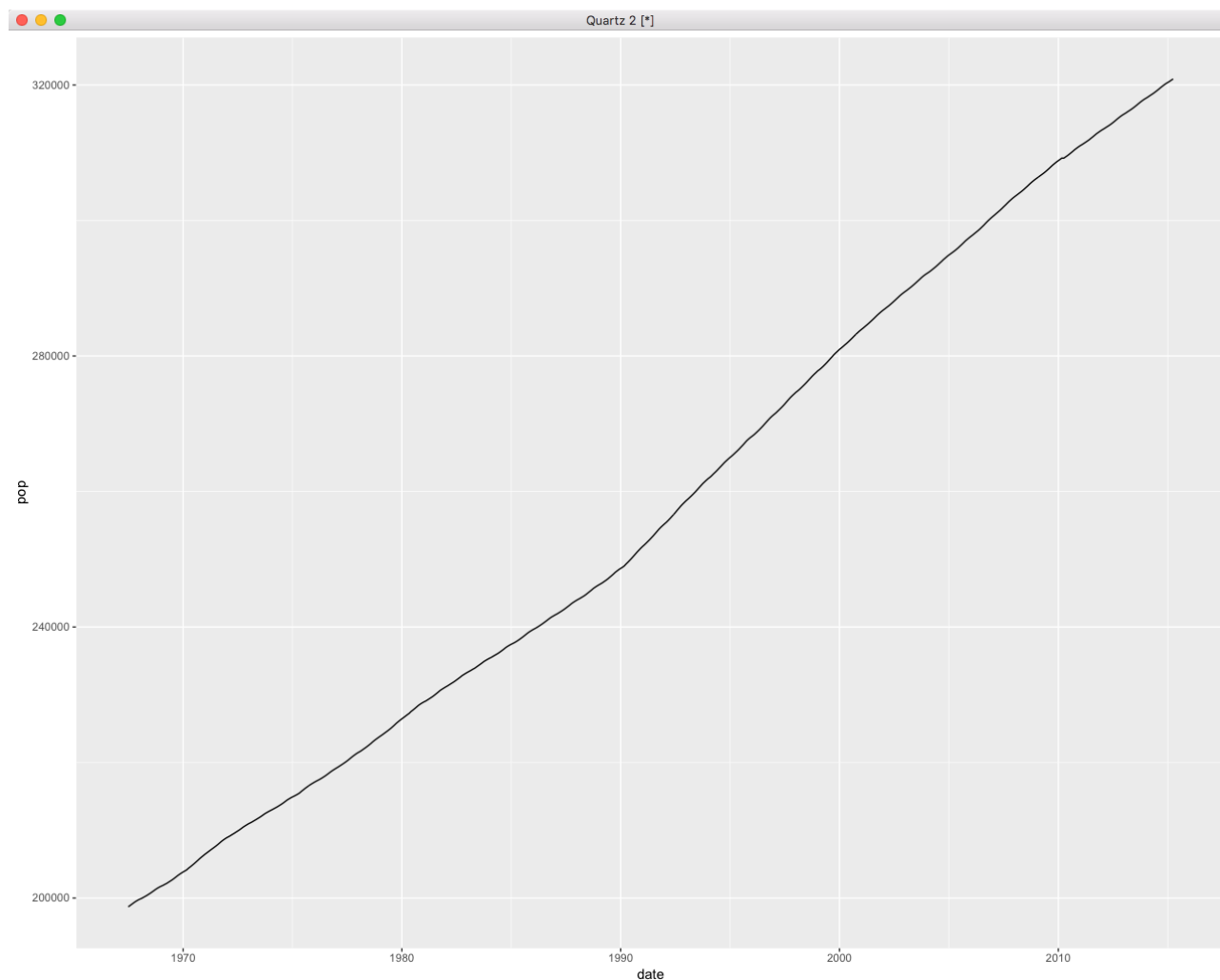
ggplot(economics, aes(x=date, y=pop)) + geom_line()

economics$year <- year(economics$date)
economics$month <- month(economics$date, label=TRUE)
```

```
library(scales)

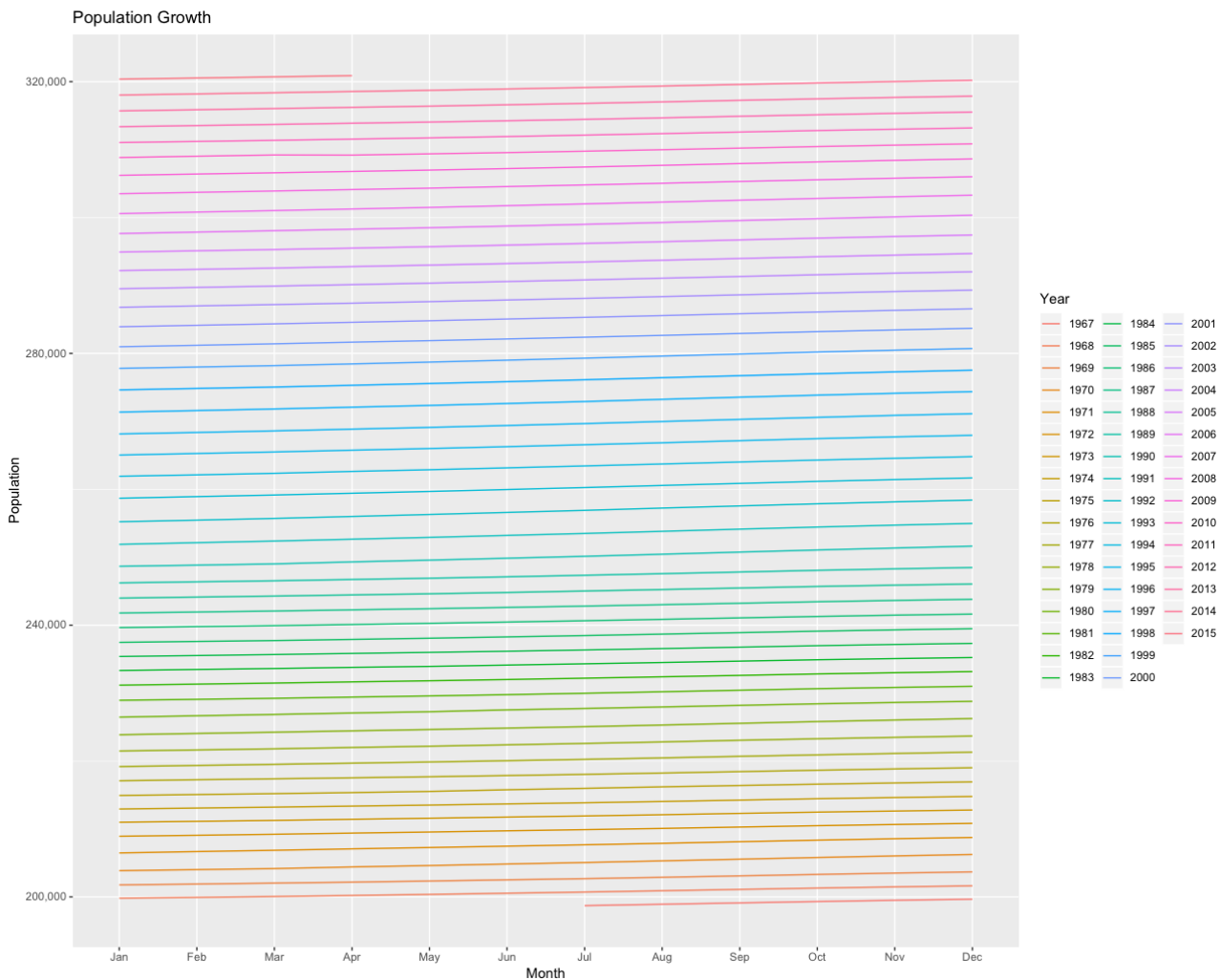
ggplot(economics, aes(x=month, y=pop)) + geom_line(aes(color=factor(year), group
  =year)) + scale_color_discrete(name="Year") + scale_y_continuous(labels=comma
  )+ labs(title="Population_Growth", x="Month", y="Population")
```

В примерните данни за икономическо развитие, нарастването на популацията във времето е представена под формата на линейна графика (Фиг. 8.12).



Фигура 8.12: Нарастване на популацията във времето

При анализирането на ръст в популацията понякога е интересно тази информация да се организира по години и да се представи в обща графика. Данните за всяка година могат да се представят с различен цвят, така че да бъде ясно кои линии за кои периоди от време се отнасят. С помощта на библиотеката *lubridate* в *economics* данните се добавят в две допълнителни колони за година и за месец. С помощта на библиотеката *scales* се постига по-добро оформление на информацията по осите.



Фигура 8.13: Визуализиране на прироста по години

Важно е да се отбележи, че информацията за годината е от тип *factor*, така че да се използва за определяне на цветовете. Също така, на ординатната ос се добавя и запетая, като разделител за хилядите. Като последна декорация е подмяната на текстовете за двете оси.

8.1.5 Тематично оформление

При генерирането на графики е от съществено значение медията на която тези графики ще бъдат представяни. При визуализация на проектор или монитор може да се използват тематично тъмни цветове, а при разпечатване на хартия е по-разумно да се използват светли цветове, така че да се намалява разхода на мастило (Листинг 8.6). Каквито и да са нуждите за представяне, в пакетът *ggthemes* са добавени възможности за цялостно преобразяване на получените графики, чрез избор на теми (Фиг. 8.14-8.17).

Листинг 8.6: Избор на теми за визуално представяне

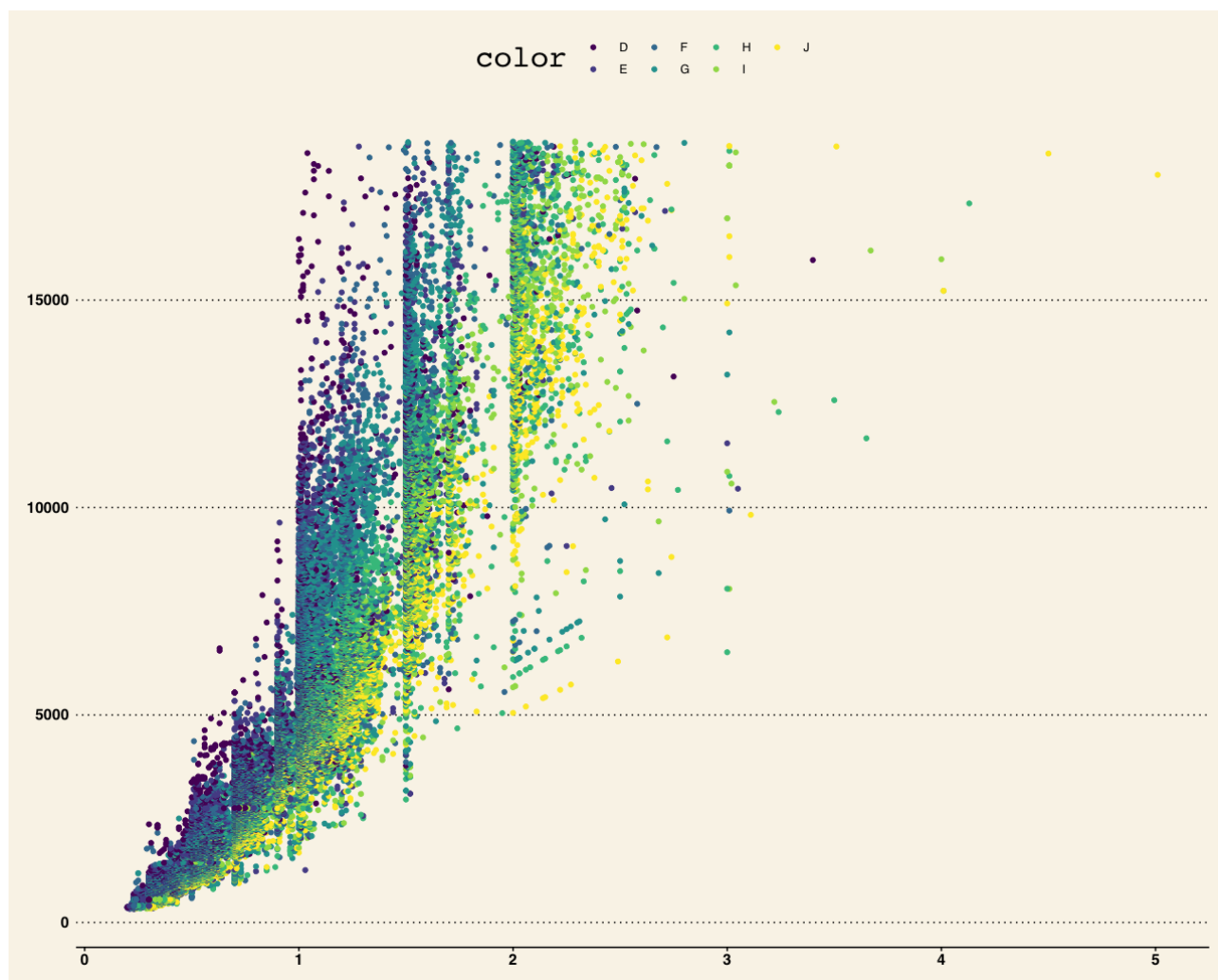
```
library(ggthemes)

ggplot(diamonds, aes(x=carat, y=price)) + geom_point(aes(color=color)) + theme_
wsj()
```

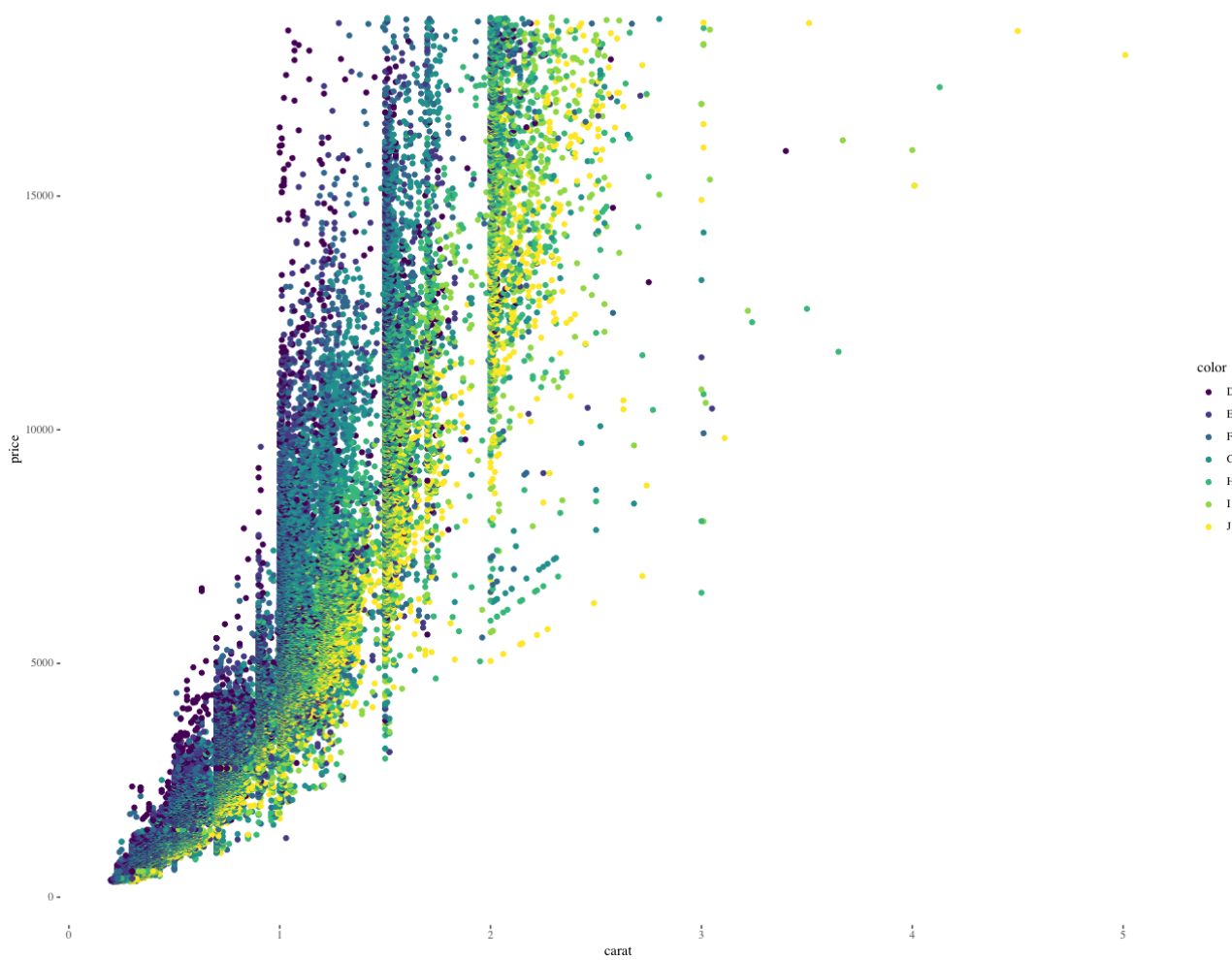
```
ggplot(diamonds, aes(x=carat, y=price)) + geom_point(aes(color=color)) + theme_
  tufte()

ggplot(diamonds, aes(x=carat, y=price)) + geom_point(aes(color=color)) + theme_
  excel()

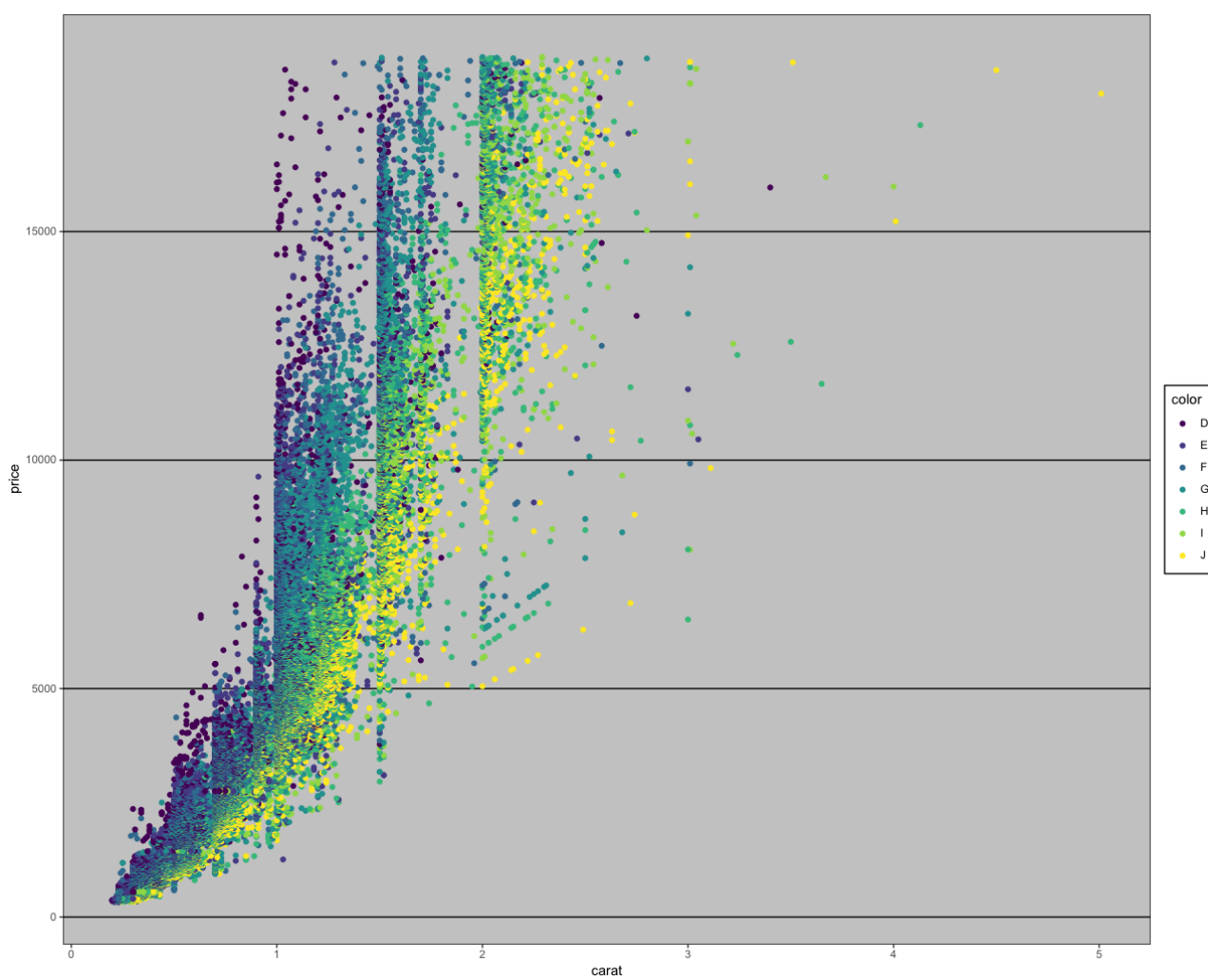
ggplot(diamonds, aes(x=carat, y=price)) + geom_point(aes(color=color)) + theme_
  economist() + scale_colour_economist()
```



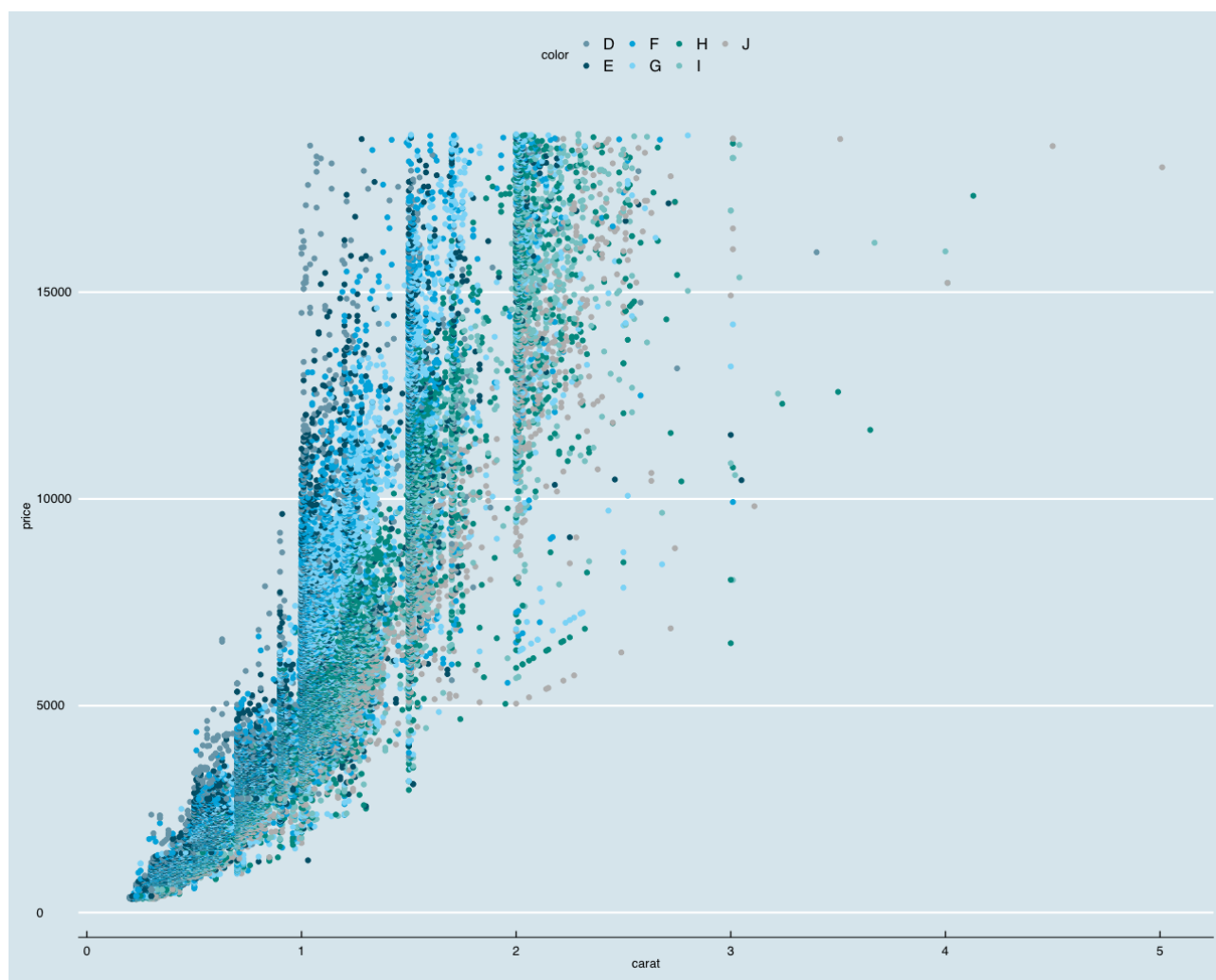
Фигура 8.14: Тема Wall Street Journal



Фигура 8.15: Тема Edward Tufte



Фигура 8.16: Тема в стил Microsoft Excel



Фигура 8.17: Тема Economist

8.2 Изследване на случайни величини

Когато се работи с данни за които няма предварителна информация, е от съществено значение да се определи какви са параметрите на вероятностното разпределение. С помощта на пакета *Rdice* е възможно да се генерират експерименти с различни зарове. Чрез генерирането и статистическото изследване на множество случайни събития се осъществява изследване наречено „Монте Карло метод“.

Листинг 8.7: Случайни величини със зарове

```
library(ggplot2)
library(Rdice)

x <- dice.roll(faces=6, dice=1, rolls=100000)

ggplot(data=x$results) + geom_histogram(aes(x=values)) + ggtitle("Single_die_
  rolled_100K_times.") + xlab("Die_Side") + ylab("Outcomes") + scale_x_
  continuous(breaks=round(seq(min(x$results$values),max(x$results$values),by
    =0.5)))
```

```
x <- dice.roll(faces=6, dice=2, rolls=100000)

ggplot(data=x$results) + geom_histogram(aes(x=(die_1+die_2))) + ggtitle("Two_
dice_rolled_100K_times.") + xlab("Dice_Sides") + ylab("Outcomes") + scale_x_
continuous(breaks=round(seq(min((x$results$die_1+x$results$die_2)),max((x$
results$die_1+x$results$die_2))),by=0.5)))

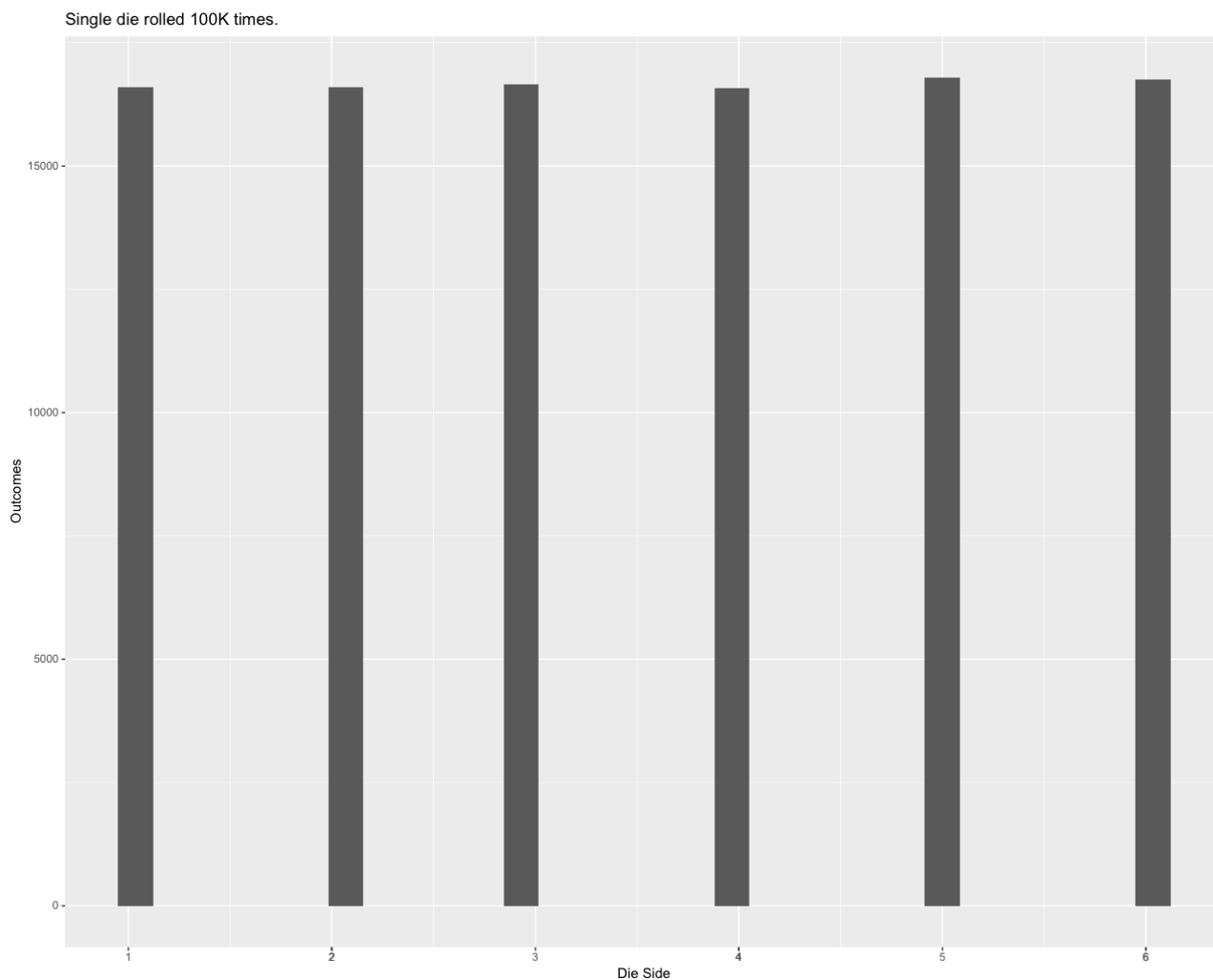
x <- dice.roll(faces=6, dice=6, rolls=100000)
x$results$values = rowSums( x$results[,1:6] )

ggplot(data=x$results) + geom_histogram(aes(x=values)) + ggtitle("Six_dice_
rolled_100K_times.") + xlab("Dice_Sides") + ylab("Outcomes") + scale_x_
continuous(breaks=round(seq(min(x$results$values),max(x$results$values)),by
=0.5)))

x <- dice.roll(faces=6, dice=10, rolls=100000)
x$results$values = rowSums( x$results[,1:10] )

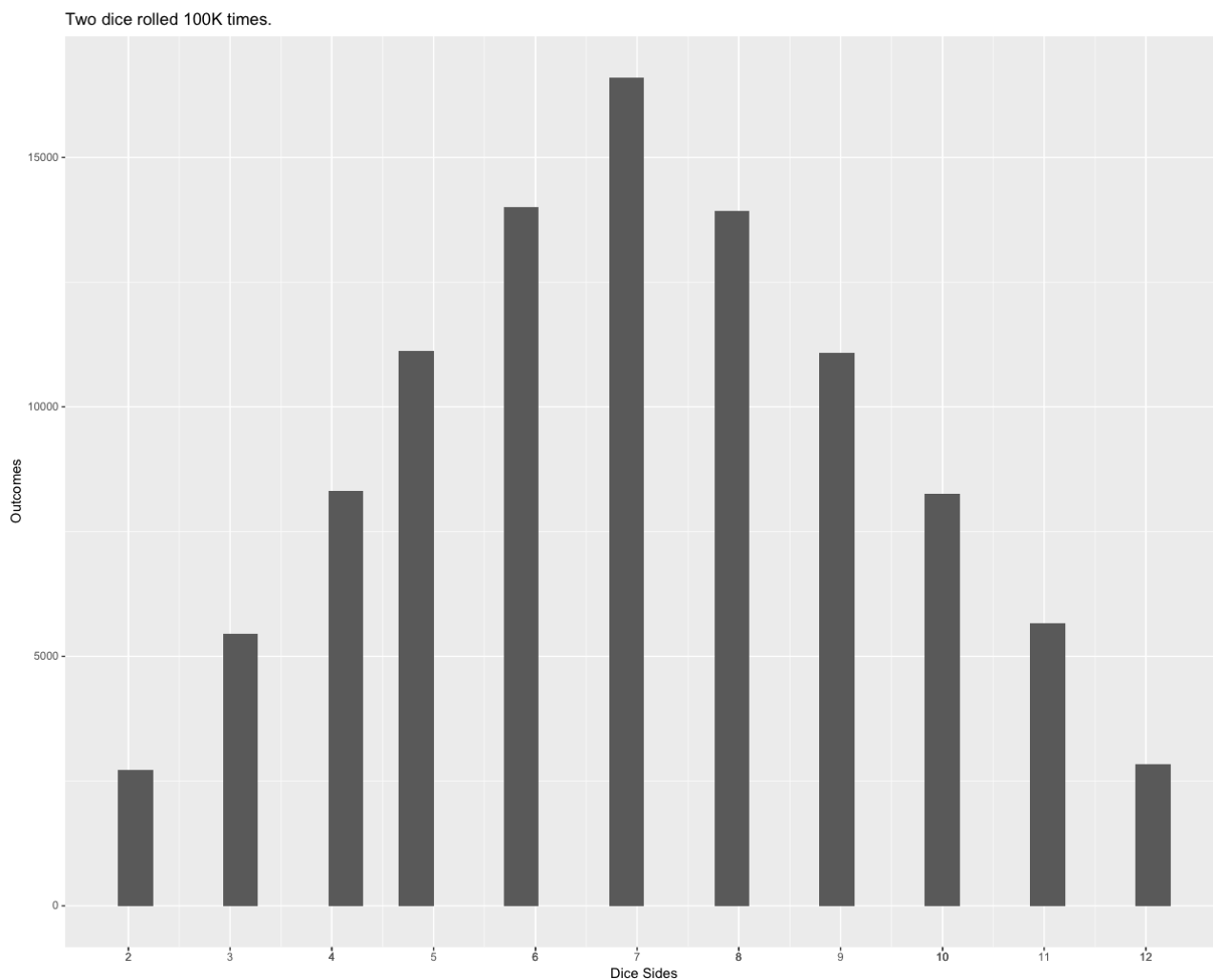
ggplot(data=x$results) + geom_density(aes(x=values)) + ggtitle("Ten_dice_rolled_
100K_times.") + xlab("Dice_Sides") + ylab("Outcomes") + scale_x_continuous(
breaks=round(seq(min(x$results$values),max(x$results$values)),by=0.5)))
```

Ако се приеме, че променливата x е резултатът от многократното хвърляне на един математически честен зар, с шест страни, то чрез изчертаване на хистограмата може да се добие представа за характера на случайната величина (Листинг 8.7). От изчертаната хистограма ясно се вижда, че случайната величина е дискретна и равномерно разпределена (Фиг. 8.18).



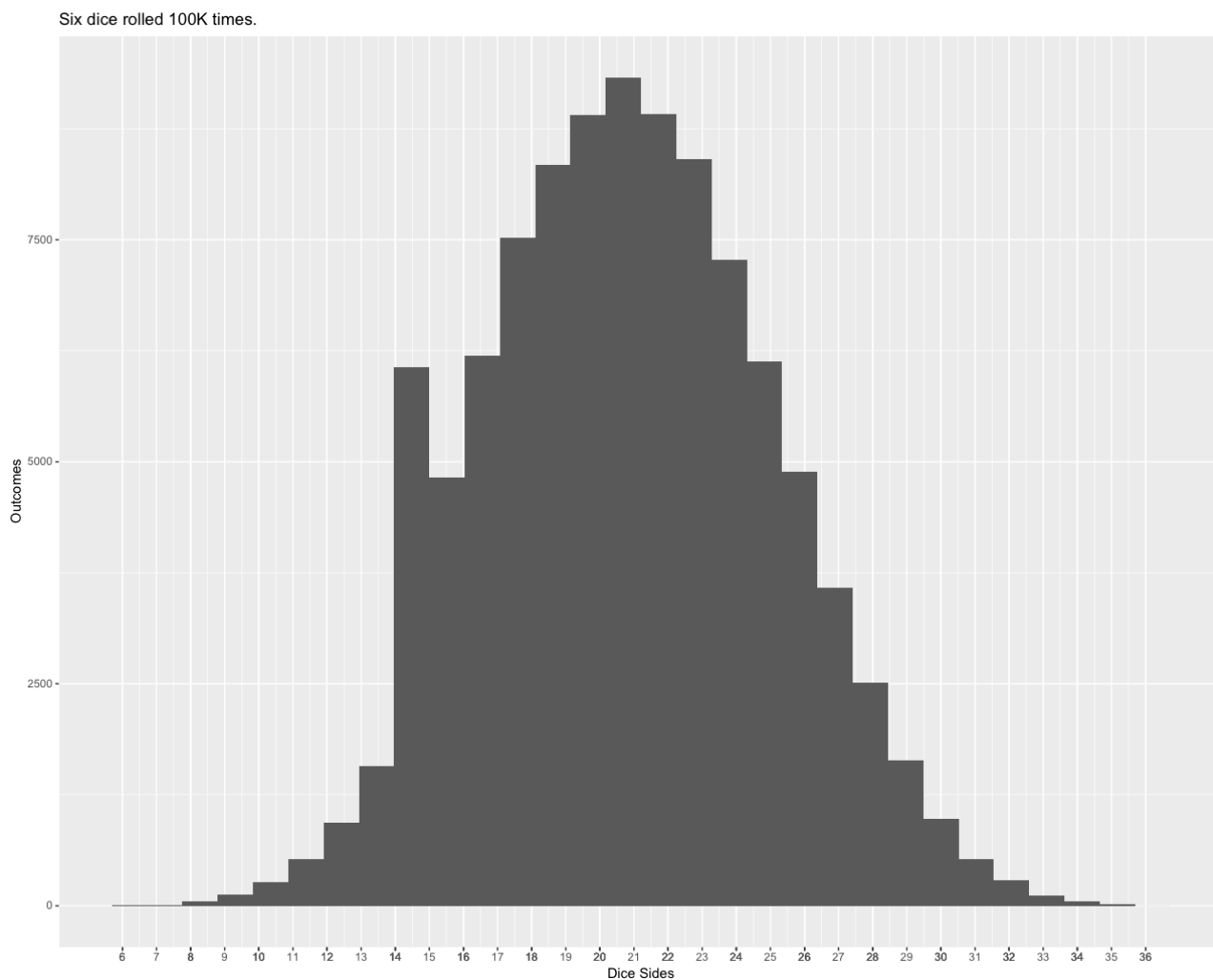
Фигура 8.18: Хистограма на хвърлянията за един зар

Когато същият експеримент бъде повторен, но вместо един зар се използват два зара, ясно се различава, че някои събития стават по-малко вероятни от други и разпределението се превръща в триъгълно (Фиг. 8.19).



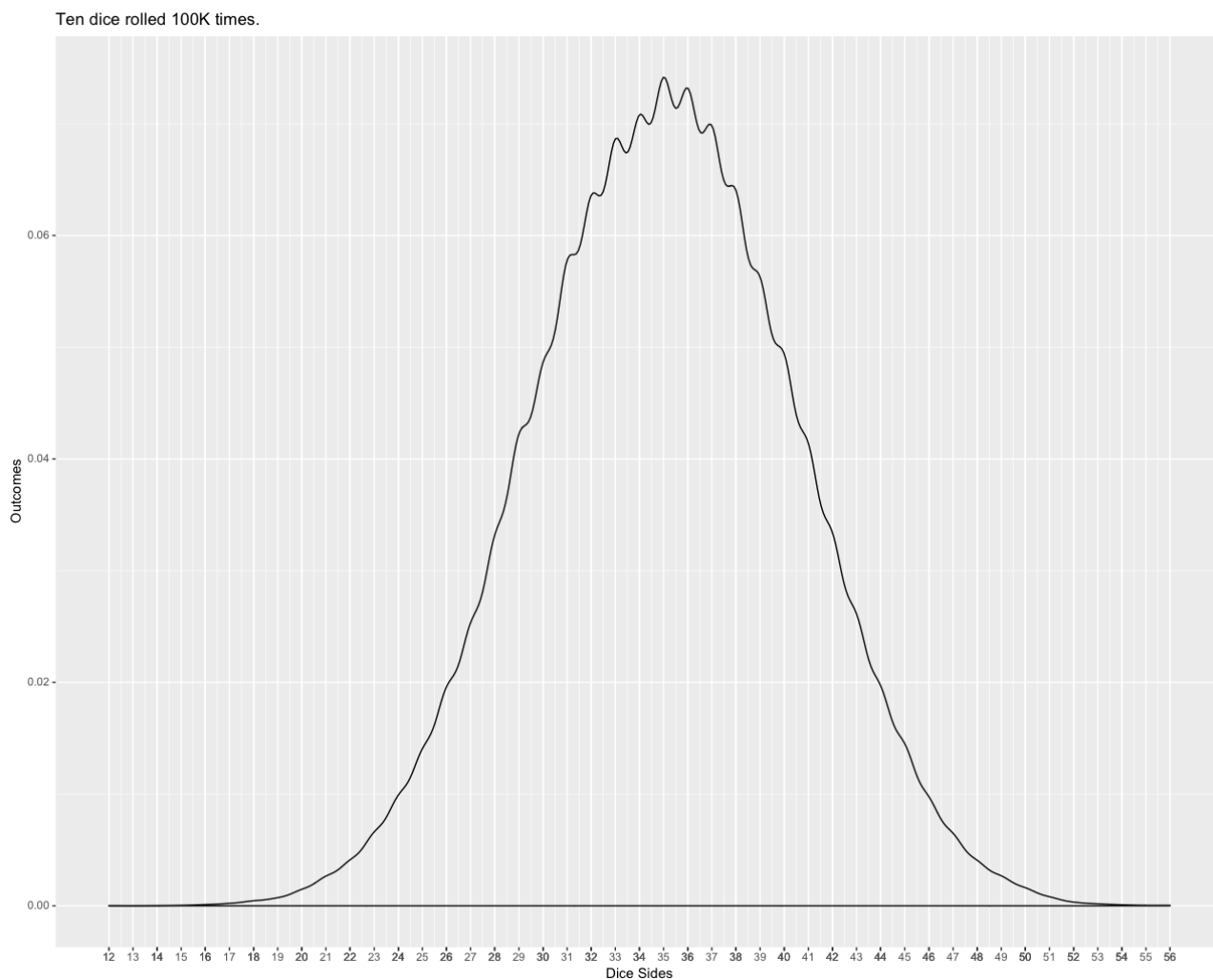
Фигура 8.19: Хистограма на хвърлянията за два зара

При шест зара ясно започва да се различава формата на нормалното разпределение (Фиг. 8.20).



Фигура 8.20: Хистограма на хвърлянията за шест зара

При десет зара и изчертаване на плътностна диаграма формата на нормалното вероятностно разпределение е ясно забележима (Фиг. 8.21).



Фигура 8.21: Плътностна диаграма на хвърлянията за десет зара

За изследването на една случайна величина, хистограмата и плътностната диаграма носят първоначална ориентировъчна информация за характеристиките на величината.

8.3 Вероятностни разпределения

В реалната практика от статистическия анализ се наблюдават множество случайни величини, които винаги се подчиняват на някакво вероятностно разпределение. Определянето на вероятностното разпределение, към което принадлежи случайната величина има изключително важна роля за адекватното и надеждно извършване на статистическия анализ. След определяне на вероятностното разпределение от съществена важност е и определянето на параметрите, които характеризират разпределението.

8.3.1 Нормално разпределение

Най-често срещаното в природата и най-използваното в статистическия анализ е нормалното разпределение. Също така, познато е и под названието на Гаусово разпределение (Формула 8.1).

$$pdf(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} \quad (8.1)$$

Нормалното разпределение се характеризира с два параметъра - средната стойност μ и стандартно отклонение σ . Формата на графиката, с която се изобразява нормалното разпределение е като камбана. Средната стойност задава къде се намира върхът на камбаната, по оста X , а стандартното отклонение определя широчината на камбаната.

Листинг 8.8: Нормално разпределение

```
library(ggplot2)

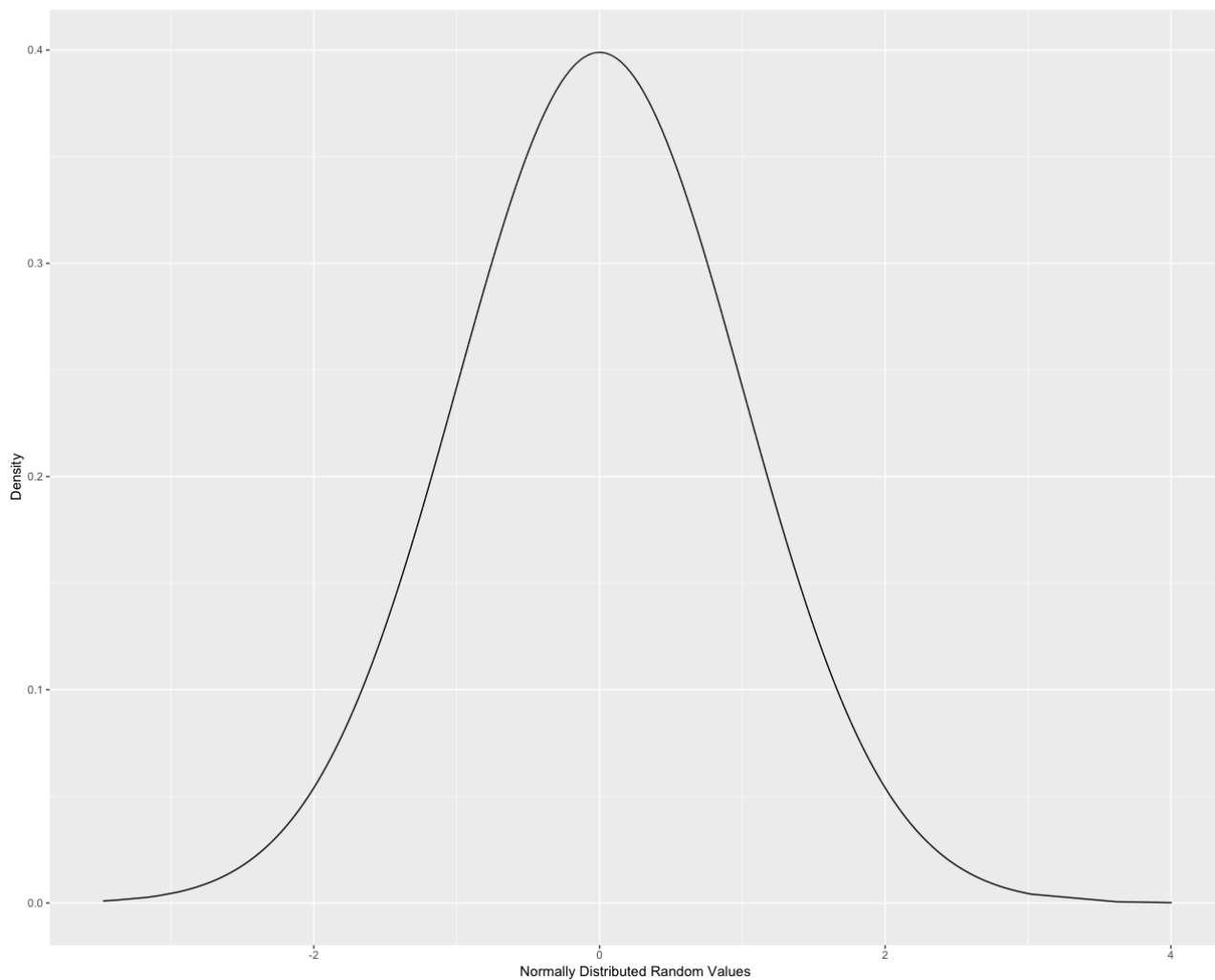
values <- rnorm(n=30000, mean=0, sd=0.85)
density <- dnorm( values )
cumulative <- pnorm( values )
quantile <- qnorm( cumulative )

ggplot(data.frame(x=values, y=density)) + aes(x=x, y=y) + geom_line() + labs(x="
Normally_Distributed_Random_Values_", y="Density")

ggplot(data.frame(x=values, y=cumulative)) + aes(x=x, y=y) + geom_line() + labs(
x="Normally_Distributed_Random_Values_", y="Cumulative_Probability")

ggplot(data.frame(x=values, y=quantile)) + aes(x=x, y=y) + geom_line() + labs(x=
"Normally_Distributed_Random_Values_", y="Quantile")
```

В езика R генерирането на нормално разпределени случайни числа става чрез функцията *rnorm*, която получава параметър за брой числа, средна стойност и стандартно отклонение. Подразбиращата се средна стойност е нула, а подразбиращото се стандартно отклонение е единица. Вероятността дадена стойност да бъде генерирана, се изчислява с функцията *dnorm*, която е полезна при изчертаването на плътностната функция (Фиг. 8.22).

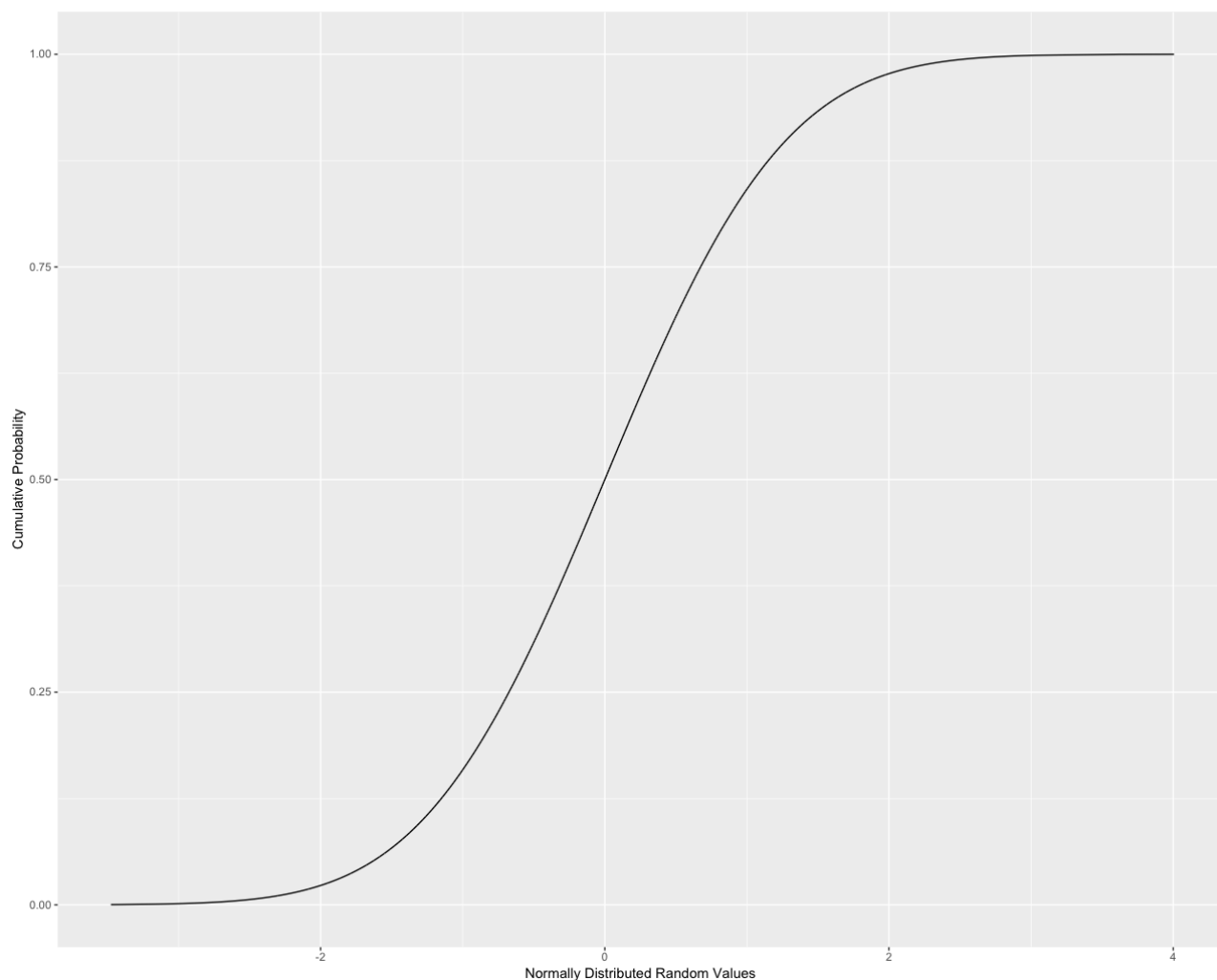


Фигура 8.22: Плътностна функция на нормално разпределение

От плътностната функция, чрез интегриране, се получава кумулативната функция на разпределението (Формула 8.2).

$$cdf(x) = \int_{-\infty}^x \frac{1}{\sigma\sqrt{2\pi}} e^{-(x-\mu)^2/2\sigma^2} dx \quad (8.2)$$

Смисълът на кумулативната функция е, че определя вероятността да се падне число по-малко от зададеното в интервала (Фиг. 8.23).



Фигура 8.23: Кумулативна функция на нормално разпределение

Обратната функция на *pnorm* е *qnorm* и тя изчислява квантила, ако е известна кумулативната вероятност.

8.3.2 Биномно разпределение

Биномното разпределение (Формула 8.3) е добре представено в R с помощта на серия функции по аналогия с функциите за нормално разпределение.

$$pdf(x) = \binom{n}{x} p^x (1-p)^{n-x} \quad (8.3)$$

Където първият множител е биномен коефициент изчисляван по Формула 8.4.

$$\binom{n}{x} = \frac{n!}{x!(n-x)!} \quad (8.4)$$

Параметърът n определя броя опити, а параметърът p задава вероятността за успех при единичен опит. Средната на биномното разпределение е np , а вариацията $np(1 - p)$. Когато параметърът n има стойност единица биномното разпределение се трансформира в Бернулиево разпределение.

Листинг 8.9: Биномно разпределение

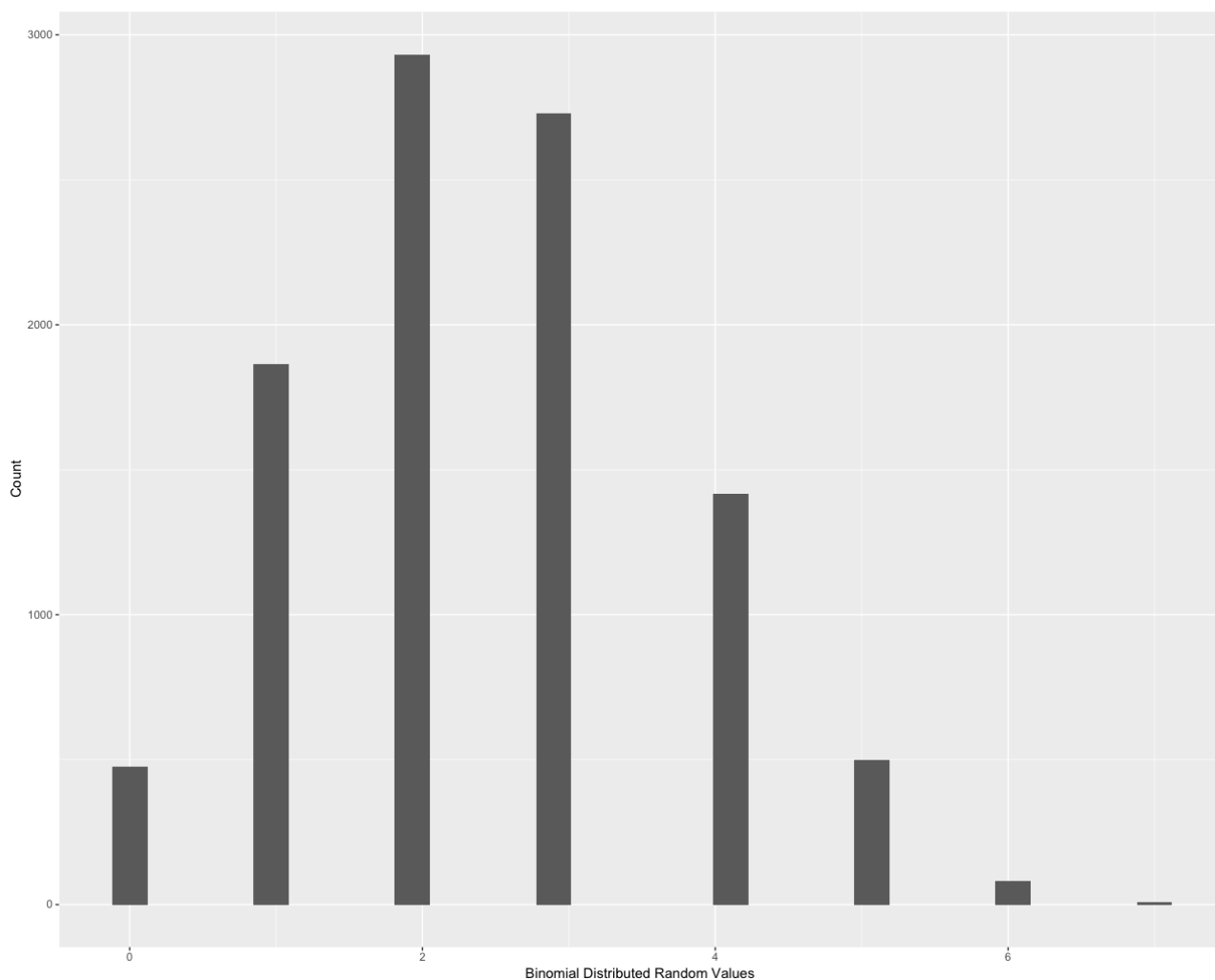
```
library(ggplot2)

values <- rbinom(n=10000, size=7, prob=0.35)
density <- dbinom(x=2, size=7, prob=0.35)
cumulative <- pbinom(q=2, size=7, prob=0.35)
quantile <- qbinom(p=0.15, size=7, prob=0.35)

ggplot(data.frame(x=values)) + aes(x=x) + geom_histogram() + labs(x="Binomial_
  Distributed_Random_Values_", y="Count")

density
cumulative
quantile
```

При биномното разпределение не просто се генерират случайни числа, а се генерират броят на успешните независими Бернулиеви експеримента. В езика R за да се генерират биномно разпределени нормални числа се използва функцията *rbinom* (Листинг 8.9). Биномното разпределение е дискретно вероятностно разпределение, тъй като отразява определен брой успешни изходи от експеримент с предварително зададена вероятност за успех. Поради тази причина визуализацията става с помощта на хистограма (Фиг. 8.24).



Фигура 8.24: Хистограма на биомно разпределение

Освен броят случайни числа които трябва да се генерират към функцията *rbinom* се подава параметър за броя експерименти и параметър за вероятността на успех при единичен експеримент. При значително увеличаване на броя експерименти биомното разпределение започва да клони към нормално разпределение.

$$cdf(x) = \sum_{i=0}^x \binom{n}{i} p^i (1-p)^{n-i} \quad (8.5)$$

С функцията *dbinom* може да се провери плътността за определена стойност (Формула 8.5), а с *rbinom* кумулативната стойност. И двете функции могат да се използват с вектор от стойности.

8.3.3 Поасоново разпределение

Популярно в практиката е също и разпределението на Поасон. То има само един параметър λ , който отразява едновременно средната стойност и дисперсията (Формула 8.6).

$$pdf(x) = \frac{\lambda^x e^{-\lambda}}{x!} \quad (8.6)$$

Разпределението е дискретно и намира приложение при случайни променливи, които отразяват случването на брой случайни събития в зададен интервал от време.

$$cdf(x) = \sum_{i=0}^x \frac{\lambda^i e^{-\lambda}}{i!} \quad (8.7)$$

Както и при повечето други вероятностни разпределения, това също започва да клони към нормалното разпределение когато параметърът λ нарасне до големи стойности.

Листинг 8.10: Разпределение на Поасон

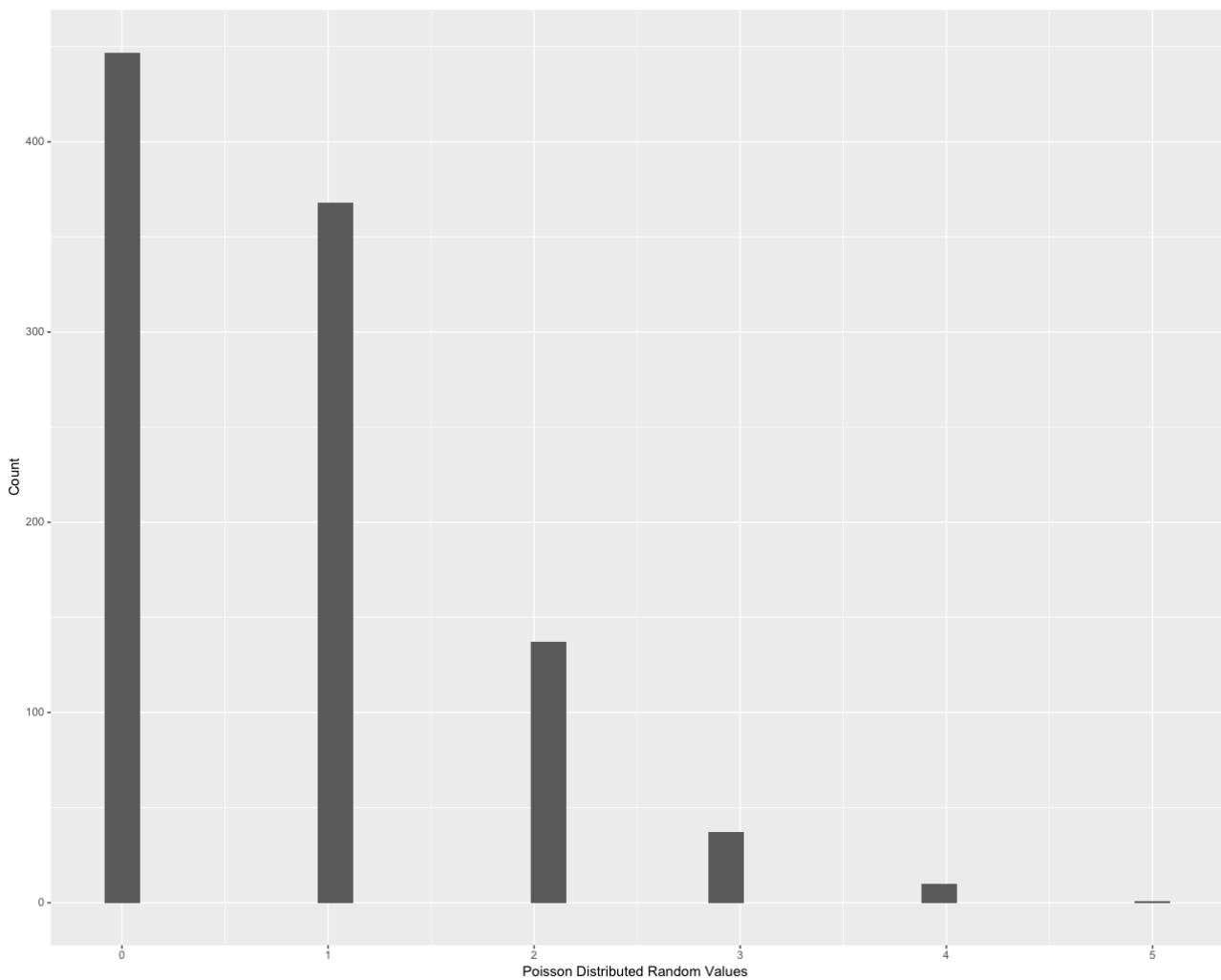
```
library(ggplot2)

values <- rpois(n=1000, lambda=0.8)
density <- dpois(x=2, lambda=0.8)
cumulative <- ppois(q=2, lambda=0.8)
quantile <- qpois(p=0.75, lambda=0.8)

ggplot(data.frame(x=values)) + aes(x=x) + geom_histogram() + labs(x="Poisson_
  Distributed_Random_Values_", y="Count")

density
cumulative
quantile
```

Визуализацията на Поасоновото разпределение става с помощта на хистограма (Фиг. 8.25).



Фигура 8.25: Хистограма на Поасоново разпределение

С функцията *dpois* може да се провери плътността за определена стойност (Формула 8.7), а с *ppois* кумулативната стойност. И двете функции могат да се използват с вектор от стойности.

8.3.4 Други разпределения

Пакетът R предлага богат набор от вероятностни разпределения (Фиг. 8.26). Една част от тези разпределения са широко използвани, други не чак толкова.

Distribution	Functions			
Beta	pbeta	qbeta	dbeta	rbeta
Binomial	pbinom	qbinom	dbinom	rbinom
Cauchy	pcauchy	qcauchy	dcauchy	rcauchy
Chi-Square	pchisq	qchisq	dchisq	rchisq
Exponential	pexp	qexp	dexp	rexp
F	pf	qf	df	rf
Gamma	pgamma	qgamma	dgamma	rgamma
Geometric	pgeom	qgeom	dgeom	rgeom
Hypergeometric	phyper	qhyper	dhyper	rhyper
Logistic	plogis	qlogis	dlogis	rlogis
Log Normal	plnorm	qlnorm	dlnorm	rlnorm
Negative Binomial	pnbinom	qnbinom	dnbinom	rnbinom
Normal	pnorm	qnorm	dnorm	rnorm
Poisson	ppois	qpois	dpois	rpois
Student t	pt	qt	dt	rt
Studentized Range	ptukey	qtukey	dtukey	rtukey
Uniform	punif	qunif	dunif	runif
Weibull	pweibull	qweibull	dweibull	rweibull
Wilcoxon Rank Sum Statistic	pwilcox	qwilcox	dwilcox	rwilcox
Wilcoxon Signed Rank Statistic	psignrank	qsignrank	dsignrank	rsignrank

Фигура 8.26: Списък с най-използваните вероятностни разпределения

Заклучение

Визуалното представяне на числените данни повишава степента за възприемане на представяната информация. Колкото по-големи са възможностите на пакетите за визуално представяне, толкова повече възможности биха имали хората, работещи с информация и нейното представяне пред широка аудитория. При анализа на случайни величини, визуализацията с хистограма и плътностна функция може да даде най-груба първоначална представа за параметрите на случайната величина. Информация, която може да бъде изключително полезна при избора на по-сложни методи за статистически анализ. От своя страна, визуализацията на случайностите най-удачно се представя, чрез някои от най-използваните вероятностни разпределения.

Глава 9

Статистическа обработка на данните

Статистическата обработка на данни се състои основно от два вида статистика – описателна статистика и сравнителна статистика. При описателната статистика се изчисляват определени параметри описващи характеристики на събраните данни, докато при сравнителната статистика се извършва сравнение между някои от описателните параметри на данните.

9.1 Описателна статистика

Параметрите при описателната статистика основно са свързани с някакво централно групиране на данните и някакво разпръскване (дисперсия) около централното групиране. Параметри за централно групиране са средната стойност, медианата и модата, а параметри за разпръскване са дисперсията и стандартното отклонение.

Листинг 9.1: Генериране на извадка от случайни числа

```
v1 <- round( rnorm(100, mean=62, sd=72) )  
  
v2 <- v1  
  
v2[sample(x=1:100, size=15, replace=FALSE)] <- NA  
  
w1 <- 1 / sample(x=1:100, size=100, replace=TRUE)
```

За да бъдат илюстрирани възможностите на R за пресмятане на описателни статистики се използва извадка от 100 нормално разпределени случайни числа (Листинг 9.1). Често в реалната практика данните съдържат липсващи измервания. При такава ситуация трябва да се вземат допълнителни мерки за пресмятане на описателните статистики. Функцията *sample* в R позволява на случаен принцип част от стойностите в определен вектор да бъдат избрани на случаен принцип, при равномерно вероятностно разпределение. Тази възможност се използва за замяна на 15% от генерираните данни с липсваща стойност (Листинг 9.1). Параметърът *replace* указва дали определено число може да бъде избрано повторно или не.

9.1.1 Средна стойност

Средната стойност е най-често използваната статистика и представлява сумата от стойностите разделена на общия брой стойности (Листинг. 9.2).

Листинг 9.2: Средна стойност

```
sum(v1) / length(v1)
mean( v1 )
mean( v2 )
mean(v2, na.rm=TRUE)
```

Когато липсват стойности при проведените измервания е невъзможно да се изчисли средната стойност без да се вземе решение как да се обработят липсващите числа. Има различни подходи за обработката на липсите, като интерполация или премахване. Независимо кой подход бъде избран, то той неизбежно води до внасяне на допълнителна грешка при пресмятанията. Ако бъде извършена интерполация, то съседните измервания биха внесли грешка в липсващите стойности. Ако бъдат премахнати липсващите измервания, то размерът на извадката намалява, а от там се увеличава и неточността на последващите пресмятания.

В някои ситуации отделните измерени стойности имат различна тежест и се налага те да участват с различен коефициент при пресмятането на средната стойност (Листинг 9.3). Такава е ситуацията когато се пресмята бал за прием в учебно заведение. Различните компоненти формиращи бал на всеки кандидат участват с различна тежест.

Листинг 9.3: Претеглена средна стойност

```
weighted.mean(x=v1, w=w1)
```

9.1.2 Минимална стойност, максимална стойност, медиана и мода

В практиката често от значение е диапазонът в който се разпростират данните. В програмния пакет R този диапазон лесно се установява с функциите `min` и `max` (Листинг 9.4).

Листинг 9.4: Минимум, максимум, медиана и мода

```
min( v1 )
max( v1 )
median( v1 )
# Mode calculation.
unique(v1)[ which.max( tabulate(match(v1, unique(v1))) ) ]
```

Един от основните недостатъци на средната стойност е, че наличието на екстремално различни отделни измервания силно може да повлияе върху общото пресмятане на средната стойност. Поради тази причина в практиката много често се използва медианата, а не средната стойност (Листинг 9.4). За да се пресметне медианата данните се сортират. При нечетен брой стойности медианата е стойността на средния елемент. При четен брой стойности медианата е средно аритметично между двата елемента в средата.

Модата е параметър, който отразява най-често срещаната стойност в множеството от данни. Освен числена стойност модата може да има и символна стойност, според характера на самите данни. В програмния пакет R няма функция, която да изчислява модата, но тя лесно може да се пресметне с комбинация от извикването на други функции (Листинг 9.4).

9.1.3 Дисперсия и стандартно отклонение

След като бъде установено някакво централно групиране в данните, от съществено значение е по какъв начин данните са разпръснати около това централно групиране. Изчисляването на дисперсията спомага за изследване на разпръскването (Листинг 9.5). Дисперсията се изчислява като сума от квадрата на разликите между всяка стойност и средната, разделена на броя стойности минус едно.

Листинг 9.5: Дисперсия и стандартно отклонение

```
sum( (v1-mean(v1))^2 ) / (length(v1) - 1)

var( v1 )

sqrt( var(v1) )

sd( v1 )
```

Основен недостатък на дисперсията е, че се изчислява с повдигане на втора степен и полученият резултат е несравним с оригиналните измервания. Примерно при серия измервания в метри резултатът от изчислението на дисперсията би имал смисъла на квадратни метри. За да бъдат сравними стойностите е достатъчно дисперсията да се подложи на корен квадратен, което коригира резултата получен с повдигане на втора степен. Резултатът от квадратния корен на дисперсията се нарича стандартно отклонение (Листинг 9.5).

9.1.4 Квантили и обобщение

Квантилите определят каква част от измерените стойности попадат в определен процент от вероятностното разпределение (Листинг 9.6).

Листинг 9.6: Квантили и обобщение

```
quantile(v1, probs=c(0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9))
# 10% 20% 30% 40% 50% 60% 70% 80% 90%
#-28.0 -0.4 19.0 41.6 66.5 84.4 110.6 121.8 150.2

summary( v1 )
```

Програмният пакет R предоставя и обобщаваща функция за описателните статистики наречена *summary* и тя визуализира стойностите за минимална, максимална, медиана, средна, първи и трети квантил.

9.2 Сравнителна статистика

Когато се работи с две или повече случайни променливи от ползва идва апарата на сравнителната статистика. Целта е да се изследва връзката между тези променливи. За променливи от едно и също измерване най-често се използват корелацията и ковариацията, а за сравнение на средни T-тест и ANOVA [15].

9.2.1 Ковариация и корелация

Корелационният коефициент е число в интервала -1 до +1 и показва възможността за наличие на линейна зависимост между две случайни променливи. Важно е да се отбележи, че корелационният коефициент не гарантира наличие на взаимна връзка, а само указва, че такава връзка е възможна.

Листинг 9.7: Ковариация и корелация

```

library(ggplot2)

cor(economics$psavert, economics$pce)

# Correlation formula.
sum((economics$psavert - mean(economics$psavert)) * (economics$pce - mean(economics$pce))) / ((nrow(economics) - 1) * sd(economics$psavert) * sd(economics$pce))

cor(economics[, c("pce", "psavert", "uempmed", "unemploy")])

cov(economics$psavert, economics$pce)

cov(economics[, c("pce", "psavert", "uempmed", "unemploy")])

# Correlation is a covariance divided by the standard deviation.
identical(cov(economics$psavert, economics$pce), cor(economics$psavert, economics$pce) * sd(economics$psavert) * sd(economics$pce))

```

В пакета `ggplot2` множеството от данни *economics* дава идеална възможност да се демонстрира концепцията за корелация и ковариация (Листинг 9.7). Колоната *psavert* отразява индивидуалното спестовно ниво, а колоната *pce* индивидуалните разходи. При относително еднакво ниво на доходите, чисто интуитивно е ясно, че който харчи повече би следвало да спестява по-малко. Това ще рече, че когато една-та стойност нараства другата стойност ще намалява и това има смисъла на отрицателна корелационна връзка. В демонстрирания пример корелационният коефициент е -0.7928546 и отразява точно факта за възможно наличие на относително силна отрицателна корелационна връзка.

$$r_{xy} = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{(n-1)s_x s_y} \quad (9.1)$$

Корелационният коефициент се пресмята като сума от разликите между множителите на отделните измервания в разлика от средната стойност, разделена на произведението на стандартните отклонения на двете променливи с броя измервания минус единица (Формула 9.1). При стойност близка до +1 е възможна силна положителна корелационна връзка, при стойност близка до -1 е възможна силна отрицателна корелационна връзка, а при стойност близка до 0 наличието на корелационна връзка е много малко вероятно.

Когато корелацията се смята между повече от две случайни променливи резултатът е корелационна матрица (Листинг 9.7). При корелационната матрица е показана възможната връзка между всяка двойка променливи.

$$v_{xy} = \frac{1}{n-1} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y}) \quad (9.2)$$

Ковариацията е подобна на корелацията, като разликата е, че стойностите не са нормирани (Формула 9.2). Може да се приеме като аналогия между дисперсията и стандартното отклонение (Листинг 9.7). Ковариацията може да бъде голямо отрицателно число или голямо положително число. В посочения пример ковариацията е със стойност -8359.069, при съответната ковариационна матрица.

Както при описателните статистики, при ковариацията и корелацията липси в измерванията възпрепятстват изчисляването на крайните стойности.

9.2.2 Тест на Стюдънт

От сравнителната статистика t -теста (или тест на Стюдънт) е един от най-разпространените подходи за сравнение на средните стойности между две извадки.

Листинг 9.8: Тестово множество за бакшиши

```
library(reshape2)

head(tips)

#   total_bill tip    sex smoker day   time size
# 1    16.99  1.01 Female    No  Sun  Dinner    2
# 2    10.34  1.66   Male    No  Sun  Dinner    3
# 3    21.01  3.50   Male    No  Sun  Dinner    3
# 4    23.68  3.31   Male    No  Sun  Dinner    2
# 5    24.59  3.61 Female    No  Sun  Dinner    4
# 6    25.29  4.71   Male    No  Sun  Dinner    4
```

За илюстриране на възможностите, които R предлага при пресмятането на t -теста е подходящо да се използва множеството данни за бакшиши (Листинг 9.8).

Тест на една извадка

При тестването на една извадка тестът се прилага за определяне на средната стойност и съответния доверителен интервал. От съществено значение е данните да са нормално разпределение което означава, че имат конкретна средна стойност и конкретно стандартно отклонение. Ако тестваната стойност попада в доверителния интервал, то може да се заключи, че това е действителната средна стойност. В противен случай не може да се приеме, че предположената средна стойност е действителната средна стойност.

Листинг 9.9: Тест на единична извадка

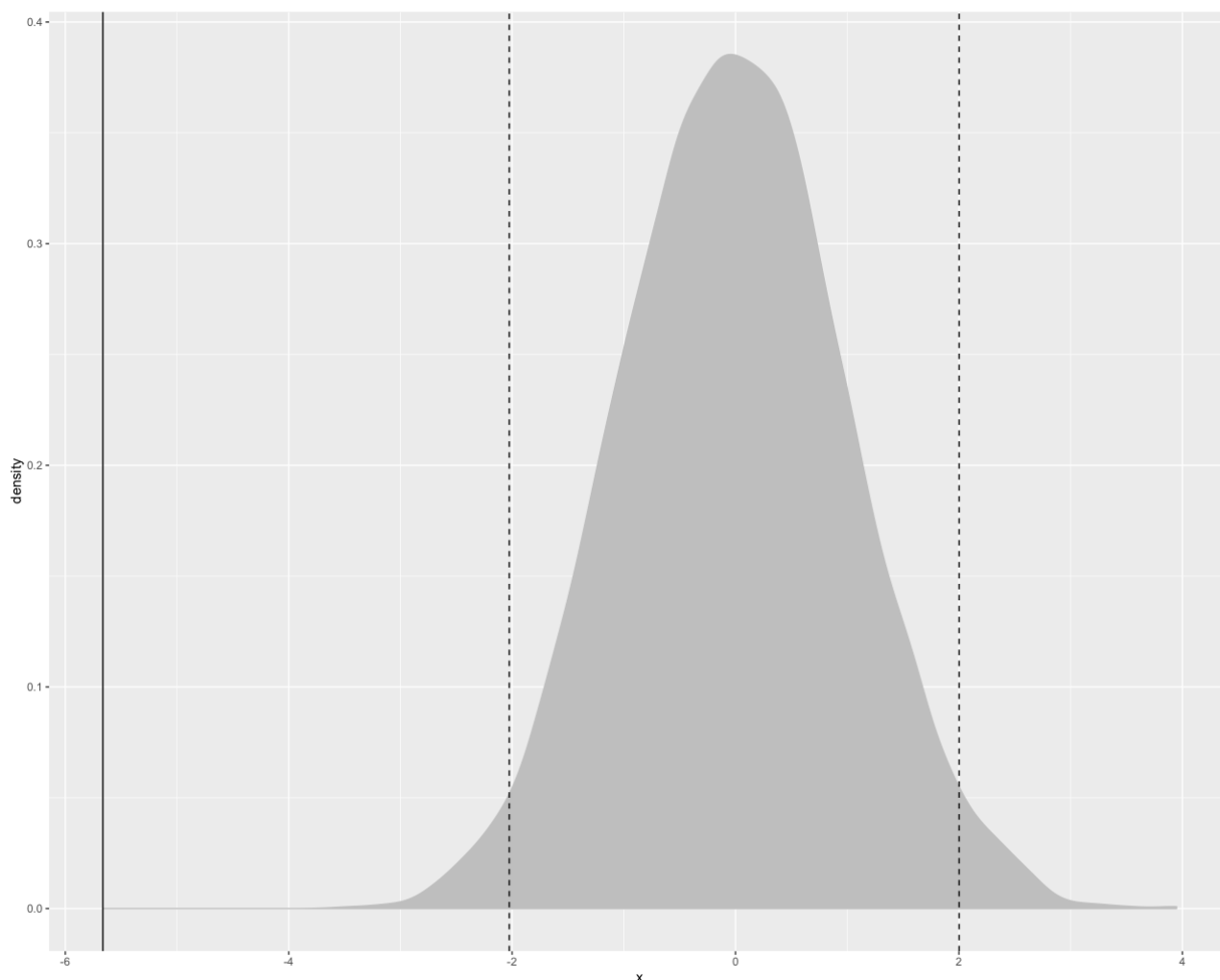
```
t.test(tips$tip, alternative="two.sided", mu=3.50)

#      One Sample t-test
#
# data:  tips$tip
# t = -5.6642, df = 243, p-value = 4.161e-08
# alternative hypothesis: true mean is not equal to 3.5
# 95 percent confidence interval:
#  2.823799 3.172758
# sample estimates:
# mean of x
#  2.998279
```

При проверката дали средната стойност на бакшишите е \$3.5 t -тестът завършва с отчет за пресметнатите стойности (Листинг 9.9), което включва - t статистиката, степените на свобода и p стойността. Също така е представена информация за 95 процентния доверителен интервал и информация за средната стойност на изследваната променлива. Получената p стойност означава, че нулевата хипотеза трябва да бъде отхвърлена. Това води до заключението, че средната стойност на бакшишите не може да е \$3.5. Нулевата хипотеза е това, което се смята за истина (в примера това е: средна стойност равна на \$3.5).

$$t = \frac{\bar{x} - \mu_0}{s_{\bar{x}}/\sqrt{n}} \quad (9.3)$$

Т-статистиката се смята по формула 9.3 и има смисъла на отношение между разликата от пресметнатото средно (\bar{x}) и предположеното средно (μ_0), а като делител е стандартната грешка на пресметнатото средно ($s_{\bar{x}}/\sqrt{n}$). В уравнението $s_{\bar{x}}$ има смисъла на стандартно отклонение, а n е броят на наблюденията.



Фигура 9.1: Т-статистика за бакшиши

Когато предположената средна стойност е вярна очакването е t -статистиката да попада в интервала от две стандартни отклонения, около средната стойност (Фиг. 9.1).

Листинг 9.10: Визуализация на t -разпределение

```
library( ggplot2 )

# Generation of t-distributed random values.
values <- rt(10000, df = NCOW( tips ) - 1)

# T-statistics calculation.
t <- t.test(tips$tip, alternative="two.sided", mu=3.50)

# Visualization.
ggplot(data.frame(x=values)) + geom_density(aes(x=x), fill="grey", color="grey")
+ geom_vline(xintercept=t$statistic) + geom_vline(xintercept=mean(values) +
```

```
c(-2, 2) * sd(values), linetype=2)
```

Изисква определено усилие за правилно интерпретиране на p -стойността. По своята същност тази стойност отразява вероятността нулевата хипотеза да е вярна. На практика измерва колко екстремна е измерената статистика (в този случай средната стойност). Ако статистиката е твърде екстремна трябва да се направи заключение, че нулевата хипотеза трябва да бъде отхвърлена. Основното усложнение с p -стойността е как да се определи кое ниво трябва да се възприеме като твърде екстремно. В практиката (емпирично) са наложени следните стойности за екстремност на p -стойността - 0.10, 0.05 и 0.01. Поради емпиричния характер за определяне на тези стойности по настоящем възникват въпроси до колко те са адекватни, спрямо съвременните тенденции в обработката на статистически данни.

Степените на свобода са втората сложна за възприемане концепция. По своята същност степените на свобода отразяват броя ефективни наблюдения в конкретен експеримент. Общата концепция за степените на свобода е броят на наблюденията минус броя параметри, които трябва да се определят. В случая на t -разпределението само един параметър трябва да се определи (стандартната грешка). В демонстрирания пример (Листинг 9.10) има 244 наблюдения и съответно $244-1 = 243$ степени на свобода.

Листинг 9.11: Едностранны t -статистика

```
t.test(tips$tip, alternative="less", mu=3.50)

#      One Sample t-test
#
# data:  tips$tip
# t = -5.6642, df = 243, p-value = 2.08e-08
# alternative hypothesis: true mean is less than 3.5
# 95 percent confidence interval:
#      -Inf 3.144535
# sample estimates:
# mean of x
# 2.998279
```

Нулевата хипотеза може да се анализира спрямо двете страни на вероятностното разпределение или да се разглежда само една от страните (по-малко или по-голямо). В този случай (Листинг 9.11) p -стойността показва, че не можем да отхвърлим нулевата хипотеза.

Тест на две извадки

Тестът на Стюдънт много по-често се прилага върху две извадки. В тестовото множество от данни за бакшишите, отделните измервания могат да се разделят по различни признаци в две групи (примерно бакшиши на мъжете и бакшиши на жените Фиг. 9.2).

Листинг 9.12: Сравнение на две извадки

```
ggplot(tips, aes(x=tip, fill=sex)) + geom_histogram(binwidth=1.0, alpha=0.8)

# Compute the variance for the two groups.
aggregate(tip ~ sex, data=tips, var)

# Test for normality.
shapiro.test(tips$tip[tips$sex == "Female"])
shapiro.test(tips$tip[tips$sex == "Male"])
shapiro.test(tips$tip)

#      Shapiro-Wilk normality test
```

```

#
# data:  tips$tip
# W = 0.89781, p-value = 8.2e-12

# Examine the equality of variances.
ansari.test(tip ~ sex, tips)

#           Ansari-Bradley test
#
# data:  tip by sex
# AB = 5582.5, p-value = 0.376
# alternative hypothesis: true ratio of scales is not equal to 1

t.test(tip ~ sex, data=tips, var.equal=TRUE)

#           Two Sample t-test
#
# data:  tip by sex
# t = -1.3879, df = 242, p-value = 0.1665
# alternative hypothesis: true difference in means is not equal to 0
# 95 percent confidence interval:
#  -0.6197558  0.1074167
# sample estimates:
# mean in group Female      mean in group Male
#           2.833448           3.089618

library(plyr)

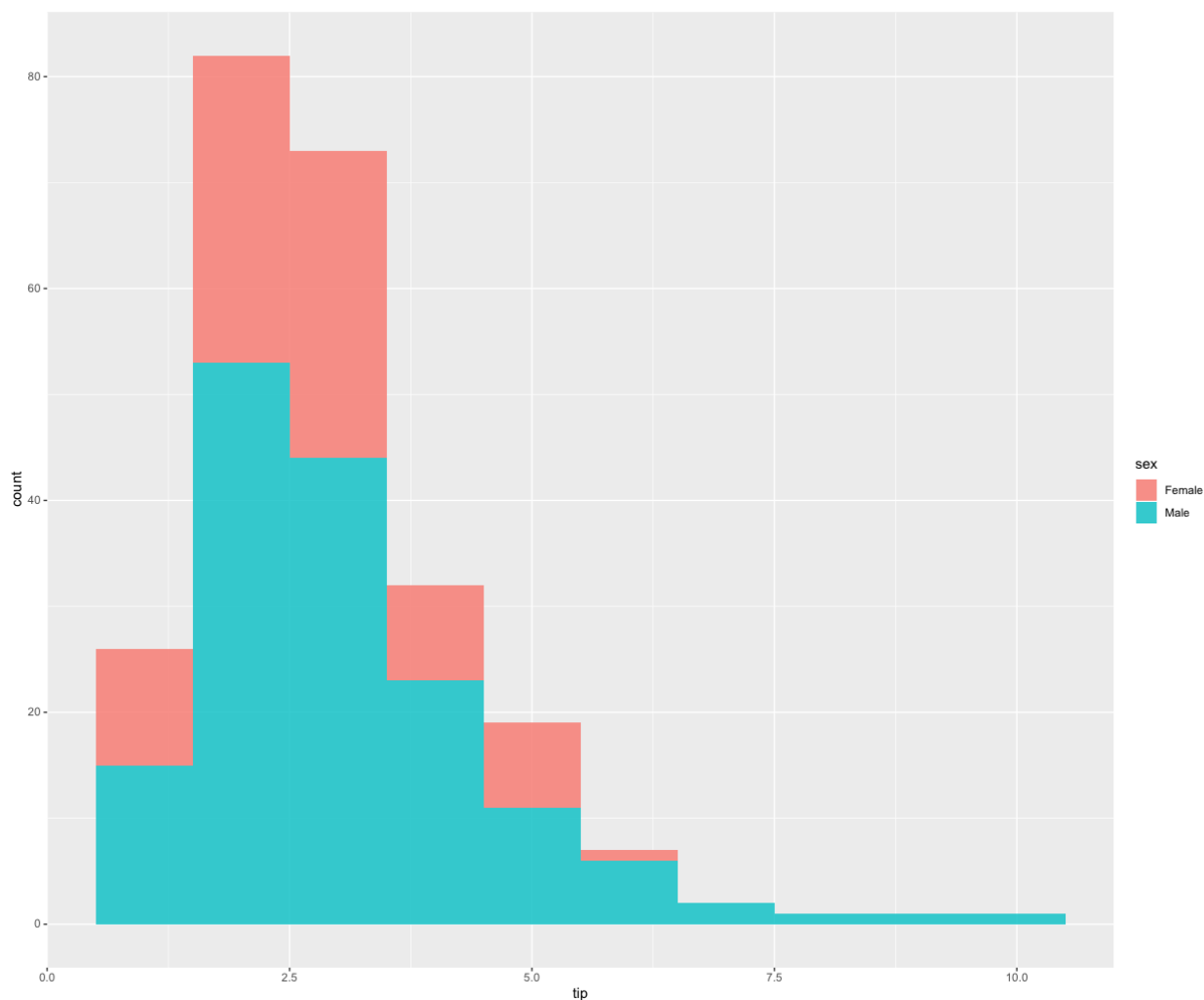
# Checking of the means in two standard deviations range.
ddply(tips, "sex", summarize, tip.mean=mean(tip), tip.sd=sd(tip), Lower=tip.mean
      -2*tip.sd/sqrt(NROW(tip)), Upper=tip.mean+2*tip.sd/sqrt(NROW(tip)))

#           sex tip.mean  tip.sd    Lower    Upper
# 1 Female  2.833448  1.159495  2.584827  3.082070
# 2 Male   3.089618  1.489102  2.851931  3.327304

ggplot(ddply(tips, "sex", summarize, tip.mean=mean(tip), tip.sd=sd(tip), Lower=
  tip.mean-2*tip.sd/sqrt(NROW(tip)), Upper=tip.mean+2*tip.sd/sqrt(NROW(tip))),
  aes(x=tip.mean, y=sex)) + geom_point() + geom_errorbarh(aes(xmin=Lower, xmax=
  Upper), height=.2)

```

За да бъде надежден традиционният t -тест, от съществено значение е дисперсията на двете извадки да бъде идентична. В ситуации където дисперсиите не са идентични се използва t -тест на Welch за две извадки.



Фигура 9.2: Разпределение на бакшишите според пола на сервитьора

На Фиг. 9.2 и при теста на Shapiro-Wilk (Листинг 9.12) ясно се вижда, че двете извадки не са нормално разпределени. Тъй като данните очевидно не са нормално разпределени за проверката на идентичността на дисперсиите се прилага тест на Ansari-Bradley. Резултатът показва, че дисперсиите са достатъчно сходни и може да се приложи *t*-тест. От получения резултат (Листинг 9.12) не може да се заключи, че нивото на бакшишите между мъжете и жените се различава достатъчно съществено.

Сдвоен тест на две извадки

В някои ситуации се налага изследването на две извадки, но на сдвоени измервания. Това е ситуация, при която всяко отделно измерване се отнася до два взаимно свързани субекта (примерно близнаци третирани с определен медикамент или измерване на бащи и синове). В такива ситуации трябва да се прилага сдвоен тест и това в пакета R се постига чрез флаг *paired = TRUE* на функцията *t.test*. За илюстриране на работата със сдвоени данни може да се използва тестовото множество от данни за височините на бащите и синовете (Листинг 9.13).

Листинг 9.13: Т-тест на сдвоени данни

```
library( UsingR )
```



```
head( father.son )

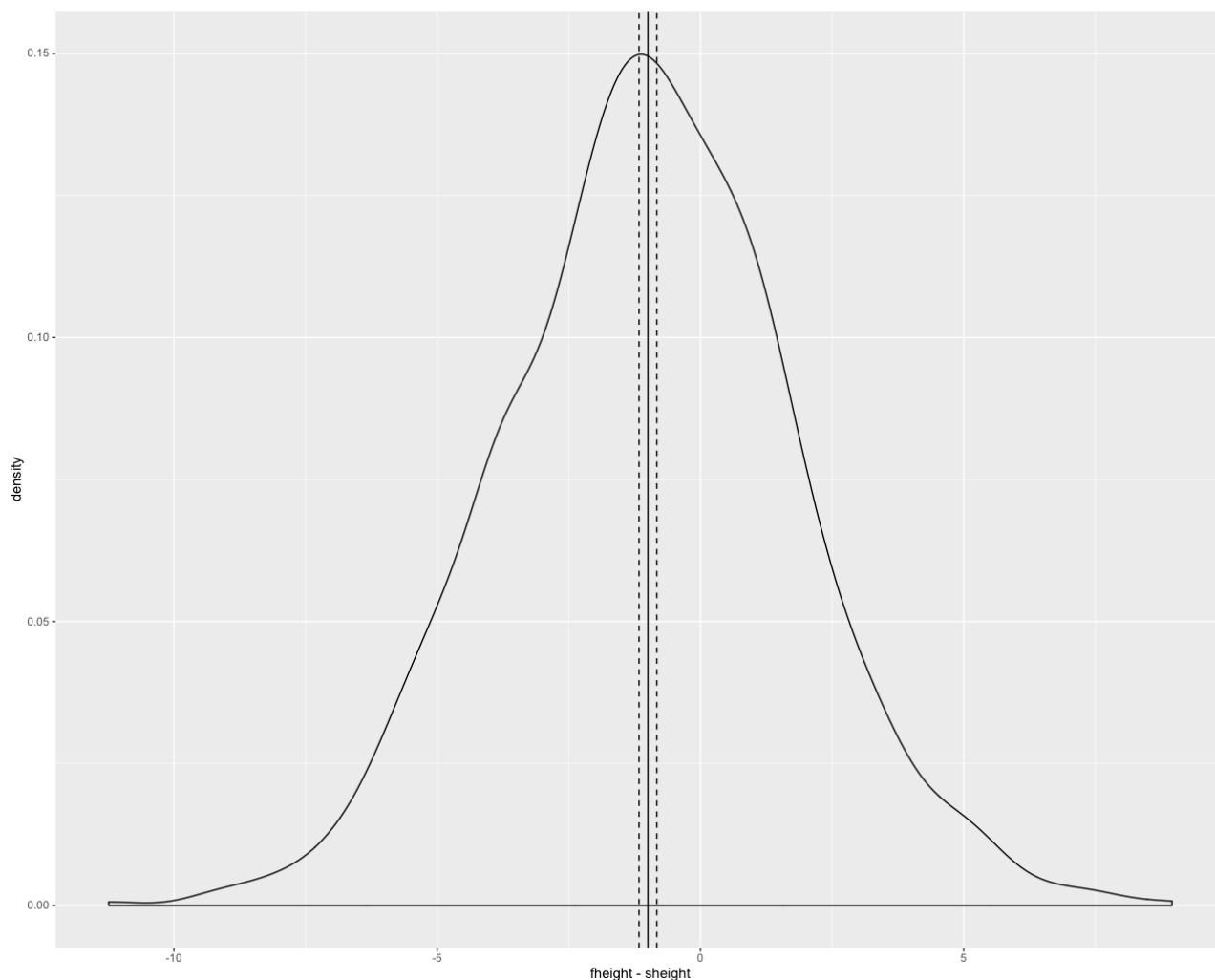
#      fheight  sheight
# 1 65.04851 59.77827
# 2 63.25094 63.21404
# 3 64.95532 63.34242
# 4 65.75250 62.79238
# 5 61.13723 64.28113
# 6 63.02254 64.24221

t.test(father.son$sheight , father.son$fheight , paired=TRUE)

#      Paired t-test
#
# data:  father.son$sheight and father.son$fheight
# t = 11.789, df = 1077, p-value < 2.2e-16
# alternative hypothesis: true difference in means is not equal to 0
# 95 percent confidence interval:
#  0.8310296 1.1629160
# sample estimates:
# mean of the differences
#                0.9969728

# Visualization of the means.
ggplot(father.son, aes(x=fheight-sheight))+geom_density()+geom_vline(xintercept=
  mean(father.son$fheight-father.son$sheight))+geom_vline(xintercept=mean(
  father.son$fheight-father.son$sheight)+2*c(-1, 1)*sd(father.son$fheight-
  father.son$sheight)/sqrt(nrow(father.son)), linetype=2)
```

Установено е, че ръстът на човека е нормално разпределена величина, така че не е нужно да се прави проверка за нормалност и идентичност на дисперсиите на данните преди да се приложи *t*-тестът.



Фигура 9.3: Визуализация на средните при сдвоения тест

При настоящия пример тестът показва, че нулевата хипотеза трябва да бъде отхвърлена и трябва да се приеме, че поне за тази извадка от данни бащите и синовете имат различна височина (Фиг. 9.3).

9.2.3 Дисперсионен анализ

Логичното развитие след сравняването на две групи от данни е сравняването на много групи от данни. Най-разпространеният начин за такова сравнение е ANOVA (analysis of variance), който се описва с Формула 9.4.

$$F = \frac{\sum_i n_i (\bar{Y}_i - \bar{Y})^2 / (K - 1)}{\sum_{ij} (Y_{ij} - \bar{Y}_i)^2 / (N - K)} \quad (9.4)$$

Където n_i е броят наблюдения в група i , \bar{Y}_i е средната стойност на група i , \bar{Y} е общата средна, Y_{ij} е j наблюдение в група i , N е общият брой наблюдения и K е броят на изследваните групи.

Листинг 9.14: ANOVA тест

```

library( plyr )
library( reshape2 )

aov(tip~day-1, tips)

# Call:
# aov(formula = tip ~ day - 1, data = tips)
#
# Terms:
#
#              day Residuals
# Sum of Squares 2203.0066  455.6866
# Deg. of Freedom      4      240
#
# Residual standard error: 1.377931
# Estimated effects are balanced

aov(tip~day-1,tips)$coefficients

#   dayFri   daySat   daySun   dayThur
# 2.734737 2.993103 3.255132 2.771452

summary( aov(tip~day-1,tips) )

#              Df Sum Sq Mean Sq F value Pr(>F)
# day              4 2203.0    550.8    290.1 <2e-16 ***
# Residuals    240   455.7       1.9
# ---
# Signif. Codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

ddply(tips, "day", plyr::summarize, tip.mean=mean(tip), tip.sd=sd(tip), Length=
  NROW(tip), tfrac=qt(p=.90, df=Length-1), Lower=tip.mean - tfrac*tip.sd/sqrt(
    Length), Upper=tip.mean + tfrac*tip.sd/sqrt(Length))

#   day tip.mean   tip.sd Length   tfrac   Lower   Upper
# 1 Fri 2.734737 1.019577    19 1.330391 2.423549 3.045925
# 2 Sat 2.993103 1.631014    87 1.291473 2.767272 3.218934
# 3 Sun 3.255132 1.234880    76 1.292941 3.071986 3.438277
# 4 Thur 2.771452 1.240223    62 1.295585 2.567386 2.975517

summary( lm(tip~day-1,data=tips) )

# Call:
# lm(formula = tip ~ day - 1, data = tips)
#
# Residuals:
#      Min       1Q   Median       3Q      Max
# -2.2451 -0.9931 -0.2347  0.5382  7.0069
#
# Coefficients:
#              Estimate Std. Error t value Pr(>|t|)
# dayFri          2.7347      0.3161   8.651 7.46e-16 ***
# daySat          2.9931      0.1477  20.261 < 2e-16 ***
# daySun          3.2551      0.1581  20.594 < 2e-16 ***
# dayThur         2.7715      0.1750  15.837 < 2e-16 ***

```

```
# ———
# Signif. Codes:  0  ***  0.001  **  0.01  *  0.05  .  0.1  1
#
# Residual standard error: 1.378 on 240 degrees of freedom
# Multiple R-squared:  0.8286, Adjusted R-squared:  0.8257
# F-statistic: 290.1 on 4 and 240 DF, p-value: < 2.2e-16
```

В програмния пакет R функцията *aov* служи за изчислението на ANOVA теста (Листинг 9.14). Функцията използва синтаксиса за запис на формули, като от лявата страна стои анализираната променлива, а от дясната страна стои променливата, използвана за групиране.

За илюстриране на възможностите при анализа на дисперсиите данните от множеството за бакшиши може да се разделят в четири групи, според различните дни в които ресторантът работи. Наличието на минус единица в дясната страна на формулата оказва, че няма да се изчислява параметър за отрез.

ANOVA тестът показва дали има значимо различие в някоя от групите, но не показва коя от групите води до това различие. За да бъде установена тази разлика може да се изчисли *p*-стойността. Резултатите от ANOVA теста могат да се потвърдят с множество *t* тестове на отделните променливи групирани по двойки.

Алтернатива на ANOVA теста може да бъде линейната регресия с една определяща променлива и без изчисление на среза.

9.3 Линейна регресия

Един от най-използваните инструменти в статистическия анализ е линейната регресия. В най-простия си вариант тя се използва за определяне на взаимовръзката между две случайни променливи. Това означава, че ако се знае стойността на едната променлива с помощта на линейната регресия може да се предполага каква би била очакваната стойност на другата променлива. Прогнозираната променлива се нарича „отговор“, а променливата която служи за показател „предиктор“.

$$y = ax + b + \epsilon \quad (9.5)$$

Основната идея при простата линейна регресия е, че прогнозираната стойност е в линейна функционална зависимост от стойността на предиктора (Формула 9.5). Където ϵ има смисъла на нормално разпределена грешка със средна стойност 0.

$$a = \frac{\sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})}{\sum_{i=1}^n (x_i - \bar{x})^2} \quad (9.6)$$

Коефициентът *a* има смисъла на наклон на правата и се изчислява по Формула 9.6.

$$b = \bar{y} - a \quad (9.7)$$

Коефициентът *b* има смисъла на срез на правата по оста *y* за стойност 0 по оста *x* и се изчислява по Формула 9.7.

Листинг 9.15: Линейна регресия

```
library(ggplot2)
library(UsingR)
```

```

head(father.son)

#      fheight  sheight
# 1 65.04851 59.77827
# 2 63.25094 63.21404
# 3 64.95532 63.34242
# 4 65.75250 62.79238
# 5 61.13723 64.28113
# 6 63.02254 64.24221

ggplot(father.son, aes(x=fheight, y=sheight)) + geom_point() + geom_smooth(
  method="lm") + labs(x="Fathers", y="Sons")

lm(sheight~fheight, data=father.son)

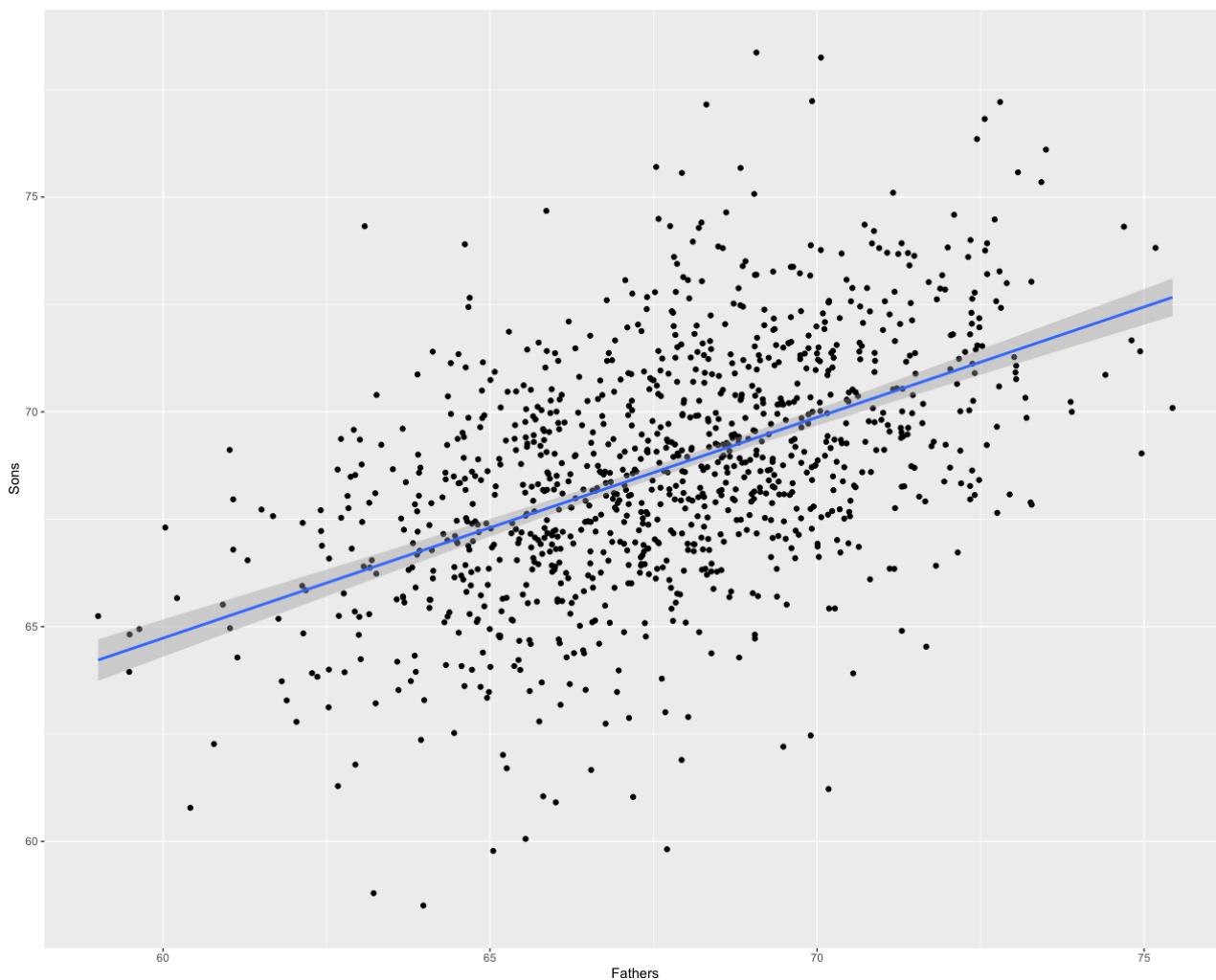
# Call:
# lm(formula = sheight ~ fheight, data = father.son)
#
# Coefficients:
# (Intercept)      fheight
#      33.8866      0.5141

summary(lm(sheight~fheight, data=father.son))

# Call:
# lm(formula = sheight ~ fheight, data = father.son)
#
# Residuals:
#      Min       1Q   Median       3Q      Max
# -8.8772 -1.5144 -0.0079  1.6285  8.9685
#
# Coefficients:
#              Estimate Std. Error t value Pr(>|t|)
# (Intercept)  33.88660    1.83235   18.49  <2e-16 ***
# fheight      0.51409    0.02705   19.01  <2e-16 ***
# ---
# Signif. Codes:  0   ***  0.001  **  0.01  *  0.05  .  0.1    1
#
# Residual standard error: 2.437 on 1076 degrees of freedom
# Multiple R-squared:  0.2513, Adjusted R-squared:  0.2506
# F-statistic: 361.2 on 1 and 1076 DF, p-value: < 2.2e-16

```

За демонстрация на линейна регресия е удачно да се използва множеството данни за височината на бащите и синовете (Листинг 9.15). Височината на бащата се използва като предиктор, а височината на сина като отговор (Фиг. 9.4).



Фигура 9.4: Линейна регресия бащи-синове

Синята линия, която преминава през точките е правата, изчислена по формулите за линейна регресия. Сивият регион около синята линия показва нивото на доверие, което може да се припише на прогнозата. Изобразяването на данните дава представа за възможностите на линейната регресия, но не показва самото изчисляване на коефициентите.

В програмния пакет R изчисляването на тези коефициенти става с функцията *lm*. За пореден път е видно приложението на записа от тип формула. От лявата страна е променливата отговор, а от дясната страна променливата предиктор. В резултат от пресмятането се изчислява наклонът на правата и срезът с ординатната ос. Изчислението на двата параметъра за линейна регресия нямат особено голям смисъл, ако не се отчете стандартната грешка. Тя дава оценка за достоверността на прогнозите. Обобщената информация получена с функцията *summary* включва стандартната грешка, *t*-статистика, *p*-стойности на коефициентите, степени на свобода и *F*-статистика.

Заклучение

Без значение дали трябва да се пресметнат описателни статистики или да се направи сравнителен анализ, програмният пакет R разполага с пълен набор от функции за извършването на тези изчисления. Най-

популярните пресмятания за средна стойност, медиана, стандартно отклонение, корелация, ковариация, дисперсионен анализ, тест на Стюдънт и линейна регресия.

Глава 10

Приближени пресмятания - подходи, методи, алгоритми

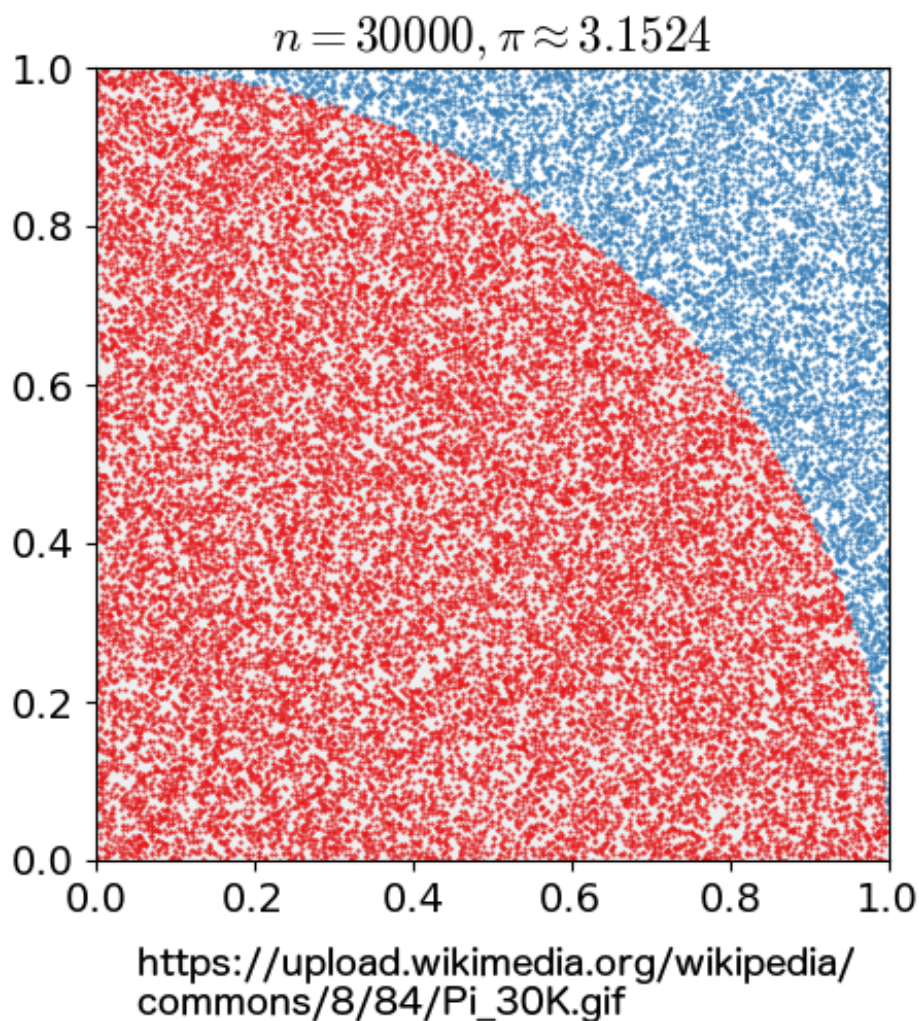
В изчислителната математика се открояват две основни направления – точните числени пресмятания и приближените числени пресмятания. Точни числени пресмятания се извършват в ситуации в които обемът на задачата позволява изчисленията да се осъществят в приемлив интервал от време. В реалната практика често поставяните задачи значително нарастват като обем и тяхното пресмятане с точни числени методи става неприемливо по отношение на нужното време за пресмятане. В такива ситуации се прибягва до множеството разработени методи за приближени пресмятания. По отношение на програмния продукт R, ще бъдат разгледани някои от най-популярните методи за приближени числени пресмятания, а именно Монте-Карло симулации [22], генетични алгоритми [21] и изкуствени невронни мрежи [20].

10.1 Монте-Карло методи

В средата на XX век, във връзка с разработването на първите ядрени оръжия, са разработени група методи за приближени пресмятания, които залагат на способ за генериране на голямо количество случайни числа и последващата им статистическа обработка. Монте-Карло методите намират най-голямо приложение в оптимизационни задачи, числено интегриране и генериране на семпли за специфични вероятностни разпределения. При повечето програмни езици вградените възможности за генериране на псевдослучайни числа се ограничават до равномерно вероятностно разпределение. В такива ситуации за да се получат псевдо случайни числа с вероятностно разпределение различно от равномерното е нужно да се направят допълнителни пресмятания. Монте-Карло методите могат да се използват за решаването на всяка задача, която има представяне в термините на вероятности и статистика.

Има вариации в реализацията на Монте-Карло методите, но те в общия случай следват няколко добре дефинирани стъпки:

1. Определяне на област от допустими стойности;
2. Генериране на извадка от случайни числа в предварително дефинираната област;
3. Извършване на точни пресмятания с така генерираните случайни числа;
4. Обобщаване на резултатите от извършените пресмятания.

Фигура 10.1: Пресмятане на числото π с Монте-Карло метод

Един от най-популярните примери за Монте-Карло пресмятане е приближеното изчисление на числото π (Фиг. 10.1). За тази цел една четвърт от окръжност се представя с обгръщащия я квадрат. Съотношението между площите на квадрата и на четвъртината окръжност е $\pi/4$. За да се апроксимира стойността на числото π с Монте-Карло метод се изпълняват следните стъпки:

1. Изчертаване на квадрат и четвъртина окръжност вписана в него;
2. Генериране на случайни равномерно разпределени числа, като координати (x,y двойки) в описаната допустима област;
3. Изброяване на точките с координати x,y които са на разстояние 1 от центъра на четвърт окръжността;
4. Съотношението между точките в четвърт окръжността и общия брой точки е $\pi/4$, което умножено по 4 дава приближена стойност за числото π .

При тази процедура за приближено пресмятане е важно да се вземат под внимание два много важни фактора:

1. Ако генерираните случайни числа не са равномерно разпределени, това ще доведе до невярна апроксимация за търсената стойност;

2. Генерирането на малък брой координати за точки в дефиниционната област води до ниска надеждност на резултата от апроксимацията. Колкото по-голям е броят на генерираните случайни числа, толкова по-надеждни са резултатите от апроксимацията.

Листинг 10.1: Монте-Карло пресмятане

```
library( MonteCarlo )

mode <- function(x) {
  values <- unique(x)
  return( values[which.max(tabulate(match(x, values)))] )
}

experiment <- function(n, d){
  x <- sample(6,n,TRUE)

  for(i in 2:d) {
    x <- x + sample(6,n,TRUE)
  }

  return( list("mean"=mean(x), "median"=median(x), "mode"=mode(x)) )
}

result <- MonteCarlo(func=experiment, nrep=1000, param_list=list("n"=c(10, 50,
  100, 150, 200), "d"=c(1,2,6)))

summary( result );

MakeTable(output=result, rows=c("d"), cols=c("n","list"), digits=2, include_meta
=FALSE)
```

Към програмния продукт R е създаден отделен пакет (автор Christian Hendrik Leschinski) за извършване на Монте-Карло пресмятания, наречен „MonteCarlo“ (Листинг 10.1).

За демонстрация на възможностите, които R дава при Монте-Карло симулации е представено сравнение между средната, медианата и модата за вероятностно разпределение на сумата от n зара. Тъй като R не предлага функция за пресмятане на мода, е необходимо тази функция да бъде предварително дефинирана (Листинг 10.1).

В пакета *MonteCarlo* основно се използват две функции - *MonteCarlo* и *MakeTable*. Функцията *MonteCarlo* има за основна задача генерирането на множеството експерименти в процеса на симулацията. Най-важният параметър за тази функция е функционален обект, който описва единичен експеримент. В програмния език R функционалните обекти по своята същност представляват потребителски дефинирани функции. В предложения пример функционалният обект се нарича *experiment*, а функцията която представлява получава два входни параметъра n и d . Потребителят на пакета *MonteCarlo* сам може да избира какъв брой параметри да има функцията за единичен експеримент. В настоящия пример n определя броя случайни числа, които да бъдат генерирани (размер на случайната извадка), а d определя колко зара ще участват във формирането на крайния резултат. При предишни примери бе показано, че вероятностното разпределение на един зар е равномерно, на два зара триъгълно, а при достатъчно много зарове разпределението клони към нормалното. В този пример се разглеждат разликите между средната, медианата и модата за 1, 2 и 6 зара.

Към функцията за единичен експеримент има следните изисквания: 1. Аргументите да бъдат скаларни стойности; 2. Върнатата стойност от функцията да бъде списък с именувани или неименувани

скаларни стойности. Потребителската функция за единичен експеримент се изпълнява в текущото работно пространство и поради тази причина всички нужни библиотеки, променливи с данни и помощни функции трябва да са заредени предварително.

Вторият важен аргумент на функцията *MonteCarlo* е *paramlist* и трябва да изпълнява следните изисквания: 1. Трябва да е списъчна структура; 2. Елементите на списъка трябва да са именувани и имената да съответстват на имената използвани за параметри в потребителската функция за единичен експеримент; 3. Всеки елемент в списъка е вектор със скаларни стойности; 4. Списъкът съдържа точно толкова на брой елементи, колкото са параметрите на потребителската функция за единичен експеримент.

Последният задължителен аргумент на функцията *MonteCarlo* е *nrep* и той определя колко пъти ще бъде изпълнен Монте-Карло експериментът. В представения пример (Листинг 10.1) се изпълняват хиляда повторения за три възможни комбинации от зарове, при пет различни броя за хвърлянето на тези зарове, а именно 10, 50, 100, 150 и 200.

list d/n	mean					median					mode				
	10	50	100	150	200	10	50	100	150	200	10	50	100	150	200
1	10.50	10.48	10.49	10.50	10.50	10.52	10.50	10.48	10.50	10.52	10.48	10.50	10.42	10.49	10.53
2	7.01	7.00	7.00	7.01	6.99	7.02	7.01	7.00	7.00	7.00	7.03	7.04	6.98	6.98	6.99
6	20.94	21.03	21.00	21.01	21.00	20.90	21.01	20.99	21.00	21.00	20.86	20.98	21.04	21.00	20.97

Таблица 10.1: Сравнение на средна, медиана и мода за експеримент със зарове

За визуализация на получените резултати от функцията *MonteCarlo* се използва функцията *MakeTable*. На функцията *MakeTable* се подава резултатът от симулацията и тя генерира таблица с резултати в *LaTeX* формат (Таб. 10.1).

Функцията *MakeTable* дава множество възможности, но само три от аргументите ѝ са задължителни. На аргумента *output* се присвоява резултатът от изпълнението на функцията *MonteCarlo*. Вторият и третият аргумент са *rows* и *cols*, като те определят по какъв начин ще се организират табличните данни. В представения пример (Листинг 10.1) по редове са организирани броят зарове, участващи в единичен експеримент, а по колони са организирани броят хвърляния на заровете, групирани по вида статистика (средна, медиана или мода).

Макар и незадължителен параметърът *digits* е определен на 2, което дава по-добра прегледност на табличната визуализация. Също незадължителен параметърът *include_metal* е установен на „лъжа“ с цел да не се генерират коментари с обобщаваща информация за извършената симулация.

10.2 Генетични алгоритми

Генетичните алгоритми са евристика за глобална оптимизация, която е вдъхновена от идеите за естествената природна еволюция. Генетичните алгоритми спадат към по-голям клас оптимизационни евристики наречени „еволюционни алгоритми“. Основното си приложение генетичните алгоритми намират в оптимизационни задачи в големи пространства. По своята същност генетичните алгоритми са вероятностни и генерират субоптимални решения, като относително рядко достигат до глобалния оптимум. Изчисленията при генетичните алгоритми са организирани в три основни операции (рекомбинация) – селекция, кръстосване и мутация. Пресмятанятията най-често започват от група случайно генерирани решения, които съставят първоначалната популация. Целта е в процеса на еволюция решенията системно да се подобряват. Популацията условно се разделя на старо и ново поколение. В новото поколение влизат решенията генерирани след рекомбинацията. За всяко решение в популацията се определя стойност на жизнеспособност. В общия случай стойността на жизнеспособност е резултатът от пресмятането на функцията,

която подлежи на оптимизация. Най-жизнеспособните решения биват подбрани да участват в създаването на новото поколение. Създаването на нови поколения е итеративен процес и най-често той приключва след изтичането на определен брой поколения или достигане на задоволително ниво за оптималност на предложеното решение.

Генетичните алгоритми представят информацията под формата на група от решения, организирани в популация. Всяко решение представлява вектор от стойности в пространството на решенията. Кодирането на задачата в термините на генетичните алгоритми е строго специфично за всяка задача. При целочислени задачи стойностите във вектора са цели числа. При пресмятане на непрекъснати задачи стойностите на вектора са реални числа. При някои комбинаторни задачи кодирането е под формата на пермутации. В повечето случаи дължината на вектора е фиксирана, но това не е задължително условие. Примерно при кодиране на серия от инструкции за конкретна машина, дължината на серията може да варира. Най-често началната популация се генерира на случаен принцип, но това не е задължително, особено ако се налага оптимизацията да продължи от вече постигнати резултати. Съществен е въпросът за размера на популацията и в практиката се е наложило този размер да се определя експериментално. При селекцията е от значение най-жизнеспособните решения да имат най-голям шанс за възпроизвеждане, но в същото време е важно и по-слабите решения да има шанс за участие в следващото поколение.

Оценката на жизнеността най-често се постига, чрез подаване на решението към целевата функция. Целевата функция е строго специфична за всяка различна задача, която се решава. Желателно е целевата функция да връща единствена стойност. Ако при оптимизационната задача има повече критерии за оптимизиране, то многокритериалната задача трябва да се сведе до еднокритериална. След избора на два (или повече) родители, операцията по кръстосване разменя части от векторите. С помощта на кръстосването се изследват обширни региони от пространството на решенията. При мутацията на случаен принцип в новото поколение се избират отделни елементи от вектора и те се модифицират. Мутацията спомага за изследването на близки околности във вече генерираните точки в пространството на решенията.

Използването на генетични алгоритми става неефективно в ситуации, в които целевата функция изисква неприемливо дълго време за пресмятане. Тъй като генетичните алгоритми ползват генерирането на голямо количество междинни решения, множеството пресмятания на целевата функция може да направи цялата оптимизация неприемливо бавна. Важно е също да се знае, че генетичните алгоритми не гарантират намирането на глобалния оптимум, най-вече когато този оптимум не е предварително известен. Генетичните алгоритми не са ефективни при задачи където, целевата функция води само до две състояния „добро“ или „лошо“. За да бъде ефективен процесът по оптимизацията, решенията в популацията трябва да подлежат на подреждане, спрямо тяхната жизнеспособност.

Предмет	Ценност	Тегло
cake	10	1
ice cream	15	10
orange juice	10	5
strawberries	30	7
grape	30	1
candies	20	5
chocolate	2	1

Таблица 10.2: Предмети с определена ценност и тегло

Възможностите за оптимизация с генетични алгоритми добре може да се илюстрира с една от най-популярните комбинаторни задачи, която се нарича „задача на раницата“. Всяка раница има ограничена вместимост и максимално тегло, което човекът може да носи. В същото време има множество предмети, които биха били полезни, ако са налични при едно пътуване. Всеки предмет има своя специфична ценност, която е важна за притежателя му, но също така има и специфично тегло. Целта на оптимизацията е да се вземат тези предмети на които сумарното тегло не надвишава лимита, но и носят максимална сумарна ценност. Тази задача е от областта на целочислената оптимизация и дали един предмет попада в групата на взетите може да се отбележи с единица, а ако не попада в тази група с нула. Макар и да изглежда проста

задачата за раницата е много трудно решима, особено когато става въпрос за много на брой предмети и силно ограничено сумарно тегло, какъвто е случаят с изстрелването на летателни апарати в космоса. В представения пример са изброени група храни и напитки, които човек би избрал за разходка в планината (Таб. 10.2).

Листинг 10.2: Оптимизация на задачата за раницата с генетични алгоритми

```
library(genalg)

data <- data.frame(item = c("cake", "ice_cream", "orange_juice", "strawberries",
  "grape", "candies", "chocolate"), points = c(10, 15, 10, 30, 30, 20, 2),
  weight = c(1, 10, 5, 7, 1, 5, 1))

limit <- 20

solution <- c(1, 0, 0, 1, 1, 0, 0)

data[solution==1, ]
#           item           points weight
# 1         cake             10      1
# 4 strawberries            30      7
# 5          grape            30      1

cat(solution %*% data$points)
# 70

# Fitness function calculation.
fitness <- function(x) {
  points <- x %*% data$points

  weight <- x %*% data$weight

  if (weight > limit) {
    return( 0 )
  } else {
    return( -points )
  }
}

iterations <- 75

model <- rbga.bin(size=7, popSize=37, iters=iterations, mutationChance=0.01,
  elitism=TRUE, evalFunc=fitness)

cat( summary(model) )
# GA Settings
#   Type                = binary chromosome
#   Population size      = 37
#   Number of Generations = 75
#   Elitism              = TRUE
#   Mutation Chance      = 0.01
#
# Search Domain
#   Var 1 = [,]
#   Var 0 = [,]
```

```

#
# GA Results
# Best Solution : 1 0 1 1 1 1 1

# Print the best found solution.
best <- c(1, 0, 1, 1, 1, 1, 1)
data[best == 1, ]
#           item           points weight
# 1         cake             10       1
# 3 orange juice             10       5
# 4 strawberries             30       7
# 5         grape             30       1
# 6        candies             20       5
# 7    chocolate              2       1

# Calculate survival points.
cat(paste(best %*% data$points, "/", sum(data$points)))
# 102 / 117

```

За използването на генетични алгоритми в R, една от възможностите е пакетът *genalg* (Листинг 10.2). Основната функция в този пакет за работа с бинарни вектори на решенията е *rbga.bin* и като резултат от извикването ѝ се получава обект с модела на извършената оптимизация. Параметърът *size* определя каква е размерността на пространството. В разглеждания пример са налични 7 обекта и поради тази причина векторът, описващ едно решение има 7 компонента. Параметърът *popSize* определя какъв да бъде размерът на популацията. Тъй като няма разработена теория за определяне на този размер, той се определя експериментално и най-подходящата стойност варира от задача до задача. Препоръчително е да се изпробват различни стойности за да се провери при коя оптимизацията протича най-ефективно. Параметърът *iters* определя от колко итерации да се състои процесът по оптимизация. При задачи в пространства с голям брой измерения този параметър може да се наложи да има голяма стойност. В представения пример бройката от 75 итерации се оказва напълно достатъчна. Параметърът *mutationChance* определя колко често, на вероятностен принцип, да се случва мутацията на отделни елементи във векторите на решенията. Емпирично правило е стойността на този параметър да бъде относително малко число. В много случаи на оптимизация с генетични алгоритми по-добри резултати се постигат, когато се използва правилото за елита, което се контролира с булев аргумент *elitism*. За да работи ефективно оптимизацията с генетични алгоритми е нужно да се изпрати и аргумент *evalFunc* към обект от тип функция, който да служи за изчисляване на жизнеността за всеки вектор на решение.

В представения пример функцията за жизненост е наречена *fitness* и като входен аргумент получава само един вектор, представляващ вектор на единично решение. При задачата за раницата сумарното тегло на избраните за натоварване предмети определя и до колко един вектор на решение е жизнеспособен. При надхвърляне на лимита се използва малък трик, стойността на жизнеността да бъде направена отрицателна (Листинг 10.2). По този начин решения, които са с голямо надхвърляне на позволения лимит биха били оценени с най-слаба стойност за жизненост.

За удобство, описанието на данните е организирано в *data.frame* структура, която съдържа вектор с названията на обектите, вектор с ценността на всеки предмет и вектор с теглото на всеки предмет (Листинг 10.2). Обектът с данните, лимитът за максимално тегло и броят итерации са представени като обекти в общото пространство на текущата R сесия, така че функцията за оценка на жизнеността да има директен достъп до тази информация. След извършване на оптимизацията с помощта на функцията *summary* може да се визуализира най-доброто открито решение.

В представения пример оптималното решение, предложено от генетичния алгоритъм изключва само един предмет *icescrea*, като достига 102 точки на полезност от максималните 117 точки на полезност. Ако експериментът бъде повторен с лимит на раницата от 15 се получава решение в което не участват *icescream* и *orangejuice*. При този лимит постигнатата оптимална стойност е 92 от 117.

Листинг 10.3: Анимирана визуализация на процеса по търсене на оптимално решение

```
library(ggplot2)
library(animation)

setwd("~/Desktop")

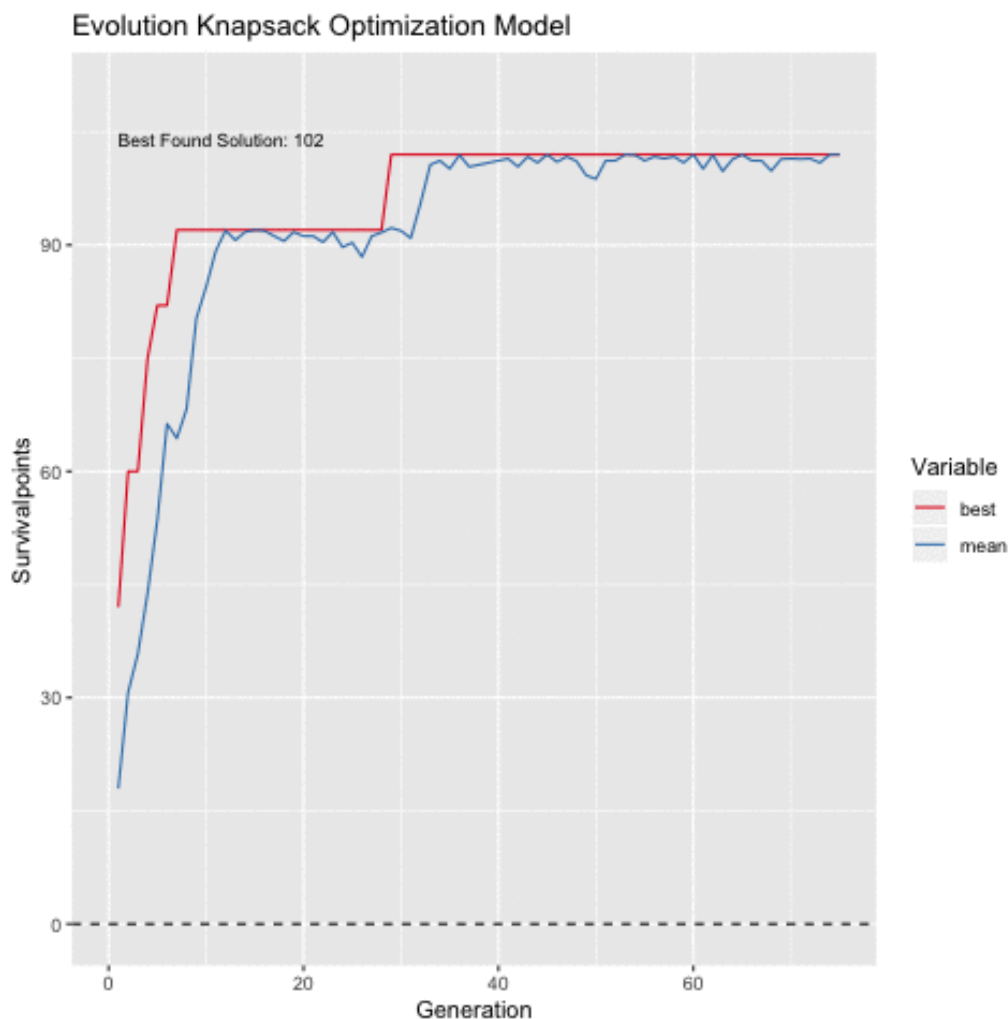
animate <- function(x) {
  for (i in seq(1, iterations)) {
    current <- data.frame(Generation = c(seq(1, i), seq(1, i)),
                          Variable = c(rep("mean", i), rep("best", i)), Survivalpoints
                          = c(-model$mean[1:i], -model$best[1:i]))

    graphics <- ggplot(current, aes(x = Generation, y =
    Survivalpoints, group = Variable, colour = Variable)) + geom_
    line() + scale_x_continuous(limits = c(0, iterations)) +
    scale_y_continuous(limits = c(0, 110)) + geom_hline(
    yintercept = 0, y = max(current$Survivalpoints), lty = 2) +
    annotate("text", x = 1, y = max(current$Survivalpoints) + 2,
    hjust = 0, size = 3, color = "black", label = paste("Best_
    Found_Solution:", max(current$Survivalpoints))) + scale_
    colour_brewer(palette = "Set1") + labs(title = "Evolution_
    Knapsack_Optimization_Model")

    print( graphics )
  }
}

saveGIF(animate(), interval=0.1, outdir=getwd())
```

При извършването на оптимизация с приближени числени методи е важно да се наблюдава сходимостта на процеса. Сходимостта изразява бързината с която се достига до достатъчно приемливо приближение на търсената оптимална стойност. Благодарение на възможностите за създаване на анимирани GIF файлове, която пакетът *animation* дава, сходимостта на процеса по оптимизация на предложената задача за раницата може да бъде наблюдаван представен визуално (Листинг 10.3).



Фигура 10.2: Сходимость на процеса по оптимизация

По своята същност кадровата анимация представлява последователност от отделни изображения, генерирани с помощта на функциите в пакета *ggplot2*. За всяка итерация от оптимизацията се генерира отделно статично растрно изображение (Листинг 10.3). В генерираната анимация се проследява текущо намерената оптимална стойност, най-доброто открито решение и средната стойност на решенията в популацията (Фиг. 10.2). Тъй като генетичните алгоритми имат стохастична природа, то всяко стартиране на кода от примера ще води до различно решение, но близко до оптималното.

10.3 Изкуствени невронни мрежи

Изкуствените невронни мрежи започват своето развитие в средата на XX век, като пред 80 години добиват нова популярност, а към настоящия момент са един от най-обещаващите методи за приближени пресмятания в областта на изкуствения интелект. Работата по изкуствените невронни мрежи започва с идеята да се направи математически модел на естествените биологични нервни системи. Разбира се, толкова амбициозна задача се оказва не толкова лесно приложима и в резултат на тези усилия се появява клон в числените пресмятания наречен „изкуствени невронни мрежи“, които за жалост са много далеч от реален модел на биологична нервна система.

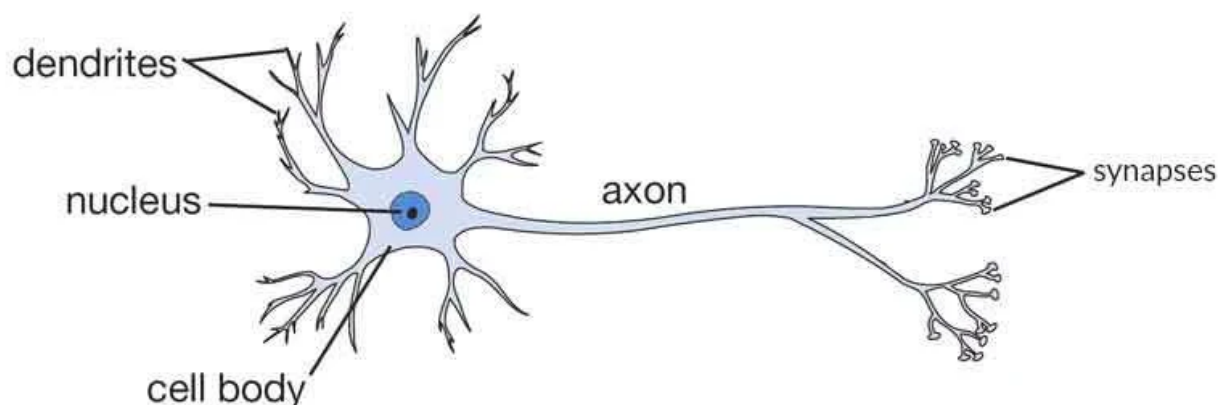
Най-голямото предимство на изкуствените невронни мрежи е тяхната възможност да се обучават с помощта на примери. В общия случай изкуствените невронни мрежи са съставени от голям брой малки, но взаимно свързани елементи наречени неврони. Обработката на информацията е нелинейна и в силно паралелна топология на мрежата. Изкуствените невронни мрежи попадат в категорията на само-адаптиращите се системи. Това означава, че мрежата може да промени своето поведение, спрямо примерите с които бива обучавана в течение не времето.

Най-основното си приложение изкуствените невронни мрежи намират при решаването на задачи, които се решават с лекота от хората, но са трудни за решаване с класическите изчислителни машини. Пример за такива задачи са разпознаването на лица, разпознаването на пръстови отпечатьци, разпознаването на печатен или ръкописен текст, прогнозирането на времеви редове и много други. Общото название на тази група задачи е „разпознаване на образи“. Под „образ“ се разбира по-широкото понятие, а не тясно свързаното с графично изображение.

Най-широко разпространеният вид изкуствена невронна мрежа е трислойният перцептрон. По своята същност този вид невронна мрежа представлява насочен тегловен граф организиран в слоеве. Теоретично е доказано, че трислойна мрежа може да научи всяка функция, представена с примери за вход изход. При съвременните реализации на многослойни перцептрони се използват значително по-голям брой слоеве, но принципът на работа остава един и същ. При многослойните перцептрони е характерно всеки слой да бъде пълно свързан с предходния и със следващия (Фиг. 10.6). Към всеки слой има допълнителен неврон, който не получава входни сигнали и служи за отместване (bias). Този неврон емитира единичен сигнал към следващия слой. Входният слой получава информация от външната среда за мрежата, тоест не е свързан с предходен слой. Изходният слой подава резултата от изчислението на мрежата към външната среда, тоест няма свързване към следващ слой. Мрежата съхранява научената информация в своите тегла и процесът на обучение е пряко свързан с такава модификация на теглата в графа, че мрежата да постига оптимални резултати в работен режим.

Най-големият недостатък на изкуствените невронни мрежи е нуждата от много време през фазата на обучението. Разработени са множество точни числени методи за модифициране на теглата в мрежата, като най-популярният е обратно разпространение на грешката. Този метод е градиентен точен числен метод и дава едни от най-добрите резултати при обучението на многослойни перцептрони. Разработени са също и приближени числени методи за модификация на теглата в мрежата, като един от най-популярните е еволюция на разликите. Еволюция на разликите е модификация на генетичен алгоритъм в частта му на операцията по мутация. Най-голямото предизвикателство в съвременното развитие на изкуствените невронни мрежи е намирането на възможно по-ефективни топологии на мрежата и намирането на възможно по-бързи алгоритми за обучение на мрежата.

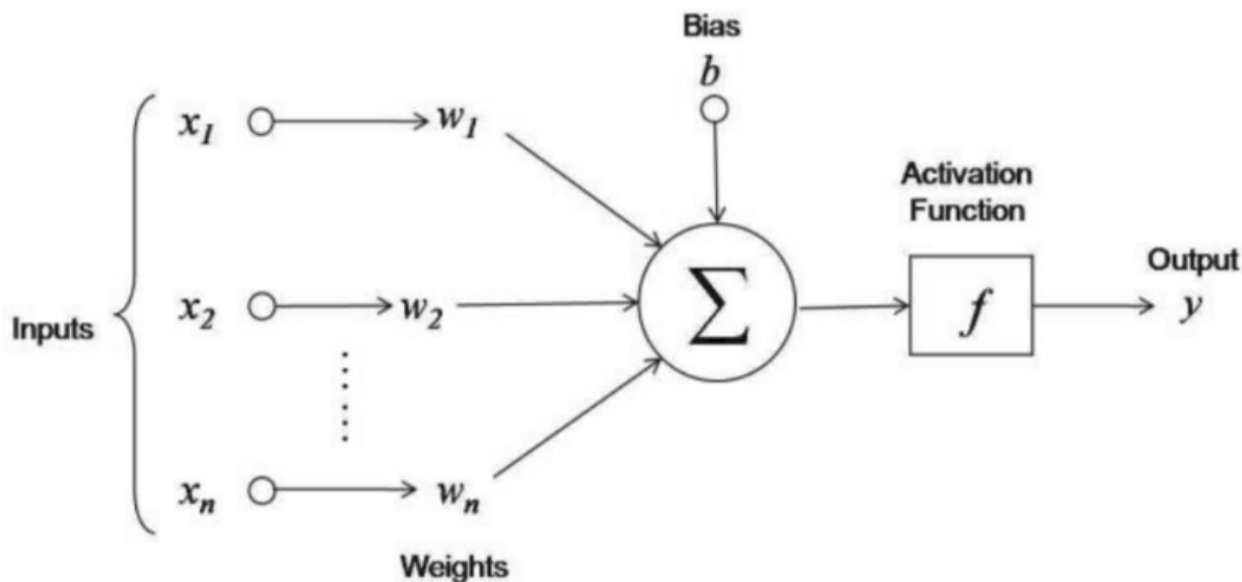
Biological Neuron



https://res.cloudinary.com/dyd911kmh/image/upload/f_auto,q_auto:best/v1547672259/1_a74o1a.png

Фигура 10.3: Схема на биологична нервна клетка

В биологичните неврони клетката има тяло, свързано с дендрити за входни сигнали, ядро, аксон, предаващ сигнала към синапсите (Фиг. 10.3). По аналогия с биологичния неврон, изкуственият неврон (Фиг. 10.4) има входни връзки с тегла (дендрити), сумираща функция (ядро на неврона), функция за активация (аксон) и изходен сигнал (синапси).



https://res.cloudinary.com/dyd911kmh/image/upload/f_auto,q_auto:best/v1547672259/2_i1cdwq.png

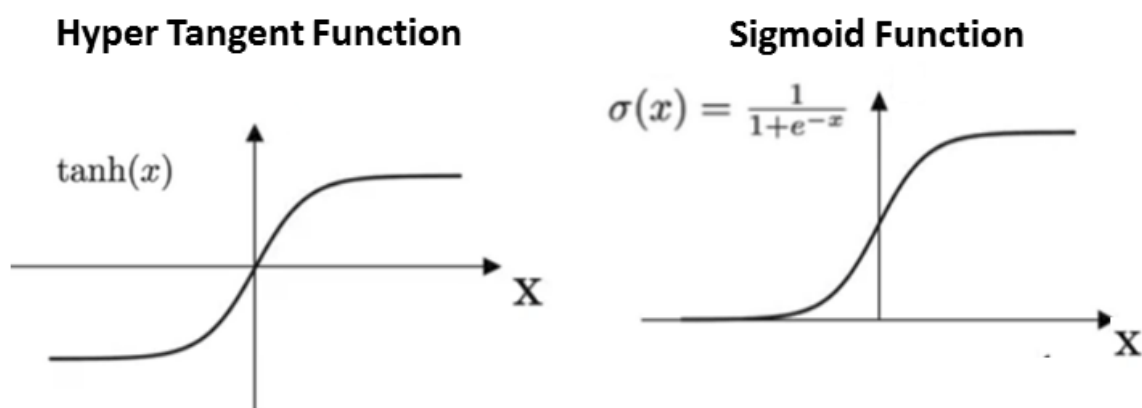
Фигура 10.4: Схема на изкуствен неврон

Като трансферна функция най-често се използва линейна зависимост. Това означава, че стойността

в ядрото на изкуствения неврон е сума от умножението на входните сигнали с прилежащите им тегла (Формула 10.1).

$$y_i = \sum (w_{ij} * y_j) + b \quad (10.1)$$

Тъй като няма ограничение за стойностите на теглата, то е възможно теглата да бъдат големи отрицателни числа или големи положителни числа. Сигналите на изхода на невроните е обичайно да бъдат нормирани или в диапазона от 0.0 до 1.0 или в диапазона от 1.0 до +1.0. Също така, броят входящи връзки към един неврон може да варират. При тези условия, когато се прилага линейна трансформация на сигналите (Формула 10.1), резултатът от пресмятането става несравним между различните неврони и не може разумно да се определи кой неврон какво влияние трябва да оказва към следващия слой. Това налага резултатът от трансферната функция да бъде подложен на нормализация чрез прилагането на функция за активация (Фиг. 10.4). Най-често прилаганите функции за активация имат асимптотична сходимост в минус безкрайност и в плюс безкрайност. Това са плавно нарастващи диференцируеми функции (Фиг. 10.5). Предимството на хиперболичния тангенс пред сигмоидната функция е, че той има симетрия спрямо абсцисната ос.



Фигура 10.5: Функции за активация на изкуствен неврон

За илюстрация на възможностите, които една трислойна неврона мрежа има ще бъде използвано множество от данни за успешно започване на работа при професията „програмист“ (Таб. 10.3). Отделните кандидати са оценени по скала от 100 точки за два критерия – технически умения и социални умения. Успешно започналите работа кандидати са обозначени с „Да“, а неуспешните кандидати с „Не“. Тази организация на данните показва нуждата от въвеждане на числена информация на входа на мрежата и определянето на резултат в два класа (Да/Не).

Technical Skills (TS)	Soft Skills (SS)	Successful Employment (SE)
30	80	Yes
10	20	No
20	90	Yes
30	40	No
80	50	Yes

Таблица 10.3: Тренировъчно множество данни

При два числени входни параметъра и един параметър за прогнозиране е съвсем естествено топологията на изкуствената невронна мрежа да бъде с два входни неврона и един изходен неврон. Остава

единствено да се вземе решение за размера на скрития слой. Тъй като не е създадена теория за избор на размера на скрития слой, този размер най-често в практиката се определя експериментално. В предложения случай се използват четири неврона в скрития слой, но приемливи резултати биха се получили и с три неврона. Съкратеният запис за такава топология на изкуствена невронна мрежа е 2-4-1 и отразява бройката на невроните във всеки от слоевете.

Листинг 10.4: Класифициране с изкуствена невронна мрежа

```
library(neuralnet)

training <- data.frame(TS=c(30,10,20,30,80), SS=c(80,20,90,40,50), SE=c
  (1,0,1,0,1))

network <- neuralnet(SE~TS+SS, data=training, hidden=4, act.fct="tanh", linear.
  output=FALSE)

plot(network)

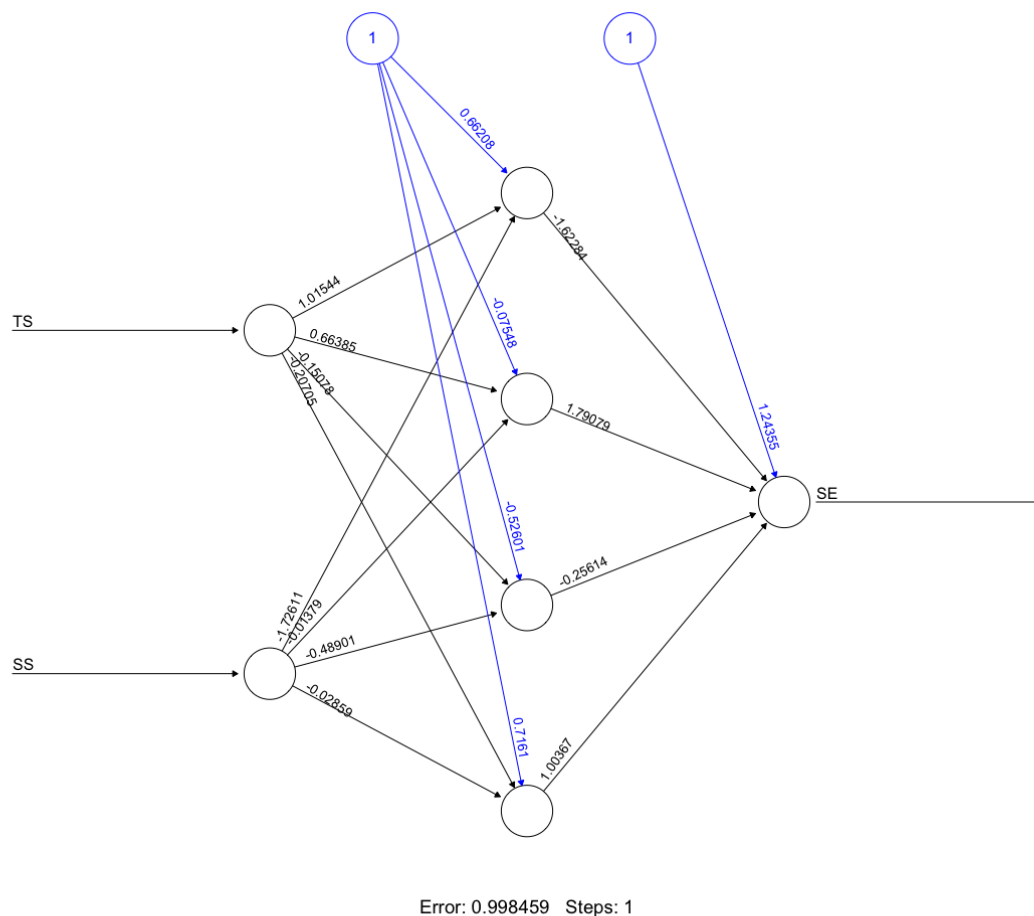
testing <- data.frame(TS=c(85,30,20), SS=c(10,20,75))

prediction <- compute(network, testing)

prediction$net.result
#           [,1]
# [1,] 0.3322677
# [2,] 0.3322677
# [3,] 0.9924267

ifelse(prediction$net.result > 0.5, TRUE, FALSE)
#           [,1]
# [1,] FALSE
# [2,] FALSE
# [3,] TRUE
```

За работа с невронни мрежи в R е създаден програмният пакет *neuralnet*. За нуждите на експеримента данните се подготвят в две множества – тренировъчно и тестово (Листинг 10.4). След което следва изграждането на модела с помощта на функцията *neuralnet*. Като първи аргумент функцията получава обект от тип формула $SE \sim TS + SS$, който показва кои променливи ще са на входа и кои на изхода. Параметърът *hidden* определя размера на скрития слой. Параметърът *act.fct* задава каква функция на активация да имат невроните. Изходът да не бъде линейно интерпретиран се задава в параметъра *linear.output*.



Фигура 10.6: Трислойна изкуствена невронна мрежа

Изчертаването на изкуствената невронна мрежа се постига с функцията *plot*, на която се подава като аргумент самата мрежа (Фиг. 10.6). С помощта на функцията *compute* тестовото множество (Таб. 10.4) се подава на изкуствената невронна мрежа и се пресмятат шансовете за наемане на кандидатите, които не са участвали в множеството за обучение на мрежата.

Technical Skills	Soft Skills	Employment Forecast
85	10	0.3322677
30	20	0.3322677
20	75	0.9924267

Таблица 10.4: Тестово множество данни

За ад се даде еднозначен отговор за шансовете на един кандидат дали ще бъде нает е удачно полученият резултат да се бинаризира (Листинг 10.4).

Заклучение

В реалната практика, когато точните числени методи не успяват да дадат желания резултат в приемливо време за пресмятане, изключително полезни могат да бъдат алгоритмите за приближени пресмятания. За симулации това са Монте-Карло методите, за оптимизация това са генетичните алгоритми, а за класификацията това са изкуствените невронни мрежи. Разбира се, съществуват множество други подходи, методи и алгоритми за приближени числени пресмятания, но представените в настоящото изложение са едни от най-използваните в практиката.

Глава 11

Оформление на резултатите за печатно и електронно представяне

Финалното оформление на получените от анализа резултати е не по-малко важно от самото им пресмятане. Спрямо аудиторията пред която резултатите ще бъдат представяни оформлението им може да бъде в различни варианти като писмен доклад, уеб страница или презентация със слайдове. За тези нужди програмният продукт *R* предлага група от пакети, спомагащи за постигането на максимална експресивност в представянето. Пакетът *knitr* подпомага оформянето на отчети и доклади. Пакетът дава възможност за работа с тагиращите езици *LaTeX* и *Markdown* [17], като резултатът от компилацията може да бъдат *PDF* документи, *HTML* [18] страници, презентации и дори *MicrosoftWord* документи.

11.1 Работа с LaTeX

LaTeX [9] е тагиращ език с широко приложение в писането на научни статии, тези, книги, постери и презентации. За да се използват възможностите на *LaTeX* е необходимо инсталирането на допълнителен софтуер за съответната операционна система. За трите най-популярни операционни системи *LaTeX* се поддържа от различни пакети, както следва: *Windows* - *MiKTeX*, *MacOS* - *MacTeX* и *Linux* - *TeXLive*.

За работа с *LaTeX* се създават обикновено текстови файлове, чието разширение е „.tex“ и може да се създават с всеки съвременен текстов редактор. Тех документите са йерархични документи с ясно дефинирана структура. На първия ред се записва инструкция за вида на документа с командата `\documentclass{...}`. Най-популярните видове документ са *report*, *beamer*, *memoir*, *letter* и други. След типа на документа следва служебна секция за зареждане на нужни за компилацията пакети и/или индекси. За включването на изображения е необходимо използването на пакета *graphicx*. В същата секция се определя авторът (`\author`), заглавието (`\title`) и датата (`\date`) на документа. Същинското съдържание на документа се разполага между инструкциите `\begin{document}` и `\end{document}`.

Изложението на документа може да бъде разделено на отделни секции с инструкцията `\section{Название на секция}`. Всичко написано след тази инструкция става част на съответната секция докато не бъде достигната следващата инструкция на нова секция. Номерирането на секциите и подсекциите се извършва автоматично от текстовия процесор на *LaTeX*. Когато са поставени етикети с инструкцията `\label{етикет}`, те могат да бъдат позовавани в други части на документа с инструкцията `\ref{етикет}`. Съдържанието на документа се генерира автоматично с помощта на инструкцията `\tableofcontents`.

Изброените инструкции са напълно достатъчни за създаване на базови документи, но далеч не покриват пълните възможности на *LaTeX*. Тъй като компилаторът използван в текстовия процесор е еднопасов, то за да се направи правилно индексирание на препратките и таблицата за съдържанието често се налага компилацията да бъде стартирана два пъти последователно.

Създаването на *LaTeX* документи с интегриране на *R* инструкции в него става чрез изготвянето на стандартен *LaTeX* документ в който се добавят фрагменти (chunks) на *R* програмен код. Тези фрагменти се предхождат от инструкция за начало на фрагмента по зададен шаблон (Листинг 11.1), а края на фрагмента се обозначава със символа маймунско а @.

Листинг 11.1: Инструкция за R фрагмент в LaTeX документ

```
<<label–value , option1=value1 , option2=value2>>=
@
```

Текстовият документ се запазва с разширение „.Rnw“. Трансформацията на комбинирания код (*LaTeX* и *R*) се транслира до *Tex* с командата *Sweave* в командния интерпретатор на *R* (Листинг 11.2). Важно е *Rnw* файлът да се намира в същата директория, в която се изпълнява командата за транслиране.

Листинг 11.2: Транслиране от Rnw до Tex

```
library( knitr )

setwd( "~/Desktop" )

Sweave( "./example0002.Rnw" )
```

В резултат на транслацията, в съответната директория се генерира *Tex* файл, който от своя страна се подава на транслатор, изпълняван в конзолата на операционната система, *Tex* към *PDF* (Листинг 11.3).

Листинг 11.3: Транслиране от Tex до PDF

```
pdflatex ./example0002.tex
```

За илюстриране на възможностите при генерирането на *PDF* документи са предложени примерните файлове достъпни на следните електронни адреси:

Листинг 11.4: Адрес на примерени LaTeX документи

```
https://raw.githubusercontent.com/TodorBalabanov/Statistical–Data–Processing–
with–R/master/code/Sweave.sty

https://raw.githubusercontent.com/TodorBalabanov/Statistical–Data–Processing–
with–R/master/code/example0002.Rnw

https://raw.githubusercontent.com/TodorBalabanov/Statistical–Data–Processing–
with–R/master/code/example0003.Rnw
```

За да бъде успешно транслирането от *Tex* към *PDF* е нужно в директорията с *Tex* файла да се помести и *Sweave.sty* файлът, който съдържа дефиниции необходими за извършването на процеса по транслация.

Листинг 11.5: Линейна регресия на разходи спрямо спестявания

```
library( knitr )

setwd( "~/Desktop" )

Sweave( "./example0003.Rnw" )

system("pdflatex ./example0003.tex", intern=TRUE)
```


При избора на имена за етикетите в *R* фрагментите, трябва да се избягва използването на специални символи като интервал и точка. Добър вариант е използването на латинските букви и символът за тире. Когато се използва флагът *echo* със стойност *FALSE*, в *Tex* файла не се включва *R* програмния код, довел до визуализацията на съответните резултати.

За визуализация на изображения (Листинг 11.5) най-удачно е междинната графика да бъде генерирана в отделен графичен файл (в случая Fig01.png), който в последствие да бъде добавен като графичен компонент в *Rnw* документа.

11.2 Работа с RMarkdown

RMarkdown става все по-използван през годините, тъй като е значително по-опростен от *LaTeX* и това създава допълнителен комфорт при оформянето на информацията. *RMarkdown* дава възможности за генериране на множество различни файлови формати, а също така позволява разширяване на шаблонен принцип (templates). Процесът по създаване на документи с *R* и *RMarkdown* е сходен с процеса, ползван за *LaTeX*. В чист текст се изписва *RMarkdown* структурата, а на подходящи места се вмъкват фрагменти *R* код (chunks). Текстовите файлове се съхраняват с разширение „.Rmd“.

R пакетът *knitr* се използва за изчислението на *R* кода в *RMarkdown*, а *pandoc* софтуерният пакет за транслирането в различните файлови формати. Транслирането на текстовия файл става с *R* функцията *render*, която използва софтуерния пакет *pandoc*. Група функции от пакета *rmarkdown* - *html_document*, *pdf_document*, *word_document* и *ioslides_presentation* се използват за най-популярните файлови формати. Допълнително пакетите *rticles*, *tuftes* и *resumer* дават още възможности.

Всеки *RMarkdown* документ започва със заглавна част в *YAML* формат, която дава подробна информация за документа. Заглавната част започва с три последователни тирета и завършва с три последователни тирета, на отделен ред. Всеки ред в заглавната част е ключ-стойност двойка, която определя параметри като: заглавие, автор, дата и целеви формат на документа. Типовете възможни документи за генериране са описани в пакетите, които отговарят за тях и в общия случай съвпадат с названието на функцията, която извършва транслацията. За пакети, различни от *rmarkdown*, типът на документа се предхожда от името на пакета, който го поддържа. Ако типът на документа изисква допълнителни опции, то те се задават като подетикети с поне два спейса подравняване.

Markdown тагирацията език е създаден с цел максимално опростяване на синтаксиса. Не разполага с възможностите и гъвкавостта на *LaTeX* или *HTML*, но пък значително ускорява писането и оформлението. Поради максималната си опростеност *Markdown* е много бърз за научаване, което винаги е положително.

В *Markdown* нов ред се получава с оставянето на празен ред между текстовете и с два или повече празни интервала в края на реда. За наклонен текст се поставя подчертаваща черта от двете страни на текста. За удебелен текст се поставят по две подчертаващи черти от двете страни на текста. За наклонен и удебелен шрифт се поставят по три подчертаващи черти от двете страни на текста. Символът по-голямо в началото на всеки ред служи за блоков цитат. Неномериран списък се създава с тире в началото на всеки нов ред или със звезда. Номериран списък се създава с цифра или буква, последвани с точка в началото на всеки ред. Списъците могат да бъдат вложени един в друг. Заглавията се обозначават със символа диез в началото на реда. Броят на диезите определя нивото на заглавие като са възможни стойности от едно до шест. При генерирането на *PDF* файл заглавията от ранг едно са за секции, а от ранг две за подсекции. Хипервръзки се създават с текст в квадратни скоби, последвани от *URL* адрес, обграден в кръгли скоби. Включването на изображения става с алтернативен текст в квадратни скоби и адрес до изображението в кръгли скоби, но пред квадратните скоби се изписва удивителен знак. Изписването на уравненията става с ограждане на уравнението от двете страни с два знака за долар. За целта двойните долар знаци е важно да са на отделни редове, а като синтаксис за формулите се използва този на *LaTeX*.

11.2.1 Статични документи

Когато финалните документи не съдържат възможности за манипулирането на компонентите в тях, то те съдържат статична информация. Най-често това са *PDF* файлове и *HTML* документи без JavaScript функционалност.

Листинг 11.6: Адрес на примерни RMarkdown документ

```
https://raw.githubusercontent.com/TodorBalabanov/Statistical-Data-Processing-
with-R/master/code/example0004.Rmd

https://raw.githubusercontent.com/TodorBalabanov/Statistical-Data-Processing-
with-R/master/code/example0005.Rmd

https://raw.githubusercontent.com/TodorBalabanov/Statistical-Data-Processing-
with-R/master/code/example0007.Rmd
```

Примерен *RMarkdown* е представен на адрес от Листинг 11.6, а транслирането до *HTML* документ става с командите представени в Листинг 11.7. За целта *Rmd* файлът трябва да се намира в Desktop директорията на локалния компютър.

Листинг 11.7: Транслиране от RMarkdown в HTML и PDF

```
library( knitr )
library( ggplot2 )
library( rmarkdown )

setwd( "~/Desktop" )

render( "./example0004.Rmd" )
render( "./example0005.Rmd" )
render( "./example0007.Rmd" )
```

Markdown документи, които съдържат фрагменти с *R* програмен код се наричат *RMarkdown*. Поведението на тези фрагменти е като на *knitr*, но се етикетират по различен начин и имат малко повече възможности. Фрагментът започва с три апострофа, отваряща фигурна скоба, буквата *r*, етикет на фрагмента, опции разделени със запетая и затваряща фигурна скоба. Фрагментът се затваря с три последователни апострофа. Всичко между отварянето и затварянето на фрагмента се третира като *R* програмен код.

Възможност за създаване на презентации с *R* има през *LaTeX* и шаблона *Beamer*, където всяка страница се явява отделен слайд. Сложността на *LaTeX* синтаксиса може да бъде избегната чрез използването на *RMarkdown*, който да се транслира до *HTML5* презентация (example0007.Rmd). Заглавието на всеки слайд се маркира с два диеза, което е еквивалентно на заглавие от втори ред. Графично оформление на слайда може да се укаже с *class*, идентификатор на слайда и стандартни *CSS* инструкции от вида: {.vcenter .flexbox #SlideID}

11.2.2 Динамични документи

С помощта на пакета *htmlwidgets* към финалния документ може да се добави JavaScript функционалност, така че отделните компоненти да дадат определена степен на интерактивност.

Въпреки че графичното представяне на информацията е значително по-информативно, в някои случаи е нужно резултатите да бъдат представени в таблична форма. С помощта на функцията *kable* от пакета *knitr* могат да се създават таблици със статично съдържание (Листинг 11.9).

Листинг 11.8: Адрес на примерни интерактивни документ

```
https://raw.githubusercontent.com/TodorBalabanov/Statistical-Data-Processing-  
with-R/master/code/example0006.Rmd
```

За да се добави интерактивност в табличното представяне е предложен пакетът *DT*. С помощта на функцията *datatable* се генерира таблица, даваща множество допълнителни възможности. Имената на колоните могат да се премахнат с опцията *rownames*. С помощта на опцията *filter* се дава възможност за филтриране на редовете. С добавяне на разширението *Scroller* се дава възможност за вертикално прелистване на редовете. Опцията *scrollX* дава възможност за хоризонтално прелистване на колоните. Опцията *dom* указва елемент за визуализация да е самата таблица (t от буквеното съчетание tiS). Допълнителна информация за таблицата се задава с буквата i в съчетанието tiS. Възможностите за прелистване се задават с буквата S в съчетанието tiS. Част от параметрите се задават на самата функция *datatable*, а друга част като списък от опции, демонстрирани в примера *example0006.Rmd* (Листинг 11.8).

Листинг 11.9: Създаване на интерактивни документи

```
library( DT )  
library( dplyr )  
library( knitr )  
library( ggplot2 )  
library( rmarkdown )  
library( d3heatmap )  
  
setwd( "~/Desktop" )  
  
render( "../example0006.Rmd" )
```

Използването на топлинни карти е значително по-информативно, когато се пресмятат корелационни матрици. С помощта на пакета *d3heatmap* могат да се създават интерактивни топлинни карти. За нуждите на примера се демонстрира икономическото множество данни. С параметъра *colors* на функцията *d3heatmap* може да се контролира цветната палитра за изобразяване на стойностите (Листинг 11.9). Примерът е поместен в работен файл с адрес посочен в Листинг 11.8.

Заклучение

Благодарение на възможностите за интегриране на *R* в *LaTeX* и *Rarkdown* документи може да се постигне бързо и стилно представяне на статистическите резултати получени след извършване на съответни пресмятания. Изложените примери представят само най-основното, но пакетите разработени за *R* дават значително повече възможности. С малко повече желание и малко повече старание всеки потребител на пакета *R* може да постигне висока степен на експресивност в начина по който представя резултатите от своята работа.

Заклучение

Без да претендира за изчерпателност настоящото учебно помагало прави въведение в статистическата обработка на данни с помощта на един от най-популярните програмни продукти, а именно програмния пакет *R*. В практическата работа на ученици, студенти, докторанти и специалисти по статистика се срещат множество особености, които до голяма степен са засегнати в изложения материал. Макар и да съществуват множество алтернативни програмни продукти, като *SPSS*, *Matlab* и *Mathematica*, програмният продукт *R* се отличава с финансова ефективност и отворен модел за разширяване. В учебното помагало не са засегнати темите за напреднали, тъй като целите на авторите са основно да провокират широката аудитория. Темите за напреднали могат да бъдат открити в множество учебници и книги в чуждоезичната литература, както и в голям брой видео уроци.

Библиография

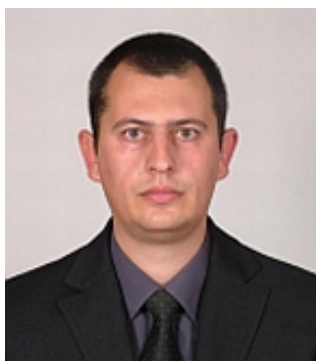
- [1] Zagumny, M.: The SPSS Boo: A Student Guide to the Statistical Package for the Social Sciences, iUniverse, 2001.
- [2] Higham. D., Higham, N.: MATLAB Guide 2nd Edition, SIAM: Society for Industrial and Applied Mathematics, 2005.
- [3] Wolfram, S.: The MATHEMATICA Book, Version 4 4th Edition, Cambridge University Press, 1999.
- [4] Cody, R., Smith, J.: Applied Statistics and the SAS Programming Language 5th Edition, Pearson, 2005.
- [5] Jamsa, K., Klander, L.: Jamsa's C/C++ Programmer's Bible 1st Edition, Cengage Learning, 1997.
- [6] Arnold, K., Gosling, J.: The Java Programming Language (Java Series), Addison-Wesley, 1997.
- [7] Collingbourne, H.: The Little Book Of C# Programming: Learn To Program C-Sharp For Beginners, Dark Neon, 2019.
- [8] Salkind, N.: Excel Statistics: A Quick Guide Second 2nd Edition, SAGE Publications, Inc, 2012.
- [9] Lamport, L.: LaTeX A Document Preparation System 2nd Edition, Addison-Wesley Professional, 1994.
- [10] Tatroe, K., MacIntyre, P., Lerdorf, R.: Programming PHP, Creating Dynamic Web Pages 3d Edition, O'Reilly Media, 2013.
- [11] Harvard, C.: Python Programming A Smarter And Faster Way To Learn Python In 7 Days: With Practical Exercises, Interview Questions, Tips And Tricks, Independently published, 2019.
- [12] Raasch, J.: JavaScript Programming Pushing the Limits 1st Edition, Wiley, 2013.
- [13] Guthals, S., Haack, P.: GitHub For Dummies (For Dummies (Computer/Tech)) 1st Edition, For Dummies, 2019.
- [14] iCode Academy: Json for Beginners Your Guide to Easily Learn Json In 7 Days, Independently published, 2017.
- [15] Goos, P., Meintrup, D.: Statistics with JMP: Hypothesis Tests, ANOVA and Regression 1st Edition, Wiley, 2016.
- [16] Verzani, J.: Getting Started with RStudio An Integrated Development Environment for R 1st Edition, O'Reilly Media, 2011.
- [17] Alam, K.: Learn Markdown The Complete Guide on Markdown Formatting, Creative Content Media, 2018.
- [18] Duckett, J.: HTML and CSS Design and Build Websites 1st Edition, John Wiley & Sons, 2011.
- [19] Turner, R.: SQL The Ultimate Beginners' Guide to Learn SQL Programming Step by Step, Amazon Digital Services LLC, 2019.

-
- [20] Jones, H.: Neural Networks An Essential Beginners Guide to Artificial Neural Networks and their Role in Machine Learning and Artificial Intelligence, Amazon Digital Services LLC, 2018.
 - [21] Goldberg, D.: Genetic Algorithms in Search, Optimization, and Machine Learning 1st Edition, Addison-Wesley Professional, 1989.
 - [22] Rubinstein, R., Kroese, D.: Simulation and the Monte Carlo Method 2nd Edition, Wiley-Interscience, 2007.
 - [23] Matloff, N.: The Art of R Programming: A Tour of Statistical Software Design 1st Edition, No Starch Press, 2011.
 - [24] Gardener, M.: Beginning R The Statistical Programming Language 1st Edition, Wrox, 2012.
 - [25] Chambers, J.: Software for Data Analysis: Programming with R (Statistics and Computing) 2nd Edition, Springer, 2009.
 - [26] Venables, W., Smith D.: An Introduction to R Notes on R: A Programming Environment for Data Analysis and Graphics,
<https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf>
 - [27] R Data Import/Export,
<https://cran.r-project.org/doc/manuals/r-release/R-data.pdf>
 - [28] R Installation and Administration,
<https://cran.r-project.org/doc/manuals/r-release/R-admin.pdf>
 - [29] Writing R Extensions,
<https://cran.r-project.org/doc/manuals/r-release/R-exts.pdf>
 - [30] R Language Definition,
<https://cran.r-project.org/doc/manuals/r-release/R-lang.pdf>
 - [31] R Internals,
<https://cran.r-project.org/doc/manuals/r-release/R-ints.pdf>
 - [32] R A Language and Environment for Statistical Computing,
<https://cran.r-project.org/doc/manuals/r-release/fullrefman.pdf>
 - [33] GNU General Public License, version 2, Free Software Foundation,
<http://www.gnu.org/licenses/old-licenses/gpl-2.0.html>
 - [34] Hungarian notation, Wikimedia Foundation, Inc.,
http://en.wikipedia.org/wiki/Hungarian_notation

Азбучен указател

- Бернулиево разпределение, 132
Монте Карло симулации, 61
Монте-Карло методи, 153, 155
агрегатни функции, 76, 86
активационна функция, 164
алтернатива при условен преход, 67
анимация, 161
анти сливане, 102
аргументи на функции, 33
аргументи на функция, 71
аргументи на функция с подразбираща се стойност, 72
аритметични операции, 25
бази данни, 50, 93
бинарни файлове, 53
бинарни операции, 25
биномни коефициенти, 131
биномно разпределение, 131
цикъл с условие за край, 69
цикъл за обхождане, 69
частично търсене, 34
четене от файл, 48
числени типове, 29
дефиниция на функция, 71
демонстрационни данни, 54
дълъг запис, 103
диаграма на разпръскване, 109
диаграма на разсейване, 58
дисперсионен анализ, 147
дисперсия, 139
документация на функции, 33
документация на операции, 34
достъп по индекс, 40, 43
дясна асоциативност, 26
експериментален модел, 62
електронни таблици, 50
елит, 159
еволюционни алгоритми, 156
еволюция на разликите, 162
файлови формати, 53
фактори, 37, 49
филтриране на редове, 89
функции с променлив брой аргументи, 72
функционални обекти, 73
функционално програмиране, 94
генетични алгоритми, 156
графични теми, 119
графика тик кутия, 113
графика тип цигулка, 115
графика тип кутия, 60
групиране по редове, 92
групово манипулиране на данни, 74
хистограма, 56, 106, 112, 124, 134
именуване на променливи, 27
инсталация на продукта, 8
интерполация, 138
изкуствен интелект, 161
изкуствени невронни мрежи, 161
изпълнение на скрипт, 63
изтегляне на инсталатор, 4
извикване на функции, 33
извличане на данни по колонии, 88
каскада от оператори за условен преход, 68
ключови полета, 86
команден интерпретатор, 15
конкатенация на символи низове, 103
контекстна зависимост на операциите, 26
корелационна матрица, 140, 172
корелация, 139
коварияция, 140
кумулятивна функция, 130
квантили, 139
линейна графика, 117
линейна регресия, 149
липсващи стойности, 37
логически операции, 66, 90
логически стойности, 67
логически тип данни, 31
лява асоциативност, 26
масиви, 46
математически изрази, 26
математически операции, 24

- матрици, 44
медиана, 138
метод Монте-Карло, 123
многослоен перцептрон, 162
мода, 138
модифициране на колони, 90
мултимедийни презентации, 168
насочен тегловен граф, 162
нормално разпределение, 129
нулева хипотеза, 141, 143, 147
обединяване на данни, 97
обхождане по елементи, 94
обобщение по колони, 91
обратно разпространение на грешката, 162
обсег на видимост, 87
операции с вектори, 35
операции за присвояване, 27
оператор за многовариантен избор, 68
оператор за условен преход, 67
оператори за цикъл, 69
оператори за преход, 66
описателна статистика, 137
отворен код, 3
пакети, 18
плътностна функция, 108, 130
поасоново разпределение, 134
полусливане, 102
потоци, 88
поточни операции, 88, 91
потребителски функции, 71
представяне на информацията, 28
прекъсване на цикли, 70
претеглена средна, 138
приблизени числени методи, 153
приблизени пресмятания, 153
приоритет на операциите, 26
прогнозиране, 166
програмен скрипт, 62
променливи, 27
рамкирани данни, 38
размер на популация, 159
разпознаване на образи, 162
регулярни изрази, 104
реорганизация на данните, 97
сходимост на процес, 160
съхраняване на обект, 54
символни низове, 30, 103
сливане от дясно, 102
сливане от ляво, 102
сложни сливания, 101
софтуерен лиценз, 11
сравнителна статистика, 137, 139
средна, 137
стандартно отклонение, 139
степени на свобода, 143
стохастичен алгоритъм, 161
широк запис, 103
таблично представяне, 172
тернарна операция, 26
тест на две извадки, 143
тест на две сдвоени извадки, 145
тест на една извадка, 141
типове данни, 27
типове за астрономическо време, 30
точни числени методи, 153
топлинна карта, 172
трансферна функция, 164
транспониране на редове и колони, 99
уеб страници, 168
унарни операции, 25
вектори, 34
вероятностно разпределение, 128
външно сливане, 102
върната стойност от функция, 72
вътрешно сливане, 102
въвеждане на информация, 48
визуализация на данни, 55
визуализация на резултатите, 48
визуално оформление, 168
зареждане на обект, 54
ANOVA, 147
JSON данни, 55
LaTeX, 168
p-стойност, 141
RMarkdown, 170
t-тест, 141



Настоящото учебно помагало съдържа серия от примери по дисциплините „Анализ на Данни с R“, която се изучава в докторантската програма на „Център за обучение“ към „Българска академия на науките“. Изложението е насочено към аудитория със сериозни познания в областта на математиката и статистиката.



Помагалото е съставено от практически примери, обхващащи цялостния цикъл за статистическа обработка на информацията, което включва събиране на информация, предварителна обработка на суровите данни, прилагане на методи за статистическа обработка, визуализация на резултатите и оформление на финалните документи. Структурата на учебното помагало позволява отделни части от него да послужат при разработването на курсови задачи и/или дипломни работи.



Свободният достъп до помагалото дава възможност то да бъде използвано в учебния процес на курсове с подобно съдържание и на други учебни заведения. Изложеният материал дава възможност за допълване на съществуващите магистърски програми, примерно в областта на приложната математика и статистиката.

Посочените в библиографията литературни източници са предимно със справочен характер, но дават възможност на заинтересуваните читатели да разширят познанията си в засегнатите области.