

R

Notes for Professionals

专业人员笔记R 专业人员笔记

Chapter 20: Reading and writing tabular data in plain-text files (CSV, TSV, etc.)

Section 20.1: Importing .csv files

Importing using base R

```
# reading the package
library(foreign)
# get the file path of a .csv included in R's utils package
# will very likely need an installation location
# e.g. "C:/Program Files/R/R-3.6.0/library/utils/resource/library/utils/datasets/csv.csv"
df <- read.csv(csv.path)
```

A user friendly option, `file.choose()`:

```
df <- read.csv(file.choose())
```

Notes

- Unlike `read.table.read.csv` defaults to header = TRUE, and uses 0 separator factors = FALSE.
- All these functions will convert strings to factor class by default unless stringAsFactors = FALSE.
- The `read.csv2` variant defaults to sep = ";" and dec = "," for ; comma is used as a decimal point and the semicolon as a field separator.

Importing using packages

The `readr` package's `read_csv` function offers much faster performance, a progress bar for large files, and more popular default options than standard `read.csv`, including `stringAsFactors = FALSE`.

Chapter 92: I/O for foreign tables (Excel, SAS, SPSS, Stata)

Section 92.1: Importing data with rio

Section 20.2: Read and write Stata, SPSS and SAS files

Chapter 106: Color schemes for graphics

Section 106.1: viridis - print and colorblind friendly palettes

Viridis (named after the `chemin.viridis`) is a recently developed color scheme for the Python library `matplotlib`.

(The video presentation by the link explains how the color scheme was developed and what are its main advantages). It is seamlessly joined to R.



The package can be installed from CRAN or GitHub. The `viridis` for `viridis` package is just brilliant. Nice feature of the `viridis` color scheme is integration with `ggplot2`. Within the package two `ggplot2`-specific functions are defined `scale_color_viridis()` and `scale_fill_viridis()`. See the example below:

```
library(viridis)
library(ggplot2)
gg1 <- ggplot(mtcars)
```

haven for Professionals

For reading in SPSS and SAS files

haven for Professionals

Notes

- Unlike `read.table.read.csv` defaults to header = TRUE, and uses 0 separator factors = FALSE.
- All these functions will convert strings to factor class by default unless stringAsFactors = FALSE.
- The `read.csv2` variant defaults to sep = ";" and dec = "," for ; comma is used as a decimal point and the semicolon as a field separator.

Importing using packages

The `readr` package's `read_csv` function offers much faster performance, a progress bar for large files, and more

popular default options than standard `read.csv`, including `stringAsFactors = FALSE`.

haven for Professionals

If notes for Professionals

Chapter 20: Reading and writing tabular data in plain-text files (CSV, TSV, etc.)

Section 20.1: Importing .csv files

Importing using base R

A very simple way to import data from many common file formats is with `rio`. This package provides a function `import()` that wraps many commonly used data import functions, thereby providing a standard interface. It works simply by passing a file name or URL to `import()`:

```
import('example.csv') # comma-separated values
import('example.tsv') # tab-separated values
import('example.xls') # Excel
import('example.sas') # SAS
import('example.spss') # SPSS
import('example.stata') # Stata
import('example.xlsx') # Excel
```

`import()` can also read from compressed directories, URLs (HTTP or HTTPS), and the clipboard. A comprehensive list of all supported file formats is available on the [rio package GitHub repository](#).

It is even possible to specify some further parameters related to the specific the format you are trying to read, passing them directly within the `import()` function:

```
import('example.csv', format = ",") # for csv file where comma is used as separator
import('example.csv', format = ";") # for csv file where semicolon is used as separator
```

Section 20.2: Read and write Stata, SPSS and SAS files

The packages `foreign` and `haven` can be used to import and export files from a variety of other statistical packages like Stata, SPSS and SAS and related software. There is a `read` function for each of the supported data types to import the files.

```
# loading the package
library(foreign)
library(haven)
library(readr)
library(rvest)
```

Some examples for the most common data types:

```
# reading Stata files with 'foreign'
read_stata('path/to/your/data')
# reading Stata files with 'haven'
read_stata('path/to/your/data')

# reading recent Stata (.dta) files with 'readstata13'
read_stata('path/to/your/data')

# haven for Professionals
```

The `foreign` package can read in stata (.dta) files for versions of Stata 7-12. According to the development plan, it is a more or less frozen and will not be updated for reading in versions 13+. For more recent version of Stata, you can use either the `readstata13` package or `haven`. For `readstata13`, the files are:

```
# reading recent Stata (.dta) files with 'readstata13'
read_stata('path/to/your/data')

# haven for Professionals
```

The `haven` package can read in SPSS and SAS files.

Notes

- Unlike `read.table.read.csv` defaults to header = TRUE, and uses 0 separator factors = FALSE.
- All these functions will convert strings to factor class by default unless stringAsFactors = FALSE.
- The `read.csv2` variant defaults to sep = ";" and dec = "," for ; comma is used as a decimal point and the semicolon as a field separator.

Importing using packages

The `readr` package's `read_csv` function offers much faster performance, a progress bar for large files, and more

popular default options than standard `read.csv`, including `stringAsFactors = FALSE`.

haven for Professionals

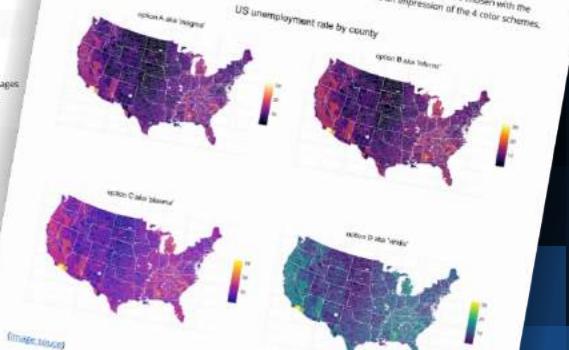
If notes for Professionals

Chapter 106: Color schemes for graphics

Section 106.1: viridis - print and colorblind friendly palettes

Viridis (named after the `chemin.viridis`) is a recently developed color scheme for the Python library `matplotlib`.

(The video presentation by the link explains how the color scheme was developed and what are its main advantages). It is seamlessly joined to R.



The package can be installed from CRAN or GitHub. The `viridis` for `viridis` package is just brilliant. Nice feature of the `viridis` color scheme is integration with `ggplot2`. Within the package two `ggplot2`-specific functions are defined `scale_color_viridis()` and `scale_fill_viridis()`. See the example below:

```
library(viridis)
library(ggplot2)
gg1 <- ggplot(mtcars)
```

haven for Professionals

For reading in SPSS and SAS files

haven for Professionals

Notes

- Unlike `read.table.read.csv` defaults to header = TRUE, and uses 0 separator factors = FALSE.
- All these functions will convert strings to factor class by default unless stringAsFactors = FALSE.
- The `read.csv2` variant defaults to sep = ";" and dec = "," for ; comma is used as a decimal point and the semicolon as a field separator.

Importing using packages

The `readr` package's `read_csv` function offers much faster performance, a progress bar for large files, and more

popular default options than standard `read.csv`, including `stringAsFactors = FALSE`.

haven for Professionals

If notes for Professionals

400多页
专业提示和技巧

目录

关于	1
第1章：R语言入门	2
第1.1节：安装R	2
第1.2节：你好，世界！	3
第1.3节：获取帮助	3
第1.4节：交互模式和R脚本	3
第2章：变量	7
第2.1节：变量、数据结构和基本操作	7
第3章：算术运算符	10
第3.1节：范围与加法	10
第3.2节：加法与减法	10
第4章：矩阵	13
第4.1节：创建矩阵	13
第5章：公式	15
第5.1节：公式基础	15
第6章：字符串的读写	17
第6.1节：字符串的打印和显示	17
第6.2节：捕获操作系统命令的输出	18
第6.3节：从文件连接读取或写入	19
第7章：使用stringi包进行字符串操作	21
第7.1节：统计字符串中的模式	21
第7.2节：字符串复制	21
第7.3节：粘贴向量	22
第7.4节：按某些固定模式拆分文本	22
第8章：类	23
第8.1节：检查类	23
第8.2节：向量和列表	23
第8.3节：向量	24
第9章：列表	25
第9.1节：列表简介	25
第9.2节：列表快速介绍	25
第9.3节：序列化：使用列表传递信息	27
第10章：哈希映射	29
第10.1节：环境作为哈希映射	29
第10.2节：package:hash	32
第10.3节：包：listenv	33
第11章：创建向量	35
第11.1节：从内置常量创建向量：字母序列和月份名称	35
第11.2节：创建命名向量	35
第11.3节：数字序列	37
第11.4节：seq()函数	37
第11.5节：向量	38
第11.6节：使用rep()函数扩展向量	39
第12章：日期和时间	41
第12.1节：当前日期和时间	41

Contents

About	1
Chapter 1: Getting started with R Language	2
Section 1.1: Installing R	2
Section 1.2: Hello World!	3
Section 1.3: Getting Help	3
Section 1.4: Interactive mode and R scripts	3
Chapter 2: Variables	7
Section 2.1: Variables, data structures and basic Operations	7
Chapter 3: Arithmetic Operators	10
Section 3.1: Range and addition	10
Section 3.2: Addition and subtraction	10
Chapter 4: Matrices	13
Section 4.1: Creating matrices	13
Chapter 5: Formula	15
Section 5.1: The basics of formula	15
Chapter 6: Reading and writing strings	17
Section 6.1: Printing and displaying strings	17
Section 6.2: Capture output of operating system command	18
Section 6.3: Reading from or writing to a file connection	19
Chapter 7: String manipulation with stringi package	21
Section 7.1: Count pattern inside string	21
Section 7.2: Duplicating strings	21
Section 7.3: Paste vectors	22
Section 7.4: Splitting text by some fixed pattern	22
Chapter 8: Classes	23
Section 8.1: Inspect classes	23
Section 8.2: Vectors and lists	23
Section 8.3: Vectors	24
Chapter 9: Lists	25
Section 9.1: Introduction to lists	25
Section 9.2: Quick Introduction to Lists	25
Section 9.3: Serialization: using lists to pass information	27
Chapter 10: Hashmaps	29
Section 10.1: Environments as hash maps	29
Section 10.2: package:hash	32
Section 10.3: package:listenv	33
Chapter 11: Creating vectors	35
Section 11.1: Vectors from build in constants: Sequences of letters & month names	35
Section 11.2: Creating named vectors	35
Section 11.3: Sequence of numbers	37
Section 11.4: seq()	37
Section 11.5: Vectors	38
Section 11.6: Expanding a vector with the rep() function	39
Chapter 12: Date and Time	41
Section 12.1: Current Date and Time	41

第12.2节：跳转到月底	41
第12.3节：跳转到月初	42
第12.4节：按月数一致地移动日期	42
第13章：日期类	44
第13.1节：日期格式化	44
第13.2节：将字符串解析为日期对象	44
第13.3节：日期	45
第14章：日期时间类（POSIXct 和 POSIXlt）	47
第14.1节：日期时间对象的格式化和打印	47
第14.2节：日期时间运算	47
第14.3节：将字符串解析为日期时间对象	48
第15章：字符类	50
第15.1节：强制转换	50
第16章：数值类和存储模式	51
第16.1节：数字	51
第17章：逻辑类	53
第17.1节：逻辑运算符	53
第17.2节：强制转换	53
第17.3节：NA的解释	53
第18章：数据框	55
第18.1节：创建空数据框	55
第18.2节：从数据框中子集选取行和列	56
第18.3节：操作数据框的便捷函数	59
第18.4节：简介	60
第18.5节：将数据框的所有列转换为字符类	61
第19章：拆分函数	63
第19.1节：在拆分-应用-合并范式中使用split	63
第19.2节：split的基本用法	64
第20章：读取和写入纯文本文件中的表格数据（CSV、TSV等）	67
第20.1节：导入.csv文件	67
第20.2节：使用data.table导入	68
第20.3节：导出.csv文件	69
第20.4节：导入多个csv文件	69
第20.5节：导入定宽文件	69
第21章：管道操作符（%>% 等）	71
第21.1节：基本使用和链式操作	71
第21.2节：函数序列	72
第21.3节：使用 %<>% 进行赋值	73
第21.4节：使用 %\$% 访问内容	73
第21.5节：使用 %T>% 创建副作用	74
第21.6节：在dplyr和ggplot2中使用管道操作符	75
第22章：线性模型（回归）	76
第22.1节：mtcars数据集上的线性回归	76
第22.2节：使用“predict”函数	78
第22.3节：加权	79
第22.4节：使用多项式回归检查非线性	81
第22.5节：绘制回归（基础）	83
第22.6节：质量评估	85
第23章：data.table	87

Section 12.2: Go to the End of the Month	41
Section 12.3: Go to First Day of the Month	42
Section 12.4: Move a date a number of months consistently by months	42
Chapter 13: The Date class	44
Section 13.1: Formatting Dates	44
Section 13.2: Parsing Strings into Date Objects	44
Section 13.3: Dates	45
Chapter 14: Date-time classes (POSIXct and POSIXlt)	47
Section 14.1: Formatting and printing date-time objects	47
Section 14.2: Date-time arithmetic	47
Section 14.3: Parsing strings into date-time objects	48
Chapter 15: The character class	50
Section 15.1: Coercion	50
Chapter 16: Numeric classes and storage modes	51
Section 16.1: Numeric	51
Chapter 17: The logical class	53
Section 17.1: Logical operators	53
Section 17.2: Coercion	53
Section 17.3: Interpretation of NAs	53
Chapter 18: Data frames	55
Section 18.1: Create an empty data.frame	55
Section 18.2: Subsetting rows and columns from a data frame	56
Section 18.3: Convenience functions to manipulate data.frames	59
Section 18.4: Introduction	60
Section 18.5: Convert all columns of a data.frame to character class	61
Chapter 19: Split function	63
Section 19.1: Using split in the split-apply-combine paradigm	63
Section 19.2: Basic usage of split	64
Chapter 20: Reading and writing tabular data in plain-text files (CSV, TSV, etc.)	67
Section 20.1: Importing .csv files	67
Section 20.2: Importing with data.table	68
Section 20.3: Exporting .csv files	69
Section 20.4: Import multiple csv files	69
Section 20.5: Importing fixed-width files	69
Chapter 21: Pipe operators (%>% and others)	71
Section 21.1: Basic use and chaining	71
Section 21.2: Functional sequences	72
Section 21.3: Assignment with %<>%	73
Section 21.4: Exposing contents with %\$%	73
Section 21.5: Creating side effects with %T>%	74
Section 21.6: Using the pipe with dplyr and ggplot2	75
Chapter 22: Linear Models (Regression)	76
Section 22.1: Linear regression on the mtcars dataset	76
Section 22.2: Using the ‘predict’ function	78
Section 22.3: Weighting	79
Section 22.4: Checking for nonlinearity with polynomial regression	81
Section 22.5: Plotting The Regression (base)	83
Section 22.6: Quality assessment	85
Chapter 23: data.table	87

第23.1节：创建data.table	87
第23.2节：data.table中的特殊符号	88
第23.3节：添加和修改列	89
第23.4节：编写兼容data.frame和data.table的代码	91
第23.5节：在data.table中设置键	93
第24章：使用data.table进行透视和反透视	95
第24.1节：使用data.table进行表格数据的透视和反透视 - 一	95
第24.2节：使用data.table进行表格数据的透视和反透视 - 二	96
第25章：条形图	98
第25.1节：barplot()函数	98
第26章：基础绘图	104
第26.1节：密度图	104
第26.2节：组合图	105
第26.3节：R绘图入门	107
第26.4节：基本图形	108
第26.5节：直方图	111
第26.6节：Matplot	113
第26.7节：经验累积分布函数	119
第27章：箱线图	121
第27.1节：使用boxplot() {graphics} 创建箱线图	121
第27.2节：箱线图的附加样式参数	125
第28章：ggplot2	128
第28.1节：显示多个图形	128
第28.2节：为绘图准备数据	131
第28.3节：向图表添加水平线和垂直线	133
第28.4节：散点图	136
第28.5节：使用qplot绘制基本图形	136
第28.6节：垂直和水平条形图	138
第28.7节：小提琴图	140
第29章：因子	143
第29.1节：使用列表合并因子水平	143
第29.2节：因子的基本创建	144
第29.3节：更改和重新排序因子	145
第29.4节：从零重建因子	150
第30章：模式匹配与替换	152
第30.1节：查找匹配项	152
第30.2节：单次匹配与全局匹配	153
第30.3节：进行替换	154
第30.4节：在大数据集中查找匹配项	154
第31章：游程编码	156
第31.1节：使用`rle`的游程编码	156
第31.2节：在基础R中识别和按游程分组	156
第31.3节：游程编码用于压缩和解压向量	157
第31.4节：在data.table中识别和按游程分组	158
第32章：加速难以向量化的代码	159
第32.1节：使用Rcpp加速难以向量化的for循环	159
第32.2节：通过字节编译加速难以向量化的for循环	159
第33章：地理地图简介	161
第33.1节：使用maps包中的map()进行基础制图	161

Section 23.1: Creating a data.table	87
Section 23.2: Special symbols in data.table	88
Section 23.3: Adding and modifying columns	89
Section 23.4: Writing code compatible with both data.frame and data.table	91
Section 23.5: Setting keys in data.table	93
Chapter 24: Pivot and unpivot with data.table	95
Section 24.1: Pivot and unpivot tabular data with data.table - I	95
Section 24.2: Pivot and unpivot tabular data with data.table - II	96
Chapter 25: Bar Chart	98
Section 25.1: barplot() function	98
Chapter 26: Base Plotting	104
Section 26.1: Density plot	104
Section 26.2: Combining Plots	105
Section 26.3: Getting Started with R_Plots	107
Section 26.4: Basic Plot	108
Section 26.5: Histograms	111
Section 26.6: Matplot	113
Section 26.7: Empirical Cumulative Distribution Function	119
Chapter 27: boxplot	121
Section 27.1: Create a box-and-whisker plot with boxplot() {graphics}	121
Section 27.2: Additional boxplot style parameters	125
Chapter 28: ggplot2	128
Section 28.1: Displaying multiple plots	128
Section 28.2: Prepare your data for plotting	131
Section 28.3: Add horizontal and vertical lines to plot	133
Section 28.4: Scatter Plots	136
Section 28.5: Produce basic plots with qplot	136
Section 28.6: Vertical and Horizontal Bar Chart	138
Section 28.7: Violin plot	140
Chapter 29: Factors	143
Section 29.1: Consolidating Factor Levels with a List	143
Section 29.2: Basic creation of factors	144
Section 29.3: Changing and reordering factors	145
Section 29.4: Rebuilding factors from zero	150
Chapter 30: Pattern Matching and Replacement	152
Section 30.1: Finding Matches	152
Section 30.2: Single and Global match	153
Section 30.3: Making substitutions	154
Section 30.4: Find matches in big data sets	154
Chapter 31: Run-length encoding	156
Section 31.1: Run-length Encoding with `rle`	156
Section 31.2: Identifying and grouping by runs in base R	156
Section 31.3: Run-length encoding to compress and decompress vectors	157
Section 31.4: Identifying and grouping by runs in data.table	158
Chapter 32: Speeding up tough-to-vectorize code	159
Section 32.1: Speeding tough-to-vectorize for loops with Rcpp	159
Section 32.2: Speeding tough-to-vectorize for loops by byte compiling	159
Chapter 33: Introduction to Geographical Maps	161
Section 33.1: Basic map-making with map() from the package maps	161

第33.2节：50个州地图及使用Google Viz的高级分级图	164
第33.3节：交互式plotly地图	165
第33.4节：使用Leaflet制作动态HTML地图	167
第33.5节：Shiny应用中的动态Leaflet地图	168
第34章：集合运算	171
第34.1节：向量对的集合运算符	171
第34.2节：向量的笛卡尔积或“叉积”	171
第34.3节：向量的集合成员资格	172
第34.4节：从向量中去重 / 删除重复项 / 选择不同元素	172
第34.5节：测量集合重叠 / 向量的韦恩图	173
第35章：tidyverse	174
第35.1节：tidyverse概述	174
第35.2节：创建tbl_df	175
第36章：Rcpp	176
第36.1节：使用插件扩展Rcpp	176
第36.2节：内联代码编译	176
第36.3节：Rcpp属性	177
第36.4节：指定额外的构建依赖	178
第37章：随机数生成器	179
第37.1节：随机排列	179
第37.2节：使用各种密度函数生成随机数	179
第37.3节：随机数生成器的可重复性	181
第38章：并行处理	182
第38.1节：使用parallel包进行并行处理	182
第38.2节：使用foreach包的并行处理	183
第38.3节：随机数生成	184
第38.4节：mparallelDo	184
第39章：子集选择	186
第39.1节：数据框	186
第39.2节：原子向量	187
第39.3节：矩阵	188
第39.4节：列表	190
第39.5节：向量索引	191
第39.6节：其他对象	192
第39.7节：矩阵逐元素操作	192
第40章：调试	194
第40.1节：使用调试	194
第40.2节：使用浏览器	194
第41章：安装软件包	196
第41.1节：从GitHub安装软件包	196
第41.2节：从仓库下载并安装软件包	197
第41.3节：从本地源安装软件包	198
第41.4节：安装软件包的本地开发版本	198
第41.5节：使用命令行界面包管理器——pacman的基本用法	199
第42章：检查软件包	200
第42.1节：查看软件包版本	200
第42.2节：查看当前会话中加载的软件包	200
第42.3节：查看软件包信息	200
第42.4节：查看软件包内置的数据集	200

Section 33.2: 50 State Maps and Advanced Choropleths with Google Viz	164
Section 33.3: Interactive plotly maps	165
Section 33.4: Making Dynamic HTML Maps with Leaflet	167
Section 33.5: Dynamic Leaflet maps in Shiny applications	168
Chapter 34: Set operations	171
Section 34.1: Set operators for pairs of vectors	171
Section 34.2: Cartesian or "cross" products of vectors	171
Section 34.3: Set membership for vectors	172
Section 34.4: Make unique / drop duplicates / select distinct elements from a vector	172
Section 34.5: Measuring set overlaps / Venn diagrams for vectors	173
Chapter 35: tidyverse	174
Section 35.1: tidyverse: an overview	174
Section 35.2: Creating tbl_df's	175
Chapter 36: Rcpp	176
Section 36.1: Extending Rcpp with Plugins	176
Section 36.2: Inline Code Compile	176
Section 36.3: Rcpp Attributes	177
Section 36.4: Specifying Additional Build Dependencies	178
Chapter 37: Random Numbers Generator	179
Section 37.1: Random permutations	179
Section 37.2: Generating random numbers using various density functions	179
Section 37.3: Random number generator's reproducibility	181
Chapter 38: Parallel processing	182
Section 38.1: Parallel processing with parallel package	182
Section 38.2: Parallel processing with foreach package	183
Section 38.3: Random Number Generation	184
Section 38.4: mparallelDo	184
Chapter 39: Subsetting	186
Section 39.1: Data frames	186
Section 39.2: Atomic vectors	187
Section 39.3: Matrices	188
Section 39.4: Lists	190
Section 39.5: Vector indexing	191
Section 39.6: Other objects	192
Section 39.7: Elementwise Matrix Operations	192
Chapter 40: Debugging	194
Section 40.1: Using debug	194
Section 40.2: Using browser	194
Chapter 41: Installing packages	196
Section 41.1: Install packages from GitHub	196
Section 41.2: Download and install packages from repositories	197
Section 41.3: Install package from local source	198
Section 41.4: Install local development version of a package	198
Section 41.5: Using a CLI package manager -- basic pacman usage	199
Chapter 42: Inspecting packages	200
Section 42.1: View Package Version	200
Section 42.2: View Loaded packages in Current Session	200
Section 42.3: View package information	200
Section 42.4: View package's built-in data sets	200

第42.5节：列出软件包导出的函数	200
第43章：使用devtools创建包	201
第43.1节：创建和分发包	201
第43.2节：创建示例文档	203
第44章：在自己的包中使用管道赋值 %<>%：如何操作？	204
第44.1节：将管道放入实用函数文件	204
第45章：Arima模型	205
第45.1节：使用Arima建模AR1过程	205
第46章：分布函数	210
第46.1节：正态分布	210
第46.2节：二项分布	210
第47章：Shiny	214
第47.1节：创建应用程序	214
第47.2节：复选框组	214
第47.3节：单选按钮	215
第47.4节：调试	216
第47.5节：选择框	216
第47.6节：启动Shiny应用	217
第47.7节：控件部件	218
第48章：空间分析	220
第48.1节：从XY数据集创建空间点	220
第48.2节：导入形状文件 (.shp)	221
第49章：sqldf	222
第49.1节：基本用法示例	222
第50章：代码性能分析	224
第50.1节：使用microbenchmark进行基准测试	224
第50.2节：proc.time()	225
第50.3节：微基准测试	226
第50.4节：System.time	227
第50.5节：行级性能分析	227
第51章：控制流结构	229
第51.1节：For循环的最优构造	229
第51.2节：基本的For循环构造	230
第51.3节：其他循环结构：while和repeat	230
第52章：按列操作	234
第52.1节：每列求和	234
第53章：JSON	236
第53.1节：JSON与R对象之间的转换	236
第54章：RODBC	238
第54.1节：通过RODBC连接Excel文件	238
第54.2节：SQL Server管理数据库连接以获取单个表	238
第54.3节：连接关系型数据库	238
第55章：lubridate	239
第55.1节：使用lubridate从字符串解析日期和日期时间	239
第55.2节：期间与持续时间的区别	240
第55.3节：瞬时点	240
第55.4节：区间、持续时间和周期	241
第55.5节：在lubridate中操作日期和时间	242

Section 42.5: List a package's exported functions	200
Chapter 43: Creating packages with devtools	201
Section 43.1: Creating and distributing packages	201
Section 43.2: Creating vignettes	203
Chapter 44: Using pipe assignment in your own package %<>%: How to ?	204
Section 44.1: Putting the pipe in a utility-functions file	204
Chapter 45: Arima Models	205
Section 45.1: Modeling an AR1 Process with Arima	205
Chapter 46: Distribution Functions	210
Section 46.1: Normal distribution	210
Section 46.2: Binomial Distribution	210
Chapter 47: Shiny	214
Section 47.1: Create an app	214
Section 47.2: Checkbox Group	214
Section 47.3: Radio Button	215
Section 47.4: Debugging	216
Section 47.5: Select box	216
Section 47.6: Launch a Shiny app	217
Section 47.7: Control widgets	218
Chapter 48: spatial analysis	220
Section 48.1: Create spatial points from XY data set	220
Section 48.2: Importing a shape file (.shp)	221
Chapter 49: sqldf	222
Section 49.1: Basic Usage Examples	222
Chapter 50: Code profiling	224
Section 50.1: Benchmarking using microbenchmark	224
Section 50.2: proc.time()	225
Section 50.3: Microbenchmark	226
Section 50.4: System.time	227
Section 50.5: Line Profiling	227
Chapter 51: Control flow structures	229
Section 51.1: Optimal Construction of a For Loop	229
Section 51.2: Basic For Loop Construction	230
Section 51.3: The Other Looping Constructs: while and repeat	230
Chapter 52: Column wise operation	234
Section 52.1: sum of each column	234
Chapter 53: JSON	236
Section 53.1: JSON to / from R objects	236
Chapter 54: RODBC	238
Section 54.1: Connecting to Excel Files via RODBC	238
Section 54.2: SQL Server Management Database connection to get individual table	238
Section 54.3: Connecting to relational databases	238
Chapter 55: lubridate	239
Section 55.1: Parsing dates and datetimes from strings with lubridate	239
Section 55.2: Difference between period and duration	240
Section 55.3: Instants	240
Section 55.4: Intervals, Durations and Periods	241
Section 55.5: Manipulating date and time in lubridate	242

第55.6节：时区	243
第55.7节：在lubridate中解析日期和时间	243
第55.8节：日期的四舍五入	243
第56章：时间序列与预测	245
第56.1节：创建ts对象	245
第56.2节：时间序列数据的探索性数据分析	245
第57章：strsplit函数	247
第57.1节：介绍	247
第58章：网页抓取与解析	248
第58.1节：使用rvest进行基础抓取	248
第58.2节：需要登录时使用rvest	248
第59章：广义线性模型	250
第59.1节：泰坦尼克号数据集上的逻辑回归	250
第60章：长格式与宽格式数据的重塑	253
第60.1节：数据重塑	253
第60.2节：reshape函数	254
第61章：RMarkdown和knitr演示	256
第61.1节：向ioslides演示文稿添加页脚	256
第61.2节：Rstudio示例	257
第62章：变量的作用域	259
第62.1节：环境与函数	259
第62.2节：函数退出	259
第62.3节：子函数	260
第62.4节：全局分配	260
第62.5节：环境和变量的显式赋值	261
第63章：执行置换检验	262
第63.1节：一个相当通用的函数	262
第64章：xgboost	265
第64.1节：使用xgboost进行交叉验证和调优	265
第65章：R代码向量化最佳实践	267
第65.1节：通过行操作	267
第66章：缺失值	270
第66.1节：检查缺失数据	270
第66.2节：读取和写入包含NA值的数据	270
第66.3节：使用不同类别的NA	270
第66.4节：TRUE/FALSE和/or NA	271
第67章：分层线性模型	272
第67.1节：基本模型拟合	272
第68章：*应用函数族（泛函）	273
第68.1节：使用内置泛函	273
第68.2节：合并多个`data.frames`(`lapply`, `mapply`)	273
第68.3节：批量文件加载	275
第68.4节：使用用户定义的泛函	275
第69章：文本挖掘	277
第69.1节：抓取数据以构建N-gram词云	277
第70章：方差分析（ANOVA）	281
第70.1节：aov()的基本用法	281
第70.2节：Anova()的基本用法	281

Section 55.6: Time Zones	243
Section 55.7: Parsing date and time in lubridate	243
Section 55.8: Rounding dates	243
Chapter 56: Time Series and Forecasting	245
Section 56.1: Creating a ts object	245
Section 56.2: Exploratory Data Analysis with time-series data	245
Chapter 57: strsplit function	247
Section 57.1: Introduction	247
Chapter 58: Web scraping and parsing	248
Section 58.1: Basic scraping with rvest	248
Section 58.2: Using rvest when login is required	248
Chapter 59: Generalized linear models	250
Section 59.1: Logistic regression on Titanic dataset	250
Chapter 60: Reshaping data between long and wide forms	253
Section 60.1: Reshaping data	253
Section 60.2: The reshape function	254
Chapter 61: RMarkdown and knitr presentation	256
Section 61.1: Adding a footer to an ioslides presentation	256
Section 61.2: Rstudio example	257
Chapter 62: Scope of variables	259
Section 62.1: Environments and Functions	259
Section 62.2: Function Exit	259
Section 62.3: Sub functions	260
Section 62.4: Global Assignment	260
Section 62.5: Explicit Assignment of Environments and Variables	261
Chapter 63: Performing a Permutation Test	262
Section 63.1: A fairly general function	262
Chapter 64: xgboost	265
Section 64.1: Cross Validation and Tuning with xgboost	265
Chapter 65: R code vectorization best practices	267
Section 65.1: By row operations	267
Chapter 66: Missing values	270
Section 66.1: Examining missing data	270
Section 66.2: Reading and writing data with NA values	270
Section 66.3: Using NAs of different classes	270
Section 66.4: TRUE/FALSE and/or NA	271
Chapter 67: Hierarchical Linear Modeling	272
Section 67.1: basic model fitting	272
Chapter 68: *apply family of functions (functionals)	273
Section 68.1: Using built-in functionals	273
Section 68.2: Combining multiple `data.frames`(`lapply`, `mapply`)	273
Section 68.3: Bulk File Loading	275
Section 68.4: Using user-defined functionals	275
Chapter 69: Text mining	277
Section 69.1: Scraping Data to build N-gram Word Clouds	277
Chapter 70: ANOVA	281
Section 70.1: Basic usage of aov()	281
Section 70.2: Basic usage of Anova()	281

第71章：栅格和图像分析	283	Chapter 71: Raster and Image Analysis	283
第71.1节：计算GLCM纹理	283	Section 71.1: Calculating GLCM Texture	283
第71.2节：数学形态学	285	Section 71.2: Mathematical Morphologies	285
第72章：生存分析	287	Chapter 72: Survival analysis	287
第72.1节：使用randomForestSRC的随机森林生存分析	287	Section 72.1: Random Forest Survival Analysis with randomForestSRC	287
第72.2节：介绍——使用survival包进行参数生存模型的基本拟合和绘图包	288	Section 72.2: Introduction - basic fitting and plotting of parametric survival models with the survival package	288
第72.3节：使用survminer的Kaplan-Meier生存曲线估计和风险集表	289	Section 72.3: Kaplan Meier estimates of survival curves and risk set tables with survminer	289
第73章：容错/弹性代码	292	Chapter 73: Fault-tolerant/resilient code	292
第73.1节：使用tryCatch()	292	Section 73.1: Using tryCatch()	292
第74章：可复现的R	295	Chapter 74: Reproducible R	295
第74.1节：数据可复现性	295	Section 74.1: Data reproducibility	295
第74.2节：包的可复现性	295	Section 74.2: Package reproducibility	295
第75章：傅里叶级数与变换	296	Chapter 75: Fourier Series and Transformations	296
第75.1节：傅里叶级数	297	Section 75.1: Fourier Series	297
第76章：.Rprofile	302	Chapter 76: .Rprofile	302
第76.1节：.Rprofile - 执行的第一段代码	302	Section 76.1: .Rprofile - the first chunk of code executed	302
第76.2节：.Rprofile 示例	303	Section 76.2: .Rprofile example	303
第77章：dplyr	304	Chapter 77: dplyr	304
第77.1节：dplyr 的单表动词	304	Section 77.1: dplyr's single table verbs	304
第77.2节：使用 %>% (管道) 操作符进行聚合	311	Section 77.2: Aggregating with %>% (pipe) operator	311
第77.3节：子集观察 (行)	312	Section 77.3: Subset Observation (Rows)	312
第77.4节：dplyr中NSE和字符串变量的示例	313	Section 77.4: Examples of NSE and string variables in dplyr	313
第78章：caret	314	Chapter 78: caret	314
第78.1节：预处理	314	Section 78.1: Preprocessing	314
第79章：压缩档案中的文件提取与列出	315	Chapter 79: Extracting and Listing Files in Compressed Archives	315
第79.1节：从.zip档案中提取文件	315	Section 79.1: Extracting files from a .zip archive	315
第80章：使用R的概率分布	316	Chapter 80: Probability Distributions with R	316
第80.1节：R中不同分布的概率密度函数 (PDF) 和概率质量函数 (PMF)	316	Section 80.1: PDF and PMF for different distributions in R	316
第81章：使用knitr在LaTeX中应用R	317	Chapter 81: R in LaTeX with knitr	317
第81.1节：使用Knitr和代码外部化在LaTeX中应用R	317	Section 81.1: R in LaTeX with Knitr and Code Externalization	317
第81.2节：使用Knitr和内联代码块在LaTeX中应用R	317	Section 81.2: R in LaTeX with Knitr and Inline Code Chunks	317
第81.3节：使用Knitr和内部代码块在LaTeX中应用R	318	Section 81.3: R in LaTeX with Knitr and Internal Code Chunks	318
第82章：R语言中的网页爬取	319	Chapter 82: Web Crawling in R	319
第82.1节：使用RCurl包的标准爬取方法	319	Section 82.1: Standard scraping approach using the RCurl package	319
第83章：使用RMarkdown创建报告	320	Chapter 83: Creating reports with RMarkdown	320
第83.1节：包含参考文献	320	Section 83.1: Including bibliographies	320
第83.2节：包含LaTeX导言命令	320	Section 83.2: Including LaTeX Preamble Commands	320
第83.3节：打印表格	321	Section 83.3: Printing tables	321
第83.4节：基本的RMarkdown文档结构	323	Section 83.4: Basic R-markdown document structure	323
第84章：GPU加速计算	326	Chapter 84: GPU-accelerated computing	326
第84.1节：gpuR gpuMatrix对象	326	Section 84.1: gpuR gpuMatrix objects	326
第84.2节：gpuR vclMatrix对象	326	Section 84.2: gpuR vclMatrix objects	326
第85章：热图和heatmap.2	327	Chapter 85: heatmap and heatmap.2	327
第85.1节：官方文档中的示例	327	Section 85.1: Examples from the official documentation	327
第85.2节：heatmap.2中的参数调优	335	Section 85.2: Tuning parameters in heatmap.2	335
第86章：使用igraph包进行网络分析	341	Chapter 86: Network analysis with the igraph package	341
第86.1节：简单有向和无向网络图绘制	341	Section 86.1: Simple Directed and Non-directed Network Graphing	341

第87章：函数式编程	343
第87.1节：内置高阶函数	343
第88章：获取用户输入	344
第88.1节：R语言中的用户输入	344
第89章：Spark API (SparkR)	345
第89.1节：设置Spark上下文	345
第89.2节：缓存数据	345
第89.3节：创建RDD (弹性分布式数据集)	346
第90章：元数据：文档指南	347
第90.1节：风格	347
第90.2节：制作良好示例	347
第91章：输入与输出	348
第91.1节：读取和写入数据框	348
第92章：外部表的输入输出 (Excel、SAS、SPSS、Stata)	350
第92.1节：使用rio导入数据	350
第92.2节：读取和写入Stata、SPSS和SAS文件	350
第92.3节：导入Excel文件	351
第92.4节：Feather文件的导入或导出	354
第93章：数据库表的输入输出	356
第93.1节：从MySQL数据库读取数据	356
第93.2节：从MongoDB数据库读取数据	356
第94章：地理数据 (shapefile等) 的输入输出	357
第94.1节：导入和导出Shapefile	357
第95章：栅格图像的输入输出	358
第95.1节：加载多层栅格	358
第96章：R二进制格式的输入输出	360
第96.1节：Rds和RData (Rda) 文件	360
第96.2节：环境	360
第97章：回收	361
第97.1节：子集中的回收使用	361
第98章：表达式：解析 + 计算	362
第98.1节：执行字符串格式的代码	362
第99章：R中的正则表达式语法	363
第99.1节：使用`grep`在字符向量中查找字符串	363
第100章：正则表达式 (regex)	365
第100.1节：Perl和POSIX正则表达式的区别	365
第100.2节：验证“YYYYMMDD”格式的日期	365
第100.3节：在R正则表达式模式中转义字符	366
第100.4节：验证美国州邮政缩写	366
第100.5节：验证美国电话号码	366
第101章：组合学	368
第101.1节：枚举指定长度的组合	368
第101.2节：计算指定长度的组合数	369
第102章：在R中求解常微分方程 (ODE)	370
第102.1节：洛伦兹模型	370
第102.2节：洛特卡-沃尔泰拉模型：猎物与捕食者	371
第102.3节：编译语言中的常微分方程——在R中的定义	373
第102.4节：编译语言中的常微分方程 (ODE) ——C语言中的定义	373

Chapter 87: Functional programming	343
Section 87.1: Built-in Higher Order Functions	343
Chapter 88: Get user input	344
Section 88.1: User input in R	344
Chapter 89: Spark API (SparkR)	345
Section 89.1: Setup Spark context	345
Section 89.2: Cache data	345
Section 89.3: Create RDDs (Resilient Distributed Datasets)	346
Chapter 90: Meta: Documentation Guidelines	347
Section 90.1: Style	347
Section 90.2: Making good examples	347
Chapter 91: Input and output	348
Section 91.1: Reading and writing data frames	348
Chapter 92: I/O for foreign tables (Excel, SAS, SPSS, Stata)	350
Section 92.1: Importing data with rio	350
Section 92.2: Read and write Stata, SPSS and SAS files	350
Section 92.3: Importing Excel files	351
Section 92.4: Import or Export of Feather file	354
Chapter 93: I/O for database tables	356
Section 93.1: Reading Data from MySQL Databases	356
Section 93.2: Reading Data from MongoDB Databases	356
Chapter 94: I/O for geographic data (shapefiles, etc.)	357
Section 94.1: Import and Export Shapefiles	357
Chapter 95: I/O for raster images	358
Section 95.1: Load a multilayer raster	358
Chapter 96: I/O for R's binary format	360
Section 96.1: Rds and RData (Rda) files	360
Section 96.2: Environments	360
Chapter 97: Recycling	361
Section 97.1: Recycling use in subsetting	361
Chapter 98: Expression: parse + eval	362
Section 98.1: Execute code in string format	362
Chapter 99: Regular Expression Syntax in R	363
Section 99.1: Use `grep` to find a string in a character vector	363
Chapter 100: Regular Expressions (regex)	365
Section 100.1: Differences between Perl and POSIX regex	365
Section 100.2: Validate a date in a "YYYYMMDD" format	365
Section 100.3: Escaping characters in R regex patterns	366
Section 100.4: Validate US States postal abbreviations	366
Section 100.5: Validate US phone numbers	366
Chapter 101: Combinatorics	368
Section 101.1: Enumerating combinations of a specified length	368
Section 101.2: Counting combinations of a specified length	369
Chapter 102: Solving ODEs in R	370
Section 102.1: The Lorenz model	370
Section 102.2: Lotka-Volterra or: Prey vs. predator	371
Section 102.3: ODEs in compiled languages - definition in R	373
Section 102.4: ODEs in compiled languages - definition in C	373

第102.5节：编译语言中的常微分方程（ODE）——Fortran语言中的定义	375
第102.6节：编译语言中的常微分方程（ODE）——基准测试	376
第103章：R语言中的特征选择——去除无关特征	378
第103.1节：去除方差为零或接近零的特征	378
第103.2节：去除缺失值（NA）数量较多的特征	378
第103.3节：移除高度相关的特征	378
第104章：RMD中的参考文献	380
第104.1节：指定参考文献和引用作者	380
第104.2节：行内引用	381
第104.3节：引用样式	382
第105章：在R中编写函数	385
第105.1节：匿名函数	385
第105.2节：RStudio代码片段	385
第105.3节：命名函数	386
第106章：图形的配色方案	388
第106.1节：viridis - 适合打印和色盲友好的调色板	388
第106.2节：一个方便查看颜色向量的函数	389
第106.3节：colorspace - 颜色的点击拖拽界面	390
第106.4节：色盲友好调色板	391
第106.5节：RColorBrewer	392
第106.6节：基本的R颜色函数	393
第107章：使用hclust进行层次聚类	394
第107.1节：示例1 - hclust的基本使用，树状图显示，聚类绘图	394
第107.2节：示例2 - 层次聚类和异常值	397
第108章：随机森林算法	400
第108.1节：基本示例 - 分类与回归	400
第109章：RESTful R服务	402
第109.1节：opencpu应用	402
第110章：机器学习	403
第110.1节：创建随机森林模型	403
第111章：使用texreg以论文格式导出模型	404
第111.1节：打印线性回归结果	404
第112章：发布	406
第112.1节：格式化表格	406
第112.2节：格式化整个文档	406
第113章：使用S4类实现状态机模式	407
第113.1节：使用状态机解析行	407
第114章：使用tidyverse进行重塑	419
第114.1节：使用spread()将长格式重塑为宽格式	419
第114.2节：使用gather()将宽格式重塑为长格式	419
第115章：通过替换修改字符串	421
第115.1节：使用捕获组重新排列字符串	421
第115.2节：消除重复的连续元素	421
第116章：非标准求值与标准求值	423
第116.1节：使用标准dplyr动词的示例	423
第117章：随机化	425
第117.1节：随机抽样和排列	425
第117.2节：设置随机种子	427

Section 102.5: ODEs in compiled languages - definition in fortran	375
Section 102.6: ODEs in compiled languages - a benchmark test	376
Chapter 103: Feature Selection in R -- Removing Extraneous Features	378
Section 103.1: Removing features with zero or near-zero variance	378
Section 103.2: Removing features with high numbers of NA	378
Section 103.3: Removing closely correlated features	378
Chapter 104: Bibliography in RMD	380
Section 104.1: Specifying a bibliography and cite authors	380
Section 104.2: Inline references	381
Section 104.3: Citation styles	382
Chapter 105: Writing functions in R	385
Section 105.1: Anonymous functions	385
Section 105.2: RStudio code snippets	385
Section 105.3: Named functions	386
Chapter 106: Color schemes for graphics	388
Section 106.1: viridis - print and colorblind friendly palettes	388
Section 106.2: A handy function to glimpse a vector of colors	389
Section 106.3: colorspace - click&drag interface for colors	390
Section 106.4: Colorblind-friendly palettes	391
Section 106.5: RColorBrewer	392
Section 106.6: basic R color functions	393
Chapter 107: Hierarchical clustering with hclust	394
Section 107.1: Example 1 - Basic use of hclust, display of dendrogram, plot clusters	394
Section 107.2: Example 2 - hclust and outliers	397
Chapter 108: Random Forest Algorithm	400
Section 108.1: Basic examples - Classification and Regression	400
Chapter 109: RESTful R Services	402
Section 109.1: opencpu Apps	402
Chapter 110: Machine learning	403
Section 110.1: Creating a Random Forest model	403
Chapter 111: Using texreg to export models in a paper-ready way	404
Section 111.1: Printing linear regression results	404
Chapter 112: Publishing	406
Section 112.1: Formatting tables	406
Section 112.2: Formatting entire documents	406
Chapter 113: Implement State Machine Pattern using S4 Class	407
Section 113.1: Parsing Lines using State Machine	407
Chapter 114: Reshape using tidyverse	419
Section 114.1: Reshape from long to wide format with spread()	419
Section 114.2: Reshape from wide to long format with gather()	419
Chapter 115: Modifying strings by substitution	421
Section 115.1: Rearrange character strings using capture groups	421
Section 115.2: Eliminate duplicated consecutive elements	421
Chapter 116: Non-standard evaluation and standard evaluation	423
Section 116.1: Examples with standard dplyr verbs	423
Chapter 117: Randomization	425
Section 117.1: Random draws and permutations	425
Section 117.2: Setting the seed	427

第118章：R语言中的面向对象编程	428
第118.1节：S3	428
第119章：强制转换	429
第119.1节：隐式强制转换	429
第120章：通过编写独立的R脚本来标准化分析	430
第120.1节：独立R程序的基本结构及其调用方法	430
第120.2节：使用littler执行R脚本	431
第121章：使用R分析推文	433
第121.1节：下载推文	433
第121.2节：获取推文文本	433
第122章：自然语言处理	435
第122.1节：创建词频矩阵	435
第123章：R Markdown笔记本（来自RStudio）	437
第123.1节：创建笔记本	437
第123.2节：插入代码块	437
第123.3节：执行代码块	438
第123.4节：执行进度	439
第123.5节：预览输出	440
第123.6节：保存与分享	440
第124章：聚合数据框	442
第124.1节：使用data.table进行聚合	442
第124.2节：使用基础R进行聚合	443
第124.3节：使用dplyr进行聚合	444
第125章：数据获取	446
第125.1节：内置数据集	446
第125.2节：访问开放数据库的包	446
第125.3节：访问受限数据的包	448
第125.4节：包内的数据集	452
第126章：通过示例回顾R	454
第126.1节：绘图（使用plot）	454
第126.2节：常用函数	454
第126.3节：数据类型	455
第127章：更新R版本	457
第127.1节：从R官网安装	457
第127.2节：使用installr包在R内更新	457
第127.3节：决定旧包的处理	457
第127.4节：更新包	459
第127.5节：检查R版本	459
鸣谢	460
你可能也喜欢	464

Chapter 118: Object-Oriented Programming in R	428
Section 118.1: S3	428
Chapter 119: Coercion	429
Section 119.1: Implicit Coercion	429
Chapter 120: Standardize analyses by writing standalone R scripts	430
Section 120.1: The basic structure of standalone R program and how to call it	430
Section 120.2: Using littler to execute R scripts	431
Chapter 121: Analyze tweets with R	433
Section 121.1: Download Tweets	433
Section 121.2: Get text of tweets	433
Chapter 122: Natural language processing	435
Section 122.1: Create a term frequency matrix	435
Chapter 123: R Markdown Notebooks (from RStudio)	437
Section 123.1: Creating a Notebook	437
Section 123.2: Inserting Chunks	437
Section 123.3: Executing Chunk Code	438
Section 123.4: Execution Progress	439
Section 123.5: Preview Output	440
Section 123.6: Saving and Sharing	440
Chapter 124: Aggregating data frames	442
Section 124.1: Aggregating with data.table	442
Section 124.2: Aggregating with base R	443
Section 124.3: Aggregating with dplyr	444
Chapter 125: Data acquisition	446
Section 125.1: Built-in datasets	446
Section 125.2: Packages to access open databases	446
Section 125.3: Packages to access restricted data	448
Section 125.4: Datasets within packages	452
Chapter 126: R memento by examples	454
Section 126.1: Plotting (using plot)	454
Section 126.2: Commonly used functions	454
Section 126.3: Data types	455
Chapter 127: Updating R version	457
Section 127.1: Installing from R Website	457
Section 127.2: Updating from within R using installr Package	457
Section 127.3: Deciding on the old packages	457
Section 127.4: Updating Packages	459
Section 127.5: Check R Version	459
Credits	460
You may also like	464

请随意免费与任何人分享此 PDF,
本书的最新版本可从以下网址下载：

<https://goalkicker.com/RBook>

本 *R* 专业人士笔记一书汇编自 [Stack Overflow Documentation](#), 内容由 Stack Overflow 的优秀人士撰写。
文本内容采用知识共享署名-相同方式共享许可协议发布, 详见本书末尾对各章节贡献者的致谢。图片版权归各自所有者所有, 除非另有说明。

这是一本非官方的免费书籍, 旨在教育用途, 与官方 R 组织或公司及 Stack Overflow 无关。所有商标和注册商标均为其各自公司所有者所有。

本书所提供的信息不保证正确或准确, 使用风险自负。

请将反馈和更正发送至 web@petercv.com

Please feel free to share this PDF with anyone for free,
latest version of this book can be downloaded from:

<https://goalkicker.com/RBook>

This *R Notes for Professionals* book is compiled from [Stack Overflow Documentation](#), the content is written by the beautiful people at Stack Overflow.
Text content is released under Creative Commons BY-SA, see credits at the end of this book whom contributed to the various chapters. Images may be copyright of their respective owners unless otherwise specified

This is an unofficial free book created for educational purposes and is not affiliated with official R group(s) or company(s) nor Stack Overflow. All trademarks and registered trademarks are the property of their respective company owners

The information presented in this book is not guaranteed to be correct nor accurate, use at your own risk

Please send feedback and corrections to web@petercv.com

第1章：R语言入门

第1.1节：安装R

你可能希望在安装R之后安装[RStudio](#)。RStudio是一个R的开发环境，简化了许多编程任务。

仅限Windows：

[Visual Studio](#)（从2015年更新3版开始）现在提供了一个名为R Tools的R开发环境，包括实时解释器、智能感知(IntelliSense)和调试模块。如果你选择这种方法，就不必按照以下章节指定的步骤安装R。

适用于Windows

1. 访问CRAN网站，点击下载Windows版R，下载最新版本的R。
2. 右键点击安装程序文件，以管理员身份运行。
3. 选择安装的操作语言。
4. 按照安装说明进行操作。

适用于OSX / macOS

备选方案1

(0. 确保已安装XQuartz)

1. 访问CRAN网站并下载最新版本的R。
2. 打开磁盘映像并运行安装程序。
3. 按照安装说明进行操作。

这将安装R和R-MacGUI。它会将GUI放在/Applications/文件夹中，命名为R.app，可以双击打开或拖动到Dock。当发布新版本时，重新安装过程会覆盖R.app，但之前的主要版本会被保留。实际的R代码位于

/Library/Frameworks/R.Framework/Versions/目录中。也可以在RStudio中使用R，使用相同的R代码但界面不同。

备选方案2

1. 按照<https://brew.sh/>上的说明安装homebrew (macOS缺失的软件包管理器)
2. brew install R

选择第二种方法的人应注意，Mac分支的维护者不建议这样做，并且不会在R-SIG-Mac邮件列表中回复有关使用困难的问题。

对于 Debian、Ubuntu 及其衍生版

您可以通过 apt-get 获取与您的发行版对应的 R 版本。然而，该版本通常会远远落后于 CRAN 上提供的最新版本。您可以将 CRAN 添加到您的“源”列表中。

`sudo apt-get install r-base`

您可以通过将 CRAN 添加到您的源列表，直接从 CRAN 获取更新的版本。请按照 CRAN 的[说明](#)进行操作。特别注意还需要执行此操作，以便您可以使用

Chapter 1: Getting started with R Language

Section 1.1: Installing R

You might wish to install [RStudio](#) after you have installed R. RStudio is a development environment for R that simplifies many programming tasks.

Windows only:

[Visual Studio](#) (starting from version 2015 Update 3) now features a development environment for R called [R Tools](#), that includes a live interpreter, IntelliSense, and a debugging module. If you choose this method, you won't have to install R as specified in the following section.

For Windows

1. Go to the [CRAN](#) website, click on download R for Windows, and download the latest version of R.
2. Right-click the installer file and RUN as administrator.
3. Select the operational language for installation.
4. Follow the instructions for installation.

For OSX / macOS

Alternative 1

(0. Ensure [XQuartz](#) is installed)

1. Go to the [CRAN](#) website and download the latest version of R.
2. Open the disk image and run the installer.
3. Follow the instructions for installation.

This will install both R and the R-MacGUI. It will put the GUI in the /Applications/ Folder as R.app where it can either be double-clicked or dragged to the Dock. When a new version is released, the (re)-installation process will overwrite R.app but prior major versions of R will be maintained. The actual R code will be in the /Library/Frameworks/R.Framework/Versions/ directory. Using R within RStudio is also possible and would be using the same R code with a different GUI.

Alternative 2

1. Install homebrew (the missing package manager for macOS) by following the instructions on <https://brew.sh/>
2. brew install R

Those choosing the second method should be aware that the maintainer of the Mac fork advises against it, and will not respond to questions about difficulties on the R-SIG-Mac Mailing List.

For Debian, Ubuntu and derivatives

You can get the version of R corresponding to your distro via apt-get. However, this version will frequently be quite far behind the most recent version available on CRAN. You can add CRAN to your list of recognized "sources".

`sudo apt-get install r-base`

You can get a more recent version directly from CRAN by adding CRAN to your sources list. Follow the [directions](#) from CRAN for more details. Note in particular the need to also execute this so that you can use

install.packages()。Linux 软件包通常以源文件形式分发，需要编译：

```
sudo apt-get install r-base-dev
```

对于 Red Hat 和 Fedora

```
sudo dnf install R
```

对于 Archlinux

R 直接在 Extra 软件包仓库中提供。

```
sudo pacman -S r
```

关于在Archlinux下使用R的更多信息可以在ArchWiki R页面找到。

第1.2节：你好，世界！

“你好，世界！”

另外，请查看关于如何、何时、是否以及为何打印字符串的详细讨论。

第1.3节：获取帮助

你可以使用函数help()或?来访问R的文档并搜索帮助。对于更广泛的搜索，你可以使用help.search()或??。

```
#关于R中help函数的帮助
```

```
help()
```

```
#关于paste函数的帮助
```

```
help(paste) #或者
```

```
help("paste") #或者
```

```
?paste #或者
```

```
?"paste"
```

访问 <https://www.r-project.org/help.html> 获取更多信息

第1.4节：交互模式和R脚本

交互模式

使用R最基本的方式是交互式模式。你输入命令，R会立即给出结果。

将R用作计算器

通过在操作系统的命令提示符下输入R或在Windows上执行RGui来启动R。下面是Linux上交互式R会话的截图：

install.packages(). Linux packages are usually distributed as source files and need compilation:

```
sudo apt-get install r-base-dev
```

For Red Hat and Fedora

```
sudo dnf install R
```

For Archlinux

R is directly available in the Extra package repo.

```
sudo pacman -S r
```

More info on using R under Archlinux can be found on the [ArchWiki R page](#).

Section 1.2: Hello World!

“Hello World!”

Also, check out the detailed discussion of how, when, whether and why to print a string.

Section 1.3: Getting Help

You can use function `help()` or `?` to access documentations and search for help in R. For even more general searches, you can use `help.search()` or `??`.

```
#For help on the help function of R  
help()
```

```
#For help on the paste function  
help(paste) #OR  
help("paste") #OR  
?paste #OR  
?"paste"
```

Visit <https://www.r-project.org/help.html> for additional information

Section 1.4: Interactive mode and R scripts

The interactive mode

The most basic way to use R is the *interactive* mode. You type commands and immediately get the result from R.

Using R as a calculator

Start R by typing R at the command prompt of your operating system or by executing RGui on Windows. Below you can see a screenshot of an interactive R session on Linux:

```
user:~$ R
```

```
R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R ist freie Software und kommt OHNE JEGLICHE GARANTIE.
Sie sind eingeladen, es unter bestimmten Bedingungen weiter zu verbreiten.
Tippen Sie 'license()' or 'licence()' für Details dazu.

R ist ein Gemeinschaftsprojekt mit vielen Beitragenden.
Tippen Sie 'contributors()' für mehr Information und 'citation()', um zu erfahren, wie R oder R packages in Publikationen zitiert werden können.

Tippen Sie 'demo()' für einige Demos, 'help()' für on-line Hilfe, oder 'help.start()' für eine HTML Browserschnittstelle zur Hilfe.
Tippen Sie 'q()', um R zu verlassen.

> 1+1
[1] 2
> |
```

```
user:~$ R
```

```
R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

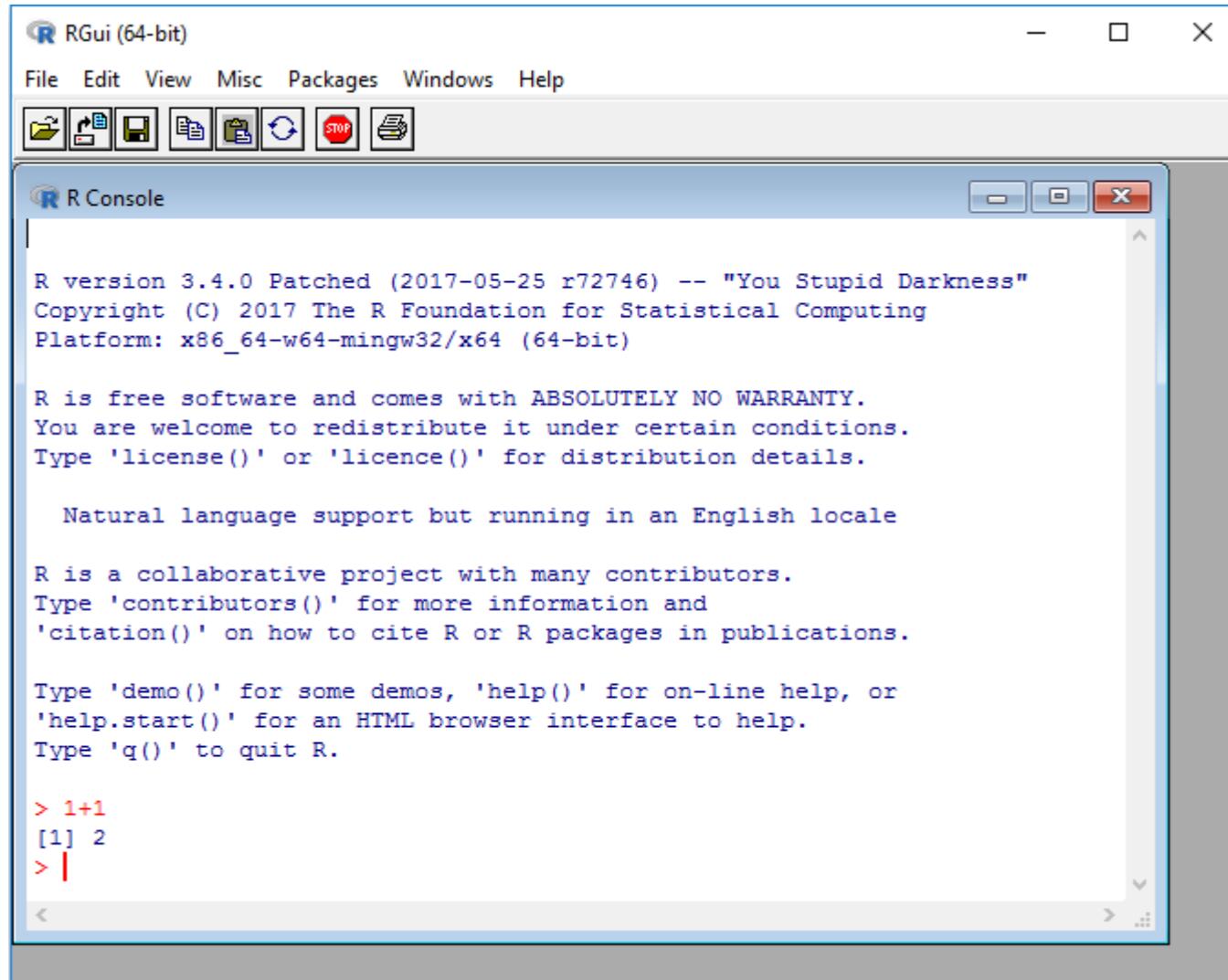
R ist freie Software und kommt OHNE JEGLICHE GARANTIE.
Sie sind eingeladen, es unter bestimmten Bedingungen weiter zu verbreiten.
Tippen Sie 'license()' or 'licence()' für Details dazu.

R ist ein Gemeinschaftsprojekt mit vielen Beitragenden.
Tippen Sie 'contributors()' für mehr Information und 'citation()', um zu erfahren, wie R oder R packages in Publikationen zitiert werden können.

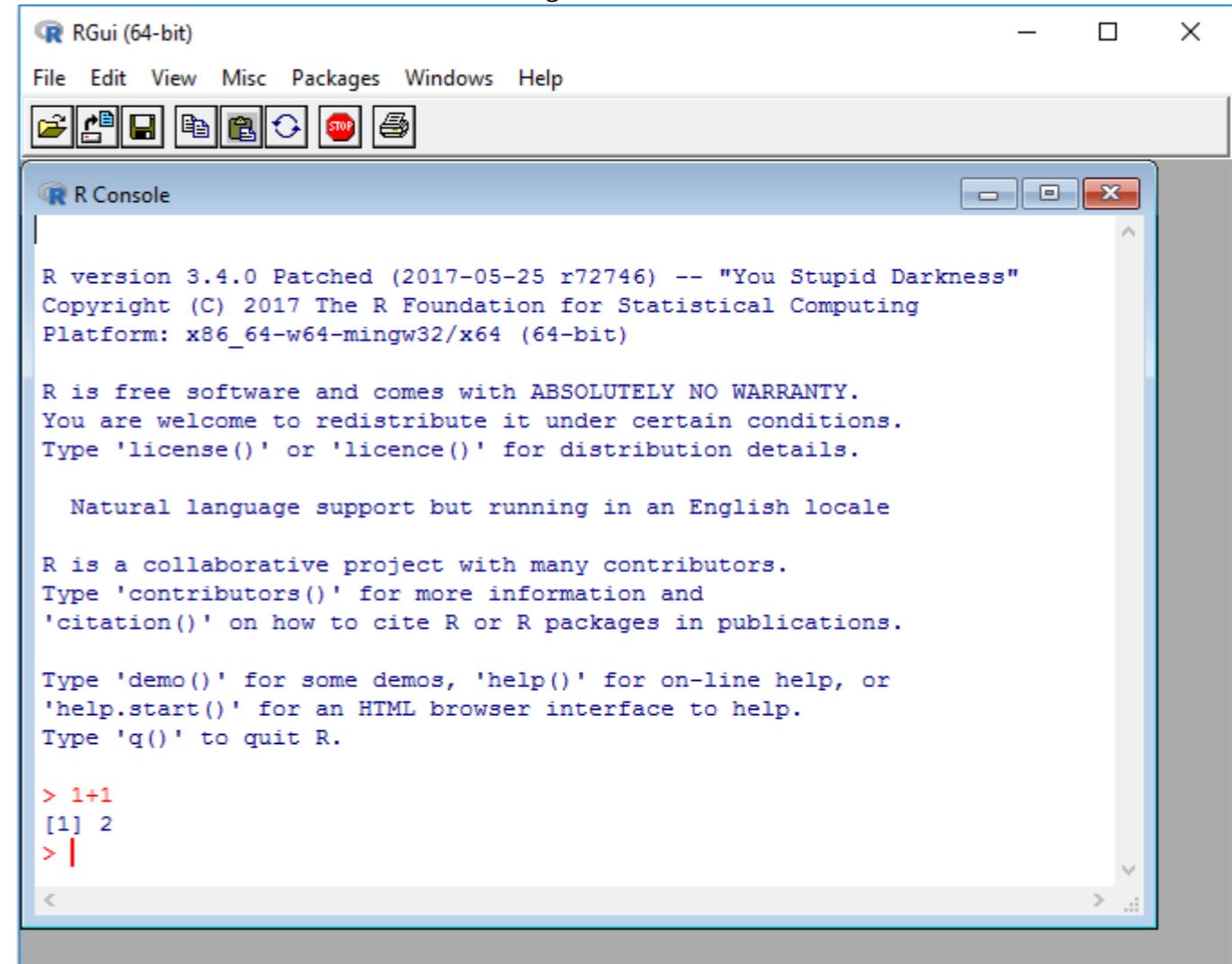
Tippen Sie 'demo()' für einige Demos, 'help()' für on-line Hilfe, oder 'help.start()' für eine HTML Browserschnittstelle zur Hilfe.
Tippen Sie 'q()', um R zu verlassen.

> 1+1
[1] 2
> |
```

这是Windows上的RGui，是Windows下R最基本的工作环境：



This is RGUi on Windows, the most basic working environment for R under Windows:



在>符号后可以输入表达式。输入表达式后，R会显示结果。在上面的截图中，R被用作计算器：输入

After the > sign, expressions can be typed in. Once an expression is typed, the result is shown by R. In the screenshot above, R is used as a calculator: Type

立即看到结果，2。前导的[1]表示R返回一个向量。在这种情况下，向量只包含一个数字（2）。

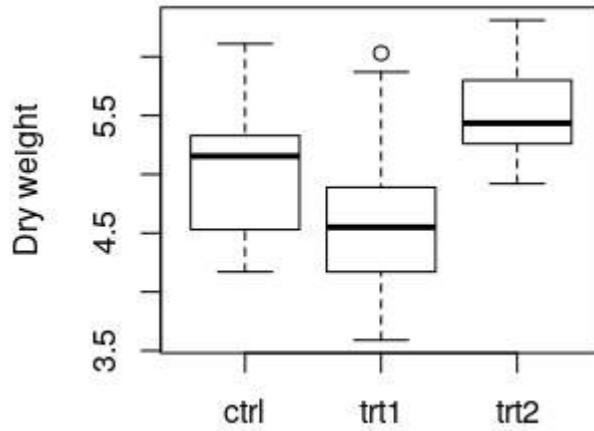
第一个图

R可以用来生成图形。以下示例使用数据集PlantGrowth，该数据集作为示例数据集随R一起提供

将以下所有不以##开头的行输入R提示符。以##开头的行用于记录R将返回的结果。

```
data(PlantGrowth)
str(PlantGrowth)
## 'data.frame': 30个观测值, 2个变量:
## $ weight: 数值 4.17 5.58 5.18 6.11 4.5 4.61 5.17 4.53 5.33 5.14 ...
## $ group : 因子, 有3个水平 "ctrl", "trt1", ...: 1 1 1 1 1 1 1 1 1 1 ...
anova(lm(weight ~ group, data = PlantGrowth))
## 方差分析表
##
## 响应变量: weight
##      自由度 平方和 均方 F值 显著性概率
## 组别    2 3.7663 1.8832 4.8461 0.01591 *
## 残差   27 10.4921 0.3886
## ---
## 显著性代码: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
boxplot(weight ~ group, data = PlantGrowth, ylab = "干重")
```

生成以下图形：



`data(PlantGrowth)` 加载示例数据集 `PlantGrowth`，该数据集记录了植物的干重，这些植物接受了两种不同的处理条件或未接受任何处理（对照组）。该数据集以 `PlantGrowth` 命名。这样的名称也称为变量名。

要加载您自己的数据，以下两个文档页面可能会有所帮助：

- 以纯文本文件格式读取和写入表格数据（CSV、TSV等）
- 外部表格的输入输出（Excel、SAS、SPSS、Stata）

`str(PlantGrowth)` 显示已加载数据集的信息。输出表明 `PlantGrowth` 是一个 `data.frame`，即 R 中表格的名称。该 `data.frame` 包含两列和 30 行。在此情况下，每行对应一株植物。两列的详细信息显示在以 \$ 开头的行中：第一列

to immediately see the result, 2. The leading [1] indicates that R returns a vector. In this case, the vector contains only one number (2).

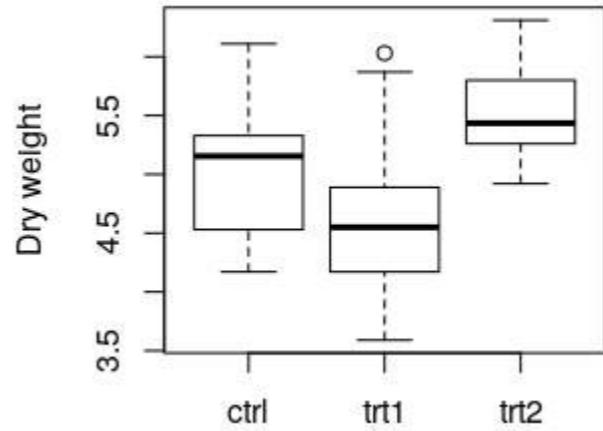
The first plot

R can be used to generate plots. The following example uses the data set `PlantGrowth`, which comes as an example data set along with R

Type int the following all lines into the R prompt which do not start with ##. Lines starting with ## are meant to document the result which R will return.

```
data(PlantGrowth)
str(PlantGrowth)
## 'data.frame': 30 obs. of 2 variables:
## $ weight: num 4.17 5.58 5.18 6.11 4.5 4.61 5.17 4.53 5.33 5.14 ...
## $ group : Factor w/ 3 levels "ctrl", "trt1", ...: 1 1 1 1 1 1 1 1 1 1 ...
anova(lm(weight ~ group, data = PlantGrowth))
## Analysis of Variance Table
##
## Response: weight
##            Df Sum Sq Mean Sq F value Pr(>F)
## group      2 3.7663 1.8832 4.8461 0.01591 *
## Residuals 27 10.4921 0.3886
## ---
## Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
boxplot(weight ~ group, data = PlantGrowth, ylab = "Dry weight")
```

The following plot is created:



`data(PlantGrowth)` loads the example data set `PlantGrowth`, which is records of dry masses of plants which were subject to two different treatment conditions or no treatment at all (control group). The data set is made available under the name `PlantGrowth`. Such a name is also called a Variable.

To load your own data, the following two documentation pages might be helpful:

- Reading and writing tabular data in plain-text files (CSV, TSV, etc.)
- I/O for foreign tables (Excel, SAS, SPSS, Stata)

`str(PlantGrowth)` shows information about the data set which was loaded. The output indicates that `PlantGrowth` is a `data.frame`, which is R's name for a table. The `data.frame` contains of two columns and 30 rows. In this case, each row corresponds to one plant. Details of the two columns are shown in the lines starting with \$: The first

该列称为weight，包含数字（num，即相应植物的干重）。第二列，group，包含植物所接受的处理。这是分类数据，在R中称为factor。阅读有关数据框的更多信息。

为了比较三组不同组的干重，使用anova(lm(...))进行单因素方差分析。weight ~ group 意思是“比较weight列的值，按group列的值分组”。这在R中称为公式。data = ... 指定数据所在表的名称。

结果显示，包括其他内容在内，三组中的某些组之间存在显著差异（列Pr(>F)，p = 0.01591）。必须进行事后检验，如Tukey检验，以确定哪些组的均值存在显著差异。

boxplot(...) 创建数据的箱线图。绘图的值来自哪里。weight ~ group 意思是：“绘制weight列的值与group列的值的关系。ylab = ... 指定y轴的标签。更多信息：基础绘图

输入 q() 或 [Ctrl]-[D] 退出 R 会话。

R 脚本

为了记录您的研究，最好将用于计算的命令保存到文件中。为此，您可以创建R脚本。R脚本是一个简单的文本文件，包含R命令。

创建一个名为plants.R的文本文件，并填入以下内容，其中一些命令来自上面的代码块：

```
data(PlantGrowth)  
  
anova(lm(weight ~ group, data = PlantGrowth))  
  
png("plant_boxplot.png", width = 400, height = 300)  
boxplot(weight ~ group, data = PlantGrowth, ylab = "Dry weight")  
dev.off()
```

通过在终端输入以下命令来执行脚本（操作系统的终端，不是像前一节那样的交互式 R 会话！）

```
R --no-save <plant.R >plant_result.txt
```

文件plant_result.txt包含您的计算结果，就像您在交互式 R 提示符中输入的一样。这样，您的计算过程得到了记录。

新命令png和dev.off用于将箱线图保存到磁盘。两个命令必须包含绘图命令，如上例所示。png("FILENAME", width = .., height = ...)会打开一个指定文件名、宽度和高度（以像素为单位）的新PNG文件。dev.off()将完成绘图并将图保存到磁盘。直到调用dev.off()之前，图形不会被保存。

column is called weight and contains numbers (num, the dry weight of the respective plant). The second column, group, contains the treatment that the plant was subjected to. This is categorial data, which is called **factor** in R. Read more information about data frames.

To compare the dry masses of the three different groups, a one-way ANOVA is performed using **anova(lm(...))**. weight ~ group means "Compare the values of the column weight, grouping by the values of the column group". This is called a Formula in R. **data = ...** specifies the name of the table where the data can be found.

The result shows, among others, that there exists a significant difference (Column Pr(>F), p = **0.01591**) between some of the three groups. Post-hoc tests, like Tukey's Test, must be performed to determine which groups' means differ significantly.

boxplot(...) creates a box plot of the data. where the values to be plotted come from. weight ~ group means: "Plot the values of the column weight *versus* the values of the column group. **ylab = ...** specifies the label of the y axis. More information: Base plotting

Type **q()** or [Ctrl]-[D] to exit from the R session.

R scripts

To document your research, it is favourable to save the commands you use for calculation in a file. For that effect, you can create **R scripts**. An R script is a simple text file, containing R commands.

Create a text file with the name plants.R, and fill it with the following text, where some commands are familiar from the code block above:

```
data(PlantGrowth)  
  
anova(lm(weight ~ group, data = PlantGrowth))  
  
png("plant_boxplot.png", width = 400, height = 300)  
boxplot(weight ~ group, data = PlantGrowth, ylab = "Dry weight")  
dev.off()
```

Execute the script by typing into your terminal (The terminal of your operating system, **not** an interactive R session like in the previous section!)

```
R --no-save <plant.R >plant_result.txt
```

The file plant_result.txt contains the results of your calculation, as if you had typed them into the interactive R prompt. Thereby, your calculations are documented.

The new commands **png** and **dev.off** are used for saving the boxplot to disk. The two commands must enclose the plotting command, as shown in the example above. **png("FILENAME", width = ..., height = ...)** opens a new PNG file with the specified file name, width and height in pixels. **dev.off()** will finish plotting and saves the plot to disk. No output is saved until **dev.off()** is called.

第2章：变量

第2.1节：变量、数据结构和基本操作

在R语言中，数据对象通过命名的数据结构进行操作。对象的名称可能被称为“变量”，尽管该术语在官方R文档中没有具体含义。R语言的名称区分大小写，且可以包含字母数字字符（a-z, A-Z, 0-9）、点号（.）和下划线（_）。要为数据结构创建名称，我们必须遵循以下规则：

- 以数字或下划线开头的名称（例如1a），或者是有效的数值表达式名称（例如.11），或者包含破折号（-'）或空格的名称只能在加引号时使用：`1a`和`.11`。名称将以反引号形式打印：

```
list( '.11' = "a")
#$`11`
#[1] "a"
```

- 所有其他字母数字字符、点和下划线的组合都可以自由使用，带或不带反引号引用都指向同一个对象。
- 以.开头的名称被视为系统名称，使用ls()函数时不一定总是可见。

变量名的字符数没有限制。

一些有效对象名称的示例有：foobar、foo.bar、foo_bar、.foobar在R语言中，

变量通过中缀赋值运算符<-来赋值。运算符=也可以用于给变量赋值，但其正确用法是在函数调用中将值与参数名关联。注意，省略运算符两边的空格可能会让用户产生混淆。表达式a<-1被解析为赋值（a <- 1），而不是逻辑比较（a < -1）。

```
> foo <- 42
> fooEquals = 43
```

因此，foo被赋值为42。在控制台输入foo会输出42，而输入fooEquals会输出43。

```
> foo
[1] 42
> fooEquals
[1] 43
```

下面的命令将值赋给名为x的变量，并同时打印该值：

```
> (x <- 5)
[1] 5
# 实际上是两个函数调用：第一个调用`<-`；第二个调用`()`函数
> is.function(``)
[1] TRUE # 在R帮助页面示例中常用，因其副作用是打印输出。
```

也可以使用->对变量进行赋值。

```
> 5 -> x
> x
```

Chapter 2: Variables

Section 2.1: Variables, data structures and basic Operations

In R, data objects are manipulated using named data structures. The names of the objects might be called "variables" although that term does not have a specific meaning in the official R documentation. R names are *case sensitive* and may contain alphanumeric characters(a-z,A-Z,0-9), the dot/period(.) and underscore(_). To create names for the data structures, we have to follow the following rules:

- Names that start with a digit or an underscore (e.g. 1a), or names that are valid numerical expressions (e.g. .11), or names with dashes ('-) or spaces can only be used when they are quoted: `1a` and `.11`. The names will be printed with backticks:

```
list( '.11' = "a")
#$`11`
#[1] "a"
```

- All other combinations of alphanumeric characters, dots and underscores can be used freely, where reference with or without backticks points to the same object.
- Names that begin with . are considered system names and are not always visible using the **ls()**-function.

There is no restriction on the number of characters in a variable name.

Some examples of valid object names are: foobar, foo.bar, foo_bar, .foobar

In R, variables are assigned values using the infix-assignment operator <- . The operator = can also be used for assigning values to variables, however its proper use is for associating values with parameter names in function calls. Note that omitting spaces around operators may create confusion for users. The expression a<-1 is parsed as assignment (a <- 1) rather than as a logical comparison (a < -1).

```
> foo <- 42
> fooEquals = 43
```

So foo is assigned the value of 42. Typing foo within the console will output 42, while typing fooEquals will output 43.

```
> foo
[1] 42
> fooEquals
[1] 43
```

The following command assigns a value to the variable named x and prints the value simultaneously:

```
> (x <- 5)
[1] 5
# actually two function calls: first one to `<-`; second one to the `()`-function
> is.function(``)
[1] TRUE # Often used in R help page examples for its side-effect of printing.
```

It is also possible to make assignments to variables using ->.

```
> 5 -> x
> x
```

数据结构类型

R中没有标量数据类型。长度为一的向量表现得像标量。

- 向量：原子向量必须是同类对象的序列：一系列数字，或一系列逻辑值，或一系列字符。`v <- c(2, 3, 7, 10)`, `v2 <- c("a", "b", "c")` 都是向量。
- 矩阵：数字、逻辑值或字符的矩阵。`a <- matrix(data = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12), nrow = 4, ncol = 3, byrow = F)`。像向量一样，矩阵必须由同类元素组成。要从矩阵中提取元素，必须指定行和列：`a[1,2]`返回`[1] 5`，即第一行第二列的元素。
- 列表：不同元素的连接 `mylist <- list(course = 'stat', date = '04/07/2009', num_isc = 7, num_cons = 6, num_mat = as.character(c(45020, 45679, 46789, 43126, 42345, 47568, 45674)), results = c(30, 19, 29, NA, 25, 26, 27))`。从列表中提取元素可以通过名称（如果列表有命名）或索引完成。在给定的例子中，`mylist$results` 和 `mylist[[6]]`获得相同的元素。警告：如果你尝试 `mylist[6]`，R不会报错，但它会以列表形式提取结果。
虽然`mylist[[6]][2]`是允许的（它给你19），但`mylist[6][2]`会报错。
- `data.frame`：具有相同长度向量列但（可能）类型不同的对象。它们不是矩阵。`exam <- data.frame(matr = as.character(c(45020, 45679, 46789, 43126, 42345, 47568, 45674)), res_S = c(30, 19, 29, NA, 25, 26, 27), res_O = c(3, 3, 1, NA, 3, 2, NA), res_TOT = c(30, 22, 30, NA, 28, 28, 27))`。列可以通过名称读取 `exam$matr`, `exam[, 'matr']` 或通过索引 `exam[1]`, `exam[, 1]`。行也可以通过名称 `exam['rowname',]` 或索引 `exam[1,]`读取。数据框实际上只是具有特定结构（行名属性和等长组件）的列表。

常见操作及一些注意事项

默认操作是逐元素进行的。请参见?Syntax了解运算符优先级规则。大多数运算符（以及base R中的许多其他函数）具有回收规则，允许参数长度不等。给定这些对象：

示例对象

```
> a <- 1
> b <- 2
> c <- c(2,3,4)
> d <- c(10,10,10)
> e <- c(1,2,3,4)
> f <- 1:6
> W <- cbind(1:4, 5:8, 9:12)
> Z <- rbind(rep(0,3), 1:3, rep(10,3), c(4,7,1))
```

一些向量运算

```
> a+b # 标量 + 标量
[1] 3
> c+d # 向量 + 向量
[1] 12 13 14
> a*b # 标量 * 标量
[1] 2
> c*d # 向量 * 向量 (逐分量)
[1] 20 30 40
> c+a # 向量 + 标量
[1] 3 4 5
> c^2 #
[1] 4 9 16
> exp(c)
```

Types of data structures

There are no scalar data types in R. Vectors of length-one act like scalars.

- Vectors: Atomic vectors must be sequence of same-class objects.: a sequence of numbers, or a sequence of logicals or a sequence of characters. `v <- c(2, 3, 7, 10)`, `v2 <- c("a", "b", "c")` are both vectors.
- Matrices: A matrix of numbers, logical or characters. `a <- matrix(data = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12), nrow = 4, ncol = 3, byrow = F)`. Like vectors, matrix must be made of same-class elements. To extract elements from a matrix rows and columns must be specified: `a[1,2]` returns `[1] 5` that is the element on the first row, second column.
- Lists: concatenation of different elements `mylist <- list(course = 'stat', date = '04/07/2009', num_isc = 7, num_cons = 6, num_mat = as.character(c(45020, 45679, 46789, 43126, 42345, 47568, 45674)), results = c(30, 19, 29, NA, 25, 26, 27))`. Extracting elements from a list can be done by name (if the list is named) or by index. In the given example `mylist$results` and `mylist[[6]]` obtains the same element. Warning: if you try `mylist[6]`, R won't give you an error, but it extract the result as a list.
While `mylist[[6]][2]` is permitted (it gives you 19), `mylist[6][2]` gives you an error.
- `data.frame`: object with columns that are vectors of equal length, but (possibly) different types. They are not matrices. `exam <- data.frame(matr = as.character(c(45020, 45679, 46789, 43126, 42345, 47568, 45674)), res_S = c(30, 19, 29, NA, 25, 26, 27), res_O = c(3, 3, 1, NA, 3, 2, NA), res_TOT = c(30, 22, 30, NA, 28, 28, 27))`. Columns can be read by name `exam$matr`, `exam[, 'matr']` or by index `exam[1]`, `exam[, 1]`. Rows can also be read by name `exam['rowname',]` or index `exam[1,]`. Dataframes are actually just lists with a particular structure (rownames-attribute and equal length components)

Common operations and some cautionary advice

Default operations are done element by element. See ?Syntax for the rules of operator precedence. Most operators (and many other functions in base R) have recycling rules that allow arguments of unequal length. Given these objects:

Example objects

```
> a <- 1
> b <- 2
> c <- c(2,3,4)
> d <- c(10,10,10)
> e <- c(1,2,3,4)
> f <- 1:6
> W <- cbind(1:4, 5:8, 9:12)
> Z <- rbind(rep(0,3), 1:3, rep(10,3), c(4,7,1))
```

Some vector operations

```
> a+b # scalar + scalar
[1] 3
> c+d # vector + vector
[1] 12 13 14
> a*b # scalar * scalar
[1] 2
> c*d # vector * vector (componentwise!)
[1] 20 30 40
> c+a # vector + scalar
[1] 3 4 5
> c^2 #
[1] 4 9 16
> exp(c)
```

```
[1] 7.389056 20.085537 54.598150
```

一些向量运算警告！

```
> c+e # 警告, 但没有错误, 因为假设回收是期望的。  
[1] 3 5 7 6  
警告信息:  
在 c + e : 较长对象的长度 不是较短对象长度的倍数
```

R 会尽可能求和, 然后重复使用较短的向量来填补空缺..... 仅因为两个向量的长度不是完全倍数才给出警告。c+f # 完全没有警告。

一些矩阵运算警告！

```
> Z+W # 矩阵 + 矩阵 #(逐元素)  
> Z*W # 矩阵 * 矩阵 #(标准乘积总是逐元素)
```

要使用矩阵乘法: V %*% W

```
> W + a # 矩阵 + 标量仍然是逐元素  
[,1] [,2] [,3]  
[1,] 2 6 10  
[2,] 3 7 11  
[3,] 4 8 12  
[4,] 5 9 13  
  
> W + c # 矩阵 + 向量... : 无警告, R 以列优先方式执行操作  
[,1] [,2] [,3]  
[1,] 3 8 13  
[2,] 5 10 12  
[3,] 7 9 14  
[4,] 6 11 16
```

“私有”变量

在 R 中, 变量或函数名以点开头通常表示该变量或函数是用于隐藏的。

因此, 声明以下变量

```
> foo <- 'foo'  
> .foo <- 'bar'
```

然后使用 ls 函数列出对象时, 只会显示第一个对象。

```
> ls()  
[1] "foo"
```

但是, 将 all.names = TRUE 传递给该函数将显示“私有”变量

```
> ls(all.names = TRUE)  
[1] ".foo" "foo"
```

```
[1] 7.389056 20.085537 54.598150
```

Some vector operation Warnings!

```
> c+e # warning but.. no errors, since recycling is assumed to be desired.  
[1] 3 5 7 6  
Warning message:  
In c + e : longer object length is not a multiple of shorter object length
```

R sums what it can and then reuses the shorter vector to fill in the blanks... The warning was given only because the two vectors have lengths that are not exactly multiples. c+f # no warning whatsoever.

Some Matrix operations Warning!

```
> Z+W # matrix + matrix #(componentwise)  
> Z*W # matrix* matrix#(Standard product is always componentwise)
```

To use a matrix multiply: V %*% W

```
> W + a # matrix+ scalar is still componentwise  
[,1] [,2] [,3]  
[1,] 2 6 10  
[2,] 3 7 11  
[3,] 4 8 12  
[4,] 5 9 13  
  
> W + c # matrix + vector... : no warnings and R does the operation in a column-wise manner  
[,1] [,2] [,3]  
[1,] 3 8 13  
[2,] 5 10 12  
[3,] 7 9 14  
[4,] 6 11 16
```

"Private" variables

A leading dot in a name of a variable or function in R is commonly used to denote that the variable or function is meant to be hidden.

So, declaring the following variables

```
> foo <- 'foo'  
> .foo <- 'bar'
```

And then using the ls function to list objects will only show the first object.

```
> ls()  
[1] "foo"
```

However, passing all.names = TRUE to the function will show the 'private' variable

```
> ls(all.names = TRUE)  
[1] ".foo" "foo"
```

第3章：算术运算符

第3.1节：范围与加法

我们来看一个将值加到范围上的例子（例如可以在循环中完成）：

```
3+1:5
```

结果是：

```
[1] 4 5 6 7 8
```

这是因为范围运算符:的优先级高于加法运算符+。

计算过程如下：

- 3+1:5
- 3+c(1, 2, 3, 4, 5) 范围运算符展开成整数向量。
- c (4, 5, 6, 7, 8) 向向量的每个元素加3。

为了避免这种行为，你必须告诉R解释器如何使用()来指定运算顺序，方法如下：

```
(3+1):5
```

现在R会先计算括号内的内容，然后再展开序列，结果为：

```
[1] 4 5
```

第3.2节：加法和减法

基本的数学运算主要在数字或向量（数字列表）上进行。

1. 使用单个数字

我们可以简单地输入用+连接的数字表示加法，用-连接的数字表示减法：

```
> 3 + 4.5
# [1] 7.5
> 3 + 4.5 + 2
# [1] 9.5
> 3 + 4.5 + 2 - 3.8
# [1] 5.7
> 3 + NA
#[1] NA
> NA + NA
#[1] NA
> NA - NA
#[1] NA
> NaN - NA
#[1] NaN
> NaN + NA
#[1] NaN
```

我们可以将数字赋值给变量（此处为常量），并执行相同的运算：

Chapter 3: Arithmetic Operators

Section 3.1: Range and addition

Let's take an example of adding a value to a range (as it could be done in a loop for example):

```
3+1:5
```

Gives:

```
[1] 4 5 6 7 8
```

This is because the range operator : has higher precedence than addition operator +.

What happens during evaluation is as follows:

- 3+1:5
- 3+c(1, 2, 3, 4, 5) expansion of the range operator to make a vector of integers.
- c (4, 5, 6, 7, 8) Addition of 3 to each member of the vector.

To avoid this behavior you have to tell the R interpreter how you want it to order the operations with () like this:

```
(3+1):5
```

Now R will compute what is inside the parentheses before expanding the range and gives:

```
[1] 4 5
```

Section 3.2: Addition and subtraction

The basic math operations are performed mainly on numbers or on vectors (lists of numbers).

1. Using single numbers

We can simple enter the numbers concatenated with + for adding and - for subtracting:

```
> 3 + 4.5
# [1] 7.5
> 3 + 4.5 + 2
# [1] 9.5
> 3 + 4.5 + 2 - 3.8
# [1] 5.7
> 3 + NA
#[1] NA
> NA + NA
#[1] NA
> NA - NA
#[1] NA
> NaN - NA
#[1] NaN
> NaN + NA
#[1] NaN
```

We can assign the numbers to *variables* (constants in this case) and do the same operations:

```

> a <- 3; B <- 4.5; cc <- 2; Dd <- 3.8 ;na<-NA;nan<-NaN
> a + B
# [1] 7.5
> a + B + cc
# [1] 9.5
> a + B + cc - Dd
# [1] 5.7
> B-nan
#[1] NaN
> a+na-na
#[1] NA
> a + na
#[1] NA
> B-nan
#[1] NaN
> a+na-na
#[1] NA

```

2. 使用向量

在这种情况下，我们创建数字向量，并使用这些向量或与单个数字的组合进行运算。此时运算是针对向量的每个元素进行的：

```

> A <- c(3, 4.5, 2, -3.8);
> A
# [1] 3.0 4.5 2.0 -3.8
> A + 2 # 加一个数字
# [1] 5.0 6.5 4.0 -1.8
> 8 - A # 数字减向量
# [1] 5.0 3.5 6.0 11.8
> n <- length(A) # 向量A的元素个数
> n
# [1] 4
> A[-n] + A[n] # 将最后一个元素加到去掉最后一个元素的同一向量上
# [1] -0.8 0.7 -1.8
> A[1:2] + 3 # 向量的前两个元素加一个数字
# [1] 6.0 7.5
> A[1:2] - A[3:4] # 向量的前两个元素减去第3和第4个元素组成的向量
# [1] 1.0 8.3

```

我们也可以使用函数 `sum` 来求向量所有元素的和：

```

> sum(A)
# [1] 5.7
> sum(-A)
# [1] -5.7
> sum(A[-n]) + A[n]
# [1] 5.7

```

我们必须注意回收，这是R的一个特性，这种行为发生在进行数学运算时向量长度不同的情况下。表达式中较短的向量会被回收，按需要的次数（可能是部分次数）重复，直到它们的长度与最长向量匹配。特别地，一个常数会被简单地重复。在这种情况下会显示警告。

```

> B <- c(3, 5, -3, 2.7, 1.8)
> B
# [1] 3.0 5.0 -3.0 2.7 1.8
> A
# [1] 3.0 4.5 2.0 -3.8

```

```

> a <- 3; B <- 4.5; cc <- 2; Dd <- 3.8 ;na<-NA;nan<-NaN
> a + B
# [1] 7.5
> a + B + cc
# [1] 9.5
> a + B + cc - Dd
# [1] 5.7
> B-nan
#[1] NaN
> a+na-na
#[1] NA
> a + na
#[1] NA
> B-nan
#[1] NaN
> a+na-na
#[1] NA

```

2. Using vectors

In this case we create vectors of numbers and do the operations using those vectors, or combinations with single numbers. In this case the operation is done considering each element of the vector:

```

> A <- c(3, 4.5, 2, -3.8);
> A
# [1] 3.0 4.5 2.0 -3.8
> A + 2 # Adding a number
# [1] 5.0 6.5 4.0 -1.8
> 8 - A # number less vector
# [1] 5.0 3.5 6.0 11.8
> n <- length(A) #number of elements of vector A
> n
# [1] 4
> A[-n] + A[n] # Add the last element to the same vector without the last element
# [1] -0.8 0.7 -1.8
> A[1:2] + 3 # vector with the first two elements plus a number
# [1] 6.0 7.5
> A[1:2] - A[3:4] # vector with the first two elements less the vector with elements 3 and 4
# [1] 1.0 8.3

```

We can also use the function `sum` to add all elements of a vector:

```

> sum(A)
# [1] 5.7
> sum(-A)
# [1] -5.7
> sum(A[-n]) + A[n]
# [1] 5.7

```

We must take care with *recycling*, which is one of the characteristics of R, a behavior that happens when doing math operations where the length of vectors is different. Shorter vectors in the expression are recycled as often as need be (perhaps fractionally) until they match the length of the longest vector. In particular a constant is simply repeated. In this case a Warning is show.

```

> B <- c(3, 5, -3, 2.7, 1.8)
> B
# [1] 3.0 5.0 -3.0 2.7 1.8
> A
# [1] 3.0 4.5 2.0 -3.8

```

```
> A + B # A的第一个元素被重复  
# [1] 6.0 9.5 -1.0 -1.1 4.8
```

警告信息:

在 A + B 中 : 较长对象的长度不是较短对象长度的倍数

```
> B - A # A的第一个元素被重复
```

```
# [1] 0.0 0.5 -5.0 6.5 -1.2
```

警告信息:

在 B - A 中 : 较长对象的长度不是较短对象长度的倍数

在这种情况下, 正确的做法是只考虑较短向量的元素 :

```
> B[1:n] + A  
# [1] 6.0 9.5 -1.0 -1.1  
> B[1:n] - A  
# [1] 0.0 0.5 -5.0 6.5
```

使用sum函数时, 函数内的所有元素都会被相加。

```
> sum(A, B)  
# [1] 15.2  
> sum(A, -B)  
# [1] -3.8  
> sum(A)+sum(B)  
# [1] 15.2  
> sum(A)-sum(B)  
# [1] -3.8
```

```
> A + B # the first element of A is repeated
```

```
# [1] 6.0 9.5 -1.0 -1.1 4.8
```

Warning message:

In A + B : longer object **length is** not a multiple of shorter object **length**

```
> B - A # the first element of A is repeated
```

```
# [1] 0.0 0.5 -5.0 6.5 -1.2
```

Warning message:

In B - A : longer object **length is** not a multiple of shorter object **length**

In this case the correct procedure will be to consider only the elements of the shorter vector:

```
> B[1:n] + A  
# [1] 6.0 9.5 -1.0 -1.1  
> B[1:n] - A  
# [1] 0.0 0.5 -5.0 6.5
```

When using the **sum** function, again all the elements inside the function are added.

```
> sum(A, B)  
# [1] 15.2  
> sum(A, -B)  
# [1] -3.8  
> sum(A)+sum(B)  
# [1] 15.2  
> sum(A)-sum(B)  
# [1] -3.8
```

第4章：矩阵

矩阵用于存储数据

第4.1节：创建矩阵

在底层，矩阵是一种具有两个维度的特殊向量。像向量一样，矩阵只能有一种数据类型。你可以使用matrix函数来创建矩阵，如下所示。

```
matrix(data = 1:6, nrow = 2, ncol = 3)
## [,1] [,2] [,3]
## [1,] 1 3 5
## [2,] 2 4 6
```

如您所见，这给我们生成了一个包含从1到6所有数字的矩阵，矩阵有两行三列。data参数接受一个数值向量，nrow 指定矩阵的行数，ncol 指定矩阵的列数。按照惯例，矩阵按列填充。默认行为可以通过下面所示的byrow参数进行更改：

```
matrix(data = 1:6, nrow = 2, ncol = 3, byrow = TRUE)
## [,1] [,2] [,3]
## [1,] 1 2 3
## [2,] 4 5 6
```

矩阵不必是数值型-任何向量都可以转换成矩阵。例如：

```
matrix(data = c(TRUE, TRUE, TRUE, FALSE, FALSE, FALSE), nrow = 3, ncol = 2)
## [,1] [,2]
## [1,] TRUE FALSE
## [2,] TRUE FALSE
## [3,] TRUE FALSE
matrix(data = c("a", "b", "c", "d", "e", "f"), nrow = 3, ncol = 2)
## [,1] [,2]
## [1,] "a" "d"
## [2,] "b" "e"
## [3,] "c" "f"
```

像向量一样，矩阵也可以作为变量存储，然后在以后调用。矩阵的行和列可以有名称。你可以使用函数rownames和colnames来查看它们。如下面所示，行和列最初没有名称，表示为NULL。但是，你可以为它们分配值。

```
mat1 <- 矩阵(数据 = 1:6, 行数 = 2, 列数 = 3, 按行填充 = TRUE)
行名(mat1)
## NULL
列名(mat1)
## NULL
行名(mat1) <- c("行 1", "行 2")
列名(mat1) <- c("列 1", "列 2", "列 3")
mat1
##     列 1 列 2 列 3
## 行 1 1 2 3
## 行 2 4 5 6
```

需要注意的是，与向量类似，矩阵只能包含一种数据类型。如果你尝试指定一个包含多种数据类型的矩阵，数据将被强制转换为等级较高的数据类型。

Chapter 4: Matrices

Matrices store data

Section 4.1: Creating matrices

Under the hood, a matrix is a special kind of vector with two dimensions. Like a vector, a matrix can only have one data class. You can create matrices using the `matrix` function as shown below.

```
matrix(data = 1:6, nrow = 2, ncol = 3)
## [,1] [,2] [,3]
## [1,] 1 3 5
## [2,] 2 4 6
```

As you can see this gives us a matrix of all numbers from 1 to 6 with two rows and three columns. The `data` parameter takes a vector of values, `nrow` specifies the number of rows in the matrix, and `ncol` specifies the number of columns. By convention the matrix is filled by column. The default behavior can be changed with the `byrow` parameter as shown below:

```
matrix(data = 1:6, nrow = 2, ncol = 3, byrow = TRUE)
## [,1] [,2] [,3]
## [1,] 1 2 3
## [2,] 4 5 6
```

Matrices do not have to be numeric – any vector can be transformed into a matrix. For example:

```
matrix(data = c(TRUE, TRUE, TRUE, FALSE, FALSE, FALSE), nrow = 3, ncol = 2)
## [,1] [,2]
## [1,] TRUE FALSE
## [2,] TRUE FALSE
## [3,] TRUE FALSE
matrix(data = c("a", "b", "c", "d", "e", "f"), nrow = 3, ncol = 2)
## [,1] [,2]
## [1,] "a" "d"
## [2,] "b" "e"
## [3,] "c" "f"
```

Like vectors matrices can be stored as variables and then called later. The rows and columns of a matrix can have names. You can look at these using the functions `rownames` and `colnames`. As shown below, the rows and columns don't initially have names, which is denoted by `NULL`. However, you can assign values to them.

```
mat1 <- matrix(data = 1:6, nrow = 2, ncol = 3, byrow = TRUE)
rownames(mat1)
## NULL
colnames(mat1)
## NULL
rownames(mat1) <- c("Row 1", "Row 2")
colnames(mat1) <- c("Col 1", "Col 2", "Col 3")
mat1
##     Col 1 Col 2 Col 3
## Row 1 1 2 3
## Row 2 4 5 6
```

It is important to note that similarly to vectors, matrices can only have one data type. If you try to specify a matrix with multiple data types the data will be coerced to the higher order data class.

class、is 和 as 函数可以用来检查和强制转换数据结构，使用方法与第一课中向量的操作相同。

```
class(mat1)
## [1] "matrix"
is.matrix(mat1)
## [1] TRUE
as.vector(mat1)
## [1] 1 4 2 5 3 6
```

The **class**, **is**, and **as** functions can be used to check and coerce data structures in the same way they were used on the vectors in class 1.

```
class(mat1)
## [1] "matrix"
is.matrix(mat1)
## [1] TRUE
as.vector(mat1)
## [1] 1 4 2 5 3 6
```

第5章：公式

第5.1节：公式基础

R中的统计函数大量使用所谓的Wilkinson-Rogers公式表示法1。

在运行如lm（线性回归）等模型函数时，需要一个formula。该formula指定了要估计的回归系数。

```
my_formula1 <- formula(mpg ~ wt)
class(my_formula1)
# 返回 "formula"

mod1 <- lm(my_formula1, data = mtcars)
coef(mod1)
# 返回 (Intercept)      wt
#           37.285126   -5.344472
```

在~符号的左侧（LHS）指定因变量，而右侧（RHS）包含自变量。从技术上讲，上述formula调用是多余的，因为波浪号操作符是一个中缀函数，返回一个公式类对象：

```
form <- mpg ~ wt
class(form)
#[1] "formula"
```

formula 函数相较于 ~ 的优势在于它还允许指定一个用于评估的环境：

```
form_mt <- formula(mpg ~ wt, env = mtcars)
```

在这种情况下，输出显示估计了变量wt的回归系数，以及（默认）截距参数。通过在formula中包含0或-1，可以排除截距或强制截距为0：

```
coef(lm(mpg ~ 0 + wt, data = mtcars))
coef(lm(mpg ~ wt -1, data = mtcars))
```

变量a和b之间的交互作用可以通过在formula中包含a:b来添加：

```
coef(lm(mpg ~ wt:vs, data = mtcars))
```

从统计学角度来看，通常建议模型中不要只有交互项而没有主效应，简单的方法是将formula扩展为a + b + a:b。这种方法可行，但可以通过写成a*b来简化，其中*操作符表示因子交叉（当两个列都是因子时）或乘法（当一个或两个列是“数值型”时）：

```
coef(lm(mpg ~ wt*vs, data = mtcars))
```

使用*符号会将一个项展开为包含所有低阶效应，例如：

```
coef(lm(mpg ~ wt*vs*hp, data = mtcars))
```

除了截距外，将会得到7个回归系数。一个是三阶交互作用，三个是二阶交互作用，三个是主效应。

Chapter 5: Formula

Section 5.1: The basics of formula

Statistical functions in R make heavy use of the so-called Wilkinson-Rogers formula notation1 .

When running model functions like lm for the Linear Regressions, they need a formula. This formula specifies which regression coefficients shall be estimated.

```
my_formula1 <- formula(mpg ~ wt)
class(my_formula1)
# gives "formula"

mod1 <- lm(my_formula1, data = mtcars)
coef(mod1)
# gives (Intercept)      wt
#           37.285126   -5.344472
```

On the left side of the ~ (LHS) the dependent variable is specified, while the right hand side (RHS) contains the independent variables. Technically the formula call above is redundant because the tilde-operator is an infix function that returns an object with formula class:

```
form <- mpg ~ wt
class(form)
#[1] "formula"
```

The advantage of the formula function over ~ is that it also allows an environment for evaluation to be specified:

```
form_mt <- formula(mpg ~ wt, env = mtcars)
```

In this case, the output shows that a regression coefficient for wt is estimated, as well as (per default) an intercept parameter. The intercept can be excluded / forced to be 0 by including 0 or -1 in the formula:

```
coef(lm(mpg ~ 0 + wt, data = mtcars))
coef(lm(mpg ~ wt -1, data = mtcars))
```

Interactions between variables a and b can added by included a:b to the formula:

```
coef(lm(mpg ~ wt:vs, data = mtcars))
```

As it is (from a statistical point of view) generally advisable not have interactions in the model without the main effects, the naive approach would be to expand the formula to a + b + a:b. This works but can be simplified by writing a*b, where the * operator indicates factor crossing (when between two factor columns) or multiplication when one or both of the columns are 'numeric':

```
coef(lm(mpg ~ wt*vs, data = mtcars))
```

Using the * notation expands a term to include all lower order effects, such that:

```
coef(lm(mpg ~ wt*vs*hp, data = mtcars))
```

will give, in addition to the intercept, 7 regression coefficients. One for the three-way interaction, three for the two-way interactions and three for the main effects.

如果有人想排除三阶交互作用，但保留所有二阶交互作用，有两种简写方式。首先，使用-可以减去任何特定项：

```
coef(lm(mpg ~ wt*vs*hp - wt:vs:hp, data = mtcars))
```

或者，我们可以使用^符号来指定所需的交互作用层级：

```
coef(lm(mpg ~ (wt + vs + hp) ^ 2, data = mtcars))
```

这两种公式写法应该会生成相同的模型矩阵。

最后，.是使用所有可用变量作为主效应的简写。在这种情况下，data参数用于获取可用变量（即不在左侧的变量）。因此：

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

会给出截距和10个自变量的系数。这种写法在机器学习包中经常使用，当需要用所有变量进行预测或分类时。注意，.的含义依赖于上下文（例如参见?update.formula以了解不同含义）。

1. G. N. Wilkinson 和 C. E. Rogers。《皇家统计学会学报》第C系列（应用统计）第22卷第3期（1973），第392-399页

If one wants, for example, to exclude the three-way interaction, but retain all two-way interactions there are two shorthands. First, using - we can subtract any particular term:

```
coef(lm(mpg ~ wt*vs*hp - wt:vs:hp, data = mtcars))
```

Or, we can use the ^ notation to specify which level of interaction we require:

```
coef(lm(mpg ~ (wt + vs + hp) ^ 2, data = mtcars))
```

Those two formula specifications should create the same model matrix.

Finally, . is shorthand to use all available variables as main effects. In this case, the **data** argument is used to obtain the available variables (which are not on the LHS). Therefore:

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

gives coefficients for the intercept and 10 independent variables. This notation is frequently used in machine learning packages, where one would like to use all variables for prediction or classification. Note that the meaning of . depends on context (see e.g. **?update.formula** for a different meaning).

1. G. N. Wilkinson and C. E. Rogers. *Journal of the Royal Statistical Society. Series C (Applied Statistics)* Vol. 22, No. 3 (1973), pp. 392-399

第6章：读取和写入字符串

第6.1节：打印和显示字符串

R有几个内置函数可以用来打印或显示信息，但print和cat是最基本的。由于R是解释型语言，你可以直接在R控制台中尝试这些函数：

```
print("Hello World")#[1]
"Hello World"cat("Hell
o World")#Hello World
```

注意这两个函数在输入和输出上的区别。（注意：用print输出时，变量 `x <- "Hello World"` 的值中没有引号字符，引号是print在输出阶段添加的。）

`cat`接受一个或多个字符向量作为参数，并将它们打印到控制台。如果字符向量长度大于1，参数之间默认用空格分隔：

```
cat(c("hello", "world", ""))#hello worl
d
```

如果没有换行符（`\n`），输出将是：

```
cat("Hello World")
#Hello World>
```

下一条命令的提示符会紧跟在输出之后出现。（某些控制台，如RStudio，可能会自动在不以换行符结尾的字符串后追加换行符。）

`print` 是一个“通用”函数的示例，这意味着传入的第一个参数的类会被检测，并使用一个特定类的方法来输出。对于像“Hello World”这样的字符向量，结果类似于`cat`的输出。

但是，字符字符串会被加引号，并且会输出一个数字[1]，表示字符向量的第一个元素（在本例中，是第一个也是唯一的元素）：character vector（在本例中，第一个且唯一的元素）：

```
print("Hello World")
#[1] "Hello World"
```

这个默认的打印方法也是当我们简单地让R打印一个变量时所看到的。注意输入s时的输出与调用`print(s)`或`print("Hello World")`的输出是相同的：

```
s <- "Hello World"
s
#[1] "Hello World"
```

甚至不赋值给任何变量时也是如此：

```
"Hello World"
#[1] "Hello World"
```

如果我们添加另一个字符字符串作为向量的第二个元素（使用`c()`函数将元素连接起来），那么`print()`的行为看起来与`cat`的行为有很大不同：

```
print(c("Hello World", "Here I am.))
```

Chapter 6: Reading and writing strings

Section 6.1: Printing and displaying strings

R has several built-in functions that can be used to print or display information, but `print` and `cat` are the most basic. As R is an [interpreted language](#), you can try these out directly in the R console:

```
print("Hello World")
#[1] "Hello World"
cat("Hello World\n")
#Hello World
```

Note the difference in both input and output for the two functions. (Note: there are no quote-characters in the value of x created with `x <- "Hello World"`. They are added by `print` at the output stage.)

`cat` takes one or more character vectors as arguments and prints them to the console. If the character vector has a length greater than 1, arguments are separated by a space (by default):

```
cat(c("hello", "world", "\n"))
#hello world
```

Without the new-line character (`\n`) the output would be:

```
cat("Hello World")
#Hello World>
```

The prompt for the next command appears immediately after the output. (Some consoles such as RStudio's may automatically append a newline to strings that do not end with a newline.)

`print` is an example of a “generic” function, which means the class of the first argument passed is detected and a class-specific *method* is used to output. For a character vector like “Hello World”，the result is similar to the output of `cat`. However, the character string is quoted and a number [1] is output to indicate the first element of a character vector (In this case, the first and only element):

```
print("Hello World")
#[1] "Hello World"
```

This default print method is also what we see when we simply ask R to print a variable. Note how the output of typing s is the same as calling `print(s)` or `print("Hello World")`:

```
s <- "Hello World"
s
#[1] "Hello World"
```

Or even without assigning it to anything:

```
"Hello World"
#[1] "Hello World"
```

If we add another character string as a second element of the vector (using the `c()` function to concatenate the elements together), then the behavior of `print()` looks quite a bit different from that of `cat`:

```
print(c("Hello World", "Here I am.))
```

```
# [1] "你好，世界" "我来了。"
```

请注意, `c()` 函数不执行字符串连接。(需要使用 `paste` 来实现该功能。) R 通过分别引用字符串向量的两个元素来显示它们。如果向量足够长以跨越多行, R 会在每行开头打印该元素的索引, 就像在第一行开头打印 [1] 一样。

```
c("Hello World", "Here I am!", "This next string is really long.")  
#[1] "Hello World" "Here I am!"  
#[3] "This next string is really long."
```

`print` 的具体行为取决于传递给该函数的对象的 `class`。

如果我们对不同类别的对象调用 `print`, 比如 "numeric" 或 "logical", 输出中将省略引号, 以表明我们处理的对象不是字符类:

```
print(1)  
#[1] 1  
print(TRUE)  
#[1] TRUE
```

因子对象的打印方式与字符变量相同, 这常常在控制台输出用于显示 SO 问题正文中的对象时造成歧义。除非在交互式环境中, 通常很少使用 `cat` 或 `print`。显式调用 `print()` 尤为罕见 (除非你想抑制引号的显示或查看函数返回的 `invisible` 对象), 因为在控制台输入 `foo` 是 `print(foo)` 的快捷方式。R 的交互式控制台被称为 REPL, 即“读取-求值-打印-循环”。`cat` 函数最好用于特殊用途 (如向打开的文件连接写入输出)。有时它在函数内部使用 (此时 `print()` 调用被抑制), 但在函数内部使用 `cat()` 向控制台生成输出是不良做法。推荐的方法是使用 `message()` 或 `warning()` 来输出中间信息; 它们的行为类似于 `cat`, 但最终用户可以选择抑制它们。最终结果应直接返回, 以便用户根据需要进行赋值存储。

```
message("hello world")  
#hello world  
suppressMessages(message("hello world"))
```

第6.2节：捕获操作系统命令的输出

返回字符向量的函数

Base R 有两个用于调用系统命令的函数。两者都需要一个额外的参数来捕获系统命令的输出。

```
system("top -a -b -n 1", intern = TRUE)  
system2("top", "-a -b -n 1", stdout = TRUE)
```

两者都返回一个字符向量。

```
[1] "top - 08:52:03 已运行 70 天, 15:09, 0 用户, 负载平均值: 0.00, 0.00, 0.00"  
[2] "任务: 125 总计, 1 运行中, 124 睡眠中, 0 停止, 0 僵尸"  
[3] "CPU(s): 0.9% 用户态, 0.3% 系统态, 0.0% 优先级调整, 98.7% 空闲, 0.1% 等待 I/O, 0.0% 硬中断, 0.0% 软中断, 0.0% 虚拟机偷取"  
[4] "内存: 12194312k 总计, 3613292k 已用, 8581020k 空闲, 216940k 缓冲区"  
[5] "交换区: 12582908k 总计, 2334156k 已用, 10248752k 空闲, 1682340k 缓存"  
[6] ""  
[7] " PID 用户 优先级 NI 虚拟内存 常驻内存 共享内存 状态 %CPU %MEM 时间+ 命令 "
```

```
# [1] "Hello World" "Here I am."
```

Observe that the `c()` function does *not* do string-concatenation. (One needs to use `paste` for that purpose.) R shows that the character vector has two elements by quoting them separately. If we have a vector long enough to span multiple lines, R will print the index of the element starting each line, just as it prints [1] at the start of the first line.

```
c("Hello World", "Here I am!", "This next string is really long.")  
#[1] "Hello World" "Here I am!"  
#[3] "This next string is really long."
```

The particular behavior of `print` depends on the *class* of the object passed to the function.

If we call `print` an object with a different class, such as "numeric" or "logical", the quotes are omitted from the output to indicate we are dealing with an object that is not character class:

```
print(1)  
#[1] 1  
print(TRUE)  
#[1] TRUE
```

Factor objects get printed in the same fashion as character variables which often creates ambiguity when console output is used to display objects in SO question bodies. It is rare to use `cat` or `print` except in an interactive context. Explicitly calling `print()` is particularly rare (unless you wanted to suppress the appearance of the quotes or view an object that is returned as `invisible` by a function), as entering `foo` at the console is a shortcut for `print(foo)`. The interactive console of R is known as a REPL, a "read-eval-print-loop". The `cat` function is best saved for special purposes (like writing output to an open file connection). Sometimes it is used inside functions (where calls to `print()` are suppressed), however **using `cat()` inside a function to generate output to the console is bad practice**. The preferred method is to `message()` or `warning()` for intermediate messages; they behave similarly to `cat` but can be optionally suppressed by the end user. The final result should simply returned so that the user can assign it to store it if necessary.

```
message("hello world")  
#hello world  
suppressMessages(message("hello world"))
```

Section 6.2: Capture output of operating system command

Functions which return a character vector

Base R has two functions for invoking a system command. Both require an additional parameter to capture the output of the system command.

```
system("top -a -b -n 1", intern = TRUE)  
system2("top", "-a -b -n 1", stdout = TRUE)
```

Both return a character vector.

```
[1] "top - 08:52:03 up 70 days, 15:09, 0 users, load average: 0.00, 0.00, 0.00"  
[2] "Tasks: 125 total, 1 running, 124 sleeping, 0 stopped, 0 zombie"  
[3] "Cpu(s): 0.9%us, 0.3%sy, 0.0%ni, 98.7%id, 0.1%wa, 0.0%hi, 0.0%si, 0.0%st"  
[4] "Mem: 12194312k total, 3613292k used, 8581020k free, 216940k buffers"  
[5] "Swap: 12582908k total, 2334156k used, 10248752k free, 1682340k cached"  
[6] ""  
[7] " PID USER PR NI VIRT RES SHR S %CPU %MEM TIME+ COMMAND "
```

```
[8] "11300 root      20  0 1278m 375m 3696 S  0.0  3.2 124:40.92 trala      "
[9] " 6093 user1     20  0 1817m 269m 1888 S  0.0  2.3 12:17.96 R      "
[10] " 4949 user2     20  0 1917m 214m 1888 S  0.0  1.8 11:16.73 R      "
```

为说明起见，使用了 UNIX 命令 `top -a -b -n 1`。这是操作系统特定的，可能需要根据您的计算机修改以运行示例。

包 `devtools` 有一个函数可以运行系统命令并捕获输出，无需额外参数。它也返回一个字符向量。

```
devtools::system_output("top", "-a -b -n 1")
```

返回数据框的函数

包 `data.table` 中的 `fread` 函数允许执行 shell 命令并像

`read.table`一样读取输出。它返回一个 `data.table` 或 `data.frame`

```
fread("top -a -b -n 1", check.names = TRUE)
      PID    用户 PR NI 虚拟内存 常驻内存 共享内存 S CPU使用率 内存使用率   时间    命令
1: 11300    root 20  0 1278m 375m 3696 S  0  3.2 124:40.92      trala
2: 6093   user1 20  0 1817m 269m 1888 S  0  2.3 12:18.56      R
3: 4949   user2 20  0 1917m 214m 1888 S  0  1.8 11:17.33      R
4: 7922   user3 20  0 3094m 131m 1892 S  0  1.1 21:04.95      R
```

注意，`fread` 自动跳过了顶部的6行表头。

这里添加了参数 `check.names = TRUE`，用于将 %CPU、%MEM 和 TIME+ 转换为语法上有效的列名。

```
[8] "11300 root      20  0 1278m 375m 3696 S  0.0  3.2 124:40.92 trala      "
[9] " 6093 user1     20  0 1817m 269m 1888 S  0.0  2.3 12:17.96 R      "
[10] " 4949 user2     20  0 1917m 214m 1888 S  0.0  1.8 11:16.73 R      "
```

For illustration, the UNIX command `top -a -b -n 1` is used. This is OS specific and may need to be amended to run the examples on your computer.

Package `devtools` has a function to run a system command and capture the output without an additional parameter. It also returns a character vector.

```
devtools::system_output("top", "-a -b -n 1")
```

Functions which return a data frame

The `fread` function in package `data.table` allows to execute a shell command and to read the output like `read.table`. It returns a `data.table` or a `data.frame`.

```
fread("top -a -b -n 1", check.names = TRUE)
      PID    USER PR NI VIRT  RES  SHR S X.CPU X.MEM   TIME.    COMMAND
1: 11300    root 20  0 1278m 375m 3696 S  0  3.2 124:40.92      trala
2: 6093   user1 20  0 1817m 269m 1888 S  0  2.3 12:18.56      R
3: 4949   user2 20  0 1917m 214m 1888 S  0  1.8 11:17.33      R
4: 7922   user3 20  0 3094m 131m 1892 S  0  1.1 21:04.95      R
```

Note, that `fread` automatically has skipped the top 6 header lines.

Here the parameter `check.names = TRUE` was added to convert %CPU, %MEM, and TIME+ to syntactically valid column names.

第6.3节：从文件连接读取或写入

我们并不总是可以自由地从本地系统路径读取或写入。例如，如果R代码流式处理map-reduce，必须读取和写入文件连接。还有其他场景是超出本地系统的，随着云计算和大数据的发展，这种情况越来越普遍。实现这一点的一种方法是按逻辑顺序进行。

使用 `file()` 命令建立文件连接以读取 ("r"表示读取模式)：

```
conn <- file("/path/example.data", "r") #当文件在本地系统时
conn1 <- file("stdin", "r") #当只有标准输入/输出可用时
```

由于这将仅建立文件连接，因此可以通过以下方式从这些文件连接中读取数据：

```
line <- readLines(conn, n=1, warn=FALSE)
```

这里我们以 `n=1` 的方式逐行从文件连接 `conn` 读取数据。可以更改 `n` 的值（例如 10、20 等）以块读取数据，从而加快读取速度（一次读取 10 或 20 行块）。若要一次性读取完整文件，将 `n` 设置为 -1。

数据处理或模型执行后；可以使用多种不同的命令如 `writeLines()`、`cat()` 等将结果写回文件连接，这些命令都能写入文件连接。但所有这些命令都将利用已建立的写入文件连接。可以使用 `file()` 命令来实现，如下所示：

Section 6.3: Reading from or writing to a file connection

Not always we have liberty to read from or write to a local system path. For example if R code streaming map-reduce must need to read and write to file connection. There can be other scenarios as well where one is going beyond local system and with advent of cloud and big data, this is becoming increasingly common. One of the way to do this is in logical sequence.

Establish a file connection to read with `file()` command ("r" is for read mode):

```
conn <- file("/path/example.data", "r") #when file is in local system
conn1 <- file("stdin", "r") #when just standard input/output for files are available
```

As this will establish just file connection, one can read the data from these file connections as follows:

```
line <- readLines(conn, n=1, warn=FALSE)
```

Here we are reading the data from file connection `conn` line by line as `n=1`. one can change value of `n` (say 10, 20 etc.) for reading data blocks for faster reading (10 or 20 lines block read in one go). To read complete file in one go set `n=-1`.

After data processing or say model execution; one can write the results back to file connection using many different commands like `writeLines()`, `cat()` etc. which are capable of writing to a file connection. However all of these commands will leverage file connection established for writing. This could be done using `file()` command as:

```
conn2 <- file("/path/result.data", "w") #当文件在本地系统时  
conn3 <- file("stdout", "w") #当仅有标准输入/输出文件时
```

然后按如下方式写入数据：

```
writeLines("text", conn2, sep = "")
```

```
conn2 <- file("/path/result.data", "w") #when file is in local system  
conn3 <- file("stdout", "w") #when just standard input/output for files are available
```

Then write the data as follows:

```
writeLines("text", conn2, sep = "\n")
```

第7章：使用 stringi 包进行字符串操作

第7.1节：统计字符串中的模式

使用固定模式

```
stri_count_fixed("babab", "b")
# [1] 3
stri_count_fixed("babab", "ba")
# [1] 2
stri_count_fixed("babab", "bab")
# [1] 1
```

原生方法：

```
length(gregexpr("b", "babab")[[1]])
# [1] 3
length(gregexpr("ba", "babab")[[1]])
# [1] 2
length(gregexpr("bab", "babab")[[1]])
# [1] 1
```

函数对字符串和模式进行了向量化处理：

```
stri_count_fixed("babab", c("b", "ba"))
# [1] 3 2
stri_count_fixed(c("babab", "bbb", "bca", "abc"), c("b", "ba"))
# [1] 3 0 1 0
```

基础R解决方案：

```
sapply(c("b", "ba"), function(x)length(gregexpr(x, "babab")[[1]]))
# b ba
# 3 2
```

使用正则表达式

第一个例子 - 查找 a及其后面的任意字符

第二个例子 - 查找 a及其后面的任意数字

```
stri_count_regex("a1 b2 a3 b4 aa", "a.")
# [1] 3
stri_count_regex("a1 b2 a3 b4 aa", "a\\d")
# [1] 2
```

第7.2节：字符串复制

```
stri_dup("abc", 3)
# [1] "abcabcabc"
```

实现相同功能的基础R解决方案如下：

Chapter 7: String manipulation with stringi package

Section 7.1: Count pattern inside string

With fixed pattern

```
stri_count_fixed("babab", "b")
# [1] 3
stri_count_fixed("babab", "ba")
# [1] 2
stri_count_fixed("babab", "bab")
# [1] 1
```

Natively:

```
length(gregexpr("b", "babab")[[1]])
# [1] 3
length(gregexpr("ba", "babab")[[1]])
# [1] 2
length(gregexpr("bab", "babab")[[1]])
# [1] 1
```

function is vectorized over string and pattern:

```
stri_count_fixed("babab", c("b", "ba"))
# [1] 3 2
stri_count_fixed(c("babab", "bbb", "bca", "abc"), c("b", "ba"))
# [1] 3 0 1 0
```

A base R solution:

```
sapply(c("b", "ba"), function(x)length(gregexpr(x, "babab")[[1]]))
# b ba
# 3 2
```

With regex

First example - find a and any character after

Second example - find a and any digit after

```
stri_count_regex("a1 b2 a3 b4 aa", "a.")
# [1] 3
stri_count_regex("a1 b2 a3 b4 aa", "a\\d")
# [1] 2
```

Section 7.2: Duplicating strings

```
stri_dup("abc", 3)
# [1] "abcabcabc"
```

A base R solution that does the same would look like this:

```
paste0(rep("abc", 3), collapse = "")  
# [1] "abcabcabc"
```

第7.3节：粘贴向量

```
stri_paste(LETTERS, "-", 1:13)  
# [1] "A-1" "B-2" "C-3" "D-4" "E-5" "F-6" "G-7" "H-8" "I-9" "J-10" "K-11" "L-12" "M-13"  
# [14] "N-1" "O-2" "P-3" "Q-4" "R-5" "S-6" "T-7" "U-8" "V-9" "W-10" "X-11" "Y-12" "Z-13"
```

在R中，我们可以通过以下方式实现：

```
> paste(LETTERS, 1:13, sep="-")  
# [1] "A-1" "B-2" "C-3" "D-4" "E-5" "F-6" "G-7" "H-8" "I-9" "J-10" "K-11" "L-12" "M-13"  
# [14] "N-1" "O-2" "P-3" "Q-4" "R-5" "S-6" "T-7" "U-8" "V-9" "W-10" "X-11" "Y-12" "Z-13"
```

第7.4节：按固定模式拆分文本

使用一个模式拆分文本向量：

```
stri_split_fixed(c("To be or not to be.", "This is very short sentence."), " ")  
# [[1]]  
# [1] "To" "be" "or" "not" "to" "be."  
#  
# [[2]]  
# [1] "这是" "非常" "简短的" "句子。"
```

使用多种模式拆分文本：

```
stri_split_fixed("苹果, 橙子和菠萝。", c(" ", "", "s"))  
# [[1]]  
# [1] "苹果," "橙子" "和" "菠萝。"  
#  
# [[2]]  
# [1] "苹果" " 橙子和菠萝。"  
#  
# [[3]]  
# [1] "苹果", " 橙子" " 和菠萝" ":"
```

```
paste0(rep("abc", 3), collapse = "")  
# [1] "abcabcabc"
```

Section 7.3: Paste vectors

```
stri_paste(LETTERS, "-", 1:13)  
# [1] "A-1" "B-2" "C-3" "D-4" "E-5" "F-6" "G-7" "H-8" "I-9" "J-10" "K-11" "L-12" "M-13"  
# [14] "N-1" "O-2" "P-3" "Q-4" "R-5" "S-6" "T-7" "U-8" "V-9" "W-10" "X-11" "Y-12" "Z-13"
```

Natively, we could do this in R via:

```
> paste(LETTERS, 1:13, sep="-")  
# [1] "A-1" "B-2" "C-3" "D-4" "E-5" "F-6" "G-7" "H-8" "I-9" "J-10" "K-11" "L-12" "M-13"  
# [14] "N-1" "O-2" "P-3" "Q-4" "R-5" "S-6" "T-7" "U-8" "V-9" "W-10" "X-11" "Y-12" "Z-13"
```

Section 7.4: Splitting text by some fixed pattern

Split vector of texts using one pattern:

```
stri_split_fixed(c("To be or not to be.", "This is very short sentence."), " ")  
# [[1]]  
# [1] "To" "be" "or" "not" "to" "be."  
#  
# [[2]]  
# [1] "This" "is" "very" "short" "sentence."
```

Split one text using many patterns:

```
stri_split_fixed("Apples, oranges and pineapllies.", c(" ", ", ", "s"))  
# [[1]]  
# [1] "Apples," "oranges" "and" "pineapllies."  
#  
# [[2]]  
# [1] "Apples" "oranges and pineapllies."  
#  
# [[3]]  
# [1] "Apple" ", orange" "and pineaplle" ":"
```

第8章：类

数据对象的类决定了哪些函数将处理其内容。class（类）属性是一个字符向量，且对象可以没有类、拥有一个类或多个类。如果没有class属性，仍然会由对象的mode（模式）隐式确定一个类。可以使用class函数查看类，也可以通过class<-函数设置或修改类。S3类系统是在S语言早期建立的，更复杂的S4类系统则是在后期建立的。

第8.1节：检查类

R 中的每个对象都有一个类。你可以使用class()来查找对象的类，使用str()来查看其结构，包括它包含的类。例如：

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

str(iris)
'data.frame': 150 个观测值, 5 个变量:
$ Sepal.Length: 数值型 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
$ Sepal.Width : 数值型 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
$ Petal.Length: 数值型 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
$ Petal.Width : 数值型 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
$ Species     : 因子型 w/ 3 水平 "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...

class(iris$Species)
[1] "factor"
```

我们看到 iris 的类是data.frame，使用str()可以检查其中的数据。iris 数据框中的变量Species 是因子类，与其他数值型变量不同。str()函数还提供变量的长度并显示前几个观测值，而class()函数仅提供对象的类。

第8.2节：向量和列表

R中的数据存储在向量中。典型的向量是一系列具有相同存储模式的值（例如，字符向量，数值向量）。详见?atomic，了解原子隐式类及其对应的存储模式："logical"，"integer"，"numeric"（同义词"double"），"complex"，"character"和"raw"。许多类只是带有class属性的原子向量：

```
x <- 1826
class(x) <- "Date"
x
# [1] "1975-01-01"
x <- as.Date("1970-01-01")
class(x)
#[1] "Date"
is(x, "Date")
#[1] TRUE
is(x, "integer")
#[1] FALSE
is(x, "numeric")
#[1] FALSE
模式(x)
#[1] "numeric"
```

Chapter 8: Classes

The class of a data-object determines which functions will process its contents. The **class**-attribute is a character vector, and objects can have zero, one or more classes. If there is no class-attribute, there will still be an implicit class determined by an object's **mode**. The class can be inspected with the function **class** and it can be set or modified by the **class<-** function. The S3 class system was established early in S's history. The more complex S4 class system was established later

Section 8.1: Inspect classes

Every object in R is assigned a class. You can use **class()** to find the object's class and **str()** to see its structure, including the classes it contains. For example:

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

str(iris)
'data.frame': 150 obs. of 5 variables:
$ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
$ Sepal.Width : num 3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
$ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
$ Petal.Width : num 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
$ Species     : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...

class(iris$Species)
[1] "factor"
```

We see that iris has the class **data.frame** and using **str()** allows us to examine the data inside. The variable Species in the iris data frame is of class factor, in contrast to the other variables which are of class numeric. The **str()** function also provides the length of the variables and shows the first couple of observations, while the **class()** function only provides the object's class.

Section 8.2: Vectors and lists

Data in R are stored in vectors. A typical vector is a sequence of values all having the same storage mode (e.g., characters vectors, numeric vectors). See **?atomic** for details on the atomic implicit classes and their corresponding storage modes: "logical"，"integer"，"numeric"（synonym "double"），"complex"，"character" and "raw"。Many classes are simply an atomic vector with a **class** attribute on top:

```
x <- 1826
class(x) <- "Date"
x
# [1] "1975-01-01"
x <- as.Date("1970-01-01")
class(x)
#[1] "Date"
is(x, "Date")
#[1] TRUE
is(x, "integer")
#[1] FALSE
is(x, "numeric")
#[1] FALSE
mode(x)
#[1] "numeric"
```

列表是一种特殊类型的向量，其中每个元素可以是任何东西，甚至是另一个列表，因此R中列表的术语是：“递归向量”：

```
mylist <- list( A = c(5,6,7,8), B = letters[1:10], CC = list( 5, "Z" ) )
```

列表有两个非常重要的用途：

- 由于函数只能返回一个值，通常会将复杂的结果放在列表中返回：

```
f <- function(x) list(xplus = x + 10, xsq = x^2)

f(7)
# $xplus
# [1] 17
#
# $xsq
# [1] 49
```

- 列表也是数据框的底层基本类。在底层，数据框是一个所有向量长度相同的列表：

```
L <- list(x = 1:2, y = c("A", "B"))
DF <- data.frame(L)
DF
#   x y
# 1 1 A
# 2 2 B
is.list(DF)
# [1] TRUE
```

另一类递归向量是R表达式，它们是“语言”对象

第8.3节：向量

R中最简单的数据结构是向量。你可以使用c()函数创建数值、逻辑值和字符字符串的向量。例如：

```
c(1, 2, 3)
## [1] 1 2 3
c(TRUE, TRUE, FALSE)
## [1] TRUE TRUE FALSE
c("a", "b", "c")
## [1] "a" "b" "c"
```

你也可以使用c()函数连接两个向量。

```
x <- c(1, 2, 5)
y <- c(3, 4, 6)
z <- c(x, y)
z
## [1] 1 2 5 3 4 6
```

关于如何创建向量的更详细说明可以在“创建向量”主题中找到

Lists are a special type of vector where each element can be anything, even another list, hence the R term for lists: “recursive vectors”:

```
mylist <- list( A = c(5,6,7,8), B = letters[1:10], CC = list( 5, "Z" ) )
```

Lists have two very important uses:

- Since functions can only return a single value, it is common to return complicated results in a list:

```
f <- function(x) list(xplus = x + 10, xsq = x^2)

f(7)
# $xplus
# [1] 17
#
# $xsq
# [1] 49
```

- Lists are also the underlying fundamental class for data frames. Under the hood, a data frame is a list of vectors all having the same length:

```
L <- list(x = 1:2, y = c("A", "B"))
DF <- data.frame(L)
DF
#   x y
# 1 1 A
# 2 2 B
is.list(DF)
# [1] TRUE
```

The other class of recursive vectors is R expressions, which are “language”- objects

Section 8.3: Vectors

The most simple data structure available in R is a vector. You can make vectors of numeric values, logical values, and character strings using the **c()** function. For example:

```
c(1, 2, 3)
## [1] 1 2 3
c(TRUE, TRUE, FALSE)
## [1] TRUE TRUE FALSE
c("a", "b", "c")
## [1] "a" "b" "c"
```

You can also join to vectors using the **c()** function.

```
x <- c(1, 2, 5)
y <- c(3, 4, 6)
z <- c(x, y)
z
## [1] 1 2 5 3 4 6
```

A more elaborate treatment of how to create vectors can be found in the “*Creating vectors*” topic

第9章：列表

第9.1节：列表简介

列表允许用户将多个元素（如向量和矩阵）存储在单个对象下。你可以使用 `list` 函数来创建列表：

```
l1 <- list(c(1, 2, 3), c("a", "b", "c"))
l1
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "a" "b" "c"
```

注意，上述列表中组成的向量属于不同的类别。列表允许用户将不同类别的元素组合在一起。列表中的每个元素也可以有一个名称。列表名称通过 `names` 函数访问，赋值方式与矩阵中行名和列名的赋值方式相同。

```
names(l1)
## NULL
names(l1) <- c("vector1", "vector2")
l1
## $vector1
## [1] 1 2 3
##
## $vector2
## [1] "a" "b" "c"
```

创建列表对象时，通常更容易且更安全地声明列表名称。

```
l2 <- list(vec = c(1, 3, 5, 7, 9),
           mat = matrix(data = c(1, 2, 3), nrow = 3))
l2
## $vec
## [1] 1 3 5 7 9
##
## $mat
## [,1]
## [1,] 1
## [2,] 2
## [3,] 3
names(l2)
## [1] "vec" "mat"
```

上述列表有两个元素，分别命名为“vec”和“mat”，分别是向量和矩阵。

第9.2节：列表快速介绍

一般来说，作为用户你会交互的大多数对象往往是向量；例如数值向量、逻辑向量。这些对象只能包含单一类型的变量（数值向量只能包含数字）。

列表可以存储任何类型的变量，使其成为可以存储我们所需任何类型变量的通用对象。

Chapter 9: Lists

Section 9.1: Introduction to lists

Lists allow users to store multiple elements (like vectors and matrices) under a single object. You can use the `list` function to create a list:

```
l1 <- list(c(1, 2, 3), c("a", "b", "c"))
l1
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "a" "b" "c"
```

Notice the vectors that make up the above list are different classes. Lists allow users to group elements of different classes. Each element in a list can also have a name. List names are accessed by the `names` function, and are assigned in the same manner row and column names are assigned in a matrix.

```
names(l1)
## NULL
names(l1) <- c("vector1", "vector2")
l1
## $vector1
## [1] 1 2 3
##
## $vector2
## [1] "a" "b" "c"
```

It is often easier and safer to declare the list names when creating the list object.

```
l2 <- list(vec = c(1, 3, 5, 7, 9),
           mat = matrix(data = c(1, 2, 3), nrow = 3))
l2
## $vec
## [1] 1 3 5 7 9
##
## $mat
## [,1]
## [1,] 1
## [2,] 2
## [3,] 3
names(l2)
## [1] "vec" "mat"
```

Above the list has two elements, named “vec” and “mat,” a vector and matrix, respectively.

Section 9.2: Quick Introduction to Lists

In general, most of the objects you would interact with as a user would tend to be a vector; e.g numeric vector, logical vector. These objects can only take in a single type of variable (a numeric vector can only have numbers inside it).

A list would be able to store any type variable in it, making it to the generic object that can store any type of variables we would need.

初始化列表的示例

```
exampleList1 <- list('a', 'b')
exampleList2 <- list(1, 2)
exampleList3 <- list('a', 1, 2)
```

为了理解列表中定义的数据，我们可以使用 `str` 函数。

```
str(exampleList1)
str(exampleList2)
str(exampleList3)
```

列表的子集操作区分提取列表的一个切片，即获得包含原列表部分元素的子列表，与提取单个元素。使用常用于向量的 `[` 操作符会生成一个新的列表。

```
# 返回列表
exampleList3[1]
exampleList3[1:2]
```

要获取单个元素，请改用 `[[` 操作符。

```
# 返回字符
exampleList3[[1]]
```

列表条目可以命名：

```
exampleList4 <- list(
  num = 1:3,
  numeric = 0.5,
  char = c('a', 'b')
)
```

可以通过名称而非索引访问命名列表中的条目。

```
exampleList4[['char']]
```

或者可以使用`$`操作符访问命名元素。

```
exampleList4$num
```

这样做的优点是输入更快且可能更易读，但需要注意一个潜在的陷阱。`$`操作符使用部分匹配来识别匹配的列表元素，可能会产生意想不到的结果。

```
exampleList5 <- exampleList4[2:3]

exampleList4$num
# c(1, 2, 3)

exampleList5$num
# 0.5

exampleList5[['num']]
# NULL
```

Example of initializing a list

```
exampleList1 <- list('a', 'b')
exampleList2 <- list(1, 2)
exampleList3 <- list('a', 1, 2)
```

In order to understand the data that was defined in the list, we can use the `str` function.

```
str(exampleList1)
str(exampleList2)
str(exampleList3)
```

Subsetting of lists distinguishes between extracting a slice of the list, i.e. obtaining a list containing a subset of the elements in the original list, and extracting a single element. Using the `[` operator commonly used for vectors produces a new list.

```
# Returns List
exampleList3[1]
exampleList3[1:2]
```

To obtain a single element use `[[` instead.

```
# Returns Character
exampleList3[[1]]
```

List entries may be named:

```
exampleList4 <- list(
  num = 1:3,
  numeric = 0.5,
  char = c('a', 'b')
)
```

The entries in named lists can be accessed by their name instead of their index.

```
exampleList4[['char']]
```

Alternatively the `$` operator can be used to access named elements.

```
exampleList4$num
```

This has the advantage that it is faster to type and may be easier to read but it is important to be aware of a potential pitfall. The `$` operator uses partial matching to identify matching list elements and may produce unexpected results.

```
exampleList5 <- exampleList4[2:3]

exampleList4$num
# c(1, 2, 3)

exampleList5$num
# 0.5

exampleList5[['num']]
# NULL
```

列表特别有用，因为它们可以存储不同长度和不同类型的对象。

```
## 数值向量
exampleVector1 <- c(12, 13, 14)
## 字符向量
exampleVector2 <- c("a", "b", "c", "d", "e", "f")
## 矩阵
exampleMatrix1 <- matrix(rnorm(4), ncol = 2, nrow = 2)
## 列表
exampleList3 <- list('a', 1, 2)

exampleList6 <- list(
  num = exampleVector1,
  char = exampleVector2,
  mat = exampleMatrix1,
  list = exampleList3
)
exampleList6
#$num
#[1] 12 13 14
#
#$char
#[1] "a" "b" "c" "d" "e" "f"
#
#$mat
#      [,1]      [,2]
#[1,] 0.5013050 -1.88801542
#[2,] 0.4295266  0.09751379
#
#$list
#$list[[1]]
#[1] "a"
#
#$list[[2]]
#[1] 1
#
#$list[[3]]
#[1] 2
```

第9.3节：序列化：使用列表传递信息

存在需要将不同类型数据放在一起的情况。例如，在Azure ML中，必须通过数据框（dataframe）将信息从一个R脚本模块传递到另一个模块。假设我们有一个数据框和一个数字：

```
> df
姓名 身高 队伍 趣味指数 职称 年龄 描述 Y
1 安德烈亚 195 拉齐奥 97 6 33 优秀 1
2 帕亚 165 佛罗伦萨 87 6 31 果断 1
3 罗罗 190 拉齐奥 65 6 28 奇怪 0
4 乔埃莱 70 拉齐奥 100 0 2 讨人喜欢 1
5 卡乔 170 尤文图斯 81 3 33 坚韧 0
6 埃多拉 171 拉齐奥 72 5 32 糊涂 1
7 萨拉米 175 国际米兰 75 3 30 双步 1
8 布劳戈 180 国际米兰 79 5 32 gjn 0
9 贝纳 158 尤文图斯 80 6 28 精疲力尽 0
10 里吉奥 182 拉齐奥 92 5 31 确定性 1
11 乔尔达诺 185 罗马 79 5 29 好1
```

Lists can be particularly useful because they can store objects of different lengths and of various classes.

```
## Numeric vector
exampleVector1 <- c(12, 13, 14)
## Character vector
exampleVector2 <- c("a", "b", "c", "d", "e", "f")
## Matrix
exampleMatrix1 <- matrix(rnorm(4), ncol = 2, nrow = 2)
## List
exampleList3 <- list('a', 1, 2)

exampleList6 <- list(
  num = exampleVector1,
  char = exampleVector2,
  mat = exampleMatrix1,
  list = exampleList3
)
exampleList6
#$num
#[1] 12 13 14
#
#$char
#[1] "a" "b" "c" "d" "e" "f"
#
#$mat
#      [,1]      [,2]
#[1,] 0.5013050 -1.88801542
#[2,] 0.4295266  0.09751379
#
#$list
#$list[[1]]
#[1] "a"
#
#$list[[2]]
#[1] 1
#
#$list[[3]]
#[1] 2
```

Section 9.3: Serialization: using lists to pass information

There exist cases in which it is necessary to put data of different types together. In Azure ML for example, it is necessary to pass information from a R script module to another one exclusively through dataframes. Suppose we have a dataframe and a number:

```
> df
name height team fun_index title age desc Y
1 Andrea 195 Lazio 97 6 33 eccezzente 1
2 Paja 165 Fiorentina 87 6 31 deciso 1
3 Roro 190 Lazio 65 6 28 strano 0
4 Gioele 70 Lazio 100 0 2 simpatico 1
5 Cacio 170 Juventus 81 3 33 duro 0
6 Edola 171 Lazio 72 5 32 svampito 1
7 Salami 175 Inter 75 3 30 doppiopasso 1
8 Braugo 180 Inter 79 5 32 gjn 0
9 Benni 158 Juventus 80 6 28 esaurito 0
10 Riggio 182 Lazio 92 5 31 certezza 1
11 Giordano 185 Roma 79 5 29 buono 1
```

```
> number <- "42"
```

我们可以访问这些信息：

```
> paste(df$name[4], "是一个", df3$team[4], "支持者。")
[1] "Gioele 是拉齐奥的支持者。"
> paste("问题的答案是", number)
[1] "问题的答案是 42"
```

为了将不同类型的数据放入数据框中，我们必须使用列表对象和序列化。特别是，我们需要将数据放入一个通用列表中，然后将该列表放入特定的数据框中：

```
l <- list(df,number)
dataframe_container <- data.frame(out2 = as.integer(serialze(l, connection=NULL)))
```

一旦我们将信息存储在数据框中，就需要反序列化它以便使用：

```
#---- 反序列化 -----
unser_obj <- unserialize(as.raw(dataframe_container$out2))
#---- 取回元素 -----
df_mod      <- unser_obj[1][[1]]
number_mod  <- unser_obj[2][[1]]
```

然后，我们可以验证数据是否被正确传输：

```
> paste(df_mod$name[4], "是一个", df_mod$team[4], "支持者。")
[1] "Gioele 是 Lazio 的支持者。"
> paste("问题的答案是", number_mod )
[1] "问题的答案是 42"
```

```
> number <- "42"
```

We can access to this information:

```
> paste(df$name[4], "is a", df3$team[4], "supporter. ")
[1] "Gioele is a Lazio supporter."
> paste("The answer to THE question is", number )
[1] "The answer to THE question is 42"
```

In order to put different types of data in a dataframe we have to use the list object and the serialization. In particular we have to put the data in a generic list and then put the list in a particular dataframe:

```
l <- list(df,number)
dataframe_container <- data.frame(out2 = as.integer(serialze(l, connection=NULL)))
```

Once we have stored the information in the dataframe, we need to deserialize it in order to use it:

```
#---- unserialize -----
unser_obj <- unserialize(as.raw(dataframe_container$out2))
#---- taking back the elements -----
df_mod      <- unser_obj[1][[1]]
number_mod  <- unser_obj[2][[1]]
```

Then, we can verify that the data are transferred correctly:

```
> paste(df_mod$name[4], "is a", df_mod$team[4], "supporter. ")
[1] "Gioele is a Lazio supporter."
> paste("The answer to THE question is", number_mod )
[1] "The answer to THE question is 42"
```

第10章：哈希映射

第10.1节：作为哈希映射的环境

注意：在后续内容中，术语 *hash map* 和 *hash table* 交替使用，指的是同一概念，即通过内部哈希函数提供高效键查找的数据结构。

介绍

虽然 R 没有提供原生的哈希表结构，但可以利用默认由 `new.env` 返回的 `environment` 对象提供的哈希键查找功能，实现类似功能。以下两条语句是等价的，因为 `hash` 参数默认为 TRUE：

```
H <- new.env(hash = TRUE)  
H <- new.env()
```

此外，可以通过 `size` 参数指定内部哈希表的预分配大小，默认值为 29。像所有其他 R 对象一样，`environment` 会管理自己的内存，并且会根据需要自动扩展容量，因此虽然不必非得为 `size` 请求非默认值，但如果对象最终将包含非常多的元素，`size` 的非默认值可能会带来轻微的性能优势。值得注意的是，通过 `size` 分配额外空间本身并不会导致对象占用更大的内存空间：

```
object.size(new.env())  
# 56 字节  
  
object.size(new.env(size = 10e4))  
# 56 字节
```

插入

元素的插入可以使用 `environment` 类提供的 `[[<-` 或 `$<-` 方法，但不能使用“单括号”赋值 `([<-)`：

```
H <- new.env()  
  
H[["key"]] <- rnorm(1)  
  
key2 <- "xyz"  
H[[key2]] <- data.frame(x = 1:3, y = letters[1:3])  
  
H$another_key <- matrix(rbinom(9, 1, 0.5) > 0, nrow = 3)  
  
H["error"] <- 42  
# 错误：在 H["error"] <- 42 中：  
# 类型为 'environment' 的对象不可被子集化
```

与 R 的其他方面类似，第一种方法 (`object[[key]] <- value`) 通常优于第二种方法 (`object$key <- value`)，因为前者可以使用变量代替字面值（例如上例中的 `key2`）。

与哈希映射实现通常的情况一样，`environment` 对象将 **not** 存储重复的键。
尝试为已存在的键插入键值对将替换之前存储的值：

Chapter 10: Hashmaps

Section 10.1: Environments as hash maps

Note: in the subsequent passages, the terms **hash map** and **hash table** are used interchangeably and refer to the [same concept](#), namely, a data structure providing efficient key lookup through use of an internal hash function.

Introduction

Although R does not provide a native hash table structure, similar functionality can be achieved by leveraging the fact that the `environment` object returned from `new.env` (by default) provides hashed key lookups. The following two statements are equivalent, as the `hash` parameter defaults to TRUE:

```
H <- new.env(hash = TRUE)  
H <- new.env()
```

Additionally, one may specify that the internal hash table is pre-allocated with a particular size via the `size` parameter, which has a default value of 29. Like all other R objects, `environments` manage their own memory and will grow in capacity as needed, so while it is not necessary to request a non-default value for `size`, there may be a slight performance advantage in doing so if the object will (eventually) contain a very large number of elements. It is worth noting that allocating extra space via `size` does not, in itself, result in an object with a larger memory footprint:

```
object.size(new.env())  
# 56 bytes  
  
object.size(new.env(size = 10e4))  
# 56 bytes
```

Insertion

Insertion of elements may be done using either of the `[[<-` or `$<-` methods provided for the `environment` class, but **not** by using “single bracket” assignment `([<-)`:

```
H <- new.env()  
  
H[["key"]] <- rnorm(1)  
  
key2 <- "xyz"  
H[[key2]] <- data.frame(x = 1:3, y = letters[1:3])  
  
H$another_key <- matrix(rbinom(9, 1, 0.5) > 0, nrow = 3)  
  
H["error"] <- 42  
# Error in H["error"] <- 42 :  
# object of type 'environment' is not subsettable
```

Like other facets of R, the first method (`object[[key]] <- value`) is generally preferred to the second (`object$key <- value`) because in the former case, a variable maybe be used instead of a literal value (e.g `key2` in the example above).

As is generally the case with hash map implementations, the `environment` object will **not** store duplicate keys.
Attempting to insert a key-value pair for an existing key will replace the previously stored value:

```
H[["key3"]] <- "original value"
H[["key3"]] <- "new value"
H[["key3"]]
#[1] "new value"
```

键查找

同样，元素可以通过[[或\$访问，但不能通过[访问：

```
H[["key"]]
#[1] 1.630631

H[[key2]] ## 假设 key2 <- "xyz"
#   x y
# 1 1 a
# 2 2 b
# 3 3 c

H$another_key
#   [,1] [,2] [,3]
# [1,] TRUE TRUE TRUE
# [2,] FALSE FALSE FALSE
# [3,] TRUE TRUE TRUE

H[1]
#Error in H[1] : object of type 'environment' is not subsettable
```

检查哈希映射

作为一个普通的environment，哈希映射可以通过典型的方法进行检查：

```
names(H)
#[1] "another_key" "xyz"      "key"      "key3"

ls(H)
#[1] "another_key" "key"      "key3"    "xyz"

str(H)
#<environment: 0x7828228>

ls.str(H)
# another_key : logi [1:3, 1:3] TRUE FALSE TRUE TRUE FALSE TRUE ...
# key : num 1.63
# key3 : chr "new value"
# xyz : 'data.frame': 3 obs. of 2 variables:
# $ x: int 1 2 3
# $ y: chr "a" "b" "c"
```

可以使用 rm 删除元素：

```
rm(list = c("key", "key3"), envir = H)

ls.str(H)
# another_key : logi [1:3, 1:3] TRUE FALSE TRUE TRUE FALSE TRUE ...
# xyz : 'data.frame': 3 obs. of 2 variables:
# $ x: int 1 2 3
```

```
H[["key3"]] <- "original value"
H[["key3"]] <- "new value"
H[["key3"]]
#[1] "new value"
```

Key Lookup

Likewise, elements may be accessed with [[or \$, but not with [:

```
H[["key"]]
#[1] 1.630631

H[[key2]] ## assuming key2 <- "xyz"
#   x y
# 1 1 a
# 2 2 b
# 3 3 c

H$another_key
#   [,1] [,2] [,3]
# [1,] TRUE TRUE TRUE
# [2,] FALSE FALSE FALSE
# [3,] TRUE TRUE TRUE

H[1]
#Error in H[1] : object of type 'environment' is not subsettable
```

Inspecting the Hash Map

Being just an ordinary [environment](#), the hash map can be inspected by typical means:

```
names(H)
#[1] "another_key" "xyz"      "key"      "key3"

ls(H)
#[1] "another_key" "key"      "key3"    "xyz"

str(H)
#<environment: 0x7828228>

ls.str(H)
# another_key : logi [1:3, 1:3] TRUE FALSE TRUE TRUE FALSE TRUE ...
# key : num 1.63
# key3 : chr "new value"
# xyz : 'data.frame': 3 obs. of 2 variables:
# $ x: int 1 2 3
# $ y: chr "a" "b" "c"
```

Elements can be removed using rm:

```
rm(list = c("key", "key3"), envir = H)

ls.str(H)
# another_key : logi [1:3, 1:3] TRUE FALSE TRUE TRUE FALSE TRUE ...
# xyz : 'data.frame': 3 obs. of 2 variables:
# $ x: int 1 2 3
```

```
# $ y: chr "a" "b" "c"
```

灵活性

使用`environment`对象作为哈希表的主要优点之一是它们能够将几乎任何类型的对象存储为值，甚至是其他`environment`对象：

```
H2 <- new.env()

H2[["a"]] <- LETTERS
H2[["b"]] <- as.list(x = 1:5, y = matrix(rnorm(10), 2))
H2[["c"]] <- head(mtcars, 3)
H2[["d"]] <- Sys.Date()
H2[["e"]] <- Sys.time()
H2[["f"]] <- (function() {
  H3 <- new.env()
  for (i in seq_along(names(H2))) {
    H3[[names(H2)[i]]] <- H2[[names(H2)[i]]]
  }
H3
})()
```

```
ls.str(H2)
# a : chr [1:26] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" ...
# b : 5个元素的列表
# $ : int 1
# $ : int 2
# $ : int 3
# $ : int 4
# $ : int 5
# c : 'data.frame': 3个观测值, 11个变量:
# $ mpg : num 21 21 22.8
# $ cyl : num 6 6 4
# $ disp: num 160 160 108
# $ hp : num 110 110 93
# $ drat: num 3.9 3.9 3.85
# $ wt : num 2.62 2.88 2.32
# $ qsec: num 16.5 17 18.6
# $ vs : num 0 0 1
# $ am : 数值 1 1 1
# $ gear: 数值 4 4 4
# $ carb: 数值 4 4 1
# d : 日期[1:1], 格式: "2016-08-03"
# e : POSIXct[1:1], 格式: "2016-08-03 19:25:14"
# f : <environment: 0x91a7cb8>
```

```
ls.str(H2$f)
# a : chr [1:26] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" ...
# b : 5个元素的列表
# $ : int 1
# $ : int 2
# $ : int 3
# $ : int 4
# $ : int 5
# c : 'data.frame': 3个观测值, 11个变量:
# $ mpg : num 21 21 22.8
# $ cyl : num 6 6 4
# $ disp: 数值 160 160 108
# $ hp : 数值 110 110 93
# $ drat: 数值 3.9 3.9 3.85
# $ wt : 数值 2.62 2.88 2.32
```

```
# $ y: chr "a" "b" "c"
```

Flexibility

One of the major benefits of using `environment` objects as hash tables is their ability to store virtually any type of object as a value, even other `environments`:

```
H2 <- new.env()

H2[["a"]] <- LETTERS
H2[["b"]] <- as.list(x = 1:5, y = matrix(rnorm(10), 2))
H2[["c"]] <- head(mtcars, 3)
H2[["d"]] <- Sys.Date()
H2[["e"]] <- Sys.time()
H2[["f"]] <- (function() {
  H3 <- new.env()
  for (i in seq_along(names(H2))) {
    H3[[names(H2)[i]]] <- H2[[names(H2)[i]]]
  }
H3
})()
```

```
ls.str(H2)
# a : chr [1:26] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" ...
# b : List of 5
# $ : int 1
# $ : int 2
# $ : int 3
# $ : int 4
# $ : int 5
# c : 'data.frame': 3 obs. of 11 variables:
# $ mpg : num 21 21 22.8
# $ cyl : num 6 6 4
# $ disp: num 160 160 108
# $ hp : num 110 110 93
# $ drat: num 3.9 3.9 3.85
# $ wt : num 2.62 2.88 2.32
# $ qsec: num 16.5 17 18.6
# $ vs : num 0 0 1
# $ am : num 1 1 1
# $ gear: num 4 4 4
# $ carb: num 4 4 1
# d : Date[1:1], format: "2016-08-03"
# e : POSIXct[1:1], format: "2016-08-03 19:25:14"
# f : <environment: 0x91a7cb8>
```

```
ls.str(H2$f)
# a : chr [1:26] "A" "B" "C" "D" "E" "F" "G" "H" "I" "J" "K" ...
# b : List of 5
# $ : int 1
# $ : int 2
# $ : int 3
# $ : int 4
# $ : int 5
# c : 'data.frame': 3 obs. of 11 variables:
# $ mpg : num 21 21 22.8
# $ cyl : num 6 6 4
# $ disp: num 160 160 108
# $ hp : num 110 110 93
# $ drat: num 3.9 3.9 3.85
# $ wt : num 2.62 2.88 2.32
```

```
# $ qsec: 数值 16.5 17 18.6
# $ vs : 数值 0 0 1
# $ am : 数值 1 1 1
# $ gear: 数值 4 4 4
# $ carb: 数值 4 4 1
# d : 日期[1:1], 格式: "2016-08-03"
# e : POSIXct[1:1], 格式: "2016-08-03 19:25:14"
```

限制

使用`environment`对象作为哈希映射的主要限制之一是，与R的许多方面不同，元素查找/插入不支持向量化：

名称(H2)

```
# [1] "a" "b" "c" "d" "e" "f"

H2[[c("a", "b")]]
#Error in H2[[c("a", "b")]] :
# 错误的环境子集参数

Keys <- c("a", "b")
H2[[Keys]]
#Error in H2[[Keys]] : 错误的环境子集参数
```

根据存储在对象中的数据性质，可能可以使用 `vapply` 或 `list2env` 来一次性赋值多个元素：

```
E1 <- new.env()
invisible({
  vapply(letters, function(x) {
    E1[[x]] <- rnorm(1)
    logical(0)
  }, FUN.VALUE = logical(0))
})

all.equal(sort(names(E1)), letters)
#[1] TRUE

Keys <- letters
E2 <- list2env(
  setNames(
    as.list(rnorm(26)),
    nm = Keys),
  envir = NULL,
  hash = TRUE
)

all.equal(sort(names(E2)), letters)
#[1] TRUE
```

以上两种方法都不是特别简洁，但当键值对数量较大时，可能比使用for循环等更可取。

第10.2节：package:hash

hash包在R中提供了哈希结构。然而，就插入和读取的时间而言，它与使用环境作为哈希相比表现不佳。本文档仅承认其存在，并基于上述原因提供了示例计时代码。当前尚未发现hash在R代码中适用的情况。

```
# $ qsec: num 16.5 17 18.6
# $ vs : num 0 0 1
# $ am : num 1 1 1
# $ gear: num 4 4 4
# $ carb: num 4 4 1
# d : Date[1:1], format: "2016-08-03"
# e : POSIXct[1:1], format: "2016-08-03 19:25:14"
```

Limitations

One of the major limitations of using `environment` objects as hash maps is that, unlike many aspects of R, vectorization is not supported for element lookup / insertion:

```
names(H2)
#[1] "a" "b" "c" "d" "e" "f"

H2[[c("a", "b")]]
#Error in H2[[c("a", "b")]] :
# wrong arguments for subsetting an environment

Keys <- c("a", "b")
H2[[Keys]]
#Error in H2[[Keys]] : wrong arguments for subsetting an environment
```

Depending on the nature of the data being stored in the object, it may be possible to use `vapply` or `list2env` for assigning many elements at once:

```
E1 <- new.env()
invisible({
  vapply(letters, function(x) {
    E1[[x]] <- rnorm(1)
    logical(0)
  }, FUN.VALUE = logical(0))
})

all.equal(sort(names(E1)), letters)
#[1] TRUE

Keys <- letters
E2 <- list2env(
  setNames(
    as.list(rnorm(26)),
    nm = Keys),
  envir = NULL,
  hash = TRUE
)

all.equal(sort(names(E2)), letters)
#[1] TRUE
```

Neither of the above are particularly concise, but may be preferable to using a `for` loop, etc. when the number of key-value pairs is large.

Section 10.2: package:hash

The `hash package` offers a hash structure in R. However, it `terms of timing` for both inserts and reads it compares unfavorably to using environments as a hash. This documentation simply acknowledges its existence and provides sample timing code below for the above stated reasons. There is no identified case where hash is an appropriate

考虑：

```
# 通用唯一字符串生成器
unique_strings <- function(n){
  string_i <- 1
  string_len <- 1
  ans <- character(n)
  chars <- c(letters, LETTERS)
  new_strings <- function(len,pfx){
    for(i in 1:length(chars)){
      if (len == 1){
        ans[string_i] <- paste(pfx,chars[i],sep=' ')
        string_i <- string_i + 1
      } else {
        new_strings(len-1,pfx=paste(pfx,chars[i],sep=' '))
      }
      if (string_i > n) return ()
    }
  }
  while(string_i <= n){
    new_strings(string_len,' ')
    string_len <- string_len + 1
  }
  sample(ans)
}

# Generate timings using an environment
timingsEnv <- plyr::adply(2^(10:15), .mar=1, .fun=function(i){
  strings <- unique_strings(i)
  ht1 <- new.env(hash=TRUE)
  lapply(strings, function(s){ ht1[[s]] <- 0L})
  data.frame(
    size=c(i,i),
    seconds=c(
      system.time(for (j in 1:i) ht1[[strings[j]]]==0L)[3]),
    type = c('1_hashedEnv')
  )
})

timingsHash <- plyr::adply(2^(10:15), .mar=1, .fun=function(i){
  strings <- unique_strings(i)
  ht <- hash::hash()
  lapply(strings, function(s) ht[[s]] <- 0L)
  data.frame(
    size=c(i,i),
    seconds=c(
      system.time(for (j in 1:i) ht[[strings[j]]]==0L)[3]),
    type = c('3_stringHash')
  )
})
```

第10.3节：package:listenv

虽然package:listenv实现了对环境的类似列表接口，但其在哈希检索方面相较于环境的性能较差。然而，如果索引是数字类型，检索速度可以相当快。不过，它们还有其他优点，例如与package:future的兼容性。针对该目的介绍此包超出了当前主题的范围。不过，这里提供的计时代码可以与package:hash的示例结合使用，以进行写入计时。

solution in R code today.

Consider:

```
# Generic unique string generator
unique_strings <- function(n){
  string_i <- 1
  string_len <- 1
  ans <- character(n)
  chars <- c(letters, LETTERS)
  new_strings <- function(len,pfx){
    for(i in 1:length(chars)){
      if (len == 1){
        ans[string_i] <- paste(pfx,chars[i],sep=' ')
        string_i <- string_i + 1
      } else {
        new_strings(len-1,pfx=paste(pfx,chars[i],sep=' '))
      }
      if (string_i > n) return ()
    }
  }
  while(string_i <= n){
    new_strings(string_len,' ')
    string_len <- string_len + 1
  }
  sample(ans)
}

# Generate timings using an environment
timingsEnv <- plyr::adply(2^(10:15), .mar=1, .fun=function(i){
  strings <- unique_strings(i)
  ht1 <- new.env(hash=TRUE)
  lapply(strings, function(s){ ht1[[s]] <- 0L})
  data.frame(
    size=c(i,i),
    seconds=c(
      system.time(for (j in 1:i) ht1[[strings[j]]]==0L)[3]),
    type = c('1_hashedEnv')
  )
})

timingsHash <- plyr::adply(2^(10:15), .mar=1, .fun=function(i){
  strings <- unique_strings(i)
  ht <- hash::hash()
  lapply(strings, function(s) ht[[s]] <- 0L)
  data.frame(
    size=c(i,i),
    seconds=c(
      system.time(for (j in 1:i) ht[[strings[j]]]==0L)[3]),
    type = c('3_stringHash')
  )
})
```

Section 10.3: package:listenv

Although package:listenv implements a list-like interface to environments, its performance relative to environments for hash-like purposes is poor on hash retrieval. However, if the indexes are numeric, it can be quite fast on retrieval. However, they have other advantages, e.g. compatibility with package:future. Covering this package for that purpose goes beyond the scope of the current topic. However, the timing code provided here can be used in conjunction with the example for package:hash for write timings.

```
timingsListEnv <- plyr::adply(2^(10:15), .mar=1, .fun=function(i){  
  strings <- unique_strings(i)  
  le <- listenv::listenv()  
  lapply(strings, function(s) le[[s]] <- 0L)  
  data.frame(  
    size=c(i,i),  
    seconds=c(  
      system.time(for (k in 1:i) le[[k]]==0L)[3]),  
    type = c('2_numericListEnv')  
  )  
})
```

```
timingsListEnv <- plyr::adply(2^(10:15), .mar=1, .fun=function(i){  
  strings <- unique_strings(i)  
  le <- listenv::listenv()  
  lapply(strings, function(s) le[[s]] <- 0L)  
  data.frame(  
    size=c(i,i),  
    seconds=c(  
      system.time(for (k in 1:i) le[[k]]==0L)[3]),  
    type = c('2_numericListEnv')  
  )  
})
```

第11章：创建向量

第11.1节：从内置常量创建向量：字母序列和月份名称

R 有许多内置常量。以下常量可用：

- **LETTERS**：罗马字母的大写26个字母
- **letters**：罗马字母的小写26个字母
- **month.abb**：英文月份名称的三字母缩写
- **month.name**：英文月份名称
- pi：圆的周长与直径的比值

可以从字母和月份常量创建向量。

1) 字母序列：

```
> 字母  
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x"  
"y" "z"  
  
> 大写字母[7:9]  
[1] "G" "H" "I"  
  
> 字母[c(1,5,3,2,4)]  
[1] "a" "e" "c" "b" "d"
```

2) 月份缩写或月份名称序列：

```
> month.abb  
[1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"  
  
> month.name[1:4]  
[1] "January" "February" "March" "April"  
  
> month.abb[c(3,6,9,12)]  
[1] "三月" "六月" "九月" "十二月"
```

第11.2节：创建命名向量

命名向量可以通过多种方式创建。使用c：

```
xc <- c('a' = 5, 'b' = 6, 'c' = 7, 'd' = 8)
```

结果为：

```
> xc  
a b c d  
5 6 7 8
```

使用list：

```
xl <- list('a' = 5, 'b' = 6, 'c' = 7, 'd' = 8)
```

结果为：

Chapter 11: Creating vectors

Section 11.1: Vectors from build in constants: Sequences of letters & month names

R has a number of build in constants. The following constants are available:

- **LETTERS**: the 26 upper-case letters of the Roman alphabet
- **letters**: the 26 lower-case letters of the Roman alphabet
- **month.abb**: the three-letter abbreviations for the English month names
- **month.name**: the English names for the months of the year
- pi: the ratio of the circumference of a circle to its diameter

From the letters and month constants, vectors can be created.

1) Sequences of letters:

```
> letters  
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t" "u" "v" "w" "x"  
"y" "z"  
  
> LETTERS[7:9]  
[1] "G" "H" "I"  
  
> letters[c(1,5,3,2,4)]  
[1] "a" "e" "c" "b" "d"
```

2) Sequences of month abbreviations or month names:

```
> month.abb  
[1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"  
  
> month.name[1:4]  
[1] "January" "February" "March" "April"  
  
> month.abb[c(3,6,9,12)]  
[1] "Mar" "Jun" "Sep" "Dec"
```

Section 11.2: Creating named vectors

Named vector can be created in several ways. With c:

```
xc <- c('a' = 5, 'b' = 6, 'c' = 7, 'd' = 8)
```

which results in:

```
> xc  
a b c d  
5 6 7 8
```

with list:

which results in:

```
> x1  
$a  
[1] 5
```

```
$b  
[1] 6
```

```
$c  
[1] 7
```

```
$d  
[1] 8
```

使用setNames函数，可以用两个相同长度的向量创建一个命名向量：

```
x <- 5:8  
y <- letters[1:4]  
  
xy <- setNames(x, y)
```

结果是一个命名的整数向量：

```
> xy  
a b c d  
5 6 7 8
```

如图所示，这与c方法得到的结果相同。

您也可以使用 names 函数来获得相同的结果：

```
xy <- 5:8  
names(xy) <- letters[1:4]
```

使用这样的向量也可以通过名称选择元素：

```
> xy["c"]
```

```
> mydf  
let  
1 c  
2 a  
3 b  
4 d
```

假设你想在mydf数据框中创建一个名为num的新变量，其值来自xy中对应行的正确值。使用match函数可以从xy中选择合适的值：

```
mydf$num <- xy[match(mydf$let, names(xy))]
```

结果为：

```
> x1  
$a  
[1] 5
```

```
$b  
[1] 6
```

```
$c  
[1] 7
```

```
$d  
[1] 8
```

With the **setNames** function, two vectors of the same length can be used to create a named vector:

```
x <- 5:8  
y <- letters[1:4]  
  
xy <- setNames(x, y)
```

which results in a named integer vector:

```
> xy  
a b c d  
5 6 7 8
```

As can be seen, this gives the same result as the c method.

You may also use the **names** function to get the same result:

```
xy <- 5:8  
names(xy) <- letters[1:4]
```

With such a vector it is also possible to select elements by name:

```
> xy["c"]  
c  
7
```

This feature makes it possible to use such a named vector as a look-up vector/table to match the values to values of another vector or column in dataframe. Considering the following dataframe:

```
mydf <- data.frame(let = c('c', 'a', 'b', 'd'))  
  
> mydf  
let  
1 c  
2 a  
3 b  
4 d
```

Suppose you want to create a new variable in the mydf dataframe called num with the correct values from xy in the rows. Using the **match** function the appropriate values from xy can be selected:

```
mydf$num <- xy[match(mydf$let, names(xy))]
```

which results in:

```
> mydf  
let num  
1 c 7  
2a 5  
3b 6  
4d 8
```

第11.3节：数字序列

使用:运算符创建数字序列，例如用于向量化代码中的较大部分：

```
x <- 1:5  
x  
## [1] 1 2 3 4 5
```

这两种情况都适用

```
10:4  
# [1] 10 9 8 7 6 5 4
```

甚至适用于浮点数

```
1.25:5  
# [1] 1.25 2.25 3.25 4.25
```

或者负数

```
-4:4  
#[1] -4 -3 -2 -1 0 1 2 3 4
```

第11.4节：seq()

seq 是一个比 : 运算符更灵活的函数，允许指定非1的步长。

该函数从 start (默认值为1) 创建一个序列，直到结束值 (包含该数)。

你可以只提供结束参数 (to)

```
seq(5)  
# [1] 1 2 3 4 5
```

也可以同时指定起始值

```
seq(2, 5) # 或 seq(from=2, to=5)  
# [1] 2 3 4 5
```

最后还可以指定步长 (by)

```
seq(2, 5, 0.5) # 或 seq(from=2, to=5, by=0.5)  
# [1] 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

当提供输出的期望长度 (length.out) 时，seq 可以选择推断 (等间距的) 步长

```
seq(2,5, length.out = 10)
```

```
> mydf  
let num  
1 c 7  
2 a 5  
3 b 6  
4 d 8
```

Section 11.3: Sequence of numbers

Use the : operator to create sequences of numbers, such as for use in vectorizing larger chunks of your code:

```
x <- 1:5  
x  
## [1] 1 2 3 4 5
```

This works both ways

```
10:4  
# [1] 10 9 8 7 6 5 4
```

and even with floating point numbers

```
1.25:5  
# [1] 1.25 2.25 3.25 4.25
```

or negatives

```
-4:4  
#[1] -4 -3 -2 -1 0 1 2 3 4
```

Section 11.4: seq()

seq is a more flexible function than the : operator allowing to specify steps other than 1.

The function creates a sequence from the start (default is 1) to the end including that number.

You can supply only the end (to) parameter

```
seq(5)  
# [1] 1 2 3 4 5
```

As well as the start

```
seq(2, 5) # or seq(from=2, to=5)  
# [1] 2 3 4 5
```

And finally the step (by)

```
seq(2, 5, 0.5) # or seq(from=2, to=5, by=0.5)  
# [1] 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

seq can optionally infer the (evenly spaced) steps when alternatively the desired length of the output (length.out) is supplied

```
seq(2,5, length.out = 10)
```

```
# [1] 2.0 2.3 2.6 2.9 3.2 3.5 3.8 4.1 4.4 4.7 5.0
```

如果序列需要与另一个向量长度相同，我们可以使用`along.with`作为
`length.out = length(x)`的简写

```
x = 1:8
seq(2,5,along.with = x)
# [1] 2.000000 2.428571 2.857143 3.285714 3.714286 4.142857 4.571429 5.000000
```

在`seq`系列中有两个有用的简化函数：`seq_along`、`seq_len`和`seq.int`。`seq_along`和
`seq_len`函数构造从1到N的自然数（计数数），其中N由
函数参数决定，`seq_along`通过向量或列表的长度确定，`seq_len`通过整数参数确定。

```
seq_along(x)
# [1] 1 2 3 4 5 6 7 8
```

注意`seq_along`返回的是现有对象的索引。

```
# 计数数字1到10
seq_len(10)
[1] 1 2 3 4 5 6 7 8 9 10
# 使用 seq_along 的现有向量（或列表）索引
letters[1:10]
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
seq_along(letters[1:10])
[1] 1 2 3 4 5 6 7 8 9 10
```

`seq.int`与 `seq` 相同，保持了向后兼容性。

还有一个旧函数 `sequence`，用于从非负参数创建序列向量。

```
sequence(4)
# [1] 1 2 3 4
sequence(c(3, 2))
# [1] 1 2 3 1 2
sequence(c(3, 2, 5))
# [1] 1 2 3 1 2 1 2 3 4 5
```

第11.5节：向量

R中的向量可以有不同的类型（例如整数、逻辑、字符）。定义向量最通用的方法是通过
使用函数 `vector()`。

```
vector('integer', 2) # 创建一个大小为2的整数向量。
vector('character', 2) # 创建一个大小为2的字符向量。
vector('logical', 2) # 创建一个大小为2的逻辑向量。
```

然而，在R语言中，简写函数通常更受欢迎。

```
integer(2) # 与 vector('integer', 2) 相同，创建一个包含两个元素的整数向量。
character(2) # 与 vector('integer', 2) 相同，创建一个包含两个元素的字符向量。
logical(2) # 与 vector('logical', 2) 相同，创建一个包含两个元素的逻辑向量。
```

创建具有非默认值的向量也是可能的。通常使用函数`c()`来实现。

`c` 是 `combine`（合并）或 `concatenate`（连接）的缩写。

```
# [1] 2.0 2.3 2.6 2.9 3.2 3.5 3.8 4.1 4.4 4.7 5.0
```

If the sequence needs to have the same length as another vector we can use the `along.with` as a shorthand for
`length.out = length(x)`

```
x = 1:8
seq(2,5,along.with = x)
# [1] 2.000000 2.428571 2.857143 3.285714 3.714286 4.142857 4.571429 5.000000
```

There are two useful simplified functions in the `seq` family: `seq_along`, `seq_len`, and `seq.int`. `seq_along` and
`seq_len` functions construct the natural (counting) numbers from 1 through N where N is determined by the
function argument, the length of a vector or list with `seq_along`, and the integer argument with `seq_len`.

```
seq_along(x)
# [1] 1 2 3 4 5 6 7 8
```

Note that `seq_along` returns the indices of an existing object.

```
# counting numbers 1 through 10
seq_len(10)
[1] 1 2 3 4 5 6 7 8 9 10
# indices of existing vector (or list) with seq_along
letters[1:10]
[1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
seq_along(letters[1:10])
[1] 1 2 3 4 5 6 7 8 9 10
```

`seq.int` is the same as `seq` maintained for ancient compatibility.

There is also an old function `sequence` that creates a vector of sequences from a non negative argument.

```
sequence(4)
# [1] 1 2 3 4
sequence(c(3, 2))
# [1] 1 2 3 1 2
sequence(c(3, 2, 5))
# [1] 1 2 3 1 2 1 2 3 4 5
```

Section 11.5: Vectors

Vectors in R can have different types (e.g. integer, logical, character). The most general way of defining a vector is by
using the function `vector()`.

```
vector('integer', 2) # creates a vector of integers of size 2.
vector('character', 2) # creates a vector of characters of size 2.
vector('logical', 2) # creates a vector of logicals of size 2.
```

However, in R, the shorthand functions are generally more popular.

```
integer(2) # is the same as vector('integer', 2) and creates an integer vector with two elements
character(2) # is the same as vector('integer', 2) and creates an character vector with two elements
logical(2) # is the same as vector('logical', 2) and creates an logical vector with two elements
```

Creating vectors with values, other than the default values, is also possible. Often the function `c()` is used for this.
The `c` is short for `combine` or `concatenate`.

```
c(1, 2) # 创建一个包含两个元素1和2的整数向量。  
c('a', 'b') # 创建一个包含两个元素a和b的字符向量。  
c(T,F) # 创建一个包含两个元素的逻辑向量：TRUE 和 FALSE。
```

这里需要注意的是，R 将任何整数（例如 1）解释为大小为一的整数向量。数值型（例如 1.1）、逻辑型（例如 T 或 F）或字符型（例如 'a'）也是如此。因此，本质上你是在组合向量，而向量本身又是向量。

请注意，你必须始终组合相似类型的向量。否则，R 会尝试将向量转换为相同类型的向量。

```
c(1,1.1,'a',T) # 所有类型（整数、数值、字符和逻辑）都会被转换为“最低”类型，即字符型。
```

可以使用 [运算符来查找向量中的元素。

```
vec_int <- c(1,2,3)  
vec_char <- c('a','b','c')  
vec_int[2] # 访问第二个元素将返回 2  
vec_char[2] # 访问第二个元素将返回 'b'
```

这也是用来修改值

```
vec_int[2] <- 5 # 将第二个值从 2 改为 5  
vec_int # 返回 [1] 1 5 3
```

最后，: 运算符（函数 seq() 的简写）可以用来快速创建一个数字向量。

```
vec_int <- 1:10  
vec_int # 返回 [1] 1 2 3 4 5 6 7 8 9 10
```

这也用来对向量进行子集选择（从简单到更复杂的子集）

```
vec_char <- c('a','b','c','d','e')  
vec_char[2:4] # 返回 [1] "b" "c" "d"  
vec_char[c(1,3,5)] # 返回 [1] "a" "c" "e"
```

第11.6节：使用 rep() 函数扩展向量

rep 函数可以用来以相当灵活的方式重复一个向量。

```
# 重复计数数字，1 到 5 两次  
rep(1:5, 2)  
[1] 1 2 3 4 5 1 2 3 4 5  
  
# 使用不完全回收重复向量  
rep(1:5, 2, length.out=7)  
[1] 1 2 3 4 5 1 2
```

each 参数对于将观测/实验单元的统计量向量扩展为包含这些单元重复观测的数据框向量特别有用。

```
# 与之前相同，只是将每个整数重复放在一起  
rep(1:5, each=2)  
[1] 1 1 2 2 3 3 4 4 5 5
```

```
c(1, 2) # creates a integer vector of two elements: 1 and 2.  
c('a', 'b') # creates a character vector of two elements: a and b.  
c(T,F) # creates a logical vector of two elements: TRUE and FALSE.
```

Important to note here is that R interprets any integer (e.g. 1) as an integer vector of size one. The same holds for numerics (e.g. 1.1), logicals (e.g. T or F), or characters (e.g. 'a'). Therefore, you are in essence combining vectors, which in turn are vectors.

Pay attention that you always have to combine similar vectors. Otherwise, R will try to convert the vectors in vectors of the same type.

```
c(1,1.1,'a',T) # all types (integer, numeric, character and logical) are converted to the 'lowest' type which is character.
```

Finding elements in vectors can be done with the [operator.

```
vec_int <- c(1,2,3)  
vec_char <- c('a','b','c')  
vec_int[2] # accessing the second element will return 2  
vec_char[2] # accessing the second element will return 'b'
```

This can also be used to change values

```
vec_int[2] <- 5 # change the second value from 2 to 5  
vec_int # returns [1] 1 5 3
```

Finally, the : operator (short for the function seq()) can be used to quickly create a vector of numbers.

```
vec_int <- 1:10  
vec_int # returns [1] 1 2 3 4 5 6 7 8 9 10
```

This can also be used to subset vectors (from easy to more complex subsets)

```
vec_char <- c('a','b','c','d','e')  
vec_char[2:4] # returns [1] "b" "c" "d"  
vec_char[c(1,3,5)] # returns [1] "a" "c" "e"
```

Section 11.6: Expanding a vector with the rep() function

The rep function can be used to repeat a vector in a fairly flexible manner.

```
# repeat counting numbers, 1 through 5 twice  
rep(1:5, 2)  
[1] 1 2 3 4 5 1 2 3 4 5  
  
# repeat vector with incomplete recycling  
rep(1:5, 2, length.out=7)  
[1] 1 2 3 4 5 1 2
```

The each argument is especially useful for expanding a vector of statistics of observational/experimental units into a vector of data.frame with repeated observations of these units.

```
# same except repeat each integer next to each other  
rep(1:5, each=2)  
[1] 1 1 2 2 3 3 4 4 5 5
```

`rep` 的一个很好的特性是，它可以将向量扩展为不平衡面板数据结构，这可以通过用一个向量替代 `length` 参数来实现，该向量决定了向量中每个元素重复的次数：

```
# 自动长度重复  
rep(1:5, 1:5)  
[1] 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5  
# 手动指定重复长度向量  
rep(1:5, c(1,1,1,2,2))  
[1] 1 2 3 4 4 5 5
```

这说明可以通过外部函数动态生成 `rep` 的第二个参数，从而根据数据动态构造扩展向量的可能性。

与 `seq` 类似，`rep` 的更快、更简化版本是 `rep_len` 和 `rep.int`。这些版本省略了 `rep` 所维护的一些属性，因此在速度优先且不需要重复向量的额外方面时可能最为有用。

```
# 重复计数数字，1 到 5 两次  
rep.int(1:5, 2)  
[1] 1 2 3 4 5 1 2 3 4 5  
  
# 使用不完全回收重复向量  
rep_len(1:5, length.out=7)  
[1] 1 2 3 4 5 1 2
```

A nice feature of `rep` regarding involving expansion to such a data structure is that expansion of a vector to an unbalanced panel can be accomplished by replacing the `length` argument with a vector that dictates the number of times to repeat each element in the vector:

```
# automated length repetition  
rep(1:5, 1:5)  
[1] 1 2 2 3 3 3 4 4 4 4 5 5 5 5 5  
# hand-fed repetition length vector  
rep(1:5, c(1,1,1,2,2))  
[1] 1 2 3 4 4 5 5
```

This should expose the possibility of allowing an external function to feed the second argument of `rep` in order to dynamically construct a vector that expands according to the data.

As with `seq`, faster, simplified versions of `rep` are `rep_len` and `rep.int`. These drop some attributes that `rep` maintains and so may be most useful in situations where speed is a concern and additional aspects of the repeated vector are unnecessary.

```
# repeat counting numbers, 1 through 5 twice  
rep.int(1:5, 2)  
[1] 1 2 3 4 5 1 2 3 4 5  
  
# repeat vector with incomplete recycling  
rep_len(1:5, length.out=7)  
[1] 1 2 3 4 5 1 2
```

第12章：日期和时间

R 包含日期、日期时间和时间差的类；请参见 `?Dates`、`?DateTimeClasses`、`?difftime`，并查看这些文档的“参见”部分以获取更多文档。相关文档：日期和日期时间类。

第12.1节：当前日期和时间

R能够访问当前的日期、时间和时区：

```
Sys.Date()          # 返回日期作为Date对象  
## [1] "2016-07-21"  
  
Sys.time()          # 返回当前区域设置的日期和时间，作为POSIXct对象  
## [1] "2016-07-21 10:04:39 CDT"  
  
as.numeric(Sys.time()) # 从UNIX纪元 (1970-01-01 00:00:00 UTC) 起的秒数  
## [1] 1469113479  
  
Sys.timezone()      # 当前所在位置的时区  
## [1] "Australia/Melbourne"
```

使用`OlsonNames()`查看当前系统中Olson/IANA数据库的时区名称：

```
str(OlsonNames())  
## chr [1:589] "Africa/Abidjan" "Africa/Accra" "Africa/Addis_Ababa" "Africa/Algiers"  
"Africa/Asmara" "Africa/Asmera" "Africa/Bamako" ...
```

第12.2节：跳转到月末

假设我们想跳转到当月的最后一天，这个函数可以帮忙实现：

```
eom <- function(x, p=as.POSIXlt(x)) as.Date(modifyList(p, list(mon=p$mon + 1, mday=0)))
```

测试：

```
x <- seq(as.POSIXct("2000-12-10"), as.POSIXct("2001-05-10"), by="months")  
> data.frame(before=x, after=eom(x))  
    before      after  
1 2000-12-10 2000-12-31  
2 2001-01-10 2001-01-31  
3 2001-02-10 2001-02-28  
4 2001-03-10 2001-03-31  
5 2001-04-10 2001-04-30  
6 2001-05-10 2001-05-31  
>
```

使用字符串格式的日期：

```
> eom('2000-01-01')  
[1] "2000-01-31"
```

Chapter 12: Date and Time

R comes with classes for dates, date-times and time differences; see `?Dates`, `?DateTimeClasses`, `?difftime` and follow the "See Also" section of those docs for further documentation. Related Docs: Dates and Date-Time Classes.

Section 12.1: Current Date and Time

R is able to access the current date, time and time zone:

```
Sys.Date()          # Returns date as a Date object  
## [1] "2016-07-21"  
  
Sys.time()          # Returns date & time at current locale as a POSIXct object  
## [1] "2016-07-21 10:04:39 CDT"  
  
as.numeric(Sys.time()) # Seconds from UNIX Epoch (1970-01-01 00:00:00 UTC)  
## [1] 1469113479  
  
Sys.timezone()      # Time zone at current location  
## [1] "Australia/Melbourne"
```

Use `OlsonNames()` to view the time zone names in Olson/IANA database on the current system:

```
str(OlsonNames())  
## chr [1:589] "Africa/Abidjan" "Africa/Accra" "Africa/Addis_Ababa" "Africa/Algiers"  
"Africa/Asmara" "Africa/Asmera" "Africa/Bamako" ...
```

Section 12.2: Go to the End of the Month

Let's say we want to go to the last day of the month, this function will help on it:

```
eom <- function(x, p=as.POSIXlt(x)) as.Date(modifyList(p, list(mon=p$mon + 1, mday=0)))
```

Test:

```
x <- seq(as.POSIXct("2000-12-10"), as.POSIXct("2001-05-10"), by="months")  
> data.frame(before=x, after=eom(x))  
    before      after  
1 2000-12-10 2000-12-31  
2 2001-01-10 2001-01-31  
3 2001-02-10 2001-02-28  
4 2001-03-10 2001-03-31  
5 2001-04-10 2001-04-30  
6 2001-05-10 2001-05-31  
>
```

Using a date in a string format:

```
> eom('2000-01-01')  
[1] "2000-01-31"
```

第12.3节：跳转到当月第一天

假设我们想跳转到某个月的第一天：

```
date <- as.Date("2017-01-20")
> as.POSIXlt(cut(date, "month"))
[1] "2017-01-01 EST"
```

第12.4节：将日期按月数一致移动

假设我们想将给定日期移动num个月。我们可以定义以下函数，使用mondate包：

```
moveNumOfMonths <- function(date, num) {
  as.Date(mondate(date) + num)
}
```

它会一致地移动日期的月份部分，并在日期指向当月最后一天时调整日期。

例如：

向前一个月：

```
> moveNumOfMonths("2017-10-30", -1)
[1] "2017-09-30"
```

向前两个月：

```
> moveNumOfMonths("2017-10-30", -2)
[1] "2017-08-30"
```

向后两个月：

```
> moveNumOfMonths("2017-02-28", 2)
[1] "2017-04-30"
```

它从二月的最后一天向后移动两个月，因此是四月的最后一天。

让我们看看当日期是月末时，向前和向后操作是如何工作的：

```
> moveNumOfMonths("2016-11-30", 2)
[1] "2017-01-31"
> moveNumOfMonths("2017-01-31", -2)
[1] "2016-11-30"
```

因为十一月有30天，向后操作时我们得到相同的日期，但：

```
> moveNumOfMonths("2017-01-30", -2)
[1] "2016-11-30"
> moveNumOfMonths("2016-11-30", 2)
[1] "2017-01-31"
```

Section 12.3: Go to First Day of the Month

Let's say we want to go to the first day of a given month:

```
date <- as.Date("2017-01-20")
> as.POSIXlt(cut(date, "month"))
[1] "2017-01-01 EST"
```

Section 12.4: Move a date a number of months consistently by months

Let's say we want to move a given date a numof months. We can define the following function, that uses the mondate package:

```
moveNumOfMonths <- function(date, num) {
  as.Date(mondate(date) + num)
}
```

It moves consistently the month part of the date and adjusting the day, in case the date refers to the last day of the month.

For example:

Back one month:

```
> moveNumOfMonths("2017-10-30", -1)
[1] "2017-09-30"
```

Back two months:

```
> moveNumOfMonths("2017-10-30", -2)
[1] "2017-08-30"
```

Forward two months:

```
> moveNumOfMonths("2017-02-28", 2)
[1] "2017-04-30"
```

It moves two months from the last day of February, therefore the last day of April.

Let's see how it works for backward and forward operations when it is the last day of the month:

```
> moveNumOfMonths("2016-11-30", 2)
[1] "2017-01-31"
> moveNumOfMonths("2017-01-31", -2)
[1] "2016-11-30"
```

Because November has 30 days, we get the same date in the backward operation, but:

```
> moveNumOfMonths("2017-01-30", -2)
[1] "2016-11-30"
> moveNumOfMonths("2016-11-30", 2)
[1] "2017-01-31"
```

因为一月有31天，所以从十一月的最后一天向后移动两个月会得到一月的最后一天。

Because January has 31 days, then moving two months from last day of November will get the last day of January.

第13章：日期类

第13.1节：日期格式化

要格式化日期，我们使用`format(date, format = "%Y-%m-%d")`函数，参数可以是`POSIXct`（由`as.POSIXct()`生成）或`POSIXlt`（由`as.POSIXlt()`生成）

```
d = as.Date("2016-07-21") # 当前日期时间戳

format(d, "%a")          # 缩写星期几
## [1] "Thu"

format(d,"%A")           # 完整星期几名称
## [1] "Thursday"

format(d,"%b")           # 缩写月份
## [1] "Jul"

format(d,"%B")           # 完整月份名称
## [1] "July"

format(d,"%m")            # 00-12 月份格式
## [1] "07"

format(d,"%d")            # 00-31 日期格式
## [1] "21"

format(d,"%e")            # 0-31 日期格式
## [1] "21"

format(d,"%y")            # 00-99 年份
## [1] "16"

format(d,"%Y")            # 带世纪的年份
## [1] "2016"
```

更多内容，请参见`?strptime`。

第13.2节：将字符串解析为日期对象

R包含一个Date类，通过`as.Date()`创建，该函数接受一个字符串或字符串向量，如果日期不是ISO 8601日期格式`YYYY-MM-DD`，则需要一个`strptime`风格的格式字符串。

```
as.Date('2016-08-01')    # ISO格式，因此不需要格式字符串
## [1] "2016-08-01"

as.Date('05/23/16', format = '%m/%d/%y')
## [1] "2016-05-23"

as.Date('March 23rd, 2016', '%B %d %Y')    # 在格式中添加分隔符和文字
## [1] "2016-03-23"

as.Date(' 2016-08-01 foo')    # 忽略前导空白和所有尾随字符
## [1] "2016-08-01"

as.Date(c('2016-01-01', '2016-01-02'))
# [1] "2016-01-01" "2016-01-02"
```

Chapter 13: The Date class

Section 13.1: Formatting Dates

To format Dates we use the `format(date, format = "%Y-%m-%d")` function with either the `POSIXct` (given from `as.POSIXct()`) or `POSIXlt` (given from `as.POSIXlt()`)

```
d = as.Date("2016-07-21") # Current Date Time Stamp

format(d, "%a")          # Abbreviated Weekday
## [1] "Thu"

format(d,"%A")           # Full Weekday
## [1] "Thursday"

format(d,"%b")           # Abbreviated Month
## [1] "Jul"

format(d,"%B")           # Full Month
## [1] "July"

format(d,"%m")            # 00-12 Month Format
## [1] "07"

format(d,"%d")            # 00-31 Day Format
## [1] "21"

format(d,"%e")            # 0-31 Day Format
## [1] "21"

format(d,"%y")            # 00-99 Year
## [1] "16"

format(d,"%Y")            # Year with Century
## [1] "2016"
```

For more, see `?strptime`.

Section 13.2: Parsing Strings into Date Objects

R contains a Date class, which is created with `as.Date()`, which takes a string or vector of strings, and if the date is not in ISO 8601 date format `YYYY-MM-DD`, a formatting string of `strptime`-style tokens.

```
as.Date('2016-08-01')    # in ISO format, so does not require formatting string
## [1] "2016-08-01"

as.Date('05/23/16', format = '%m/%d/%y')
## [1] "2016-05-23"

as.Date('March 23rd, 2016', '%B %d %Y')    # add separators and literals to format
## [1] "2016-03-23"

as.Date(' 2016-08-01 foo')    # leading whitespace and all trailing characters are ignored
## [1] "2016-08-01"

as.Date(c('2016-01-01', '2016-01-02'))
# [1] "2016-01-01" "2016-01-02"
```

第13.3节：日期

要将变量强制转换为日期，请使用as.Date()函数。

```
> x <- as.Date("2016-8-23")
> x
[1] "2016-08-23"
> class(x)
[1] "日期"
```

as.Date() 函数允许你提供一个 format 参数。默认格式是 %Y-%m-%d，即年-月-日。

```
> as.Date("23-8-2016", format="%d-%m-%Y") # 读取欧洲格式的日期
[1] "2016-08-23"
```

格式字符串可以用单引号或双引号括起来。日期通常有多种表达形式，如：“d-m-yy” 或 “d-m-YYYY” 或 “m-d-yy” 或 “m-d-YYYY” 或 “YYYY-m-d” 或 “YYYY-d-m”。

这些格式也可以通过将 “-” 替换为 “/” 来表示。此外，日期还可以表示为 “Nov 6, 1986”、“November 6, 1986”、“6 Nov, 1986”、“6 November, 1986” 等形式。as.Date() 函数接受所有这些字符串，并且当我们指定字符串的正确格式时，它总是输出 “YYYY-m-d” 形式的日期。

假设我们有一个日期字符串 “9-6-1962”，格式为 “%d-%m-%Y”。

```
#  
# 它尝试将字符串解释为 YYYY-m-d  
#
> as.Date("9-6-1962")
[1] "0009-06-19"      #解释为 "%Y-%m-%d"
>
as.Date("9/6/1962")
[1] "0009-06-19"      # 仍然被解释为 "%Y-%m-%d"
>
# 如果日期格式为 YYYY-m-d 或 YYYY/m/d，理解上没有问题
#
> as.Date("1962-6-9")
[1] "1962-06-09"      # 没有问题
> as.Date("1962/6/9")
[1] "1962-06-09"      # 没有问题
>
```

通过指定输入字符串的正确格式，我们可以得到期望的结果。我们使用以下代码为as.Date()函数指定格式。

格式代码	含义
%d	日
%m	月份
%y	两位数年份
%Y	四位数年份
%b	三字符缩写月份
%B	月份全称

考虑以下指定format参数的示例：

Section 13.3: Dates

To coerce a variable to a date use the **as.Date()** function.

```
> x <- as.Date("2016-8-23")
> x
[1] "2016-08-23"
> class(x)
[1] "Date"
```

The **as.Date()** function allows you to provide a format argument. The default is %Y-%m-%d, which is Year-month-day.

```
> as.Date("23-8-2016", format="%d-%m-%Y") # To read in an European-style date
[1] "2016-08-23"
```

The format string can be placed either within a pair of single quotes or double quotes. Dates are usually expressed in a variety of forms such as: “d-m-yy” or “d-m-YYYY” or “m-d-yy” or “m-d-YYYY” or “YYYY-m-d” or “YYYY-d-m”. These formats can also be expressed by replacing “-” by “/”. Further, dates are also expressed in the forms, say, “Nov 6, 1986” or “November 6, 1986” or “6 Nov, 1986” or “6 November, 1986” and so on. The **as.Date()** function accepts all such character strings and when we mention the appropriate format of the string, it always outputs the date in the form “YYYY-m-d”.

Suppose we have a date string “9-6-1962” in the format “%d-%m-%Y”.

```
#  
# It tries to interprets the string as YYYY-m-d  
#
> as.Date("9-6-1962")
[1] "0009-06-19"      #interprets as "%Y-%m-%d"
>
as.Date("9/6/1962")
[1] "0009-06-19"      #again interprets as "%Y-%m-%d"
>
# It has no problem in understanding, if the date is in form YYYY-m-d or YYYY/m/d
#
> as.Date("1962-6-9")
[1] "1962-06-09"      # no problem
> as.Date("1962/6/9")
[1] "1962-06-09"      # no problem
>
```

By specifying the correct format of the input string, we can get the desired results. We use the following codes for specifying the formats to the **as.Date()** function.

Format Code	Meaning
%d	day
%m	month
%y	year in 2-digits
%Y	year in 4-digits
%b	abbreviated month in 3 chars
%B	full name of the month

Consider the following example specifying the **format** parameter:

```
> as.Date("9-6-1962", format="%d-%m-%Y")
[1] "1962-06-09"
>
```

参数名format可以省略。

```
> as.Date("9-6-1962", "%d-%m-%Y")
[1] "1962-06-09"
>
```

有时，月份名称会缩写为前三个字母来书写日期。在这种情况下，我们使用格式说明符%b。

```
> as.Date("6Nov1962", "%d%b%Y")
[1] "1962-11-06"
>
```

注意，日期字符串中各部分之间没有'-'、'/'或空格。格式字符串应与输入字符串完全匹配。请看以下示例：

```
> as.Date("6 Nov, 1962", "%d %b, %Y")
[1] "1962-11-06"
>
```

注意，日期字符串中有逗号，因此格式说明中也有逗号。如果格式字符串中省略逗号，则结果为NA。以下是%B格式说明符的示例用法：

```
> as.Date("2016年10月12日", "%B %d, %Y")
[1] "2016-10-12"

as.Date("2016年10月12日", "%d %B, %Y")
[1] "2016-10-12"
>
```

%y 格式是系统特定的，因此应谨慎使用。与此函数一起使用的其他参数有 origin 和 tz (时区)。

```
> as.Date("9-6-1962", format="%d-%m-%Y")
[1] "1962-06-09"
>
```

The parameter name **format** can be omitted.

```
> as.Date("9-6-1962", "%d-%m-%Y")
[1] "1962-06-09"
>
```

Some times, names of the months abbreviated to the first three characters are used in the writing the dates. In which case we use the format specifier %b.

```
> as.Date("6Nov1962", "%d%b%Y")
[1] "1962-11-06"
>
```

Note that, there are no either '-' or '/' or white spaces between the members in the date string. The format string should exactly match that input string. Consider the following example:

```
> as.Date("6 Nov, 1962", "%d %b, %Y")
[1] "1962-11-06"
>
```

Note that, there is a comma in the date string and hence a comma in the format specification too. If comma is omitted in the format string, it results in an NA. An example usage of %B format specifier is as follows:

```
> as.Date("October 12, 2016", "%B %d, %Y")
[1] "2016-10-12"
>
> as.Date("12 October, 2016", "%d %B, %Y")
[1] "2016-10-12"
>
```

%y format is system specific and hence, should be used with caution. Other parameters used with this function are **origin** and **tz** (time zone).

第14章：日期时间类（POSIXct 和 POSIXlt）

R 包含两种日期时间类——POSIXct 和 POSIXlt——参见 ?DateTimeClasses。

第14.1节：格式化和打印日期时间对象

```
# 测试日期时间对象
options(digits.secs = 3)
d = as.POSIXct("2016-08-30 14:18:30.58", tz = "UTC")

format(d,"%S") # 00-61 秒，整数表示
## [1] "30"

format(d,"%OS") # 00-60.99... 秒（小数部分）
## [1] "30.579"

format(d,"%M") # 00-59 分钟
## [1] "18"

format(d,"%H") # 00-23 小时
## [1] "14"

format(d,"%I") # 01-12 小时
## [1] "02"

format(d,"%p") # 上午/下午 指示符
## [1] "PM"

format(d,"%z") # 带符号的时区偏移
## [1] "+0000"

format(d,"%Z") # 时区缩写
## [1] "UTC"
```

有关格式字符串的详细信息以及其他格式，请参见?strptime。

第14.2节：日期时间运算

要进行时间的加减，使用POSIXct，因为它以秒为单位存储时间

```
## 加减时间 - 60秒
as.POSIXct("2016-01-01") + 60
# [1] "2016-01-01 00:01:00 AEDT"

## 加3小时14分钟15秒
as.POSIXct("2016-01-01") + ( (3 * 60 * 60) + (14 * 60) + 15 )
# [1] "2016-01-01 03:14:15 AEDT"
```

更正式地，可以使用as.difftime来指定要加到日期或日期时间对象上的时间段。例如：

```
as.POSIXct("2016-01-01") +
  as.difftime(3, units="hours") +
  as.difftime(14, units="mins") +
  as.difftime(15, units="secs")
# [1] "2016-01-01 03:14:15 AEDT"
```

Chapter 14: Date-time classes (POSIXct and POSIXlt)

R includes two date-time classes -- POSIXct and POSIXlt -- see ?DateTimeClasses.

Section 14.1: Formatting and printing date-time objects

```
# test date-time object
options(digits.secs = 3)
d = as.POSIXct("2016-08-30 14:18:30.58", tz = "UTC")

format(d,"%S") # 00-61 Second as integer
## [1] "30"

format(d,"%OS") # 00-60.99... Second as fractional
## [1] "30.579"

format(d,"%M") # 00-59 Minute
## [1] "18"

format(d,"%H") # 00-23 Hours
## [1] "14"

format(d,"%I") # 01-12 Hours
## [1] "02"

format(d,"%p") # AM/PM Indicator
## [1] "PM"

format(d,"%z") # Signed offset
## [1] "+0000"

format(d,"%Z") # Time Zone Abbreviation
## [1] "UTC"
```

See ?[strptime](#) for details on the format strings here, as well as other formats.

Section 14.2: Date-time arithmetic

To add/subtract time, use POSIXct, since it stores times in seconds

```
## adding/subtracting times - 60 seconds
as.POSIXct("2016-01-01") + 60
# [1] "2016-01-01 00:01:00 AEDT"

## adding 3 hours, 14 minutes, 15 seconds
as.POSIXct("2016-01-01") + ( (3 * 60 * 60) + (14 * 60) + 15 )
# [1] "2016-01-01 03:14:15 AEDT"
```

More formally, [as.difftime](#) can be used to specify time periods to add to a date or datetime object. E.g.:

```
as.POSIXct("2016-01-01") +
  as.difftime(3, units="hours") +
  as.difftime(14, units="mins") +
  as.difftime(15, units="secs")
# [1] "2016-01-01 03:14:15 AEDT"
```

要计算日期/时间之间的差异，使用 `difftime()`，可计算秒、分钟、小时、天或周的差异。

```
# 使用POSIXct对象
difftime(
  as.POSIXct("2016-01-01 12:00:00"),
  as.POSIXct("2016-01-01 11:59:59"),
  unit = "secs")
# 时间差为1秒
```

要生成日期时间序列，请使用`seq.POSIXt()`或简单地使用`seq`。

第14.3节：将字符串解析为日期时间对象

将字符串解析为POSIXct和POSIXlt的函数接受类似的参数并返回类似的结果，但存储该日期时间的方式有所不同；详见“备注”。

```
as.POSIXct("11:38",           # 时间字符串
           format = "%H:%M")      # 格式化字符串
## [1] "2016-07-21 11:38:00 CDT"
strptime("11:38",            # 相同，但生成POSIXlt对象
         format = "%H:%M")
## [1] "2016-07-21 11:38:00 CDT"

as.POSIXct("11 AM",          # time string
           format = "%I %p")
## [1] "2016-07-21 11:00:00 CDT"
```

注意日期和时区是推断的。

```
as.POSIXct("11:38:22",        # 无时区的时间字符串
           format = "%H:%M:%S",
           tz = "America/New_York")  # 设置时区
## [1] "2016-07-21 11:38:22 EDT"

as.POSIXct("2016-07-21 00:00:00",
           format = "%F %T")       # "%Y-%m-%d" 和 "%H:%M:%S" 的快捷符号
```

有关格式字符串的详细信息，请参见 `?strptime`。

注意事项

缺失的元素

- 如果未提供日期元素，则使用当前日期的对应元素。
- 如果未提供时间元素，则使用午夜时间，即0秒。
- 如果字符串或 `tz` 参数中均未提供时区，则使用本地时区。

时区

- `tz` 的可接受值取决于位置。
 - CST 可用 `"CST6CDT"` 或 `"America/Chicago"`
- 支持的位置和时区请使用：
 - 在 R 中：`OlsonNames()`
 - 或者，在 R 中尝试：`system("cat $R_HOME/share/zoneinfo/zone.tab")`
- 这些位置由[互联网号码分配局 \(IANA\)](#)
 - [时区数据库时区列表 \(维基百科\)](#) 提供

To find the difference between dates/times use `difftime()` for differences in seconds, minutes, hours, days or weeks.

```
# using POSIXct objects
difftime(
  as.POSIXct("2016-01-01 12:00:00"),
  as.POSIXct("2016-01-01 11:59:59"),
  unit = "secs")
# Time difference of 1 secs
```

To generate sequences of date-times use `seq.POSIXt()` or simply `seq`.

Section 14.3: Parsing strings into date-time objects

The functions for parsing a string into POSIXct and POSIXlt take similar parameters and return a similar-looking result, but there are differences in how that date-time is stored; see "Remarks."

```
as.POSIXct("11:38",           # time string
           format = "%H:%M")      # formatting string
## [1] "2016-07-21 11:38:00 CDT"
strptime("11:38",            # identical, but makes a POSIXlt object
         format = "%H:%M")
## [1] "2016-07-21 11:38:00 CDT"

as.POSIXct("11 AM",          # identical, but makes a POSIXlt object
           format = "%I %p")
## [1] "2016-07-21 11:00:00 CDT"
```

Note that date and timezone are imputed.

```
as.POSIXct("11:38:22",        # time string without timezone
           format = "%H:%M:%S",
           tz = "America/New_York")  # set time zone
## [1] "2016-07-21 11:38:22 EDT"

as.POSIXct("2016-07-21 00:00:00",
           format = "%F %T")       # shortcut tokens for "%Y-%m-%d" and "%H:%M:%S"
```

See `?strptime` for details on the format strings here.

Notes

Missing elements

- If a date element is not supplied, then that from the current date is used.
- If a time element is not supplied, then that from midnight is used, i.e. 0s.
- If no timezone is supplied in either the string or the `tz` parameter, the local timezone is used.

Time zones

- The accepted values of `tz` depend on the location.
 - CST is given with `"CST6CDT"` or `"America/Chicago"`
- For supported locations and time zones use:
 - In R: `OlsonNames()`
 - Alternatively, try in R: `system("cat $R_HOME/share/zoneinfo/zone.tab")`
- These locations are given by [Internet Assigned Numbers Authority \(IANA\)](#)
 - [List of tz database time zones \(Wikipedia\)](#)

◦ [IANA 时区数据 \(2016e\)](#)

◦ [IANA TZ Data \(2016e\)](#)

第15章：字符类

字符是其他语言所称的“字符串向量”。

第15.1节：强制转换

要检查一个值是否为字符，使用`is.character()`函数。要将变量强制转换为字符，使用`as.character()`函数。

```
x <- "敏捷的棕色狐狸跳过了懒狗"  
class(x)  
[1] "character"  
is.character(x)  
[1] TRUE
```

注意，数值可以被强制转换为字符，但尝试将字符强制转换为数值可能会导致NA。

```
as.numeric("2")  
[1] 2  
as.numeric("fox")  
[1] NA  
警告信息:  
通过强制转换引入的NA
```

Chapter 15: The character class

Characters are what other languages call 'string vectors.'

Section 15.1: Coercion

To check whether a value is a character use the `is.character()` function. To coerce a variable to a character use the `as.character()` function.

```
x <- "The quick brown fox jumps over the lazy dog"  
class(x)  
[1] "character"  
is.character(x)  
[1] TRUE
```

Note that numerics can be coerced to characters, but attempting to coerce a character to numeric may result in NA.

```
as.numeric("2")  
[1] 2  
as.numeric("fox")  
[1] NA  
Warning message:  
NAs introduced by coercion
```

第16章：数值类和存储模式

第16.1节：数值型

数值型表示整数和双精度数，是分配给数字向量的默认模式。该函数 `is.numeric()` 将评估一个向量是否为数值型。需要注意的是，虽然整数和双精度数会通过 `is.numeric()`，函数 `as.numeric()` 总是尝试转换为双精度类型。

```
x <- 12.3
y <- 12L

# 确认类型
typeof(x)
[1] "double"
typeof(y)
[1] "integer"

# 确认两者均为数值
is.numeric(x)
[1] TRUE
is.numeric(y)
[1] TRUE

# 逻辑型转数值型
as.numeric(TRUE)
[1] 1

# 虽然 TRUE == 1，但它是双精度数而非整数
is.integer(as.numeric(TRUE))
[1] FALSE
```

双精度数是 R 的默认数值类型。它们是双精度向量，意味着向量中每个值占用 8 字节内存。R 没有单精度数据类型，因此所有实数都以双精度格式存储。

```
is.double(1)
TRUE
is.double(1.0)
TRUE
is.double(1L)
FALSE
```

整数是可以写成没有小数部分的整数。整数用一个数字后面跟一个L来表示。任何没有L的数字都将被视为双精度数（double）。

```
typeof(1)
[1] "double"
class(1)
[1] "numeric"
typeof(1L)
[1] "integer"
class(1L)
[1] "integer"
```

虽然在大多数情况下使用整数或双精度数没有区别，但有时用整数替代双精度数会

Chapter 16: Numeric classes and storage modes

Section 16.1: Numeric

Numeric represents integers and doubles and is the default mode assigned to vectors of numbers. The function `is.numeric()` will evaluate whether a vector is numeric. It is important to note that although integers and doubles will pass `is.numeric()`, the function `as.numeric()` will always attempt to convert to type double.

```
x <- 12.3
y <- 12L

# confirm types
typeof(x)
[1] "double"
typeof(y)
[1] "integer"

# confirm both numeric
is.numeric(x)
[1] TRUE
is.numeric(y)
[1] TRUE

# logical to numeric
as.numeric(TRUE)
[1] 1

# While TRUE == 1, it is a double and not an integer
is.integer(as.numeric(TRUE))
[1] FALSE
```

Doubles are R's default numeric value. They are double precision vectors, meaning that they take up 8 bytes of memory for each value in the vector. R has no single precision data type and so all real numbers are stored in the double precision format.

```
is.double(1)
TRUE
is.double(1.0)
TRUE
is.double(1L)
FALSE
```

Integers are whole numbers that can be written without a fractional component. Integers are represented by a number with an L after it. Any number without an L after it will be considered a double.

```
typeof(1)
[1] "double"
class(1)
[1] "numeric"
typeof(1L)
[1] "integer"
class(1L)
[1] "integer"
```

Though in most cases using an integer or double will not matter, sometimes replacing doubles with integers will

消耗更少的内存和运行时间。双精度向量每个元素使用8字节，而整数向量每个元素只使用4字节。随着向量大小的增加，使用合适的类型可以显著加快处理速度。

```
# 测试大量算术运算的速度
microbenchmark(
  for( i in 1:100000){
    2L * i
    10L + i
  },
  for( i in 1:100000){
    2.0 * i
    10.0 + i
  }
)
单位:毫秒
expr      min       lq     mean   median      uq
max neval
for (i in 1:1e+05) { 2L * i 10L + i } 40.74775 42.34747 50.70543 42.99120 65.46864
94.11804 100
for (i in 1:1e+05) { 2 * i 10 + i } 41.07807 42.38358 53.52588 44.26364 65.84971
83.00456 100
```

consume less memory and operational time. A double vector uses 8 bytes per element while an integer vector uses only 4 bytes per element. As the size of vectors increases, using proper types can dramatically speed up processes.

```
# test speed on lots of arithmetic
microbenchmark(
  for( i in 1:100000){
    2L * i
    10L + i
  },
  for( i in 1:100000){
    2.0 * i
    10.0 + i
  }
)
Unit: milliseconds
expr      min       lq     mean   median      uq
max neval
for (i in 1:1e+05) { 2L * i 10L + i } 40.74775 42.34747 50.70543 42.99120 65.46864
94.11804 100
for (i in 1:1e+05) { 2 * i 10 + i } 41.07807 42.38358 53.52588 44.26364 65.84971
83.00456 100
```

第17章：逻辑类

Logical 是向量的一个模式（以及一个隐式类）。

第17.1节：逻辑运算符

逻辑运算符有两种：一种接受并返回任意长度的向量（元素级运算符：`!`, `|`, `&`, `xor()`）和另一种只计算每个参数的第一个元素（`&&`, `||`）。第二种主要用作if函数的cond参数。

逻辑运算符	含义	语法
<code>!</code>	不	<code>!x</code>
<code>&</code>	逐元素（向量化）且	<code>x & y</code>
<code>&&</code>	且（仅单个元素）	<code>x && y</code>
<code> </code>	逐元素（向量化）或	<code>x y</code>
<code> </code>	或（仅单个元素）	<code>x y</code>
异或	逐元素（向量化）异或 <code>xor(x,y)</code>	

注意，`||` 运算符会先计算左侧条件，如果左侧条件为 TRUE，则右侧永远不会被计算。如果第一个条件是复杂操作的结果，这可以节省时间。类似地，`&&` 运算符当第一个参数的第一个元素为 FALSE 时，也会返回 FALSE 而不计算第二个参数。

```
> x <- 5
> x > 6 || stop("X 太小")
错误: X 太小
> x > 3 || stop("X 太小")
[1] TRUE
```

要检查一个值是否为逻辑值，可以使用 `is.logical()` 函数。

第17.2节：强制转换

要将变量强制转换为逻辑值，使用 `as.logical()` 函数。

```
> x <- 2
> z <- x > 4
> z
[1] FALSE
> class(x)
[1] "numeric"
> as.logical(2)
[1] TRUE
```

当对逻辑值应用`as.numeric()`时，将返回一个双精度数值。NA是一个逻辑值，且带有NA的逻辑运算符如果结果不确定，将返回NA。

第17.3节：NA的解释

详情请参见缺失值。

```
> TRUE & NA
[1] NA
> FALSE & NA
```

Chapter 17: The logical class

Logical is a mode (and an implicit class) for vectors.

Section 17.1: Logical operators

There are two sorts of logical operators: those that accept and return vectors of any length (elementwise operators: `!`, `|`, `&`, `xor()`) and those that only evaluate the first element in each argument (`&&`, `||`). The second sort is primarily used as the cond argument to the `if` function.

Logical Operator	Meaning	Syntax
<code>!</code>	Not	<code>!x</code>
<code>&</code>	element-wise (vectorized) and	<code>x & y</code>
<code>&&</code>	and (single element only)	<code>x && y</code>
<code> </code>	element-wise (vectorized) or	<code>x y</code>
<code> </code>	or (single element only)	<code>x y</code>
<code>xor</code>	element-wise (vectorized) exclusive OR <code>xor(x,y)</code>	

Note that the `||` operator evaluates the left condition and if the left condition is TRUE the right side is never evaluated. This can save time if the first is the result of a complex operation. The `&&` operator will likewise return FALSE without evaluation of the second argument when the first element of the first argument is FALSE.

```
> x <- 5
> x > 6 || stop("X is too small")
Error: X is too small
> x > 3 || stop("X is too small")
[1] TRUE
```

To check whether a value is a logical you can use the `is.logical()` function.

Section 17.2: Coercion

To coerce a variable to a logical use the `as.logical()` function.

```
> x <- 2
> z <- x > 4
> z
[1] FALSE
> class(x)
[1] "numeric"
> as.logical(2)
[1] TRUE
```

When applying `as.numeric()` to a logical, a double will be returned. NA is a logical value and a logical operator with an NA will return NA if the outcome is ambiguous.

Section 17.3: Interpretation of NAs

See Missing values for details.

```
> TRUE & NA
[1] NA
> FALSE & NA
```

```
[1] FALSE  
> TRUE || NA  
[1] TRUE  
> FALSE || NA  
[1] NA
```

```
[1] FALSE  
> TRUE || NA  
[1] TRUE  
> FALSE || NA  
[1] NA
```

第18章：数据框

第18.1节：创建空数据框

数据框是一种特殊的列表：它是矩形的。列表的每个元素（列）长度相同，每行都有一个“行名”。每列有自己的类，但一列的类可以与另一列不同（不同于矩阵，矩阵中所有元素必须具有相同的类）。

原则上，数据框可以没有行也没有列：

```
> str(结构(列表(), 类 = "data.frame"))
NULL
<0 行> (或 0-长度 行名)
```

但这很少见。数据框（data.frame）通常有很多列和很多行。这里有一个包含三行两列的数据框（a 是数值型，b 是字符型）：

```
> str(结构(list(a = 1:3, b = letters[1:3]), class = "data.frame"))
[1] a b
<0 行> (或 0-长度 行名)
```

为了让数据框能够打印，我们需要提供一些行名。这里我们使用数字 1:3 作为行名：

```
> str(结构(list(a = 1:3, b = letters[1:3]), class = "data.frame", row.names = 1:3))
  a b
1 1 a
2 2 b
3 3 c
```

现在很明显我们有一个包含3行2列的数据框。你可以使用 nrow()、ncol() 和 dim() 来检查：

```
> x <- str(结构(list(a = numeric(3), b = character(3)), class = "data.frame", row.names = 1:3))
> nrow(x)
[1] 3
> ncol(x)
[1] 2
> dim(x)
[1] 3 2
```

R 提供了另外两个函数（除了 structure() 之外）可以用来创建 data.frame。第一个函数直观地称为 data.frame()。它会检查你提供的列名是否有效，列表元素是否长度相同，并自动生成一些行名。这意味着 data.frame() 的输出可能并不总是完全符合你的预期：

```
> str(data.frame("a a a" = numeric(3), "b-b-b" = character(3)))
'data.frame': 3 观测值, 2 个变量:
$ a.a.a: 数值 0 0 0
$ b.b.b: 因子 w/ 1 水平 "" : 1 1 1
```

另一个函数叫做 as.data.frame()。它可以用来将非 data.frame 对象强制转换为 data.frame，方法是通过 data.frame() 运行该对象。举个例子，考虑一个矩阵：

```
> m <- matrix(letters[1:9], nrow = 3)
> m
```

Chapter 18: Data frames

Section 18.1: Create an empty data.frame

A data.frame is a special kind of list: it is *rectangular*. Each element (column) of the list has same length, and where each row has a "row name". Each column has its own class, but the class of one column can be different from the class of another column (unlike a matrix, where all elements must have the same class).

In principle, a data.frame could have no rows and no columns:

```
> str(结构(list(character()), class = "data.frame"))
NULL
<0 rows> (or 0-length row.names)
```

But this is unusual. It is more common for a data.frame to have many columns and many rows. Here is a data.frame with three rows and two columns (a is numeric class and b is character class):

```
> str(结构(list(a = 1:3, b = letters[1:3]), class = "data.frame"))
[1] a b
<0 rows> (or 0-length row.names)
```

In order for the data.frame to print, we will need to supply some row names. Here we use just the numbers 1:3:

```
> str(结构(list(a = 1:3, b = letters[1:3]), class = "data.frame", row.names = 1:3))
  a b
1 1 a
2 2 b
3 3 c
```

Now it becomes obvious that we have a data.frame with 3 rows and 2 columns. You can check this using nrow(), ncol(), and dim():

```
> str(结构(list(a = numeric(3), b = character(3)), class = "data.frame", row.names = 1:3))
> nrow(x)
[1] 3
> ncol(x)
[1] 2
> dim(x)
[1] 3 2
```

R provides two other functions (besides structure()) that can be used to create a data.frame. The first is called, intuitively, data.frame(). It checks to make sure that the column names you supplied are valid, that the list elements are all the same length, and supplies some automatically generated row names. This means that the output of data.frame() might now always be exactly what you expect:

```
> str(data.frame("a a a" = numeric(3), "b-b-b" = character(3)))
'data.frame': 3 obs. of 2 variables:
$ a.a.a: num 0 0 0
$ b.b.b: Factor w/ 1 level "" : 1 1 1
```

The other function is called as.data.frame(). This can be used to coerce an object that is not a data.frame into being a data.frame by running it through data.frame(). As an example, consider a matrix:

```
> m <- matrix(letters[1:9], nrow = 3)
> m
```

```
[,1] [,2] [,3]
[1,] "a"  "d"  "g"
[2,] "b"  "e"  "h"
[3,] "c"  "f"  "i"
```

结果是：

```
> as.data.frame(m)
V1 V2 V3
1 a d g
2 b e h
3 c f i
> str(as.data.frame(m))
'data.frame': 3 观测值, 3 变量:
$ V1: 因子 w/ 3 水平 "a", "b", "c": 1 2 3
$ V2: 因子 w/ 3 水平 "d", "e", "f": 1 2 3
$ V3: 因子 w/ 3 水平 "g", "h", "i": 1 2 3
```

第18.2节：从数据框中子集选取行和列

访问行和列的语法：[, [] 和 \$

本节介绍访问数据框中特定行和列的最常用语法。这些语法包括

- 类似于矩阵，使用单括号 `data[行, 列]`
 - 使用行号和列号
 - 使用列名（和行名）
- 类似于列表：
 - 使用单括号 `data[列]` 获取数据框
 - 使用双括号 `data[[某列]]` 获取向量
- 使用 \$ 获取单列 `data$列名`

我们将使用内置的 `mtcars` 数据框进行说明。

像一个矩阵：`data[行, 列]`

带有数字索引

使用内置的数据框 `mtcars`，我们可以使用带逗号的[]括号提取行和列。

逗号前的索引是行：

```
# 获取第一行
mtcars[1, ]
# 获取前五行
mtcars[1:5, ]
```

同样，逗号后的索引是列：

```
# 获取第一列
mtcars[, 1]
# 获取第一、第三和第五列：
mtcars[, c(1, 3, 5)]
```

如上所示，如果行或列留空，则全部被选中。`mtcars[1,]` 表示第一行及所有列。

带有列（和行）名称

```
[,1] [,2] [,3]
[1,] "a"  "d"  "g"
[2,] "b"  "e"  "h"
[3,] "c"  "f"  "i"
```

And the result:

```
> as.data.frame(m)
V1 V2 V3
1 a d g
2 b e h
3 c f i
> str(as.data.frame(m))
'data.frame': 3 obs. of 3 variables:
$ V1: Factor w/ 3 levels "a", "b", "c": 1 2 3
$ V2: Factor w/ 3 levels "d", "e", "f": 1 2 3
$ V3: Factor w/ 3 levels "g", "h", "i": 1 2 3
```

Section 18.2: Subsetting rows and columns from a data frame

Syntax for accessing rows and columns: [, [], and \$

This topic covers the most common syntax to access specific rows and columns of a data frame. These are

- Like a `matrix` with single brackets `data[rows, columns]`
 - Using row and column numbers
 - Using column (and row) names
- Like a `list`:
 - With single brackets `data[columns]` to get a data frame
 - With double brackets `data[[one_column]]` to get a vector
- With \$ for a single column `data$column_name`

We will use the built-in `mtcars` data frame to illustrate.

Like a matrix: `data[rows, columns]`

With numeric indexes

Using the built in data frame `mtcars`, we can extract rows and columns using [] brackets with a comma included. Indices before the comma are rows:

```
# get the first row
mtcars[1, ]
# get the first five rows
mtcars[1:5, ]
```

Similarly, after the comma are columns:

```
# get the first column
mtcars[, 1]
# get the first, third and fifth columns:
mtcars[, c(1, 3, 5)]
```

As shown above, if either rows or columns are left blank, all will be selected. `mtcars[1,]` indicates the first row with *all* the columns.

With column (and row) names

到目前为止，这与访问矩阵的行和列的方式相同。对于data.frame，大多数情况下更倾向于使用列名而非列索引。这是通过使用带有列名的character而不是带有列号的numeric来实现的：

```
# 获取 mpg 列  
mtcars[, "mpg"]  
# 获取 mpg、cyl 和 disp 列  
mtcars[, c("mpg", "cyl", "disp")]
```

虽然不太常见，行名也可以使用：

```
mtcars["Mazda Rx4", ]
```

行和列一起

行和列参数可以一起使用：

```
# mpg 列的前四行  
mtcars[1:4, "mpg"]  
  
# mpg、cyl 和 disp 列的第 2 行和第 5 行  
mtcars[c(2, 5), c("mpg", "cyl", "disp")]
```

关于维度的警告：

使用这些方法时，如果提取多个列，将返回一个数据框。然而，如果提取一列，默认选项下将返回一个向量，而不是数据框。

```
## 多列返回数据框  
class(mtcars[, c("mpg", "cyl")])  
# [1] "data.frame"  
## 单列返回向量  
class(mtcars[, "mpg"])  
# [1] "numeric"
```

有两种解决方法。一种是将数据框视为列表（见下文），另一种是添加一个 drop = FALSE 参数。这告诉 R 不要“丢弃未使用的维度”：

```
class(mtcars[, "mpg", drop = FALSE])  
# [1] "data.frame"
```

注意矩阵的工作方式相同——默认情况下，单列或单行会变成向量，但如果指定 drop = FALSE，则可以保持为一列或一行的矩阵。

像列表一样

数据框本质上是 **list**，即它们是列向量的列表（所有列向量必须长度相同）。列表可以使用单括号 [进行子列表的子集操作，或双括号 [[访问单个元素。

使用单括号 data[columns]

当你使用单括号且不带逗号时，你将得到列，因为数据框是列的列表。

```
mtcars["mpg"]  
mtcars[c("mpg", "cyl", "disp")]  
my_columns <- c("mpg", "cyl", "hp")  
mtcars[my_columns]
```

So far, this is identical to how rows and columns of matrices are accessed. With **data.frames**, most of the time it is preferable to use a column name to a column index. This is done by using a **character** with the column name instead of **numeric** with a column number:

```
# get the mpg column  
mtcars[, "mpg"]  
# get the mpg, cyl, and disp columns  
mtcars[, c("mpg", "cyl", "disp")]
```

Though less common, row names can also be used:

```
mtcars["Mazda Rx4", ]
```

Rows and columns together

The row and column arguments can be used together:

```
# first four rows of the mpg column  
mtcars[1:4, "mpg"]  
  
# 2nd and 5th row of the mpg, cyl, and disp columns  
mtcars[c(2, 5), c("mpg", "cyl", "disp")]
```

A warning about dimensions:

When using these methods, if you extract multiple columns, you will get a data frame back. However, if you extract a *single* column, you will get a vector, not a data frame under the default options.

```
## multiple columns returns a data frame  
class(mtcars[, c("mpg", "cyl")])  
# [1] "data.frame"  
## single column returns a vector  
class(mtcars[, "mpg"])  
# [1] "numeric"
```

There are two ways around this. One is to treat the data frame as a list (see below), the other is to add a **drop = FALSE** argument. This tells R to not "drop the unused dimensions":

```
class(mtcars[, "mpg", drop = FALSE])  
# [1] "data.frame"
```

Note that matrices work the same way - by default a single column or row will be a vector, but if you specify **drop = FALSE** you can keep it as a one-column or one-row matrix.

Like a list

Data frames are essentially **lists**, i.e., they are a list of column vectors (that all must have the same length). Lists can be subset using single brackets [for a sub-list, or double brackets [[for a single element.

With single brackets data[columns]

When you use single brackets and no commas, you will get column back because data frames are lists of columns.

```
mtcars["mpg"]  
mtcars[c("mpg", "cyl", "disp")]  
my_columns <- c("mpg", "cyl", "hp")  
mtcars[my_columns]
```

单括号像列表与单括号像矩阵

`data[columns]`和`data[, columns]`之间的区别在于，当将`data.frame`视为list（括号内无逗号）时，返回的对象将是一个`data.frame`。如果使用逗号将`data.frame`视为matrix，则选择单列会返回一个向量，但选择多列会返回一个`data.frame`。

```
## 选择单列时  
## 像列表会返回一个数据框  
class(mtcars["mpg"])  
# [1] "data.frame"  
## 像矩阵会返回一个向量  
class(mtcars[, "mpg"])  
# [1] "numeric"
```

使用双括号`data[[one_column]]`

当将`data.frame`视为list时，若要提取单列作为向量，可以使用双括号`[[`。

这只适用于一次处理单列。

```
# 按名称提取单列作为向量  
mtcars[["mpg"]]  
  
# 按名称提取单列作为数据框（同上）  
mtcars["mpg"]
```

使用 \$ 访问列

可以使用神奇的快捷方式`$`提取单列，无需使用带引号的列名：

```
# 获取列 "mpg"  
mtcars$mpg
```

通过`$`访问的列始终是向量，而不是数据框。

使用 \$ 访问列的缺点

`$`可以是一个方便的快捷方式，尤其是在像 RStudio 这样的环境中，它会自动补全列名。然而，`$`也有缺点：它使用了非标准求值来避免使用引号，这意味着如果你的列名存储在变量中，它将无法工作。

```
my_column <- "mpg"  
# 下面的代码将无法运行  
mtcars$my_column  
# 但以下代码可以运行  
mtcars[, my_column] # 向量  
mtcars[my_column] # 单列数据框  
mtcars[[my_column]] # 向量
```

鉴于这些问题，`$`最好在交互式 R 会话中使用，当你的列名是固定不变时。对于编程使用，例如编写一个可泛化的函数，用于不同数据集且列名不同的情况，`$`应该避免使用。

还要注意，默认行为是仅在通过`$`从递归对象（环境除外）中提取时使用部分匹配

```
# 给你 "mpg" 列的值  
# 因为 "mtcars" 中只有一列的名称以 "m" 开头
```

Single brackets like a list vs. single brackets like a matrix

The difference between `data[columns]` and `data[, columns]` is that when treating the `data.frame` as a `list` (no comma in the brackets) the object returned will be a `data.frame`. If you use a comma to treat the `data.frame` like a `matrix` then selecting a single column will return a vector but selecting multiple columns will return a `data.frame`.

```
## When selecting a single column  
## like a list will return a data frame  
class(mtcars["mpg"])  
# [1] "data.frame"  
## like a matrix will return a vector  
class(mtcars[, "mpg"])  
# [1] "numeric"
```

With double brackets `data[[one_column]]`

To extract a single column as a vector when treating your `data.frame` as a `list`, you can use double brackets `[[`.

This will only work for a single column at a time.

```
# extract a single column by name as a vector  
mtcars[["mpg"]]  
  
# extract a single column by name as a data frame (as above)  
mtcars["mpg"]
```

Using \$ to access columns

A single column can be extracted using the magical shortcut`$` without using a quoted column name:

```
# get the column "mpg"  
mtcars$mpg
```

Columns accessed by`$`will always be vectors, not data frames.

Drawbacks of \$ for accessing columns

The`$`can be a convenient shortcut, especially if you are working in an environment (such as RStudio) that will autocomplete the column name in this case. **However**,`$`has drawbacks as well: it uses *non-standard evaluation* to avoid the need for quotes, which means it *will not work* if your column name is stored in a variable.

```
my_column <- "mpg"  
# the below will not work  
mtcars$my_column  
# but these will work  
mtcars[, my_column] # vector  
mtcars[my_column] # one-column data frame  
mtcars[[my_column]] # vector
```

Due to these concerns,`$`is best used in *interactive* R sessions when your column names are constant. For *programmatic* use, for example in writing a generalizable function that will be used on different data sets with different column names,`$`should be avoided.

Also note that the default behaviour is to use partial matching only when extracting from recursive objects (except environments) by`$`

```
# give you the values of "mpg" column  
# as "mtcars" has only one column having name starting with "m"
```

```
mtcars$m  
# 将返回 "NULL"  
# 因为 "mtcars" 中有多列名称以 "d" 开头  
mtcars$d
```

高级索引：负数和逻辑索引

每当我们可以使用数字作为索引时，也可以使用负数来省略某些索引，或者使用布尔（逻辑）向量来精确指示要保留的项目。

负数索引省略元素

```
mtcars[1, ] # 第一行  
mtcars[-1, ] # 除第一行外的所有行  
mtcars[-(1:10), ] # 除前10行外的所有行
```

逻辑向量指示要保留的特定元素

我们可以使用诸如`<`的条件生成逻辑向量，并仅提取满足条件的行：

```
# 逻辑向量，当行的mpg小于15时为TRUE  
# 当行的mpg大于等于15时为FALSE  
test <- mtcars$mpg < 15  
  
# 从数据框中提取这些行  
mtcars[test, ]
```

我们也可以跳过保存中间变量的步骤

```
# 提取 cyl 值为 4 的所有行的所有列。  
mtcars[mtcars$cyl == 4, ]  
# 提取 cyl 值为 4 的 cyl、mpg 和 hp 列  
mtcars[mtcars$cyl == 4, c("cyl", "mpg", "hp")]
```

第18.3节：操作数据框的便捷函数

一些操作data.frames的便捷函数有subset()、transform()、with()和within()。

subset

`subset()`函数允许你以更方便的方式对data.frame进行子集操作（subset也适用于其他类）：

```
subset(mtcars, subset = cyl == 6, select = c("mpg", "hp"))  
          mpg  hp  
Mazda RX4   21.0 110  
Mazda RX4 Wag 21.0 110  
Hornet 4 Drive 21.4 110  
Valiant    18.1 105  
Merc 280    19.2 123  
Merc 280C   17.8 123  
Ferrari Dino 19.7 175
```

在上面的代码中，我们只请求了`cyl == 6`的行，以及`mpg`和`hp`这两列。你也可以使用`[]`来实现相同的结果，代码如下：

```
mtcars[mtcars$cyl == 6, c("mpg", "hp")]
```

```
mtcars$m  
# will give you "NULL"  
# as "mtcars" has more than one columns having name starting with "d"  
mtcars$d
```

Advanced indexing: negative and logical indices

Whenever we have the option to use numbers for a index, we can also use negative numbers to omit certain indices or a boolean (logical) vector to indicate exactly which items to keep.

Negative indices omit elements

```
mtcars[1, ] # first row  
mtcars[-1, ] # everything but the first row  
mtcars[-(1:10), ] # everything except the first 10 rows
```

Logical vectors indicate specific elements to keep

We can use a condition such as`<`to generate a logical vector, and extract only the rows that meet the condition:

```
# logical vector indicating TRUE when a row has mpg less than 15  
# FALSE when a row has mpg >= 15  
test <- mtcars$mpg < 15  
  
# extract these rows from the data frame  
mtcars[test, ]
```

We can also bypass the step of saving the intermediate variable

```
# extract all columns for rows where the value of cyl is 4.  
mtcars[mtcars$cyl == 4, ]  
# extract the cyl, mpg, and hp columns where the value of cyl is 4  
mtcars[mtcars$cyl == 4, c("cyl", "mpg", "hp")]
```

Section 18.3: Convenience functions to manipulate data.frames

Some convenience functions to manipulate data.frames are `subset()`, `transform()`, `with()` and `within()`.

subset

The `subset()` function allows you to subset a data.frame in a more convenient way (subset also works with other classes):

```
subset(mtcars, subset = cyl == 6, select = c("mpg", "hp"))  
          mpg  hp  
Mazda RX4   21.0 110  
Mazda RX4 Wag 21.0 110  
Hornet 4 Drive 21.4 110  
Valiant    18.1 105  
Merc 280    19.2 123  
Merc 280C   17.8 123  
Ferrari Dino 19.7 175
```

In the code above we asking only for the lines in which`cyl == 6`and for the columns`mpg`and`hp`. You could achieve the same result using`[]`with the following code:

```
mtcars[mtcars$cyl == 6, c("mpg", "hp")]
```

transform

transform()函数是一个方便的函数，用于更改data.frame中的列。例如，以下代码向mtcars数据框中添加了另一个名为mpg2的列，其值为mpg的平方：

```
mtcars <- transform(mtcars, mpg2 = mpg^2)
```

with 和 within

with()和within()都允许你在data.frame环境中计算表达式，使语法更简洁，减少了使用\$或[]的次数。

例如，如果你想在airquality数据框中创建、更改和/或删除多个列：

```
aq <- within(airquality, {  
  Ozone <- log(Ozone) # 创建新列  
  Month <- factor(month.abb[Month]) # 更改Month列  
  cTemp <- round(Temp - 32) * 5/9, 1) # 创建新列  
  S.cT <- Solar.R / cTemp # 创建新列  
  rm(Day, Temp) # 删除列  
})
```

第18.4节：介绍

数据框很可能是你在分析中最常用的数据结构。数据框是一种特殊的列表，存储长度相同但类型不同的向量。你可以使用data.frame函数来创建数据框。下面的示例通过将数值向量和字符向量组合成数据框来展示这一点。它使用了:运算符，该运算符会创建一个包含从1到3所有整数的向量。

```
df1 <- data.frame(x = 1:3, y = c("a", "b", "c"))  
df1  
##   x y  
## 1 1 a  
## 2 2 b  
## 3 3 c  
class(df1)  
## [1] "data.frame"
```

数据框对象打印时不会带引号，因此列的类型并不总是显而易见的。

```
df2 <- data.frame(x = c("1", "2", "3"), y = c("a", "b", "c"))  
df2  
##   x y  
## 1 1 a  
## 2 2 b  
## 3 3 c
```

如果不进一步检查，无法区分 df1 和 df2 中的"x"列。可以使用str函数来更详细地描述对象，而不仅仅是查看其类。

```
str(df1)  
## 'data.frame': 3 个观测值, 2 个变量:  
## $ x: 整数 1 2 3  
## $ y: 因子, 3个水平 "a","b","c": 1 2 3  
str(df2)  
## 'data.frame': 3 个观测值, 2 个变量:
```

transform

The **transform()** function is a convenience function to change columns inside a **data.frame**. For instance the following code adds another column named mpg2 with the result of mpg² to the **mtcars data.frame**:

```
mtcars <- transform(mtcars, mpg2 = mpg^2)
```

with and within

Both **with()** and **within()** let you to evaluate expressions inside the **data.frame** environment, allowing a somewhat cleaner syntax, saving you the use of some \$ or [].

For example, if you want to create, change and/or remove multiple columns in the **airquality data.frame**:

```
aq <- within(airquality, {  
  logOzone <- log(Ozone) # creates new column  
  Month <- factor(month.abb[Month]) # changes Month Column  
  cTemp <- round((Temp - 32) * 5/9, 1) # creates new column  
  S.cT <- Solar.R / cTemp # creates new column  
  rm(Day, Temp) # removes columns  
})
```

Section 18.4: Introduction

Data frames are likely the data structure you will used most in your analyses. A data frame is a special kind of list that stores same-length vectors of different classes. You create data frames using the **data.frame** function. The example below shows this by combining a numeric and a character vector into a data frame. It uses the : operator, which will create a vector containing all integers from 1 to 3.

```
df1 <- data.frame(x = 1:3, y = c("a", "b", "c"))  
df1  
##   x y  
## 1 1 a  
## 2 2 b  
## 3 3 c  
class(df1)  
## [1] "data.frame"
```

Data frame objects do not print with quotation marks, so the class of the columns is not always obvious.

```
df2 <- data.frame(x = c("1", "2", "3"), y = c("a", "b", "c"))  
df2  
##   x y  
## 1 1 a  
## 2 2 b  
## 3 3 c
```

Without further investigation, the "x" columns in df1 and df2 cannot be differentiated. The **str** function can be used to describe objects with more detail than class.

```
str(df1)  
## 'data.frame': 3 obs. of 2 variables:  
## $ x: int 1 2 3  
## $ y: Factor w/ 3 levels "a","b","c": 1 2 3  
str(df2)  
## 'data.frame': 3 obs. of 2 variables:
```

```
## $ x: 因子, 3个水平 "1","2","3": 1 2 3  
## $ y: 因子, 3个水平 "a","b","c": 1 2 3
```

这里你可以看到 df1 是一个 **data.frame** 包含2个变量“x”和“y”的3个观测值。然后告诉你“x”的数据类型是整数（对本课程不重要，但对我们来说它表现得像数值型），“y”是一个有三个水平的因子（另一种我们不讨论的数据类型）。**需要注意的是，默认情况下，数据框会将字符串强制转换为因子。**默认行为可以通过 stringsAsFactors 参数更改：

```
df3 <- data.frame(x = 1:3, y = c("a", "b", "c"), stringsAsFactors = FALSE)  
str(df3)  
## 'data.frame': 3 个观测值, 2 个变量:  
## $ x: 整数 1 2 3  
## $ y: 字符 "a" "b" "c"
```

现在“y”列是字符型。如上所述，数据框的每一“列”必须长度相同。尝试用不同长度的向量创建数据框会导致错误。（试着运行 **data.frame**(x = 1:3, y = 1:4)，看看产生的错误。）

作为数据框的测试用例，R默认提供了一些数据。其中之一是iris，加载方式如下：

```
mydataframe <- 鸢尾花数据集  
str(mydataframe)
```

第18.5节：将**data.frame**的所有列转换为字符类

一个常见的任务是将**data.frame**的所有列转换为字符类，以便于操作，例如在将**data.frame**发送到关系数据库管理系统（RDBMS）或合并包含因子且输入**data.frame**之间级别可能不同的**data.frame**时。

执行此操作的最佳时机是在读取数据时——几乎所有创建**data.frame**的输入方法都有一个选项stringsAsFactors，可以设置为FALSE。

如果数据已经创建，可以按如下所示将因子列转换为字符列。

```
bob <- data.frame(jobs = c("scientist", "analyst"),  
                   pay = c(160000, 100000), age = c(30, 25))  
str(bob)
```

```
'data.frame': 2 观测值, 3 个变量:  
$ jobs: 因子 w/ 2 级别 "analyst", "scientist": 2 1  
$ pay : 数值 160000 100000  
$ age : 数值 30 25
```

```
# 将所有列*转换为字符  
bob[] <- lapply(bob, as.character)  
str(bob)
```

```
'data.frame': 2 观测值, 3 个变量:  
$ jobs: 字符 "科学家" "分析师"  
$ pay : 字符 "160000" "1e+05"  
$ age : 字符 "30" "25"
```

```
# 仅将因子列转换为字符型  
bob[] <- lapply(bob, function(x) {
```

```
## $ x: Factor w/ 3 levels "1","2","3": 1 2 3  
## $ y: Factor w/ 3 levels "a","b","c": 1 2 3
```

Here you see that df1 is a **data.frame** and has 3 observations of 2 variables, "x" and "y." Then you are told that "x" has the data type integer (not important for this class, but for our purposes it behaves like a numeric) and "y" is a factor with three levels (another data class we are not discussing). **It is important to note that, by default, data frames coerce characters to factors.** The default behavior can be changed with the stringsAsFactors parameter:

```
df3 <- data.frame(x = 1:3, y = c("a", "b", "c"), stringsAsFactors = FALSE)  
str(df3)  
## 'data.frame': 3 obs. of 2 variables:  
## $ x: int 1 2 3  
## $ y: chr "a" "b" "c"
```

Now the "y" column is a character. As mentioned above, each "column" of a data frame must have the same length. Trying to create a data.frame from vectors with different lengths will result in an error. (Try running **data.frame**(x = 1:3, y = 1:4) to see the resulting error.)

As test-cases for data frames, some data is provided by R by default. One of them is iris, loaded as follows:

```
mydataframe <- iris  
str(mydataframe)
```

Section 18.5: Convert all columns of a **data.frame** to character class

A common task is to convert all columns of a **data.frame** to character class for ease of manipulation, such as in the cases of sending **data.frames** to a RDBMS or merging **data.frames** containing factors where levels may differ between input **data.frames**.

The best time to do this is when the data is read in - almost all input methods that create **data.frames** have an option **stringsAsFactors** which can be set to FALSE.

If the data has already been created, factor columns can be converted to character columns as shown below.

```
bob <- data.frame(jobs = c("scientist", "analyst"),  
                   pay = c(160000, 100000), age = c(30, 25))  
str(bob)
```

```
'data.frame': 2 obs. of 3 variables:  
$ jobs: Factor w/ 2 levels "analyst", "scientist": 2 1  
$ pay : num 160000 100000  
$ age : num 30 25
```

```
# Convert *all columns* to character  
bob[] <- lapply(bob, as.character)  
str(bob)
```

```
'data.frame': 2 obs. of 3 variables:  
$ jobs: chr "scientist" "analyst"  
$ pay : chr "160000" "1e+05"  
$ age : chr "30" "25"
```

```
# Convert only factor columns to character  
bob[] <- lapply(bob, function(x) {
```

```
如果是因子(x) x <- 转换为字符型(x)
返回(x)
})
```

```
if is.factor(x) x <- as.character(x)
return(x)
})
```

第19章：split函数

第19.1节：在split-apply-combine范式中使用split

一种流行的数据分析形式是split-apply-combine，即将数据分组，对每个组应用某种处理，然后合并结果。

我们考虑一个数据分析，想要获得内置mtcars数据集中每个气缸数(cyl)对应的两辆最佳每加仑英里数(mpg)的汽车。首先，我们按气缸数将mtcars数据框拆分：

```
(spl <- split(mtcars, mtcars$cyl))
# $4`  
#          mpg cyl disp hp drat wt qsec vs am gear carb  
# Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1  
# Merc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2  
# Merc 230 22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2  
# Fiat 128 32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1  
# ...  
#  
# $`6`  
#          mpg 气缸数 排量 马力 传动比 重量 1/4英里加速时间 引擎形状 变速箱 齿轮数 化油器数  
# 马自达 RX4 21.0 6 160.0 110 3.90 2.620 16.46 0 1 4 4  
# 马自达 RX4 Wag 21.0 6 160.0 110 3.90 2.875 17.02 0 1 4 4  
# 大黄蜂 4 驱动 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1  
# 勇士 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1  
# ...  
#  
# $`8`  
#          mpg 气缸数 排量 马力 传动比 重量 1/4英里加速时间 引擎形状 变速箱 齿轮数 化油器数  
# 大黄蜂运动版 18.7 8 360.0 175 3.15 3.440 17.02 0 0 3 2  
# 尘埃360 14.3 8 360.0 245 3.21 3.570 15.84 0 0 3 4  
# 梅赛德斯 450SE 16.4 8 275.8 180 3.07 4.070 17.40 0 0 3 3  
# 梅赛德斯 450SL 17.3 8 275.8 180 3.07 3.730 17.60 0 0 3 3  
# ...
```

这返回了一个数据框列表，每个气缸数对应一个数据框。如输出所示，我们可以通过 `spl$`4``、`spl$`6`` 和 `spl$`8`` 获取相关的数据框（有些人可能觉得使用 `spl["4"]` 或 `spl[["4"]]` 更加直观）。

现在，我们可以使用 `lapply` 遍历这个列表，应用我们的函数，从每个列表元素中提取油耗(mpg)最好的两辆车：

```
(best2 <- lapply(spl, function(x) tail(x[order(x$mpg), ], 2)))
# $4`  
#          mpg 气缸数 排量 马力 传动比 重量 1/4英里加速时间 引擎形状 变速箱 齿轮数 化油器数  
# 菲亚特 128 32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1  
# 丰田卡罗拉 33.9 4 71.1 65 4.22 1.835 19.90 1 1 4 1  
#  
# $`6`  
#          mpg 气缸数 排量 马力 传动比 重量 1/4英里加速时间 引擎形状 变速箱 齿轮数 化油器数  
# 马自达 RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4  
# 大黄蜂 4 驱动 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1  
#  
# $`8`  
#          mpg 气缸数 排量 马力 传动比 重量 1/4英里加速时间 引擎形状 变速箱 齿轮数 化油器数  
# 大黄蜂运动版 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2  
# 庞蒂亚克火鸟 19.2 8 400 175 3.08 3.845 17.05 0 0 3 2
```

Chapter 19: Split function

Section 19.1: Using split in the split-apply-combine paradigm

A popular form of data analysis is [split-apply-combine](#), in which you split your data into groups, apply some sort of processing on each group, and then combine the results.

Let's consider a data analysis where we want to obtain the two cars with the best miles per gallon (mpg) for each cylinder count (cyl) in the built-in mtcars dataset. First, we split the `mtcars` data frame by the cylinder count:

```
(spl <- split(mtcars, mtcars$cyl))
# $`4`  
#          mpg cyl disp hp drat wt qsec vs am gear carb  
# Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1  
# Merc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2  
# Merc 230 22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2  
# Fiat 128 32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1  
# ...  
#  
# $`6`  
#          mpg cyl disp hp drat wt qsec vs am gear carb  
# Mazda RX4 21.0 6 160.0 110 3.90 2.620 16.46 0 1 4 4  
# Mazda RX4 Wag 21.0 6 160.0 110 3.90 2.875 17.02 0 1 4 4  
# Hornet 4 Drive 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1  
# Valiant 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1  
# ...  
#  
# $`8`  
#          mpg cyl disp hp drat wt qsec vs am gear carb  
# Hornet Sportabout 18.7 8 360.0 175 3.15 3.440 17.02 0 0 3 2  
# Duster 360 14.3 8 360.0 245 3.21 3.570 15.84 0 0 3 4  
# Merc 450SE 16.4 8 275.8 180 3.07 4.070 17.40 0 0 3 3  
# Merc 450SL 17.3 8 275.8 180 3.07 3.730 17.60 0 0 3 3  
# ...
```

This has returned a list of data frames, one for each cylinder count. As indicated by the output, we could obtain the relevant data frames with `spl$`4``, `spl$`6``, and `spl$`8`` (some might find it more visually appealing to use `spl["4"]` or `spl[["4"]]` instead).

Now, we can use `lapply` to loop through this list, applying our function that extracts the cars with the best 2 mpg values from each of the list elements:

```
(best2 <- lapply(spl, function(x) tail(x[order(x$mpg), ], 2)))
# $`4`  
#          mpg cyl disp hp drat wt qsec vs am gear carb  
# Fiat 128 32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1  
# Toyota Corolla 33.9 4 71.1 65 4.22 1.835 19.90 1 1 4 1  
#  
# $`6`  
#          mpg cyl disp hp drat wt qsec vs am gear carb  
# Mazda RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4  
# Hornet 4 Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1  
#  
# $`8`  
#          mpg cyl disp hp drat wt qsec vs am gear carb  
# Hornet Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2  
# Pontiac Firebird 19.2 8 400 175 3.08 3.845 17.05 0 0 3 2
```

最后，我们可以使用 `rbind` 将所有内容合并在一起。我们想调用 `rbind(best2[["4"]], best2[["6"]], best2[["8"]])`，但如果列表很大，这将非常繁琐。因此，我们使用：

```
do.call(rbind, best2)
#          mpg cyl disp hp drat wt qsec vs am gear carb
# 4.Fiat 128    32.4   4 78.7 66 4.08 2.200 19.47 1 1 4 1
# 4.Toyota Corolla 33.9   4 71.1 65 4.22 1.835 19.90 1 1 4 1
# 6.Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02 0 1 4 4
# 6.Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44 1 0 3 1
# 8.Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02 0 0 3 2
# 8.Pontiac Firebird 19.2   8 400.0 175 3.08 3.845 17.05 0 0 3 2
```

这将返回 `rbind`（参数1，一个函数）与 `best2`（参数2，一个列表）的所有元素作为参数传递的结果。

对于像这样的简单分析，将整个拆分-应用-合并过程写成一行代码可能更简洁（但可能可读性大大降低！）：

```
do.call(rbind, lapply(split(mtcars, mtcars$cyl), function(x) tail(x[order(x$mpg)], 2)))
```

同样值得注意的是，`lapply(split(x,f), FUN)` 组合也可以用 `?by` 函数来替代：

```
by(mtcars, mtcars$cyl, function(x) tail(x[order(x$mpg)], 2))
do.call(rbind, by(mtcars, mtcars$cyl, function(x) tail(x[order(x$mpg)], 2)))
```

第19.2节：split的基本用法

`split` 允许根据因子/分组变量将向量或数据框划分为多个桶。这个划分成桶的过程以列表的形式呈现，随后可以用来进行分组计算（`for` 循环或 `lapply/sapply`）。

第一个例子展示了 `split` 在向量上的用法：

考虑以下字母向量：

```
testdata <- c("e", "o", "r", "g", "a", "y", "w", "q", "i", "s", "b", "v", "x", "h", "u")
```

目标是将这些字母分为元音和辅音，即根据字母类型进行拆分。

首先创建一个分组向量：

```
vowels <- c('a', 'e', 'i', 'o', 'u', 'y')
letter_type <- ifelse(testdata %in% vowels, "vowels", "consonants")
```

注意 `letter_type` 的长度与向量 `testdata` 相同。现在我们可以将测试数据拆分为两个组，`vowels` 和 `consonants`：

```
split(testdata, letter_type)
#$consonants
#[1] "r" "g" "w" "q" "s" "b" "v" "x" "h"
#$vowels
#[1] "e" "o" "a" "y" "i" "u"
```

Finally, we can combine everything together using `rbind`. We want to call `rbind(best2[["4"]], best2[["6"]], best2[["8"]])`，但这样会非常繁琐。因此，我们使用：

```
do.call(rbind, best2)
#          mpg cyl disp hp drat wt qsec vs am gear carb
# 4.Fiat 128    32.4   4 78.7 66 4.08 2.200 19.47 1 1 4 1
# 4.Toyota Corolla 33.9   4 71.1 65 4.22 1.835 19.90 1 1 4 1
# 6.Mazda RX4 Wag 21.0   6 160.0 110 3.90 2.875 17.02 0 1 4 4
# 6.Hornet 4 Drive 21.4   6 258.0 110 3.08 3.215 19.44 1 0 3 1
# 8.Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02 0 0 3 2
# 8.Pontiac Firebird 19.2   8 400.0 175 3.08 3.845 17.05 0 0 3 2
```

This returns the result of `rbind` (argument 1, a function) with all the elements of `best2` (argument 2, a list) passed as arguments.

With simple analyses like this one, it can be more compact (and possibly much less readable!) to do the whole split-apply-combine in a single line of code:

```
do.call(rbind, lapply(split(mtcars, mtcars$cyl), function(x) tail(x[order(x$mpg)], 2)))
```

It is also worth noting that the `lapply(split(x,f), FUN)` combination can be alternatively framed using the `?by` function:

```
by(mtcars, mtcars$cyl, function(x) tail(x[order(x$mpg)], 2))
do.call(rbind, by(mtcars, mtcars$cyl, function(x) tail(x[order(x$mpg)], 2)))
```

Section 19.2: Basic usage of split

`split` 允许将向量或数据框划分为多个桶。这个划分成桶的过程以列表的形式呈现，随后可以用来进行分组计算（`for` 循环或 `lapply/sapply`）。

First example shows the usage of `split` on a vector:

Consider following vector of letters:

```
testdata <- c("e", "o", "r", "g", "a", "y", "w", "q", "i", "s", "b", "v", "x", "h", "u")
```

Objective is to separate those letters into vowels and consonants, ie split it accordingly to letter type.

Let's first create a grouping vector:

```
vowels <- c('a', 'e', 'i', 'o', 'u', 'y')
letter_type <- ifelse(testdata %in% vowels, "vowels", "consonants")
```

Note that `letter_type` has the same length that our vector `testdata`. Now we can `split` this test data in the two groups, `vowels` and `consonants`:

```
split(testdata, letter_type)
#$consonants
#[1] "r" "g" "w" "q" "s" "b" "v" "x" "h"
#$vowels
#[1] "e" "o" "a" "y" "i" "u"
```

因此，结果是一个列表，其名称来自我们的分组向量/因子letter_type。

split 也有处理数据框 (data.frames) 的方法。

例如考虑iris数据：

`data(iris)`

通过使用 split，可以创建一个列表，其中包含每个鸢尾花物种 (变量：Species) 对应的数据框：

```
> liris <- split(iris, iris$Species)
> names(liris)
[1] "setosa"    "versicolor" "virginica"
> head(liris$setosa)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4          0.2   setosa
2          4.9         3.0          1.4          0.2   setosa
3          4.7         3.2          1.3          0.2   setosa
4          4.6         3.1          1.5          0.2 山鸢尾
5          5.0         3.6          1.4          0.2 山鸢尾
6          5.4         3.9          1.7          0.4 山鸢尾
```

(仅包含山鸢尾组的数据)。

一个示例操作是计算每个鸢尾花种类的相关矩阵；然后可以使用lapply：

```
> (lcor <- lapply(liris, FUN=function(df) cor(df[,1:4])))
```

\$setosa
萼片长度 萼片宽度 花瓣长度 花瓣宽度
萼片长度 1.0000000 0.7425467 0.2671758 0.2780984
萼片宽度 0.7425467 1.0000000 0.1777000 0.2327520
花瓣长度 0.2671758 0.1777000 1.0000000 0.3316300
花瓣宽度 0.2780984 0.2327520 0.3316300 1.0000000

\$versicolor
萼片长度 萼片宽度 花瓣长度 花瓣宽度
萼片长度 1.0000000 0.5259107 0.7540490 0.5464611
萼片宽度 0.5259107 1.0000000 0.5605221 0.6639987
花瓣长度 0.7540490 0.5605221 1.0000000 0.7866681
花瓣宽度 0.5464611 0.6639987 0.7866681 1.0000000

\$virginica
萼片长度 萼片宽度 花瓣长度 花瓣宽度
萼片长度 1.0000000 0.4572278 0.8642247 0.2811077
萼片宽度 0.4572278 1.0000000 0.4010446 0.5377280
花瓣长度 0.8642247 0.4010446 1.0000000 0.3221082
花瓣宽度 0.2811077 0.5377280 0.3221082 1.0000000

然后我们可以为每个组检索相关性最强的变量对：（相关矩阵被重塑/熔化，对角线被过滤，随后选择最佳记录）

```
> 库(reshape)
> (topcor <- lapply(lcor, FUN=function(cormat){
  correlations <- melt(cormat, variable_name="correlatio");
  filtered <- correlations[correlations$X1 != correlations$X2,];
  filtered[which.max(filtered$correlation),]
}))
```

Hence, the result is a list which names are coming from our grouping vector/factor letter_type.

split 也有一个方法来处理 data.frames。

考虑 iris 数据：

`data(iris)`

通过使用 split，可以创建一个列表，其中包含一个 data.frame 每个 iris 种类 (variable: Species)：

```
> liris <- split(iris, iris$Species)
> names(liris)
[1] "setosa"    "versicolor" "virginica"
> head(liris$setosa)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4          0.2   setosa
2          4.9         3.0          1.4          0.2   setosa
3          4.7         3.2          1.3          0.2   setosa
4          4.6         3.1          1.5          0.2   setosa
5          5.0         3.6          1.4          0.2   setosa
6          5.4         3.9          1.7          0.4   setosa
```

(只包含 setosa 组的数据)。

一个示例操作是计算每个鸢尾花种类的相关矩阵；然后可以使用 lapply：

```
> (lcor <- lapply(liris, FUN=function(df) cor(df[,1:4])))
```

\$setosa
Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length 1.0000000 0.7425467 0.2671758 0.2780984
Sepal.Width 0.7425467 1.0000000 0.1777000 0.2327520
Petal.Length 0.2671758 0.1777000 1.0000000 0.3316300
Petal.Width 0.2780984 0.2327520 0.3316300 1.0000000

\$versicolor
Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length 1.0000000 0.5259107 0.7540490 0.5464611
Sepal.Width 0.5259107 1.0000000 0.5605221 0.6639987
Petal.Length 0.7540490 0.5605221 1.0000000 0.7866681
Petal.Width 0.5464611 0.6639987 0.7866681 1.0000000

\$virginica
Sepal.Length Sepal.Width Petal.Length Petal.Width
Sepal.Length 1.0000000 0.4572278 0.8642247 0.2811077
Sepal.Width 0.4572278 1.0000000 0.4010446 0.5377280
Petal.Length 0.8642247 0.4010446 1.0000000 0.3221082
Petal.Width 0.2811077 0.5377280 0.3221082 1.0000000

然后我们可以为每个组检索相关性最强的变量对：（相关矩阵被重塑/熔化，对角线被过滤，随后选择最佳记录）

```
> library(reshape)
> (topcor <- lapply(lcor, FUN=function(cormat){
  correlations <- melt(cormat, variable_name="correlatio");
  filtered <- correlations[correlations$X1 != correlations$X2,];
  filtered[which.max(filtered$correlation),]
}))
```

```
$setosa
X1      X2      correlation
2 Sepal.Width Sepal.Length    0.7425467

$versicolor
X1      X2      correlation
12 Petal.Width Petal.Length   0.7866681

$virginica
X1      X2      correlation
3 Petal.Length Sepal.Length   0.8642247
```

注意，虽然计算是在这样的分组层面上进行的，但可能有人希望将结果合并，这可以通过以下方式完成：

```
> (result <- do.call("rbind", topcor))

      X1      X2      correlation
setosa  Sepal.Width Sepal.Length    0.7425467
versicolor  Petal.Width Petal.Length   0.7866681
virginica  Petal.Length Sepal.Length   0.8642247
```

```
$setosa
X1      X2      correlation
2 Sepal.Width Sepal.Length    0.7425467

$versicolor
X1      X2      correlation
12 Petal.Width Petal.Length   0.7866681

$virginica
X1      X2      correlation
3 Petal.Length Sepal.Length   0.8642247
```

Note that one computations are performed on such groupwise level, one may be interested in stacking the results, which can be done with:

```
> (result <- do.call("rbind", topcor))

      X1      X2      correlation
setosa  Sepal.Width Sepal.Length    0.7425467
versicolor  Petal.Width Petal.Length   0.7866681
virginica  Petal.Length Sepal.Length   0.8642247
```

第20章：读取和写入纯文本文件中的表格数据（CSV、TSV等）

参数	详细信息
文件	要读取的CSV文件名
表头	逻辑值：.csv文件是否包含带有列名的表头行？
分隔符	字符：分隔每行单元格的符号
引号	字符：用于引用字符串的符号
小数点	字符：用作小数分隔符的符号
填充	逻辑值：当为 TRUE 时，长度不等的行将用空字段填充。
comment.char	字符：在 csv 文件中用作注释的字符。以该字符开头的行将被忽略。
...	传递给 <code>read.table</code> 的额外参数

第 20.1 节：导入 .csv 文件

使用基础 R 导入

逗号分隔值文件 (CSV) 可以使用 `read.csv` 导入，该函数封装了 `read.table`，但使用 `sep = ","` 将分隔符设置为逗号。

```
# 获取 R 的 utils 包中包含的 CSV 文件路径
csv_path <- system.file("misc", "exDIF.csv", package = "utils")

# 路径会根据安装位置有所不同
csv_path
## [1] "/Library/Frameworks/R.framework/Resources/library/utils/misc/exDIF.csv"

df <- read.csv(csv_path)

df
##   Var1 Var2
## 1  2.70   A
## 2  3.14   B
## 3 10.00   A
## 4 -7.00   A
```

一个用户友好的选项，`file.choose`，允许浏览目录：

```
df <- read.csv(file.choose())
```

注意事项

- 与 `read.table` 不同，`read.csv` 默认 `header = TRUE`，使用第一行作为列名。
- 所有这些函数默认会将字符串转换为 `factor` 类，除非设置了 `as.is = TRUE` 或 `stringsAsFactors = FALSE`。
- `read.csv2` 变体默认使用 `sep = ";"` 和 `dec = ","`，适用于逗号作为小数点、分号作为字段分隔符的国家的数据。

使用包导入

`readr` 包的 `read_csv` 函数提供更快的性能、大文件时的进度条，以及比标准 `read.csv` 更受欢迎的默认选项，包括 `stringsAsFactors = FALSE`。

Chapter 20: Reading and writing tabular data in plain-text files (CSV, TSV, etc.)

Parameter	Details
file	name of the CSV file to read
header	logical: does the .csv file contain a header row with column names?
sep	character: symbol that separates the cells on each row
quote	character: symbol used to quote character strings
dec	character: symbol used as decimal separator
fill	logical: when TRUE, rows with unequal length are filled with blank fields.
comment.char	character: character used as comment in the csv file. Lines preceded by this character are ignored.
...	extra arguments to be passed to <code>read.table</code>

Section 20.1: Importing .csv files

Importing using base R

Comma separated value files (CSVs) can be imported using `read.csv`, which wraps `read.table`, but uses `sep = ","` to set the delimiter to a comma.

```
# get the file path of a CSV included in R's utils package
csv_path <- system.file("misc", "exDIF.csv", package = "utils")

# path will vary based on installation location
csv_path
## [1] "/Library/Frameworks/R.framework/Resources/library/utils/misc/exDIF.csv"

df <- read.csv(csv_path)

df
##   Var1 Var2
## 1  2.70   A
## 2  3.14   B
## 3 10.00   A
## 4 -7.00   A
```

A user friendly option, `file.choose`, allows to browse through the directories:

```
df <- read.csv(file.choose())
```

Notes

- Unlike `read.table`, `read.csv` defaults to `header = TRUE`, and uses the first row as column names.
- All these functions will convert strings to `factor` class by default unless either `as.is = TRUE` or `stringsAsFactors = FALSE`.
- The `read.csv2` variant defaults to `sep = ";"` and `dec = ","` for use on data from countries where the comma is used as a decimal point and the semicolon as a field separator.

Importing using packages

The `readr` package's `read_csv` function offers much faster performance, a progress bar for large files, and more popular default options than standard `read.csv`, including `stringsAsFactors = FALSE`.

```

library(readr)

df <- read_csv(csv_path)

df
## # 一个包含4行2列的tibble
##   Var1  Var2
##   <dbl> <chr>
## 1 2.70   A
## 2 3.14   B
## 3 10.00  A
## 4 -7.00  A

```

第20.2节：使用data.table导入数据

data.table包引入了函数fread。虽然它类似于read.table，但fread通常更快且更灵活，能够自动猜测文件的分隔符。

```

# 获取R的utils包中包含的CSV文件路径
csv_path <- system.file("misc", "exDIF.csv", package = "utils")

# 路径会根据R的安装位置而变化
csv_path
## [1] "/Library/Frameworks/R.framework/Resources/library/utils/misc/exDIF.csv"

dt <- fread(csv_path)

dt
##   Var1  Var2
## 1 2.70   A
## 2 3.14   B
## 3 10.00  A
## 4 -7.00  A

```

参数input是一个字符串，表示：

- 文件名（例如 "filename.csv"），一个
- 作用于文件的shell命令（例如 "grep 'word' filename"），或者输入本身（例如 "input1, input2 A, B C, D"）。

fread返回一个data.table类的对象，该类继承自data.frame类，适合用于data.table的[]操作。若要返回普通的data.frame，请将data.table参数设置为FALSE：

```

df <- fread(csv_path, data.table = FALSE)

class(df)
## [1] "data.frame"

df
##   Var1  Var2
## 1 2.70   A
## 2 3.14   B
## 3 10.00  A
## 4 -7.00  A

```

注意事项

- fread 没有 read.table 的所有相同选项。缺少的一个参数是 na.comment，这可能导致

```

library(readr)

df <- read_csv(csv_path)

df
## # A tibble: 4 x 2
##   Var1  Var2
##   <dbl> <chr>
## 1 2.70   A
## 2 3.14   B
## 3 10.00  A
## 4 -7.00  A

```

Section 20.2: Importing with data.table

The data.table package introduces the function [fread](#). While it is similar to [read.table](#), fread is usually faster and more flexible, guessing the file's delimiter automatically.

```

# get the file path of a CSV included in R's utils package
csv_path <- system.file("misc", "exDIF.csv", package = "utils")

# path will vary based on R installation location
csv_path
## [1] "/Library/Frameworks/R.framework/Resources/library/utils/misc/exDIF.csv"

dt <- fread(csv_path)

dt
##   Var1  Var2
## 1 2.70   A
## 2 3.14   B
## 3 10.00  A
## 4 -7.00  A

```

Where argument `input` is a string representing:

- the filename (e.g. `"filename.csv"`),
- a shell command that acts on a file (e.g. `"grep 'word' filename"`), or
- the input itself (e.g. `"input1, input2 \n A, B \n C, D"`).

fread returns an object of class `data.table` that inherits from class `data.frame`, suitable for use with the data.table's usage of `[]`. To return an ordinary data.frame, set the `data.table` parameter to FALSE:

```

df <- fread(csv_path, data.table = FALSE)

class(df)
## [1] "data.frame"

df
##   Var1  Var2
## 1 2.70   A
## 2 3.14   B
## 3 10.00  A
## 4 -7.00  A

```

Notes

- fread does not have all same options as [read.table](#). One missing argument is `na.comment`, which may lead

- 如果源文件包含 #，可能会出现不希望的行为。
- fread 仅使用 " 作为 quote 参数。
 - fread 使用少量 (5) 行来猜测变量类型。

第20.3节：导出.csv文件

使用基础R导出

数据可以使用 `write.csv()` 写入CSV文件：

```
write.csv(mtcars, "mtcars.csv")
```

常用参数包括`row.names=FALSE`和`na=""`。

使用包导出

`readr::write_csv`比`write.csv`快得多，并且不会写入行名。

```
library(readr)
```

```
write_csv(mtcars, "mtcars.csv")
```

第20.4节：导入多个csv文件

```
files = list.files(pattern=".csv")
data_list = lapply(files, read.table, header = TRUE)
```

这会读取每个文件并将其添加到列表中。之后，如果所有`data.frame`结构相同，则可以合并成一个大的`data.frame`：

```
df <- do.call(rbind, data_list)
```

第20.5节：导入定宽文件

定宽文件是文本文件，其中列之间没有任何字符分隔符，如、或;，而是具有固定的字符长度（宽度）。数据通常用空格填充。

举个例子：

Column1	Column2	Column3	Column4	Column5
1647	pi	'important'	3.141596.28318	
1731	euler	'quite important'	2.718285.43656	
1979	answer	'The Answer.'	42	42

假设该数据表存在于工作目录中的本地文件constants.txt中。

使用基础R导入

```
df <- read.fwf('constants.txt', widths = c(8,10,18,7,8), header = FALSE, skip = 1)
```

df	V1	V2	V3	V4	V5
#>					

- in unwanted behaviors if the source file contains #.
- fread uses only " for quote parameter.
 - fread uses few (5) lines to guess variables types.

Section 20.3: Exporting .csv files

Exporting using base R

Data can be written to a CSV file using `write.csv()`:

```
write.csv(mtcars, "mtcars.csv")
```

Commonly-specified parameters include `row.names = FALSE` and `na = ""`.

Exporting using packages

`readr::write_csv` is significantly faster than `write.csv` and does not write row names.

```
library(readr)
```

```
write_csv(mtcars, "mtcars.csv")
```

Section 20.4: Import multiple csv files

```
files = list.files(pattern=".csv")
data_list = lapply(files, read.table, header = TRUE)
```

This read every file and adds it to a list. Afterwards, if all `data.frame` have the same structure they can be combined into one big `data.frame`:

```
df <- do.call(rbind, data_list)
```

Section 20.5: Importing fixed-width files

Fixed-width files are text files in which columns are not separated by any character delimiter, like , or ;, but rather have a fixed character length (*width*). Data is usually padded with white spaces.

An example:

Column1	Column2	Column3	Column4	Column5
1647	pi	'important'	3.141596.28318	
1731	euler	'quite important'	2.718285.43656	
1979	answer	'The Answer.'	42	42

Let's assume this data table exists in the local file constants.txt in the working directory.

Importing with base R

```
df <- read.fwf('constants.txt', widths = c(8,10,18,7,8), header = FALSE, skip = 1)
```

df	V1	V2	V3	V4	V5
#>					

```
#> 1 1647 pi      'important' 3.14159 6.28318
#> 2 1731 euler  'quite important' 2.71828 5.43656
#> 3 1979 answer 'The Answer.' 42        42.0000
```

注意：

- 列标题不需要用字符分隔 (Column4Column5)
- widths参数定义每列的宽度
- 未分隔的表头无法通过read.fwf()读取

使用readr导入

```
library(readr)
```

```
df <- read_fwf('constants.txt',
fwf_cols(Year = 8, Name = 10, Importance = 18, Value = 7, Doubled = 8),
skip = 1)
df
#> # 一个3行5列的tibble
#>   Year     Name    Importance   Value   Doubled
#>   <int>   <chr>    <chr>       <dbl>    <dbl>
#> 1 1647     pi      'important'  3.14159  6.28318
#> 2 1731     euler   'quite important' 2.71828  5.43656
#> 3 1979     answer  'The Answer.' 42.00000 42.00000
```

注意：

- readr的fwf_*辅助函数提供了指定列长度的替代方法，包括自动猜测 (fwf_empty)
- readr比基础R更快
- 列标题无法从数据文件中自动导入

```
#> 1 1647 pi      'important' 3.14159 6.28318
#> 2 1731 euler  'quite important' 2.71828 5.43656
#> 3 1979 answer 'The Answer.' 42        42.0000
```

Note:

- Column titles don't need to be separated by a character (Column4Column5)
- The widths parameter defines the width of each column
- Non-separated headers are not readable with `read.fwf()`

Importing with readr

```
library(readr)
```

```
df <- read_fwf('constants.txt',
fwf_cols(Year = 8, Name = 10, Importance = 18, Value = 7, Doubled = 8),
skip = 1)
df
#> # A tibble: 3 x 5
#>   Year     Name    Importance   Value   Doubled
#>   <int>   <chr>    <chr>       <dbl>    <dbl>
#> 1 1647     pi      'important'  3.14159  6.28318
#> 2 1731     euler   'quite important' 2.71828  5.43656
#> 3 1979     answer  'The Answer.' 42.00000 42.00000
```

Note:

- readr's ffw_* helper functions offer alternative ways of specifying column lengths, including automatic guessing (fwf_empty)
- readr is faster than base R
- Column titles cannot be automatically imported from data file

第21章：管道操作符 (%>% 及其他)

左侧值 右侧值
一个值或 magrittr 占位符。使用 magrittr 语义的函数调用

管道操作符，可在magrittr、dplyr及其他 R 包中使用，通过一系列操作处理数据对象，使用中缀操作符将一步的结果作为下一步的输入，而非更典型的 R 嵌套函数调用方式。

请注意，管道操作符的主要目的是提高代码的可读性。有关性能方面的考虑，请参见备注部分。

第21.1节：基本用法与链式调用

管道操作符%>%用于将一个参数插入函数中。它不是语言的基础特性，只有在加载提供该操作符的包（如magrittr）后才能使用。管道操作符将管道左侧（LHS）的值作为右侧（RHS）函数的第一个参数。例如：

```
library(magrittr)  
  
1:10 %>% mean  
# [1] 5.5  
  
# 等同于  
平均值(1:10)  
# [1] 5.5
```

管道可以用来替代一系列函数调用。多个管道允许我们从左到右读取和写入序列，而不是从内向外。例如，假设我们有一个定义为因子的years，但想将其转换为数值型。为了防止可能的信息丢失，我们先转换为字符型，然后再转换为数值型：

```
years <- factor(2008:2012)  
  
# 嵌套调用  
as.numeric(as.character(years))  
  
# 管道操作  
years %>% as.character %>% as.numeric
```

如果我们不希望左侧（LHS）作为右侧（RHS）第一个参数使用，有一些变通方法，比如命名参数或使用.来指示管道输入的位置。

```
# 使用grep的示例  
# 其语法：  
# grep(pattern, x, ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)  
  
# 注意，`substring`的结果是grep的第2个参数  
grep("Wo", substring("Hello World", 7, 11))  
  
# 命名其他参数的管道操作  
"Hello World" %>% substring(7, 11) %>% grep(pattern = "Wo")
```

Chapter 21: Pipe operators (%>% and others)

lhs A value or the magrittr placeholder. A function call using the magrittr semantics

Pipe operators, available in `magrittr`, `dplyr`, and other R packages, process a data-object using a sequence of operations by passing the result of one step as input for the next step using infix-operators rather than the more typical R method of nested function calls.

Note that the intended aim of pipe operators is to increase human readability of written code. See Remarks section for performance considerations.

Section 21.1: Basic use and chaining

The pipe operator, `%>%`, is used to insert an argument into a function. It is not a base feature of the language and can only be used after attaching a package that provides it, such as `magrittr`. The pipe operator takes the left-hand side (LHS) of the pipe and uses it as the first argument of the function on the right-hand side (RHS) of the pipe. For example:

```
library(magrittr)  
  
1:10 %>% mean  
# [1] 5.5  
  
# is equivalent to  
mean(1:10)  
# [1] 5.5
```

The pipe can be used to replace a sequence of function calls. Multiple pipes allow us to read and write the sequence from left to right, rather than from inside to out. For example, suppose we have years defined as a factor but want to convert it to a numeric. To prevent possible information loss, we first convert to character and then to numeric:

```
years <- factor(2008:2012)  
  
# nesting  
as.numeric(as.character(years))  
  
# piping  
years %>% as.character %>% as.numeric
```

If we don't want the LHS (Left Hand Side) used as the *first* argument on the RHS (Right Hand Side), there are workarounds, such as naming the arguments or using . to indicate where the piped input goes.

```
# example with grep  
# its syntax:  
# grep(pattern, x, ignore.case = FALSE, perl = FALSE, fixed = FALSE, useBytes = FALSE)  
  
# note that the `substring` result is the *2nd* argument of grep  
grep("Wo", substring("Hello World", 7, 11))  
  
# piping while naming other arguments  
"Hello World" %>% substring(7, 11) %>% grep(pattern = "Wo")
```

```

# 使用 . 进行管道操作。
"Hello World" %>% substring(7, 11) %>% grep("Wo", .)

# 使用 . 和花括号进行管道操作
"Hello World" %>% substring(7, 11) %>% { c(paste('Hi', .)) }
#[1] "Hi World"

# 在带有花括号和 . 的参数中多次使用左侧值 (LHS)
"Hello World" %>% substring(7, 11) %>% { c(paste(. , 'Hi', .)) }
#[1] "World Hi World"

```

第21.2节：函数序列

给定一系列我们反复使用的步骤，通常将其存储在函数中会很方便。管道允许通过以点开头的序列以可读的格式保存此类函数，如下所示：

. %>% 右侧表达式 (RHS)

举例来说，假设我们有因子型日期，想要提取年份：

```

library(magrittr) # 需要包含管道操作符
library(lubridate)
read_year <- . %>% as.character %>% as.Date %>% year

# 创建数据集
df <- data.frame(now = "2015-11-11", before = "2012-01-01")
#       now      before
# 1 2015-11-11 2012-01-01

# 示例 1：对单个字符向量应用 `read_year`
df$now %>% read_year
# [1] 2015

# 示例 2：对 `df` 的所有列应用 `read_year`
df %>% lapply(read_year) %>% as.data.frame # 隐式调用 `lapply(df, read_year)`
#       now      before
# 1 2015    2012

# 示例 3：使用 `mutate_all`，效果同上
library(dplyr)
df %>% mutate_all(funs(read_year))
# 如果是旧版本的 dplyr 使用 `mutate_each`'
#       now      before
# 1 2015    2012

```

我们可以通过输入函数名或使用functions来查看函数的组成：

```

read_year
# 由以下组件组成的函数序列：
#
# 1. as.character()
# 2. as.Date()
# 3. year()
#
# 使用 'functions' 来提取各个单独的函数。

```

我们也可以通过函数在序列中的位置来访问每个函数：

```
read_year[[2]]
```

```

# piping with .
"Hello World" %>% substring(7, 11) %>% grep("Wo", .)

# piping with . and curly braces
"Hello World" %>% substring(7, 11) %>% { c(paste('Hi', .)) }
#[1] "Hi World"

#using LHS multiple times in argument with curly braces and .
"Hello World" %>% substring(7, 11) %>% { c(paste(. , 'Hi', .)) }
#[1] "World Hi World"

```

Section 21.2: Functional sequences

Given a sequence of steps we use repeatedly, it's often handy to store it in a function. Pipes allow for saving such functions in a readable format by starting a sequence with a dot as in:

. %>% RHS

As an example, suppose we have factor dates and want to extract the year:

```

library(magrittr) # needed to include the pipe operators
library(lubridate)
read_year <- . %>% as.character %>% as.Date %>% year

# Creating a dataset
df <- data.frame(now = "2015-11-11", before = "2012-01-01")
#       now      before
# 1 2015-11-11 2012-01-01

# Example 1: applying `read_year` to a single character-vector
df$now %>% read_year
# [1] 2015

# Example 2: applying `read_year` to all columns of `df`
df %>% lapply(read_year) %>% as.data.frame # implicit `lapply(df, read_year)`
#       now      before
# 1 2015    2012

# Example 3: same as above using `mutate_all`
library(dplyr)
df %>% mutate_all(funs(read_year))
# if an older version of dplyr use `mutate_each`'
#       now      before
# 1 2015    2012

```

We can review the composition of the function by typing its name or using functions:

```

read_year
# Functional sequence with the following components:
#
# 1. as.character()
# 2. as.Date()
# 3. year()
#
# Use 'functions' to extract the individual functions.

```

We can also access each function by its position in the sequence:

```
read_year[[2]]
```

```
# function ()
# as.Date()
```

通常，当清晰度比速度更重要时，这种方法可能会很有用。

第21.3节：带有 %<>% 的赋值

magrittr包包含一个复合赋值中缀操作符%<>%，该操作符通过先将一个值传递给一个或多个rhs表达式，然后再赋值来更新该值。这消除了需要输入对象名两次（赋值操作符<-两边各一次）的需求。%<>%必须是链中第一个中缀操作符：

```
library(magrittr)
library(dplyr)

df <- mtcars
```

代替写成

```
df <- df %>% select(1:3) %>% filter(mpg > 20, cyl == 6)
```

或者

```
df %>% select(1:3) %>% filter(mpg > 20, cyl == 6) -> df
```

复合赋值操作符将同时进行管道传递并重新赋值给 df：

```
df %<>% select(1:3) %>% filter(mpg > 20, cyl == 6)
```

第21.4节：使用 %\$% 暴露内容

暴露管道操作符%\$%将左侧对象中的列名作为R符号暴露给右侧表达式。当管道传递给没有data参数的函数时（例如不像lm那样），以及不接受data.frame和列名作为参数的函数（大多数主要的dplyr函数），该操作符非常有用。

指数管道操作符%\$%允许用户在需要引用列名时避免中断管道。例如，假设你想过滤一个数据框，然后对两列运行相关性检验cor.test：

```
library(magrittr)
library(dplyr)
mtcars %>%
  filter(wt > 2) %$%
  cor.test(hp, mpg)

#>
#> 皮尔逊积矩相关系数检验

#> 数据：hp 和 mpg
#> t = -5.9546, df = 26, p-value = 2.768e-06
#> 备择假设：真实相关系数不等于 0
#> 95% 置信区间：
#> -0.8825498 -0.5393217
#> 样本估计值：
#>      cor
```

```
# function (.)
# as.Date(.)
```

Generally, this approach may be useful when clarity is more important than speed.

Section 21.3: Assignment with %<>%

The magrittr package contains a compound assignment infix-operator, %<>%, that updates a value by first piping it into one or more rhs expressions and then assigning the result. This eliminates the need to type an object name twice (once on each side of the assignment operator <-). %<>% must be the first infix-operator in a chain:

```
library(magrittr)
library(dplyr)

df <- mtcars
```

Instead of writing

```
df <- df %>% select(1:3) %>% filter(mpg > 20, cyl == 6)
```

or

```
df %>% select(1:3) %>% filter(mpg > 20, cyl == 6) -> df
```

The compound assignment operator will both pipe and reassign df:

```
df %<>% select(1:3) %>% filter(mpg > 20, cyl == 6)
```

Section 21.4: Exposing contents with %\$%

The exposition pipe operator, %\$%, exposes the column names as R symbols within the left-hand side object to the right-hand side expression. This operator is handy when piping into functions that do not have a `data` argument (unlike, say, `lm`) and that don't take a `data.frame` and column names as arguments (most of the main dplyr functions).

The exposition pipe operator %\$% allows a user to avoid breaking a pipeline when needing to refer to column names. For instance, say you want to filter a data.frame and then run a correlation test on two columns with `cor.test`:

```
library(magrittr)
library(dplyr)
mtcars %>%
  filter(wt > 2) %$%
  cor.test(hp, mpg)

#>
#> Pearson's product-moment correlation
#>
#> data: hp and mpg
#> t = -5.9546, df = 26, p-value = 2.768e-06
#> alternative hypothesis: true correlation is not equal to 0
#> 95 percent confidence interval:
#> -0.8825498 -0.5393217
#> sample estimates:
#>      cor
```

```
#> -0.7595673
```

这里标准的%>%管道将数据框传递给filter()，而%\$%管道则将列名暴露给cor.test()。

指数管道的工作原理类似于基础R中的with()函数的可管道版本，且接受相同的左侧对象作为输入。

第21.5节：使用 %T>% 创建副作用

R 中的一些函数会产生副作用（例如保存、打印、绘图等），并不总是返回有意义或期望的值。

%T>% (tee 操作符) 允许你将一个值传递给产生副作用的函数，同时保持原始lhs 值不变。换句话说：tee 操作符的作用类似于 %>%，但返回值是 lhs 本身，而不是rhs 函数/表达式的结果。

示例：创建、管道传递、写入并返回一个对象。如果在此示例中用 %>% 替代 %T>%，则变量 all_letters 会包含NULL，而不是排序对象的值。

```
all_letters <- c(letters, LETTERS) %>%  
  sort %T>%  
  write.csv(file = "all_letters.csv")
```

```
read.csv("all_letters.csv") %>% head()  
#> #> #> #> #> #>  
#>   X  
#> 1 a  
#> 2 A  
#> 3 b  
#> 4 B  
#> 5 c  
#> 6 C
```

警告：将未命名对象通过 save() 管道传递时，加载到工作区时会生成名为 . 的对象，使用 load() 加载时也是如此。不过，可以使用辅助函数的变通方法（也可以以内联匿名函数的形式编写）。

```
all_letters <- c(letters, LETTERS) %>%  
  sort %T>%  
  save(file = "all_letters.RData")  
  
load("all_letters.RData", e <- new.env())
```

```
get("all_letters", envir = e)  
#> Error in get("all_letters", envir = e) : object 'all_letters' not found
```

```
get(".", envir = e)  
#> [1] "a" "A" "b" "B" "c" "C" "d" "D" "e" "E" "f" "F" "g" "G" "h" "H" "i" "I" "j" "J"  
#> [21] "k" "K" "l" "L" "m" "M" "n" "N" "o" "O" "p" "P" "q" "Q" "r" "R" "s" "S" "t" "T"  
#> [41] "u" "U" "v" "V" "w" "W" "x" "X" "y" "Y" "z" "Z"
```

```
# 解决方法  
save2 <- function(. = ., name, file = stop("file' 必须被指定")) {  
  assign(name, .)  
  call_save <- call("save", ... = name, file = file)  
  eval(call_save)  
}
```

```
#> -0.7595673
```

Here the standard %>% pipe passes the data.frame through to `filter()`, while the %\$% pipe exposes the column names to `cor.test()`.

The exposition pipe works like a pipe-able version of the base R `with()` functions, and the same left-hand side objects are accepted as inputs.

Section 21.5: Creating side effects with %T>%

Some functions in R produce a side effect (i.e. saving, printing, plotting, etc) and do not always return a meaningful or desired value.

%T>% (tee operator) allows you to forward a value into a side-effect-producing function while keeping the original lhs value intact. In other words: the tee operator works like %>%, except the return values is lhs itself, and not the result of the rhs function/expression.

Example: Create, pipe, write, and return an object. If %>% were used in place of %T>% in this example, then the variable all_letters would contain NULL rather than the value of the sorted object.

```
all_letters <- c(letters, LETTERS) %>%  
  sort %T>%  
  write.csv(file = "all_letters.csv")
```

```
read.csv("all_letters.csv") %>% head()  
#> #> #> #> #> #>  
#>   X  
#> 1 a  
#> 2 A  
#> 3 b  
#> 4 B  
#> 5 c  
#> 6 C
```

Warning: Piping an unnamed object to `save()` will produce an object named . when loaded into the workspace with `load()`. However, a workaround using a helper function is possible (which can also be written inline as an anonymous function).

```
all_letters <- c(letters, LETTERS) %>%  
  sort %T>%  
  save(file = "all_letters.RData")
```

```
load("all_letters.RData", e <- new.env())  
  
get("all_letters", envir = e)  
#> Error in get("all_letters", envir = e) : object 'all_letters' not found
```

```
get(".", envir = e)  
#> [1] "a" "A" "b" "B" "c" "C" "d" "D" "e" "E" "f" "F" "g" "G" "h" "H" "i" "I" "j" "J"  
#> [21] "k" "K" "l" "L" "m" "M" "n" "N" "o" "O" "p" "P" "q" "Q" "r" "R" "s" "S" "t" "T"  
#> [41] "u" "U" "v" "V" "w" "W" "x" "X" "y" "Y" "z" "Z"
```

```
# Work-around  
save2 <- function(. = ., name, file = stop("file' must be specified")) {  
  assign(name, .)  
  call_save <- call("save", ... = name, file = file)  
  eval(call_save)  
}
```

```
all_letters <- c(letters, LETTERS) %>%
  sort %T>%
save2("all_letters", "all_letters.RData")
```

第21.6节：在dplyr和ggplot2中使用管道操作符

%>% 操作符也可以用来将dplyr的输出传递给ggplot。这创建了一个统一的探索性数据分析（EDA）流程，且易于定制。该方法比在ggplot内部进行聚合更快，并且避免了不必要的中间变量。

```
library(dplyr)
library(ggplot)

diamonds %>%
  filter(depth > 60) %>%
  group_by(cut) %>%
  summarize(mean_price = mean(price)) %>%
  ggplot(aes(x = cut, y = mean_price)) +
  geom_bar(stat = "identity")
```

```
all_letters <- c(letters, LETTERS) %>%
  sort %T>%
save2("all_letters", "all_letters.RData")
```

Section 21.6: Using the pipe with dplyr and ggplot2

The %>% operator can also be used to pipe the dplyr output into ggplot. This creates a unified exploratory data analysis (EDA) pipeline that is easily customizable. This method is faster than doing the aggregations internally in ggplot and has the added benefit of avoiding unnecessary intermediate variables.

```
library(dplyr)
library(ggplot)

diamonds %>%
  filter(depth > 60) %>%
  group_by(cut) %>%
  summarize(mean_price = mean(price)) %>%
  ggplot(aes(x = cut, y = mean_price)) +
  geom_bar(stat = "identity")
```

第22章：线性模型（回归）

参数	含义
公式	Wilkinson-Rogers 符号法中的一个公式；response ~ ...，其中...包含对应于环境中或由data参数指定的数据框中的变量的项
数据	包含响应变量和预测变量的数据框
subset	指定要使用的观测子集的向量：可以用data中变量的逻辑表达式表示
权重	分析权重（参见上文的Weights部分）
na.action	如何处理缺失值（NA）：参见?na.action
方法	如何执行拟合。唯一选项为 "qr" 或 "model.frame"（后者返回模型框架但不拟合模型，与指定model=TRUE相同）
模型	是否在拟合对象中存储模型框架
x	是否在拟合对象中存储模型矩阵
y	是否将模型响应存储在拟合对象中
qr	是否将QR分解存储在拟合对象中
singular.ok	是否允许奇异拟合，即具有共线性预测变量的模型（此时部分系数将自动被设置为NA）
对比	用于模型中特定因子的对比列表；参见contrasts.arg参数的说明，位于?model.matrix.default。对比也可以通过options()（参见contrasts参数）设置，或通过赋值因子的contrast属性设置（参见?contrasts）
偏置项	用于指定模型中已知的先验成分。也可以作为公式的一部分指定。参见?model.offset
...	传递给底层拟合函数(lm.fit()或lm.wfit())的其他参数

第22.1节：mtcars数据集上的线性回归

内置的mtcars数据框包含了32辆车的信息，包括它们的重量、燃油效率（每加仑英里数）、速度等。（要了解更多关于该数据集的信息，请使用help(mtcars)。）

如果我们对燃油效率（mpg）和重量（wt）之间的关系感兴趣，可以先用以下命令绘制这些变量的图形：

```
plot(mpg ~ wt, data = mtcars, col=2)
```

图表显示了一个（线性）关系！如果我们想进行线性回归以确定线性模型的系数，可以使用lm函数：

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

这里的~表示“由.....解释”，所以公式mpg ~ wt表示我们用wt来解释预测mpg。查看输出最有用的方法是：

```
summary(fit)
```

输出结果如下：

调用：
`lm(公式 = mpg ~ wt, 数据 = mtcars)`

残差：
最小值 第一四分位数 中位数 第三四分位数 最大值

Chapter 22: Linear Models (Regression)

Parameter	Meaning
formula	a formula in Wilkinson-Rogers notation; response ~ ... where ... contains terms corresponding to variables in the environment or in the data frame specified by the <code>data</code> argument
data	data frame containing the response and predictor variables
subset	a vector specifying a subset of observations to be used: may be expressed as a logical statement in terms of the variables in <code>data</code>
weights	analytical weights (see <code>Weights</code> section above)
na.action	how to handle missing (NA) values: see ? <code>na.action</code>
method	how to perform the fitting. Only choices are "qr" or "model.frame" (the latter returns the model frame without fitting the model, identical to specifying <code>model=TRUE</code>)
model	whether to store the model frame in the fitted object
x	whether to store the model matrix in the fitted object
y	whether to store the model response in the fitted object
qr	whether to store the QR decomposition in the fitted object
singular.ok	whether to allow <i>singular fits</i> , models with collinear predictors (a subset of the coefficients will automatically be set to NA in this case)
contrasts	a list of contrasts to be used for particular factors in the model; see the <code>contrasts.arg</code> argument of ? <code>model.matrix.default</code> . Contrasts can also be set with <code>options()</code> (see the <code>contrasts</code> argument) or by assigning the contrast attributes of a factor (see ? <code>contrasts</code>)
offset	used to specify an <i>a priori</i> known component in the model. May also be specified as part of the formula. See ? <code>model.offset</code>
...	additional arguments to be passed to lower-level fitting functions (<code>lm.fit()</code> or <code>lm.wfit()</code>)

Section 22.1: Linear regression on the mtcars dataset

The built-in mtcars data frame contains information about 32 cars, including their weight, fuel efficiency (in miles-per-gallon), speed, etc. (To find out more about the dataset, use `help(mtcars)`).

If we are interested in the relationship between fuel efficiency (mpg) and weight (wt) we may start plotting those variables with:

```
plot(mpg ~ wt, data = mtcars, col=2)
```

The plots shows a (linear) relationship!. Then if we want to perform linear regression to determine the coefficients of a linear model, we would use the `lm` function:

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

The ~ here means "explained by", so the formula `mpg ~ wt` means we are predicting mpg as explained by wt. The most helpful way to view the output is with:

```
summary(fit)
```

Which gives the output:

Call:
`lm(formula = mpg ~ wt, data = mtcars)`

Residuals:
Min 1Q Median 3Q Max

```
-4.5432 -2.3647 -0.1252 1.4096 6.8727
```

系数：
估计值 标准误差 t 值 Pr(>|t|)
(截距) 37.2851 1.8776 19.858 < 2e-16 ***
wt -5.3445 0.5591 -9.559 1.29e-10 ***

显著性代码: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

残差标准误差: 3.046 自由度 30
多重R平方: 0.7528, 调整后的R平方: 0.7446
F统计量: 91.38 自由度 1 和 30, p值: 1.294e-10

这提供了以下信息：

- 每个系数 (wt 和 y截距) 的估计斜率, 表明mpg的最佳拟合预测为 $37.2851 + (-5.3445) * \text{wt}$
- 每个系数的p值, 表明截距和重量可能不是偶然的
- 拟合的整体估计, 如R^2和调整后的R^2, 显示模型解释了多少 mpg的变异

我们可以在第一个图中添加一条线来显示预测的mpg：

```
abline(fit,col=3,lwd=2)
```

也可以将方程添加到该图中。首先, 使用coef获取系数。然后使用paste0将系数与适当的变量和+/-合并, 构建方程。最后, 使用mtext将其添加到图中：

```
bs <- round(coef(fit), 3)
lmlab <- paste0("mpg = ", bs[1],
  ifelse(sign(bs[2]) == 1, " + ", " - "), abs(bs[2]), " wt ")
mtext(lmlab, 3, line=-2)
```

结果是：

```
-4.5432 -2.3647 -0.1252 1.4096 6.8727
```

Coefficients:
Estimate Std. Error t value Pr(>|t|)
(Intercept) 37.2851 1.8776 19.858 < 2e-16 ***
wt -5.3445 0.5591 -9.559 1.29e-10 ***

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 3.046 on 30 degrees of freedom
Multiple R-squared: 0.7528, Adjusted R-squared: 0.7446
F-statistic: 91.38 on 1 and 30 DF, p-value: 1.294e-10

This provides information about:

- the estimated slope of each coefficient (wt and the y-intercept), which suggests the best-fit prediction of mpg is $37.2851 + (-5.3445) * \text{wt}$
- The p-value of each coefficient, which suggests that the intercept and weight are probably not due to chance
- Overall estimates of fit such as R^2 and adjusted R^2, which show how much of the variation in mpg is explained by the model

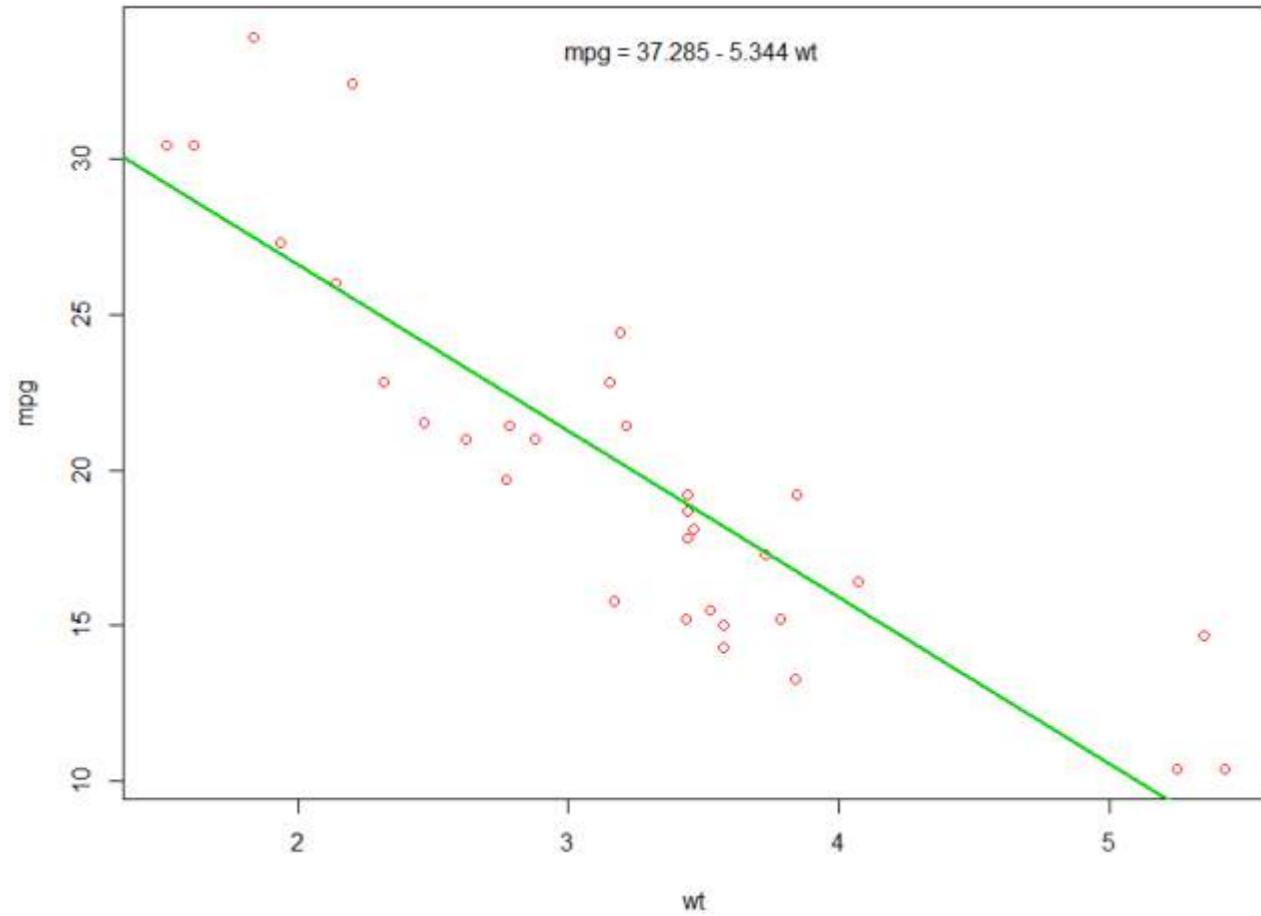
We could add a line to our first plot to show the predicted mpg:

```
abline(fit,col=3,lwd=2)
```

It is also possible to add the equation to that plot. First, get the coefficients with **coef**. Then using **paste0** we collapse the coefficients with appropriate variables and +/-, to built the equation. Finally, we add it to the plot using **mtext**:

```
bs <- round(coef(fit), 3)
lmlab <- paste0("mpg = ", bs[1],
  ifelse(sign(bs[2]) == 1, " + ", " - "), abs(bs[2]), " wt ")
mtext(lmlab, 3, line=-2)
```

The result is:



第22.2节：使用 'predict' 函数

一旦建立了模型，`predict`是用新数据进行测试的主要函数。我们的示例将使用内置数据集mtcars来回归每加仑英里数与排量：

```
my_mdl <- lm(mpg ~ disp, data=mtcars)
my_mdl
```

调用:

```
lm(formula = mpg ~ disp, data = mtcars)
```

系数:

(截距)	disp
29.59985	-0.04122

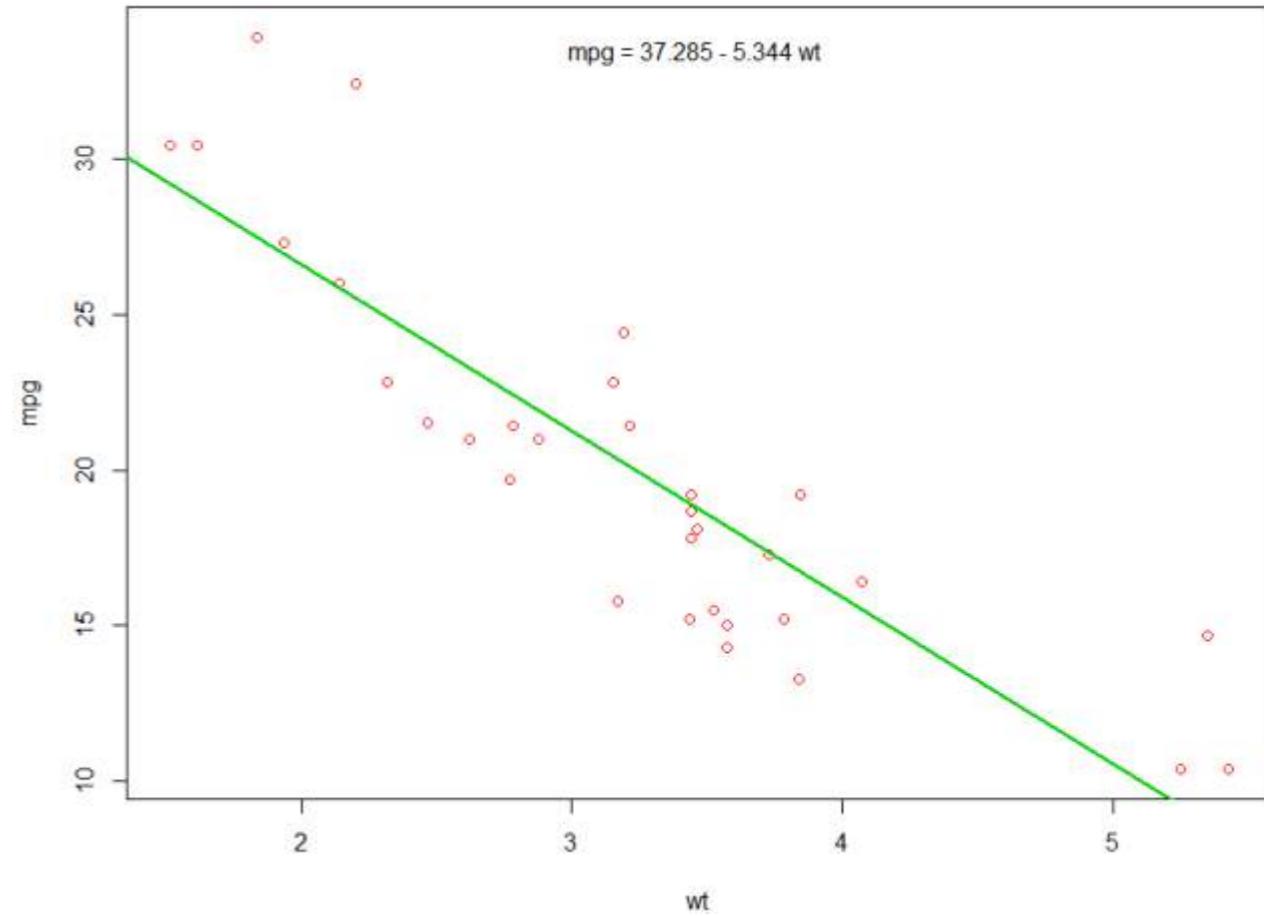
如果我有一个带位移的新数据源，我就能看到估计的每加仑英里数。

```
set.seed(1234)
newdata <- sample(mtcars$disp, 5)
newdata
[1] 258.0 71.1 75.7 145.0 400.0

newdf <- data.frame(disp=newdata)
predict(my_mdl, newdf)
1 2 3 4 5
18.96635 26.66946 26.47987 23.62366 13.11381
```

该过程最重要的部分是创建一个与原始数据具有相同列名的新数据框。

在本例中，原始数据有一个名为`disp`的列，我确保新数据也使用相同的名称。



Section 22.2: Using the 'predict' function

Once a model is built `predict` is the main function to test with new data. Our example will use the `mtcars` built-in dataset to regress miles per gallon against displacement:

```
my_mdl <- lm(mpg ~ disp, data=mtcars)
my_mdl
```

Call:

```
lm(formula = mpg ~ disp, data = mtcars)
```

Coefficients:

(Intercept)	disp
29.59985	-0.04122

If I had a new data source with displacement I could see the estimated miles per gallon.

```
set.seed(1234)
newdata <- sample(mtcars$disp, 5)
newdata
[1] 258.0 71.1 75.7 145.0 400.0

newdf <- data.frame(disp=newdata)
predict(my_mdl, newdf)
1 2 3 4 5
18.96635 26.66946 26.47987 23.62366 13.11381
```

The most important part of the process is to create a new data frame with the same column names as the original data. In this case, the original data had a column labeled `disp`, I was sure to call the new data that same name.

注意

让我们来看几个常见的陷阱：

- 在新对象中未使用data.frame：

```
predict(my_mdl, newdata)
Error in eval(predvars, data, env) :
  numeric 'envir' 参数长度不为一
```

- 在新数据框中不使用相同的名称：

```
newdf2 <- data.frame(newdata)
predict(my_mdl, newdf2)
错误 in eval(expr, envir, enclos) : 找不到对象 'disp'
```

准确率

要检查预测的准确性，您需要新数据的实际 y 值。在此示例中，newdf 需要包含 'mpg' 和 'disp' 两列。

```
newdf <- data.frame(mpg=mtcars$mpg[1:10], disp=mtcars$disp[1:10])
#   mpg   disp
# 1 21.0 160.0
# 2 21.0 160.0
# 3 22.8 108.0
# 4 21.4 258.0
# 5 18.7 360.0
# 6 18.1 225.0
# 7 14.3 360.0
# 8 24.4 146.7
# 9 22.8 140.8
# 10 19.2 167.6

p <- predict(my_mdl, newdf)

#均方根误差
sqrt(mean((p - newdf$mpg)^2, na.rm=TRUE))
[1] 2.325148
```

第22.3节：加权

有时我们希望模型对某些数据点或样本赋予比其他样本更高的权重。这可以通过在学习模型时为输入数据指定权重来实现。通常有两种情形我们可能会对样本使用非均匀权重：

- 分析权重：反映不同观测值的不同精度水平。例如，在分析数据时，如果每个观测值是某地理区域的平均结果，则分析权重与估计方差的倒数成正比。在处理数据中的平均值时非常有用，因为它根据观测次数提供了一个比例权重。来源
- 抽样权重（逆概率权重 - IPW）：一种统计技术，用于计算标准化到与数据收集时不同的人群的统计量。具有不同抽样人群和目标推断人群（目标人群）的研究设计在应用中很常见。处理存在缺失值的数据时非常有用。来源

Caution

Let's look at a few common pitfalls:

- not using a data.frame in the new object:

```
predict(my_mdl, newdata)
Error in eval(predvars, data, env) :
  numeric 'envir' arg not of length one
```

- not using same names in new data frame:

```
newdf2 <- data.frame(newdata)
predict(my_mdl, newdf2)
Error in eval(expr, envir, enclos) : object 'disp' not found
```

Accuracy

To check the accuracy of the prediction you will need the actual y values of the new data. In this example, newdf will need a column for 'mpg' and 'disp'.

```
newdf <- data.frame(mpg=mtcars$mpg[1:10], disp=mtcars$disp[1:10])
#   mpg   disp
# 1 21.0 160.0
# 2 21.0 160.0
# 3 22.8 108.0
# 4 21.4 258.0
# 5 18.7 360.0
# 6 18.1 225.0
# 7 14.3 360.0
# 8 24.4 146.7
# 9 22.8 140.8
# 10 19.2 167.6

p <- predict(my_mdl, newdf)

#root mean square error
sqrt(mean((p - newdf$mpg)^2, na.rm=TRUE))
[1] 2.325148
```

Section 22.3: Weighting

Sometimes we want the model to give more weight to some data points or examples than others. This is possible by specifying the weight for the input data while learning the model. There are generally two kinds of scenarios where we might use non-uniform weights over the examples:

- Analytic Weights: Reflect the different levels of precision of different observations. For example, if analyzing data where each observation is the average results from a geographic area, the analytic weight is proportional to the inverse of the estimated variance. Useful when dealing with averages in data by providing a proportional weight given the number of observations. [Source](#)
- Sampling Weights (Inverse Probability Weights - IPW): a statistical technique for calculating statistics standardized to a population different from that in which the data was collected. Study designs with a disparate sampling population and population of target inference (target population) are common in application. Useful when dealing with data that have missing values. [Source](#)

函数 lm() 进行分析加权。对于抽样权重，使用 survey 包构建调查设计对象并运行 svyglm()。默认情况下，survey 包使用抽样权重。（注意：lm() 和带有 family gaussian() 的 svyglm() 都会产生相同的点估计，因为它们都通过最小化加权最小二乘法来求解系数。它们的区别在于标准误差的计算方式。）

测试数据

```
data <- structure(list(lexptot = c(9.1595012302023, 9.86330744180814,
8.92372556833205, 8.58202430280175, 10.1133857229336), progvilm = c(1L,
1L, 1L, 1L, 0L), sexhead = c(1L, 1L, 0L, 1L, 1L), agehead = c(79L,
43L, 52L, 48L, 35L), weight = c(1.04273509979248, 1.01139605045319,
1.01139605045319, 1.01139605045319, 0.76305216550827)), .Names = c("lexptot",
"progvilm", "sexhead", "agehead", "weight"), class = c("tbl_df",
"tbl", "data.frame"), row.names = c(NA, -5L))
```

分析权重

```
lm.analytic <- lm(lexptot ~ progvilm + sexhead + agehead,
                  data = data, weight = weight)
summary(lm.analytic)
```

输出

调用:
`lm(公式 = lexptot ~ progvilm + sexhead + agehead, 数据 = data,
权重 = weight)`

加权残差:

1	2	3	4	5
9.249e-02	5.823e-01	0.000e+00	-6.762e-01	-1.527e-16

系数:

估计值	标准误差	t 值	Pr(> t)
(截距) 10.016054	1.744293	5.742	0.110
progvilm	-0.781204	1.344974	-0.581
sexhead	0.306742	1.040625	0.295
agehead	-0.005983	0.032024	-0.187
			0.882

残差标准误差: 0.8971 自由度 1

多重R平方: 0.467, 调整后R平方: -1.132

F统计量: 0.2921 自由度 3 和 1, p值: 0.8386

抽样权重 (IPW)

```
library(survey)
data$X <- 1:nrow(data) # 创建唯一ID

# 使用唯一ID、ipw和数据框构建调查设计对象
des1 <- svydesign(id = ~X, 权重 = ~weight, 数据 = data)

# 使用调查设计对象运行glm
prog.lm <- svyglm(lexptot ~ progvilm + sexhead + agehead, 设计=des1)
```

输出

调用:
`svyglm(公式 = lexptot ~ progvilm + sexhead + agehead, 设计 = des1)`

The `lm()` function does analytic weighting. For sampling weights the survey package is used to build a survey design object and run `svyglm()`. By default, the survey package uses sampling weights. (NOTE: `lm()`, and `svyglm()` with family `gaussian()` will all produce the same point estimates, because they both solve for the coefficients by minimizing the weighted least squares. They differ in how standard errors are calculated.)

Test Data

```
data <- structure(list(lexptot = c(9.1595012302023, 9.86330744180814,
8.92372556833205, 8.58202430280175, 10.1133857229336), progvilm = c(1L,
1L, 1L, 1L, 0L), sexhead = c(1L, 1L, 0L, 1L, 1L), agehead = c(79L,
43L, 52L, 48L, 35L), weight = c(1.04273509979248, 1.01139605045319,
1.01139605045319, 1.01139605045319, 0.76305216550827)), .Names = c("lexptot",
"progvilm", "sexhead", "agehead", "weight"), class = c("tbl_df",
"tbl", "data.frame"), row.names = c(NA, -5L))
```

Analytic Weights

```
lm.analytic <- lm(lexptot ~ progvilm + sexhead + agehead,
                  data = data, weight = weight)
summary(lm.analytic)
```

Output

Call:
`lm(formula = lexptot ~ progvilm + sexhead + agehead, data = data,
weights = weight)`

Weighted Residuals:

1	2	3	4	5
9.249e-02	5.823e-01	0.000e+00	-6.762e-01	-1.527e-16

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	10.016054	1.744293	5.742	0.110
progvilm	-0.781204	1.344974	-0.581	0.665
sexhead	0.306742	1.040625	0.295	0.818
agehead	-0.005983	0.032024	-0.187	0.882

Residual standard error: 0.8971 on 1 degrees of freedom
Multiple R-squared: 0.467, Adjusted R-squared: -1.132
F-statistic: 0.2921 on 3 and 1 DF, p-value: 0.8386

Sampling Weights (IPW)

```
library(survey)
data$X <- 1:nrow(data) # Create unique id

# Build survey design object with unique id, ipw, and data.frame
des1 <- svydesign(id = ~X, weights = ~weight, data = data)

# Run glm with survey design object
prog.lm <- svyglm(lexptot ~ progvilm + sexhead + agehead, design=des1)
```

Output

Call:
`svyglm(formula = lexptot ~ progvilm + sexhead + agehead, design = des1)`

```
调查设计:  
svydesign(id = ~X, weights = ~weight, data = data)
```

```
系数:  
估计值 标准误差 t 值 Pr(>|t|)  
(截距) 10.016054 0.183942 54.452 0.0117 *  
progvillm -0.781204 0.640372 -1.220 0.4371  
sexhead 0.306742 0.397089 0.772 0.5813  
agehead -0.005983 0.014747 -0.406 0.7546  
---  
显著性代码: 0 *** 0.001 ** 0.01 * 0.05 . 0.1 ' ' 1
```

(高斯族的离散参数取为 0.2078647)

Fisher评分迭代次数: 2

第22.4节：使用多项式回归检查非线性

有时在进行线性回归时，我们需要检查数据中的非线性。一种方法是拟合多项式模型，并检查其是否比线性模型更好地拟合数据。还有其他原因，比如理论上的考虑，表明应拟合二次或更高阶模型，因为变量之间的关系本质上是多项式的。

让我们为mtcars数据集拟合一个二次模型。线性模型请参见mtcars数据集上的线性回归。

首先，我们绘制变量mpg（每加仑英里数）、disp（排量（立方英寸））和wt（重量（1000磅））的散点图。变量mpg和disp之间的关系看起来是非线性的。

```
plot(mtcars[,c("mpg","disp","wt")])
```

```
Survey design:  
svydesign(id = ~X, weights = ~weight, data = data)
```

```
Coefficients:  
Estimate Std. Error t value Pr(>|t|)  
(Intercept) 10.016054 0.183942 54.452 0.0117 *  
progvillm -0.781204 0.640372 -1.220 0.4371  
sexhead 0.306742 0.397089 0.772 0.5813  
agehead -0.005983 0.014747 -0.406 0.7546  
---
```

Signif. codes: 0 *** 0.001 ** 0.01 * 0.05 . 0.1 ' ' 1

(Dispersion parameter for gaussian family taken to be 0.2078647)

Number of Fisher Scoring iterations: 2

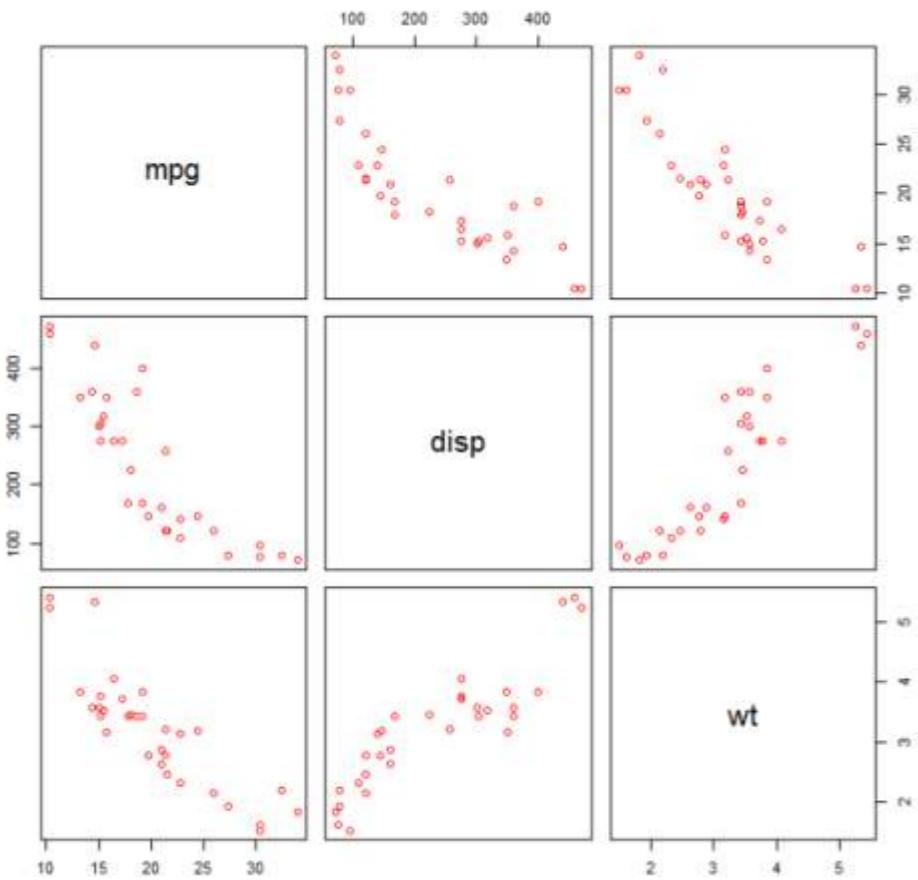
Section 22.4: Checking for nonlinearity with polynomial regression

Sometimes when working with linear regression we need to check for non-linearity in the data. One way to do this is to fit a polynomial model and check whether it fits the data better than a linear model. There are other reasons, such as theoretical, that indicate to fit a quadratic or higher order model because it is believed that the variables relationship is inherently polynomial in nature.

Let's fit a quadratic model for the `mtcars` dataset. For a linear model see Linear regression on the mtcars dataset.

First we make a scatter plot of the variables mpg (Miles/gallon), disp (Displacement (cu.in.)), and wt (Weight (1000 lbs)). The relationship among mpg and disp appears non-linear.

```
plot(mtcars[,c("mpg","disp","wt")])
```



线性拟合将显示 disp 不显著。

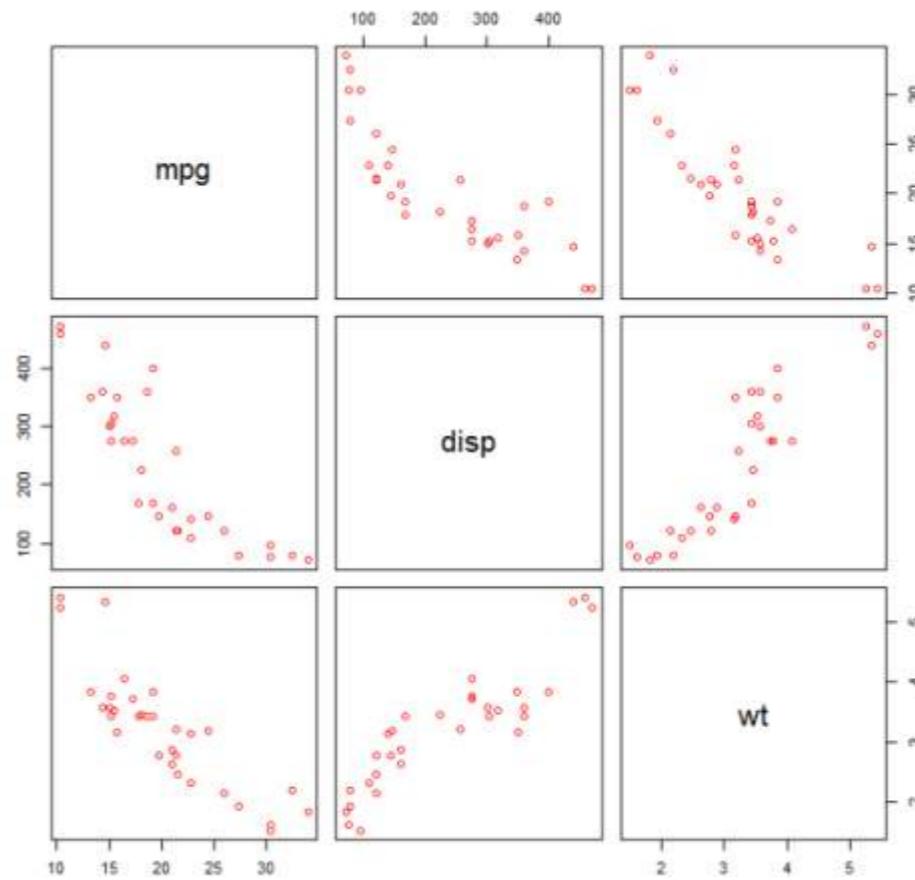
```
fit0 = lm(mpg ~ wt+disp, mtcars)
summary(fit0)

# 系数：
# 估计值 标准误差 t 值 Pr(>|t|)
#(截距) 34.96055 2.16454 16.151 4.91e-16 ***
#wt -3.35082 1.16413 -2.878 0.00743 **
#disp -0.01773 0.00919 -1.929 0.06362 .
#---
#显著性代码： 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ''
#残差标准误差：2.917，自由度29
#多重R平方： 0.7809，调整后的R平方： 0.7658
```

然后，为了得到二次模型的结果，我们添加了 `I(disp^2)`。观察时新模型表现更好 R^2 且所有变量均显著。

```
fit1 = lm(mpg ~ wt+disp+I(disp^2), mtcars)
summary(fit1)

# 系数：
# 估计值 标准误差 t 值 Pr(>|t|)
#(截距) 41.4019837 2.4266906 17.061 2.5e-16 ***
#wt -3.4179165 0.9545642 -3.581 0.001278 **
#disp -0.0823950 0.0182460 -4.516 0.000104 ***
#I(disp^2) 0.0001277 0.0000328 3.892 0.000561 ***
#---
#显著性代码： 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ''
#残差标准误差：2.391，自由度28
```



A linear fit will show that disp is not significant.

```
fit0 = lm(mpg ~ wt+disp, mtcars)
summary(fit0)

# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
#(Intercept) 34.96055 2.16454 16.151 4.91e-16 ***
#wt         -3.35082 1.16413 -2.878 0.00743 **
#disp        -0.01773 0.00919 -1.929 0.06362 .
#---
#Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ''
#Residual standard error: 2.917 on 29 degrees of freedom
#Multiple R-squared: 0.7809, Adjusted R-squared: 0.7658
```

Then, to get the result of a quadratic model, we added `I(disp^2)`. The new model appears better when looking at R^2 and all variables are significant.

```
fit1 = lm(mpg ~ wt+disp+I(disp^2), mtcars)
summary(fit1)

# Coefficients:
#             Estimate Std. Error t value Pr(>|t|)
#(Intercept) 41.4019837 2.4266906 17.061 2.5e-16 ***
#wt         -3.4179165 0.9545642 -3.581 0.001278 **
#disp        -0.0823950 0.0182460 -4.516 0.000104 ***
#I(disp^2) 0.0001277 0.0000328 3.892 0.000561 ***
#---
#Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ''
#Residual standard error: 2.391 on 28 degrees of freedom
```

```
#多重R平方 : 0.8578, 调整后的R平方 : 0.8426
```

由于我们有三个变量，拟合的模型是一个由以下公式表示的曲面：

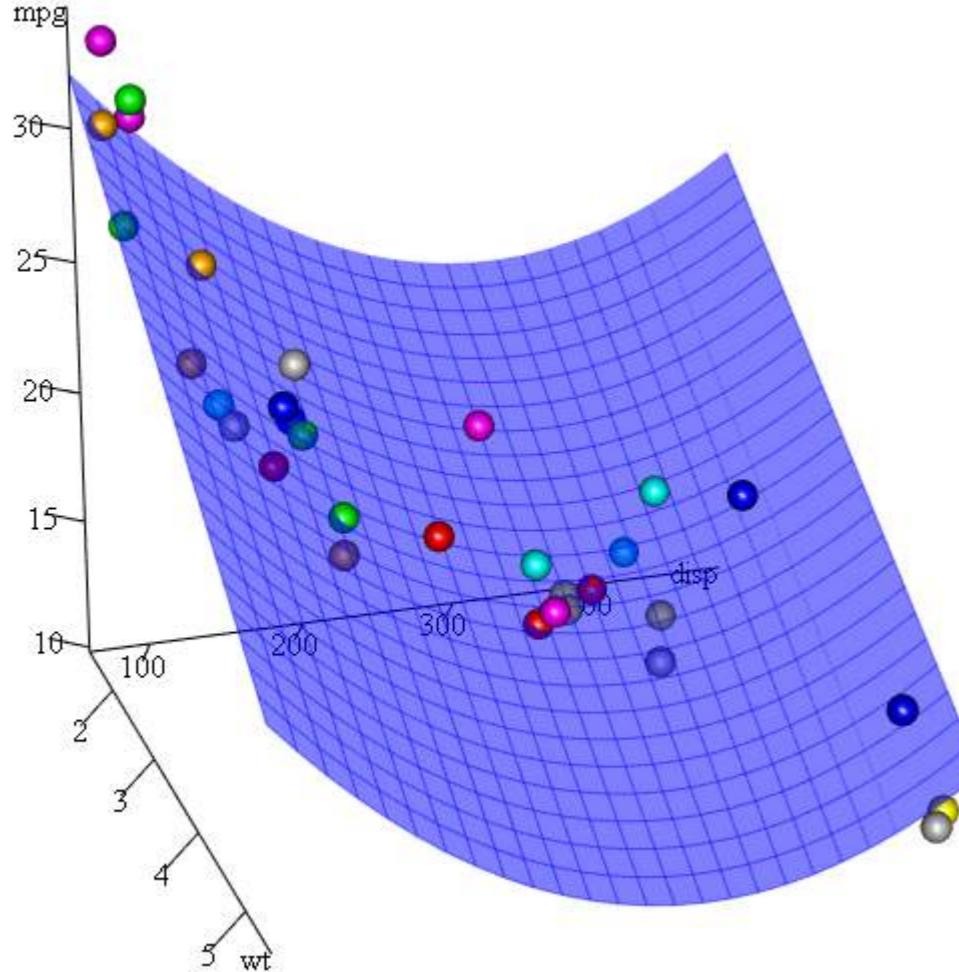
```
mpg = 41.4020-3.4179*wt-0.0824*disp+0.0001277*disp^2
```

指定多项式回归的另一种方法是使用**poly**，参数**raw=TRUE**，否则将考虑*orthogonal polynomials* (详见`help(poly)`)。我们使用以下代码得到相同的结果：

```
summary(lm(mpg ~ wt+poly(disp, 2, raw=TRUE),mtcars))
```

最后，如果我们需要展示估计曲面的图形怎么办？在R中有许多制作3D图的选项。这里我们使用来自p3d包的Fit3d。

```
library(p3d)
Init3d(family="serif", cex = 1)
Plot3d(mpg ~ disp+wt, mtcars)
Axes3d()
Fit3d(fit1)
```



第22.5节：绘制回归图（基础）

继续使用mtcars示例，这里有一种简单的方法来生成你的线性回归图，可能适合发表。

首先拟合线性模型，然后

```
#Multiple R-squared:  0.8578,   Adjusted R-squared:  0.8426
```

As we have three variables, the fitted model is a surface represented by:

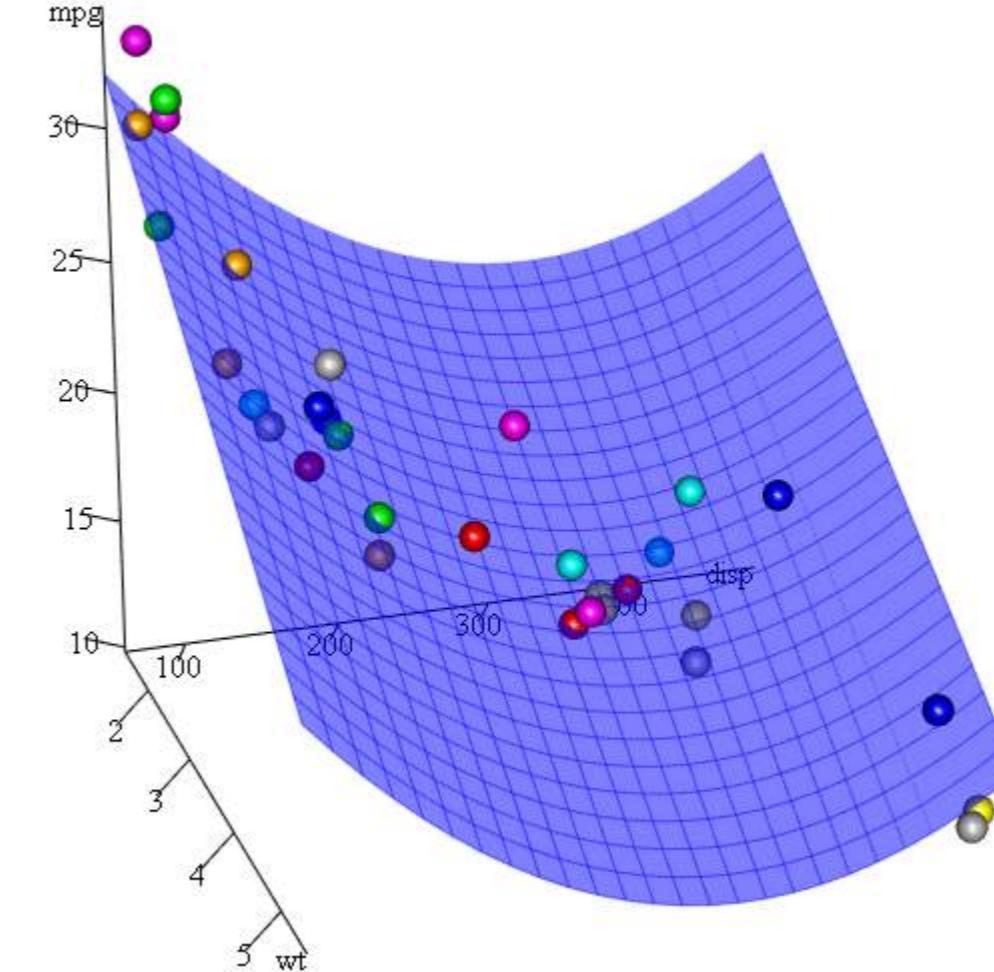
```
mpg = 41.4020-3.4179*wt-0.0824*disp+0.0001277*disp^2
```

Another way to specify polynomial regression is using **poly** with parameter **raw=TRUE**, otherwise *orthogonal polynomials* will be considered (see the `help(poly)` for more information). We get the same result using:

```
summary(lm(mpg ~ wt+poly(disp, 2, raw=TRUE),mtcars))
```

Finally, what if we need to show a plot of the estimated surface? Well there are many options to make 3D plots in R. Here we use Fit3d from p3dpackage.

```
library(p3d)
Init3d(family="serif", cex = 1)
Plot3d(mpg ~ disp+wt, mtcars)
Axes3d()
Fit3d(fit1)
```



Section 22.5: Plotting The Regression (base)

Continuing on the `mtcars` example, here is a simple way to produce a plot of your linear regression that is potentially suitable for publication.

First fit the linear model and

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

绘制两个感兴趣的变量，并在定义域内添加回归线：

```
plot(mtcars$wt, mtcars$mpg, pch=18, xlab = 'wt', ylab = 'mpg')
lines(c(min(mtcars$wt), max(mtcars$wt)),
as.numeric(predict(fit, data.frame(wt=c(min(mtcars$wt),max(mtcars$wt))))))
```

快完成了！最后一步是在图中添加回归方程、决定系数以及相关系数。这是通过vector函数完成的：

```
rp = vector('expression',3)
rp[1] = substitute(expression(italic(y) == MYOTHERVALUE3 + MYOTHERVALUE4 %%% x),
list(MYOTHERVALUE3 = format(fit$coefficients[1], digits = 2),
MYOTHERVALUE4 = format(fit$coefficients[2], digits = 2)))[2]
rp[2] = substitute(expression(italic(R)^2 == MYVALUE),
list(MYVALUE = format(summary(fit)$adj.r.squared, dig=3)))[2]
rp[3] = substitute(expression(Pearson-R == MYOTHERVALUE2),
list(MYOTHERVALUE2 = format(cor(mtcars$wt, mtcars$mpg), digits = 2)))[2]

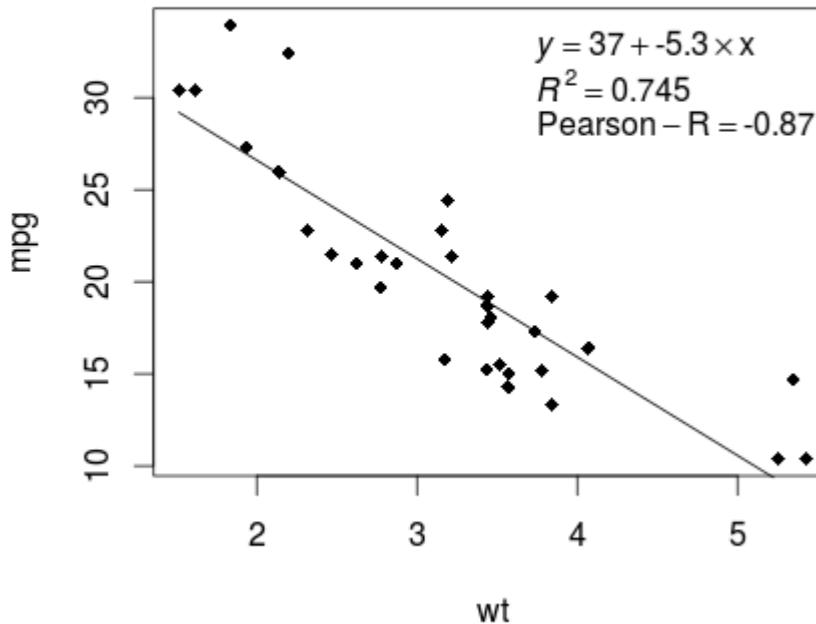
legend("topright", legend = rp, bty = 'n')
```

请注意，您可以通过调整向量函数添加任何其他参数，例如均方根误差（RMSE）。假设您想要一个包含10个元素的图例。向量定义如下：

```
rp = vector('expression',10)
```

并且您需要定义 r[1].... 到 r[10]

以下是输出结果：



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

Then plot the two variables of interest and add the regression line within the definition domain:

```
plot(mtcars$wt, mtcars$mpg, pch=18, xlab = 'wt', ylab = 'mpg')
lines(c(min(mtcars$wt), max(mtcars$wt)),
as.numeric(predict(fit, data.frame(wt=c(min(mtcars$wt),max(mtcars$wt))))))
```

Almost there! The last step is to add to the plot, the regression equation, the rsquare as well as the correlation coefficient. This is done using the **vector** function:

```
rp = vector('expression',3)
rp[1] = substitute(expression(italic(y) == MYOTHERVALUE3 + MYOTHERVALUE4 %%% x),
list(MYOTHERVALUE3 = format(fit$coefficients[1], digits = 2),
MYOTHERVALUE4 = format(fit$coefficients[2], digits = 2)))[2]
rp[2] = substitute(expression(italic(R)^2 == MYVALUE),
list(MYVALUE = format(summary(fit)$adj.r.squared, dig=3)))[2]
rp[3] = substitute(expression(Pearson-R == MYOTHERVALUE2),
list(MYOTHERVALUE2 = format(cor(mtcars$wt, mtcars$mpg), digits = 2)))[2]

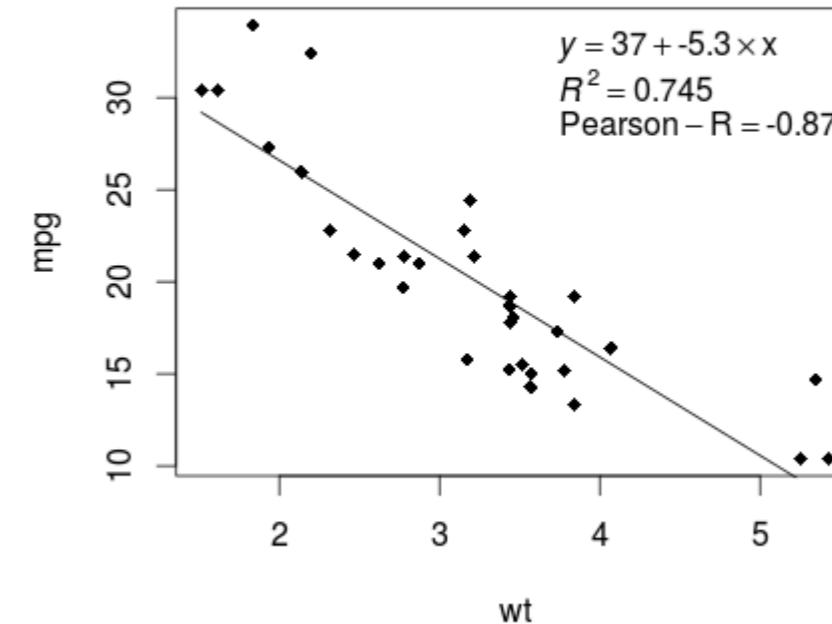
legend("topright", legend = rp, bty = 'n')
```

Note that you can add any other parameter such as the RMSE by adapting the vector function. Imagine you want a legend with 10 elements. The vector definition would be the following:

```
rp = vector('expression',10)
```

and you will need to defined r[1].... to r[10]

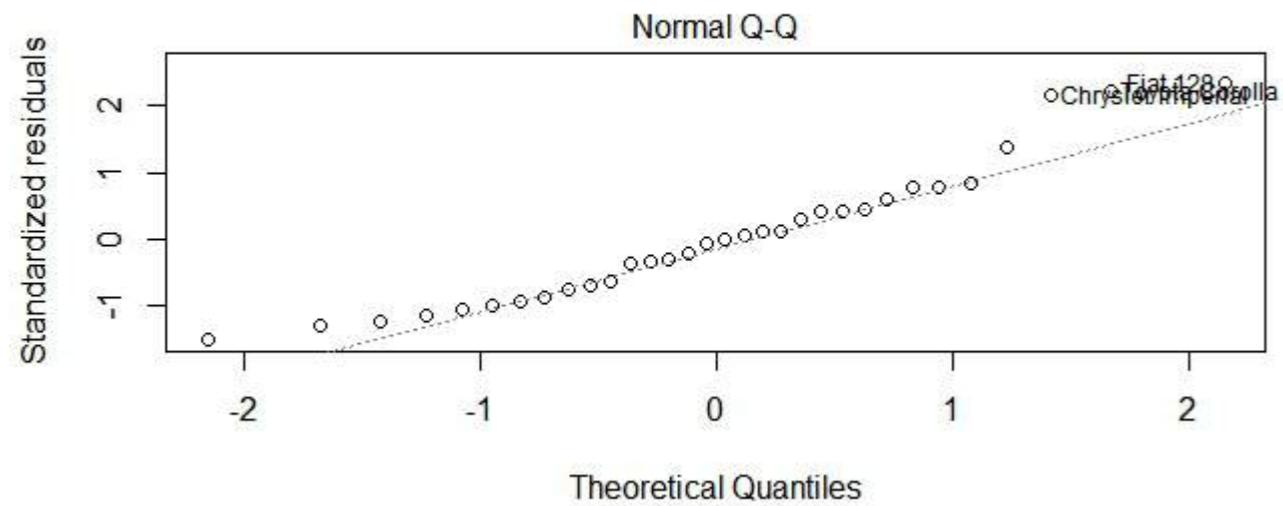
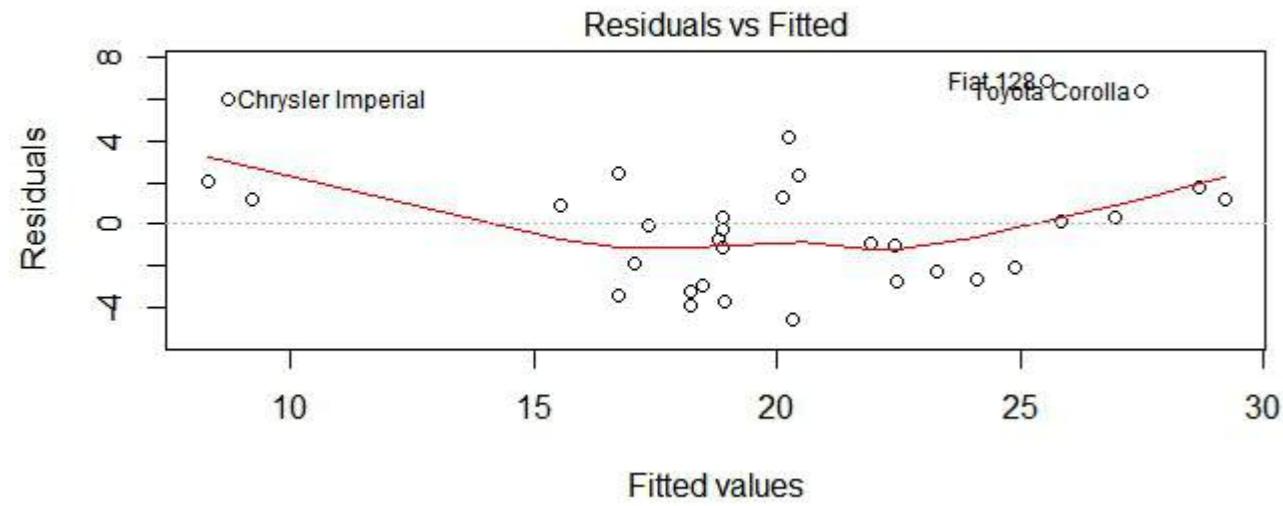
Here is the output:



第22.6节：质量评估

建立回归模型后，重要的是检查结果并判断模型是否合适且能很好地适应手头的数据。这可以通过检查残差图以及其他诊断图来完成。

```
# 拟合模型  
fit <- lm(mpg ~ wt, data = mtcars)  
#  
par(mfrow=c(2,1))  
# 绘制模型对象  
plot(fit, which =1:2)
```



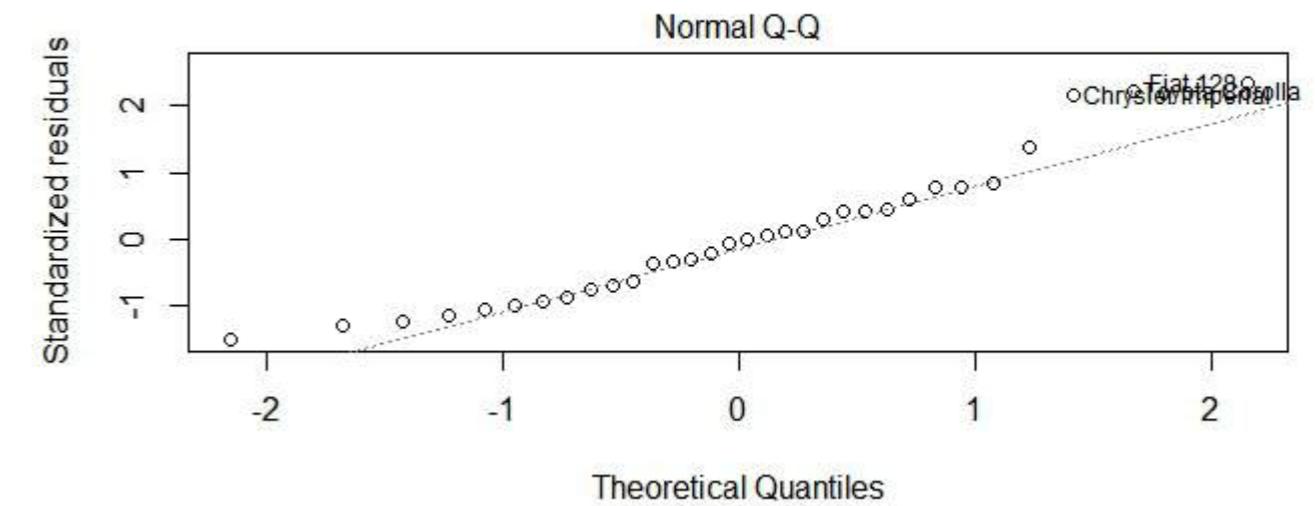
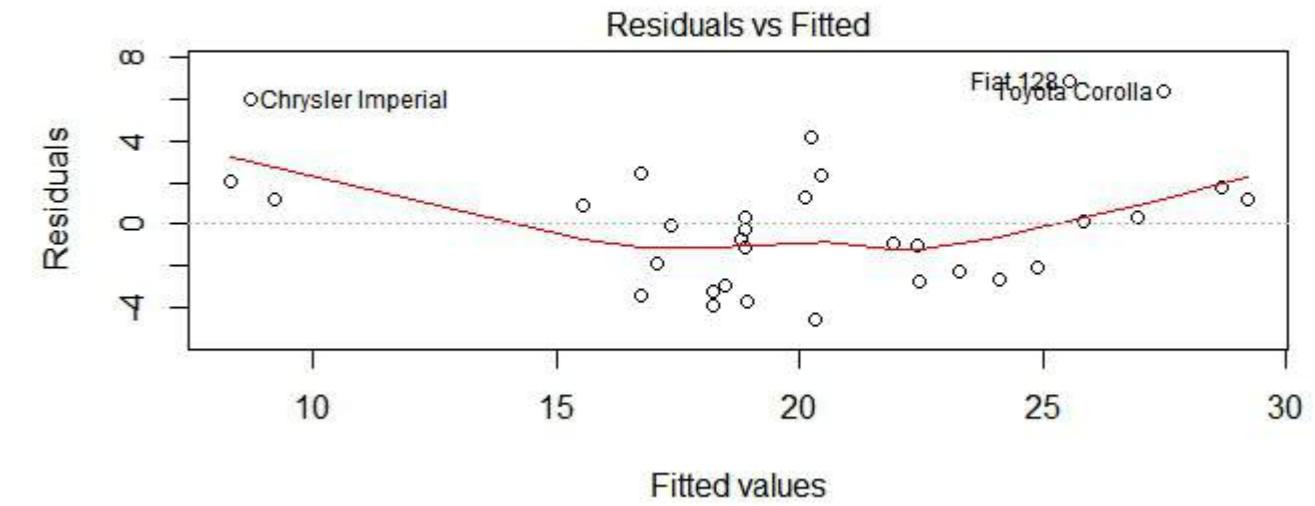
这些图检查了构建模型时所做的两个假设：

1. 预测变量的期望值（在本例中为 mpg）是由以下线性组合给出的预测变量（在本例中为wt）。我们期望该估计是无偏的。因此，残差应围绕所有预测变量的均值集中。在本例中，我们看到残差在两端倾向于正，中间为负，表明变量之间存在非线性关系。
2. 实际预测变量应围绕其估计值呈正态分布。因此，残差应为

Section 22.6: Quality assessment

After building a regression model it is important to check the result and decide if the model is appropriate and works well with the data at hand. This can be done by examining the residuals plot as well as other diagnostic plots.

```
# fit the model  
fit <- lm(mpg ~ wt, data = mtcars)  
#  
par(mfrow=c(2,1))  
# plot model object  
plot(fit, which =1:2)
```



These plots check for two assumptions that were made while building the model:

1. That the expected value of the predicted variable (in this case mpg) is given by a linear combination of the predictors (in this case wt). We expect this estimate to be unbiased. So the residuals should be centered around the mean for all values of the predictors. In this case we see that the residuals tend to be positive at the ends and negative in the middle, suggesting a non-linear relationship between the variables.
2. That the actual predicted variable is normally distributed around its estimate. Thus, the residuals should be

正态分布。对于正态分布的数据，正态Q-Q图中的点应位于对角线或其附近。这里两端存在一定程度的偏斜。

normally distributed. For normally distributed data, the points in a normal Q-Q plot should lie on or close to the diagonal. There is some amount of skew at the ends here.

第23章：data.table

Data.table是一个扩展了基础R中数据框功能的包，特别是在性能和语法上有所提升。详情请参见该包的文档区“Getting started with data.table”。

第23.1节：创建data.table

data.table是基础R中data.frame类的增强版本。因此，其class()属性是向量“data.table” “data.frame”，对data.frame有效的函数同样适用于data.table。有多种方法可以创建、加载或强制转换为data.table。

构建

别忘了安装并激活data.table包

```
library(data.table)
```

有一个同名的构造函数：

```
DT <- data.table(  
  x = letters[1:5],  
  y = 1:5,  
  z = (1:5) > 3  
)  
#   x y   z  
# 1: a 1 FALSE  
# 2: b 2 FALSE  
# 3: c 3 FALSE  
# 4: d 4  TRUE  
# 5: e 5  TRUE
```

与data.frame不同，data.table不会将字符串强制转换为因子：

```
sapply(DT, class)  
#           x     y     z  
# "character" "integer" "logical"
```

读取数据

我们可以从文本文件中读取数据：

```
dt <- fread("my_file.csv")
```

与read.csv不同，fread会将字符串读取为字符串，而不是因子。

修改数据框

为了提高效率，data.table 提供了一种方法，可以原地修改 data.frame 或列表，使其变成 data.table（无需复制或更改其内存位置）：

```
# 示例数据框  
DF <- data.frame(x = letters[1:5], y = 1:5, z = (1:5) > 3)  
# 修改  
setDT(DF)
```

请注意，我们不<-赋值结果，因为对象DF已被原地修改。该类的属性

Chapter 23: data.table

Data.table is a package that extends the functionality of data frames from base R, particularly improving on their performance and syntax. See the package's Docs area at Getting started with data.table for details.

Section 23.1: Creating a data.table

A data.table is an enhanced version of the data.frame class from base R. As such, its `class()` attribute is the vector “`data.table`” “`data.frame`” and functions that work on a data.frame will also work with a data.table. There are many ways to create, load or coerce to a data.table.

Build

Don't forget to install and activate the `data.table` package

```
library(data.table)
```

There is a constructor of the same name:

```
DT <- data.table(  
  x = letters[1:5],  
  y = 1:5,  
  z = (1:5) > 3  
)  
#   x y   z  
# 1: a 1 FALSE  
# 2: b 2 FALSE  
# 3: c 3 FALSE  
# 4: d 4  TRUE  
# 5: e 5  TRUE
```

Unlike `data.frame`, `data.table` will not coerce strings to factors:

```
sapply(DT, class)  
#           x     y     z  
# "character" "integer" "logical"
```

Read in

We can read from a text file:

```
dt <- fread("my_file.csv")
```

Unlike `read.csv`, `fread` will read strings as strings, not as factors.

Modify a data.frame

For efficiency, `data.table` offers a way of altering a `data.frame` or list to make a `data.table` in-place (without making a copy or changing its memory location):

```
# example data.frame  
DF <- data.frame(x = letters[1:5], y = 1:5, z = (1:5) > 3)  
# modification  
setDT(DF)
```

Note that we do not `<-` assign the result, since the object `DF` has been modified in-place. The class attributes of the

data.frame 将被保留：

```
sapply(DF, class)
#      x     y     z
# "factor" "integer" "logical"
```

强制将对象转换为data.table

如果你有一个list、data.frame或data.table，应该使用setDT函数将其转换为data.table因为它是通过引用进行转换，而不是像as.data.table那样复制。这在处理大型数据集时非常重要。

如果你有其他R对象（例如矩阵），必须使用as.data.table将其强制转换为data.table。

```
mat <- matrix(0, ncol = 10, nrow = 10)

DT <- as.data.table(mat)
# 或者
DT <- data.table(mat)
```

第23.2节：data.table中的特殊符号

.SD

.SD指的是每个分组的data.table子集，排除了所有在by中使用的列。

.SD结合lapply可以用于对data.table中每个分组的多列应用任意函数

我们将继续使用相同的内置数据集mtcars：

```
mtcars = data.table(mtcars) # 为了简化，不包含行名
```

按气缸数cyl计算数据集中所有列的均值：

```
mtcars[ , lapply(.SD, mean), by = cyl]

#   cyl      mpg      disp       hp      drat       wt      qsec       vs       am      gear
carb
#1:  6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714 0.5714286 0.4285714 3.857143
3.428571
#2:  4 26.66364 105.1364  82.63636 4.070909 2.285727 19.13727 0.9090909 0.7272727 4.090909
1.545455
#3:  8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214 0.0000000 0.1428571 3.285714
3.500000
```

除了cyl，数据集中还有其他类别型列，如vs、am、gear和carb。对这些列取均值实际上没有意义。因此我们排除这些列。这时.SDcols就派上用场了。

.SDcols

.SDcols指定包含在.SD中的data.table的列。

按齿轮数gear和气缸数cyl计算数据集中所有列（连续列）的均值，并按gear和cyl排序：

```
# 数据集中所有的连续变量
```

data.frame will be retained:

```
sapply(DF, class)
#      x     y     z
# "factor" "integer" "logical"
```

Coerce object to data.table

If you have a list, data.frame, or data.table, you should use the setDT function to convert to a data.table because it does the conversion by reference instead of making a copy (which as.data.table does). This is important if you are working with large datasets.

If you have another R object (such as a matrix), you must use as.data.table to coerce it to a data.table.

```
mat <- matrix(0, ncol = 10, nrow = 10)

DT <- as.data.table(mat)
# or
DT <- data.table(mat)
```

Section 23.2: Special symbols in data.table

.SD

.SD refers to the subset of the data.table for each group, excluding all columns used in by.

.SD along with lapply can be used to apply any function to multiple columns by group in a data.table

We will continue using the same built-in dataset, mtcars:

```
mtcars = data.table(mtcars) # Let's not include rownames to keep things simpler
```

Mean of all columns in the dataset by number of cylinders, cyl:

```
mtcars[ , lapply(.SD, mean), by = cyl]

#   cyl      mpg      disp       hp      drat       wt      qsec       vs       am      gear
carb
#1:  6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714 0.5714286 0.4285714 3.857143
3.428571
#2:  4 26.66364 105.1364  82.63636 4.070909 2.285727 19.13727 0.9090909 0.7272727 4.090909
1.545455
#3:  8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214 0.0000000 0.1428571 3.285714
3.500000
```

Apart from cyl, there are other categorical columns in the dataset such as vs, am, gear and carb. It doesn't really make sense to take the mean of these columns. So let's exclude these columns. This is where .SDcols comes into the picture.

.SDcols

.SDcols specifies the columns of the data.table that are included in .SD.

Mean of all columns (continuous columns) in the dataset by number of gears gear, and number of cylinders, cyl, arranged by gear and cyl:

```
# All the continuous variables in the dataset
```

```

cols_chosen <- c("mpg", "disp", "hp", "drat", "wt", "qsec")

mtcars[order(gear, cyl), lapply(.SD, mean), by = .(gear, cyl), .SDcols = cols_chosen]

#   gear cyl   mpg   disp     hp   drat      wt   qsec
#1:  3    4 21.500 120.1000 97.0000 3.700000 2.465000 20.0100
#2:  3    6 19.750 241.5000 107.5000 2.920000 3.337500 19.8300
#3:  3    8 15.050 357.6167 194.1667 3.120833 4.104083 17.1425
#4:  4    4 26.925 102.6250 76.0000 4.110000 2.378125 19.6125
#5:  4    6 19.750 163.8000 116.5000 3.910000 3.093750 17.6700
#6:  5    4 28.200 107.7000 102.0000 4.100000 1.826500 16.8000
#7:  5    6 19.700 145.0000 175.0000 3.620000 2.770000 15.5000
#8:  5    8 15.400 326.0000 299.5000 3.880000 3.370000 14.5500

```

也许我们不想按组计算mean。要计算数据集中所有汽车的平均值，我们不指定by变量。

```

mtcars[ , lapply(.SD, mean), .SDcols = cols_chosen]

#       mpg   disp     hp   drat      wt   qsec
#1: 20.09062 230.7219 146.6875 3.596563 3.21725 17.84875

```

注意：

- 事先定义cols_chosen并非必要。.SDcols可以直接接受列名。
- .SDcols也可以直接接受列号向量。在上述示例中，这将是mtcars[, lapply(.SD, mean), .SDcols = c(1,3:7)]

.N

.N是组内行数的简写。

```

iris[, .(count=.N), by=Species]

#       Species count
#1:      setosa     50
#2: versicolor     50
#3: virginica     50

```

第23.3节：添加和修改列

DT[where, select|update|do, by] 语法用于操作data.table的数据列。

- “where”部分是 i 参数
- “select|update|do”部分是 j 参数

这两个参数通常按位置传递，而不是按名称传递。

下面是我们的示例数据

```
mtcars = data.table(mtcars, keep.rownames = TRUE)
```

编辑整列

在 j 中使用 := 操作符来赋值新列：

```
mtcars[, mpg_sq := mpg^2]
```

```
cols_chosen <- c("mpg", "disp", "hp", "drat", "wt", "qsec")
```

```

mtcars[order(gear, cyl), lapply(.SD, mean), by = .(gear, cyl), .SDcols = cols_chosen]

#   gear cyl   mpg   disp     hp   drat      wt   qsec
#1:  3    4 21.500 120.1000 97.0000 3.700000 2.465000 20.0100
#2:  3    6 19.750 241.5000 107.5000 2.920000 3.337500 19.8300
#3:  3    8 15.050 357.6167 194.1667 3.120833 4.104083 17.1425
#4:  4    4 26.925 102.6250 76.0000 4.110000 2.378125 19.6125
#5:  4    6 19.750 163.8000 116.5000 3.910000 3.093750 17.6700
#6:  5    4 28.200 107.7000 102.0000 4.100000 1.826500 16.8000
#7:  5    6 19.700 145.0000 175.0000 3.620000 2.770000 15.5000
#8:  5    8 15.400 326.0000 299.5000 3.880000 3.370000 14.5500

```

Maybe we don't want to calculate the mean by groups. To calculate the mean for all the cars in the dataset, we don't specify the by variable.

```

mtcars[ , lapply(.SD, mean), .SDcols = cols_chosen]

#       mpg   disp     hp   drat      wt   qsec
#1: 20.09062 230.7219 146.6875 3.596563 3.21725 17.84875

```

Note:

- It is not necessary to define cols_chosen beforehand. .SDcols can directly take column names
- .SDcols can also directly take a vector of column numbers. In the above example this would be mtcars[, lapply(.SD, mean), .SDcols = c(1,3:7)]

.N

.N is shorthand for the number of rows in a group.

```

iris[, .(count=.N), by=Species]

#       Species count
#1:      setosa     50
#2: versicolor     50
#3: virginica     50

```

Section 23.3: Adding and modifying columns

DT[where, select|update|do, by] syntax is used to work with columns of a data.table.

- The “where” part is the i argument
- The “select|update|do” part is the j argument

These two arguments are usually passed by position instead of by name.

Our example data below is

```
mtcars = data.table(mtcars, keep.rownames = TRUE)
```

Editing entire columns

Use the := operator inside j to assign new columns:

```
mtcars[, mpg_sq := mpg^2]
```

通过设置为NULL来删除列：

```
mtcars[, mpg_sq := NULL]
```

使用:=操作符的多变量格式添加多列：

```
mtcars[, `:=`(mpg_sq = mpg^2, wt_sqrt = sqrt(wt))]  
# 或者  
mtcars[, c("mpg_sq", "wt_sqrt") := .(mpg^2, sqrt(wt))]
```

如果列之间有依赖关系且必须按顺序定义，一种方法是：

```
mtcars[, c("mpg_sq", "mpg2_hp") := .(temp1 <- mpg^2, temp1/hp)]
```

.()语法用于LHS := RHS右侧是列列表的情况。

对于动态确定的列名，使用括号：

```
vn = "mpg_sq"  
mtcars[, (vn) := mpg^2]
```

列也可以用set进行修改，尽管这很少有必要：

```
set(mtcars, j = "hp_over_wt", v = mtcars$hp/mtcars$wt)
```

编辑列的子集

使用 i 参数来子集化到“在哪些”行应该进行编辑：

```
mtcars[1:3, newvar := "Hello"]  
# 或者  
set(mtcars, j = "newvar", i = 1:3, v = "Hello")
```

和data.frame一样，我们可以使用行号或逻辑测试进行子集化。也可以在 i 中使用“连接”，但那个更复杂的任务在另一个示例中讲解。

编辑列属性

编辑属性的函数，如levels<-或names<-，实际上是用修改后的副本替换对象。即使只用于data.table中的一列，整个对象也会被复制并替换。

要在不复制对象的情况下修改对象，使用setnames来更改data.table或data.frame的列名，使用setattr来更改任何对象的属性。

```
# 每当 data.table 被复制时，向控制台打印一条消息  
tracemem(mtcars)  
mtcars[, cyl2 := factor(cyl)]  
  
# 这些语句都不会复制 data.table  
setnames(mtcars, old = "cyl2", new = "cyl_fac")  
setattr(mtcars$cyl_fac, "levels", c("four", "six", "eight"))  
  
# 这些语句都会复制 data.table  
names(mtcars)[names(mtcars) == "cyl_fac"] <- "cf"  
levels(mtcars$cf) <- c("IV", "VI", "VIII")
```

请注意，这些更改是通过引用进行的，因此它们是全局的。在一个环境中更改它们

Remove columns by setting to NULL:

```
mtcars[, mpg_sq := NULL]
```

Add multiple columns by using the := operator's multivariate format:

```
mtcars[, `:=`(mpg_sq = mpg^2, wt_sqrt = sqrt(wt))]  
# or  
mtcars[, c("mpg_sq", "wt_sqrt") := .(mpg^2, sqrt(wt))]
```

If the columns are dependent and must be defined in sequence, one way is:

```
mtcars[, c("mpg_sq", "mpg2_hp") := .(temp1 <- mpg^2, temp1/hp)]
```

The .() syntax is used when the right-hand side of LHS := RHS is a list of columns.

For dynamically-determined column names, use parentheses:

```
vn = "mpg_sq"  
mtcars[, (vn) := mpg^2]
```

Columns can also be modified with set, though this is rarely necessary:

```
set(mtcars, j = "hp_over_wt", v = mtcars$hp/mtcars$wt)
```

Editing subsets of columns

Use the i argument to subset to rows "where" edits should be made:

```
mtcars[1:3, newvar := "Hello"]  
# or  
set(mtcars, j = "newvar", i = 1:3, v = "Hello")
```

As in a data.frame, we can subset using row numbers or logical tests. It is also possible to use a "join" in i, but that more complicated task is covered in another example.

Editing column attributes

Functions that edit attributes, such as levels<- or names<-, actually replace an object with a modified copy. Even if only used on one column in a data.table, the entire object is copied and replaced.

To modify an object without copies, use setnames to change the column names of a data.table or data.frame and setattr to change an attribute for any object.

```
# Print a message to the console whenever the data.table is copied  
tracemem(mtcars)  
mtcars[, cyl2 := factor(cyl)]  
  
# Neither of these statements copy the data.table  
setnames(mtcars, old = "cyl2", new = "cyl_fac")  
setattr(mtcars$cyl_fac, "levels", c("four", "six", "eight"))  
  
# Each of these statements copies the data.table  
names(mtcars)[names(mtcars) == "cyl_fac"] <- "cf"  
levels(mtcars$cf) <- c("IV", "VI", "VIII")
```

Be aware that these changes are made by reference, so they are *global*. Changing them within one environment

会影响所有环境中的对象。

```
# 该函数也会更改全局环境中的 levels  
edit_levels <- function(x) setattr(x, "levels", c("low", "med", "high"))  
edit_levels(mtcars$cyl_factor)
```

第23.4节：编写兼容 data.frame 和 data.table 的代码

子集语法的差异

data.table 是 R 中几种二维数据结构之一，除了 data.frame、matrix 和 (二维) array 之外。所有这些类都使用非常相似但不完全相同的子集语法，即 A[行, 列] 模式。

考虑以下存储在 matrix、data.frame 和 data.table 中的数据：

```
ma <- matrix(rnorm(12), nrow=4, dimnames=list(letters[1:4], c('X', 'Y', 'Z')))  
df <- as.data.frame(ma)  
dt <- as.data.table(ma)
```

```
ma[2:3] #--> 返回第2和第3个元素，就像'ma'是一个向量（因为它确实是！）  
df[2:3] #--> 返回第2和第3列  
dt[2:3] #--> 返回第2和第3行！
```

如果你想确定返回的内容，最好明确指定。

要获取特定的行，只需在范围后加逗号：

```
ma[2:3, ] # \  
df[2:3, ] # }--> 返回第2和第3行  
dt[2:3, ] # /
```

但是，如果你想子集列，有些情况的解释会不同。所有三者都可以用整数或字符索引以相同方式子集，不存储在变量中。

```
ma[, 2:3] # \  
df[, 2:3] # \  
dt[, 2:3] # }--> 返回第2和第3列  
ma[, c("Y", "Z")] # /  
df[, c("Y", "Z")] # /  
dt[, c("Y", "Z")] # /
```

但是，对于未加引号的变量名，它们有所不同

```
mycols <- 2:3  
ma[, mycols] # \  
df[, mycols] # }--> 返回第2和第3列  
dt[, mycols, with = FALSE] # /  
  
dt[, mycols] # --> 报错
```

在最后一种情况下，mycols 被当作列名进行评估。因为 dt 找不到名为 mycols 的列，所以会引发错误。

注意：对于 data.table 包 1.9.8 之前的版本，这种行为略有不同。列索引中的任何内容都会使用 dt 作为环境进行评估。因此，dt[, 2:3] 和 dt[, mycols] 都会

affects the object in all environments.

```
# This function also changes the levels in the global environment  
edit_levels <- function(x) setattr(x, "levels", c("low", "med", "high"))  
edit_levels(mtcars$cyl_factor)
```

Section 23.4: Writing code compatible with both data.frame and data.table

Differences in subsetting syntax

A data.table is one of several two-dimensional data structures available in R, besides **data.frame**, **matrix** and (2D) **array**. All of these classes use a very similar but not identical syntax for subsetting, the A[rows, cols] schema.

Consider the following data stored in a **matrix**, a **data.frame** and a **data.table**:

```
ma <- matrix(rnorm(12), nrow=4, dimnames=list(letters[1:4], c('X', 'Y', 'Z')))  
df <- as.data.frame(ma)  
dt <- as.data.table(ma)
```

```
ma[2:3] #--> returns the 2nd and 3rd items, as if 'ma' were a vector (because it is!)  
df[2:3] #--> returns the 2nd and 3rd columns  
dt[2:3] #--> returns the 2nd and 3rd rows!
```

If you want to be sure of what will be returned, it is better to be *explicit*.

To get specific **rows**, just add a comma after the range:

```
ma[2:3, ] # \  
df[2:3, ] # }--> returns the 2nd and 3rd rows  
dt[2:3, ] # /
```

But, if you want to subset **columns**, some cases are interpreted differently. All three can be subset the same way with integer or character indices *not* stored in a variable.

```
ma[, 2:3] # \  
df[, 2:3] # \  
dt[, 2:3] # }--> returns the 2nd and 3rd columns  
ma[, c("Y", "Z")] # /  
df[, c("Y", "Z")] # /  
dt[, c("Y", "Z")] # /
```

However, they differ for unquoted variable names

```
mycols <- 2:3  
ma[, mycols] # \  
df[, mycols] # }--> returns the 2nd and 3rd columns  
dt[, mycols, with = FALSE] # /  
  
dt[, mycols] # --> Raises an error
```

In the last case, mycols is evaluated as the name of a column. Because dt cannot find a column named mycols, an error is raised.

Note: For versions of the data.table package prior to 1.9.8, this behavior was slightly different. Anything in the column index would have been evaluated using dt as an environment. So both **dt[, 2:3]** and **dt[, mycols]** would

返回向量 2:3。第二种情况不会引发错误，因为变量 mycols 确实存在于父环境中。

保持与 data.frame 和 data.table 兼容的策略

有很多理由编写保证同时适用于 data.frame 和 data.table 的代码。也许你被迫使用 data.frame，或者你需要共享一些你不知道将如何被使用的代码。因此，有一些主要策略可以实现这一点，按便利性排序如下：

1. 使用对两种类行为相同的语法。
2. 使用一个通用函数，执行与最简语法相同的操作。
3. 强制 data.table 表现得像 data.frame (例如：调用特定方法 print.data.frame)。
4. 把它们当作列表，因为它们最终就是列表。
5. 在进行任何操作之前，将表转换为 data.frame (如果表非常大，这不是一个好主意)。
6. 如果依赖关系不是问题，将表转换为 data.table。

子集行。很简单，只需使用 [,] 选择器，带逗号：

```
A[1:10, ]
A[A$var > 17, ] # A[var > 17, ] 对于 data.table 来说就是这样用的
```

子集列。如果你想要单列，使用 \$ 或 [[]] 选择器：

```
A$var
colname <- 'var'
A[[colname]]
A[[1]]
```

如果你想用统一的方式获取多列，就需要稍微变通一下：

```
B <- ` [.data.frame`(A, 2:4)

# 我们可以给它一个更好的名字
select <- ` [.data.frame` 
B <- select(A, 2:4)
C <- select(A, c('foo', 'bar'))
```

子集“索引”行。虽然 data.frame 有 row.names，data.table 有其独特的 key 特性。最好的做法是完全避免使用 row.names，并在可能的情况下利用 data.table 中已有的优化。

```
B <- A[A$var != 0, ]
# 或者...
B <- with(A, A[var != 0, ]) # data.table 会在子集操作前默默地按 var 索引 A

stuff <- c('a', 'c', 'f')
C <- A[match(stuff, A$name), ] # 这实际上比 setkey(A); A[stuff, ] 差很多
```

获取一列表，获取一行作为向量。这些用到目前为止学到的方法都很简单：

```
B <- select(A, 2)    #--> 仅包含第二列的表
C <- unlist(A[1, ]) #--> 第一行作为向量 (如有必要则强制转换)
```

return the vector 2:3. No error would be raised for the second case, because the variable mycols does exist in the parent environment.

Strategies for maintaining compatibility with data.frame and data.table

There are many reasons to write code that is guaranteed to work with data.frame and data.table. Maybe you are forced to use data.frame, or you may need to share some code that you don't know how will be used. So, there are some main strategies for achieving this, in order of convenience:

1. Use syntax that behaves the same for both classes.
2. Use a common function that does the same thing as the shortest syntax.
3. Force data.table to behave as data.frame (ex.: call the specific method print.data.frame).
4. Treat them as list, which they ultimately are.
5. Convert the table to a data.frame before doing anything (bad idea if it is a huge table).
6. Convert the table to data.table, if dependencies are not a concern.

Subset rows. Its simple, just use the [,] selector, with the comma:

```
A[1:10, ]
A[A$var > 17, ] # A[var > 17, ] just works for data.table
```

Subset columns. If you want a single column, use the \$ or the [[]] selector:

```
A$var
colname <- 'var'
A[[colname]]
A[[1]]
```

If you want a uniform way to grab more than one column, it's necessary to appeal a bit:

```
B <- ` [.data.frame`(A, 2:4)

# We can give it a better name
select <- ` [.data.frame` 
B <- select(A, 2:4)
C <- select(A, c('foo', 'bar'))
```

Subset 'indexed' rows. While data.frame has row.names, data.table has its unique key feature. The best thing is to avoid row.names entirely and take advantage of the existing optimizations in the case of data.table when possible.

```
B <- A[A$var != 0, ]
# or...
B <- with(A, A[var != 0, ]) # data.table will silently index A by var before subsetting

stuff <- c('a', 'c', 'f')
C <- A[match(stuff, A$name), ] # really worse than: setkey(A); A[stuff, ]
```

Get a 1-column table, get a row as a vector. These are easy with what we have seen until now:

```
B <- select(A, 2)    #--> a table with just the second column
C <- unlist(A[1, ]) #--> the first row as a vector (coerced if necessary)
```

第23.5节：在data.table中设置键

是的，1.9.6之前需要使用SETKEY

过去（1.9.6之前），通过将列设置为表的键，特别是对于大型表，可以加快data.table的速度。[参见2015年9月版本的intro vignette第5页，其中搜索速度提高了544倍。]你可能会发现旧代码使用'setkey'设置键，或者在设置表时使用'key='列。

```
library(data.table)
DT <- data.table(
  x = letters[1:5],
  y = 5:1,
  z = (1:5) > 3
)

#> DT
#   x y     z
#1: a 5 FALSE
#2: b 4 FALSE
#3: c 3 FALSE
#4: d 2  TRUE
#5: e 1  TRUE
```

使用setkey命令设置键。键可以包含多个列。

```
setkey(DT, y)
```

在tables()中检查表的键

```
tables()

> tables()
名称 行数 列数 内存 列 键
[1,] 数据表      5    3  1 x,y,z y
总计: 1MB
```

注意，这将重新排序您的数据。

```
#> DT
#   x y     z
#1: e 1  TRUE
#2: d 2  TRUE
#3: c 3 FALSE
#4: b 4 FALSE
#5: a 5 FALSE
```

现在不再需要

在v1.9.6之前，您必须为某些操作（尤其是连接表）设置键。data.table的开发者加快了速度，并引入了一个"on="功能，可以替代对键的依赖。详见SO答案这里的详细讨论。

2017年1月，开发者撰写了一个关于二级索引的说明文档，解释了"on"语法，并允许识别其他列以实现快速索引。

创建二级索引？

Section 23.5: Setting keys in data.table

Yes, you need to SETKEY pre 1.9.6

In the past (pre 1.9.6), your data.table was sped up by setting columns as keys to the table, particularly for large tables. [See [intro vignette page 5](#) of September 2015 version, where speed of search was 544 times better.] You may find older code making use of this setting keys with 'setkey' or setting a 'key=' column when setting up the table.

```
library(data.table)
DT <- data.table(
  x = letters[1:5],
  y = 5:1,
  z = (1:5) > 3
)

#> DT
#   x y     z
#1: a 5 FALSE
#2: b 4 FALSE
#3: c 3 FALSE
#4: d 2  TRUE
#5: e 1  TRUE
```

Set your key with the setkey command. You can have a key with multiple columns.

```
setkey(DT, y)
```

Check your table's key in tables()

```
tables()

> tables()
      NAME  NROW  NCOL  MB COLS  KEY
[1,] DT      5    3  1 x,y,z  y
Total: 1MB
```

Note this will re-sort your data.

```
#> DT
#   x y     z
#1: e 1  TRUE
#2: d 2  TRUE
#3: c 3 FALSE
#4: b 4 FALSE
#5: a 5 FALSE
```

Now it is unnecessary

Prior to v1.9.6 you had to have set a key for certain operations especially joining tables. The developers of data.table have sped up and introduced a "on=" feature that can replace the dependency on keys. See [SO answer here for a detailed discussion](#).

In Jan 2017, the developers have written a [vignette around secondary indices](#) which explains the "on" syntax and allows for other columns to be identified for fast indexing.

Creating secondary indices?

与 key 类似，您可以使用 `setindex(DT, key.col)` 或 `setindexv(DT, "key.col.string")`，其中 DT 是您的 `data.table`。使用 `setindex(DT, NULL)` 可以移除所有索引。

使用`indices(DT)`查看你的二级索引。

为什么使用二级索引？

这不会对表进行排序（不同于主键），但允许使用“on”语法进行快速索引。注意，表中只能有一个主键，但可以使用多个二级索引，这样就不必重新设置主键和重新排序表。这将加快你在更改要子集的列时的子集操作速度。

回想一下，上面的例子中y是表DT的主键：

```
DT
# x y z
# 1: e 1 TRUE
# 2: d 2 TRUE
# 3: c 3 FALSE
# 4: b 4 FALSE
# 5: a 5 FALSE

# 让我们将x设置为索引
setindex(DT, x)

# 使用indices查看已设置的索引
indices(DT)
# [1] "x"

# 使用索引而非主键列进行快速子集
DT["c", on = "x"]
#x y z
#1: c 3 FALSE

# 旧方法可能是将数据表的键从 y 重新设为 x，进行子集操作，
# 然后可能再将键设回 y (现在我们节省了两次排序)
# 上面是一个示例，但在大数据集上会更有价值
```

In a manner similar to key, you can `setindex(DT, key.col)` or `setindexv(DT, "key.col.string")`, where DT is your `data.table`. Remove all indices with `setindex(DT, NULL)`.

See your secondary indices with `indices(DT)`.

Why secondary indices?

This **does not sort** the table (unlike key), but does allow for quick indexing using the "on" syntax. Note there can be only one key, but you can use multiple secondary indices, which saves having to rekey and resort the table. This will speed up your subsetting when changing the columns you want to subset on.

Recall, in example above y was the key for table DT:

```
DT
# x y z
# 1: e 1 TRUE
# 2: d 2 TRUE
# 3: c 3 FALSE
# 4: b 4 FALSE
# 5: a 5 FALSE

# Let us set x as index
setindex(DT, x)

# Use indices to see what has been set
indices(DT)
# [1] "x"

# fast subset using index and not keyed column
DT[ "c", on = "x"]
#x y z
#1: c 3 FALSE

# old way would have been rekeying DT from y to x, doing subset and
# perhaps keying back to y (now we save two sorts)
# This is a toy example above but would have been more valuable with big data sets
```

第24章：使用

`data.table` 进行透视和反透视

参数

	详细信息
<code>id.vars</code>	告诉 <code>melt</code> 保留哪些列
<code>variable.name</code>	告诉 <code>melt</code> 如何命名包含类别标签的列
<code>value.name</code>	告诉 <code>melt</code> 如何命名包含与类别标签相关联的值的列
<code>value.var</code>	告诉 <code>dcast</code> 在哪些列中查找要转换的值
公式	告诉 <code>dcast</code> 哪些列保留以形成唯一记录标识符（左侧），哪些列包含类别标签（右侧）
<code>fun.aggregate</code>	指定在转换操作生成每个单元格的值列表时使用的函数

Chapter 24: Pivot and unpivot with `data.table`

Parameter

	Details
<code>id.vars</code>	tell <code>melt</code> which columns to retain
<code>variable.name</code>	tell <code>melt</code> what to call the column with category labels
<code>value.name</code>	tell <code>melt</code> what to call the column that has values associated with category labels
<code>value.var</code>	tell <code>dcast</code> where to find the values to cast in columns
<code>formula</code>	tell <code>dcast</code> which columns to retain to form a unique record identifier (LHS) and which one holds the category labels (RHS)
<code>fun.aggregate</code>	specify the function to use when the casting operation generates a list of values in each cell

Section 24.1: Pivot and unpivot tabular data with `data.table` - I

从宽格式转换为长格式

加载[数据USArrests](#)来自datasets。

```
data("USArrests")
head(USArrests)
```

	谋杀	攻击	城市人口	强奸
Alabama	13.2	236	58	21.2
Alaska	10.0	263	48	44.5
Arizona	8.1	294	80	31.0
Arkansas	8.8	190	50	19.5
California	9.0	276	91	40.6
Colorado	7.9	204	78	38.7

使用?USArrests了解更多。首先，转换为data.table。州名是原始数据框中的行名。

```
library(data.table)
DT <- as.data.table(USArrests, keep.rownames=TRUE)
```

这是宽格式数据。每个变量都有一列。数据也可以存储为长格式而不丢失信息。长格式有一列存储变量名，另一列存储变量值。USArrests 的长格式如下。

```
州    犯罪类型    发生率
1:    阿拉巴马州    谋杀    13.2
2:    阿拉斯加州    谋杀    10.0
3:    亚利桑那州    谋杀    8.1
4:    阿肯色州    谋杀    8.8
5:    加利福尼亚州    谋杀    9.0
---
196:   弗吉尼亚州    强奸    20.7
197:   华盛顿州    强奸    26.2
198: 西弗吉尼亚州    强奸    9.3
199:   威斯康星州    强奸    10.8
200:   怀俄明州    强奸    15.6
```

我们使用melt函数将数据从宽格式转换为长格式。

```
data("USArrests")
head(USArrests)
```

	Murder	Assault	UrbanPop	Rape
Alabama	13.2	236	58	21.2
Alaska	10.0	263	48	44.5
Arizona	8.1	294	80	31.0
Arkansas	8.8	190	50	19.5
California	9.0	276	91	40.6
Colorado	7.9	204	78	38.7

Use ?USArrests to find out more. First, convert to `data.table`. The names of states are row names in the original `data.frame`.

```
library(data.table)
DT <- as.data.table(USArrests, keep.rownames=TRUE)
```

This is data in the wide form. It has a column for each variable. The data can also be stored in long form without loss of information. The long form has one column that stores the variable names. Then, it has another column for the variable values. The long form of `USArrests` looks like so.

	State	Crime	Rate
1:	Alabama	Murder	13.2
2:	Alaska	Murder	10.0
3:	Arizona	Murder	8.1
4:	Arkansas	Murder	8.8
5:	California	Murder	9.0

196:	Virginia	Rape	20.7
197:	Washington	Rape	26.2
198:	West Virginia	Rape	9.3
199:	Wisconsin	Rape	10.8
200:	Wyoming	Rape	15.6

We use the `melt` function to switch from wide form to long form.

```
DTm <- melt(DT)
names(DTm) <- c("State", "Crime", "Rate")
```

默认情况下，melt 将所有包含数值数据的列视为具有数值的变量。在 USArests 数据集中，变量 UrbanPop 代表一个州的城市人口百分比。它不同于其他变量 Murder、Assault 和 Rape，这些是每10万人报告的暴力犯罪数量。假设我们想保留 UrbanPop 列。我们通过如下设置 id.vars 来实现这一点。

```
DTmu <- melt(DT, id.vars=c("rn", "UrbanPop"),
               variable.name='Crime', value.name = "Rate")
names(DTmu)[1] <- "State"
```

请注意，我们已经指定了包含类别名称（谋杀、袭击等）的列的名称 variable.name 和包含值的列 value.name。我们的数据如下所示。

	州	城市人口比例	犯罪率
1:	阿拉巴马州	58	谋杀 13.2
2:	阿拉斯加	48	谋杀 10.0
3:	亚利桑那州	80	谋杀 8.1
4:	阿肯色州	50	谋杀 8.8
5:	加利福尼亚州	91	谋杀 9.0

使用分割-应用-合并 (split-apply-combine) 风格的方法生成汇总非常简单。例如，要按州汇总暴力犯罪？

```
DTmu[, .(ViolentCrime = sum(Rate)), by=State]
```

结果如下：

	州	暴力犯罪
1:	阿拉巴马州	270.4
2:	阿拉斯加州	317.5
3:	亚利桑那州	333.1
4:	阿肯色州	218.3
5:	加利福尼亚	325.6
6:	科罗拉多	250.6

第24.2节：使用data.table进行数据表的旋转和反旋转 - II

从长格式转换为宽格式

要恢复前面的示例数据，使用 dcast 如下。

```
DTc <- dcast(DTmu, 州 + 城市人口 ~ 犯罪)
```

这将数据恢复为原始的宽格式。

	州	城市人口	谋杀	袭击	强奸
1:	阿拉巴马	58	13.2	236	21.2
2:	阿拉斯加	48	10.0	263	44.5
3:	亚利桑那	80	8.1	294	31.0
4:	阿肯色州	50	8.8	190	19.5
5:	加利福尼亚州	91	9.0	276	40.6

```
DTm <- melt(DT)
names(DTm) <- c("State", "Crime", "Rate")
```

By default, melt treats all columns with numeric data as variables with values. In [USArests](#), the variable UrbanPop represents the percentage urban population of a state. It is different from the other variables, Murder, Assault and Rape, which are violent crimes reported per 100,000 people. Suppose we want to retain UrbanPop column. We achieve this by setting id.vars as follows.

```
DTmu <- melt(DT, id.vars=c("rn", "UrbanPop"),
               variable.name='Crime', value.name = "Rate")
names(DTmu)[1] <- "State"
```

Note that we have specified the names of the column containing category names (Murder, Assault, etc.) with variable.name and the column containing the values with value.name. Our data looks like so.

	State	UrbanPop	Crime	Rate
1:	Alabama	58	Murder	13.2
2:	Alaska	48	Murder	10.0
3:	Arizona	80	Murder	8.1
4:	Arkansas	50	Murder	8.8
5:	California	91	Murder	9.0

Generating summaries with with split-apply-combine style approach is a breeze. For example, to summarize violent crimes by state?

```
DTmu[, .(ViolentCrime = sum(Rate)), by=State]
```

This gives:

	State	ViolentCrime
1:	Alabama	270.4
2:	Alaska	317.5
3:	Arizona	333.1
4:	Arkansas	218.3
5:	California	325.6
6:	Colorado	250.6

Section 24.2: Pivot and unpivot tabular data with data.table - II

Convert from long form to wide form

To recover data from the previous example, use dcast like so.

```
DTc <- dcast(DTmu, State + UrbanPop ~ Crime)
```

This gives the data in the original wide form.

	State	UrbanPop	Murder	Assault	Rape
1:	Alabama	58	13.2	236	21.2
2:	Alaska	48	10.0	263	44.5
3:	Arizona	80	8.1	294	31.0
4:	Arkansas	50	8.8	190	19.5
5:	California	91	9.0	276	40.6

这里，公式符号用于指定构成唯一记录标识符的列（左侧）和包含新列名称类别标签的列（右侧）。数值应使用哪一列？默认情况下，`dcast` 使用公式指定后剩余的第一列数值。若要明确指定，请使用参数`value.var`并指定列名。

当操作在每个单元格中产生值列表时，`dcast` 提供了一个`fun.aggregate`方法来处理这种情况。假设我在调查犯罪率时对城市人口相似的州感兴趣。我添加了一列`Decile`，包含计算信息。

```
DTmu[, Decile := cut(UrbanPop, quantile(UrbanPop, probs = seq(0, 1, by=0.1)))]
levels(DTmu$Decile) <- paste0(1:10, "D")
```

现在，使用`Decile ~ Crime`进行转换会在每个单元格产生多个值。我可以使用`fun.aggregate`来决定如何处理这些值。文本和数值都可以用这种方式处理。

```
dcast(DTmu, Decile ~ Crime, value.var="Rate", fun.aggregate=sum)
```

结果如下：

```
dcast(DTmu, 十分位数 ~ 犯罪, value.var="Rate", fun.aggregate=mean)
```

结果如下：

	州	城市人口	犯罪率	十分位数
1:	阿拉巴马	58	谋杀 13.2	4D
2:	阿拉斯加	48	谋杀 10.0	2D
3:	亚利桑那	80	谋杀 8.1	8D
4:	阿肯色	50	谋杀 8.8	2D
5:	加利福尼亚	91	谋杀 9.0	10D

每个城市人口的十分位数中包含多个州。使用`fun.aggregate`来指定这些数据应如何处理。

```
dcast(DTmu, Decile ~ Crime, value.var="Rate", fun.aggregate=sum)
```

这会对相似州的数据进行求和，得到如下结果。

	十分位数	谋杀	袭击	强奸
1:	1D	39.4	808	62.6
2:	2D	35.3	815	94.3
3:	3D	22.6	451	67.7
4:	4D	54.9	898	106.0
5:	5D	42.4	758	107.6

Here, the formula notation is used to specify the columns that form a unique record identifier (LHS) and the column containing category labels for new column names (RHS). Which column to use for the numeric values? By default, `dcast` uses the first column with numerical values left over when from the formula specification. To make explicit, use the parameter `value.var` with column name.

When the operation produces a list of values in each cell, `dcast` provides a `fun.aggregate` method to handle the situation. Say I am interested in states with similar urban population when investigating crime rates. I add a column `Decile` with computed information.

```
DTmu[, Decile := cut(UrbanPop, quantile(UrbanPop, probs = seq(0, 1, by=0.1)))]
levels(DTmu$Decile) <- paste0(1:10, "D")
```

Now, casting `Decile ~ Crime` produces multiple values per cell. I can use `fun.aggregate` to determine how these are handled. Both text and numerical values can be handle this way.

```
dcast(DTmu, Decile ~ Crime, value.var="Rate", fun.aggregate=sum)
```

This gives:

```
dcast(DTmu, Decile ~ Crime, value.var="Rate", fun.aggregate=mean)
```

This gives:

	State	UrbanPop	Crime	Rate	Decile
1:	Alabama	58	Murder	13.2	4D
2:	Alaska	48	Murder	10.0	2D
3:	Arizona	80	Murder	8.1	8D
4:	Arkansas	50	Murder	8.8	2D
5:	California	91	Murder	9.0	10D

There are multiple states in each decile of the urban population. Use `fun.aggregate` to specify how these should be handled.

```
dcast(DTmu, Decile ~ Crime, value.var="Rate", fun.aggregate=sum)
```

This sums over the data for like states, giving the following.

	Decile	Murder	Assault	Rape
1:	1D	39.4	808	62.6
2:	2D	35.3	815	94.3
3:	3D	22.6	451	67.7
4:	4D	54.9	898	106.0
5:	5D	42.4	758	107.6

第25章：条形图

条形图的目的是显示因子变量各水平的频率（或比例）。例如，条形图用于形象地显示不同社会经济（因子）组（水平-高、中、低）中个体的频率（或比例）。这样的图表有助于对各因子水平进行直观比较。

第25.1节：barplot()函数

在条形图中，因子水平放置在x轴上，各因子水平的频率（或比例）则考虑在y轴上。对于每个因子水平，构建一个宽度均匀、高度与因子水平频率（或比例）成正比的条形。

barplot()函数属于R系统库中的graphics包。barplot()函数至少需要一个参数。R帮助文档称该参数为heights，必须是向量或矩阵。如果是向量，其成员即为各个因子水平。

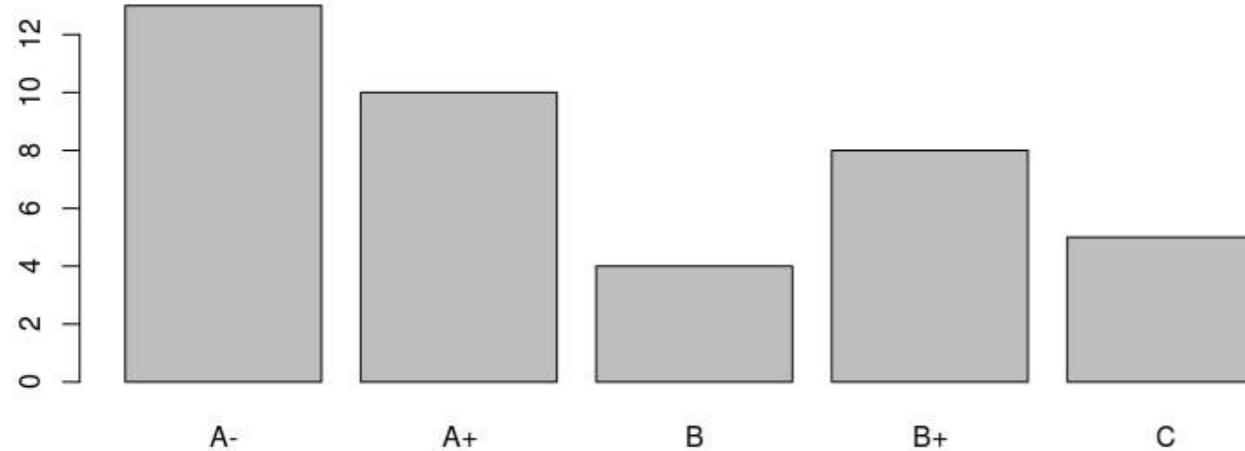
为了说明barplot()函数，考虑以下数据准备：

```
> 等级<-c("A+", "A-", "B+", "B", "C")
> 分数<-样本(等级,40,替换=T,概率=c(.2,.3,.25,.15,.1))
> 分数
[1] "A+" "A-" "B+" "A-" "A+" "B+" "A-" "B" "A+" "A-"
[13] "A-" "B+" "A-" "A-" "A-" "A+" "A-" "A+" "C" "C"
[25] "B" "C" "B+" "C" "B+" "B+" "A+" "B+" "A-" "A+"
[37] "A-" "B" "C" "A+
>
```

从中得到Marks向量的条形图

```
> barplot(table(Marks),main="算法中的中期成绩")
```

Mid-Marks in Algorithms



注意，barplot()函数将因子水平按字典顺序放置在x轴上。

使用参数 names.arg，可以按照向量grades中指定的顺序排列图中的条形。

Chapter 25: Bar Chart

The purpose of the bar plot is to display the frequencies (or proportions) of levels of a factor variable. For example, a bar plot is used to pictorially display the frequencies (or proportions) of individuals in various socio-economic (factor) groups(levels-high, middle, low). Such a plot will help to provide a visual comparison among the various factor levels.

Section 25.1: barplot() function

In barplot, factor-levels are placed on the x-axis and frequencies (or proportions) of various factor-levels are considered on the y-axis. For each factor-level one bar of uniform width with heights being proportional to factor level frequency (or proportion) is constructed.

The `barplot()` function is in the graphics package of the R's System Library. The `barplot()` function must be supplied at least one argument. The R help calls this as `heights`, which must be either vector or a matrix. If it is vector, its members are the various factor-levels.

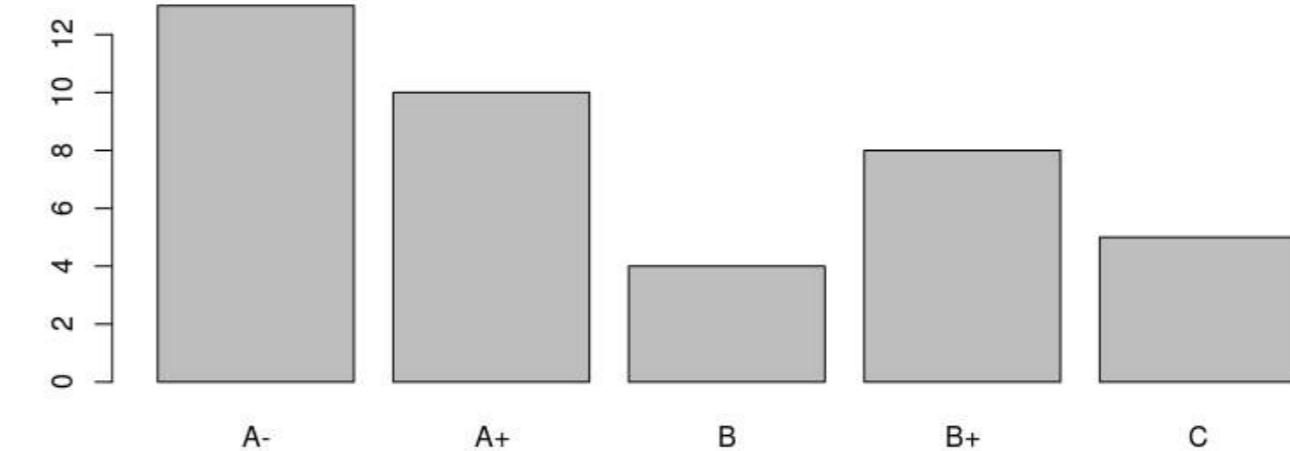
To illustrate `barplot()`, consider the following data preparation:

```
> grades<-c("A+", "A-", "B+", "B", "C")
> Marks<-sample(grades,40,replace=T,prob=c(.2,.3,.25,.15,.1))
> Marks
[1] "A+" "A-" "B+" "A-" "A+" "B+" "A-" "B" "A+" "A-"
[13] "A-" "B+" "A-" "A-" "A-" "A+" "A-" "A+" "A+" "C"
[25] "B" "C" "B+" "C" "B+" "B+" "A+" "B+" "A-" "A+"
[37] "A-" "B" "C" "A+
>
```

A bar chart of the Marks vector is obtained from

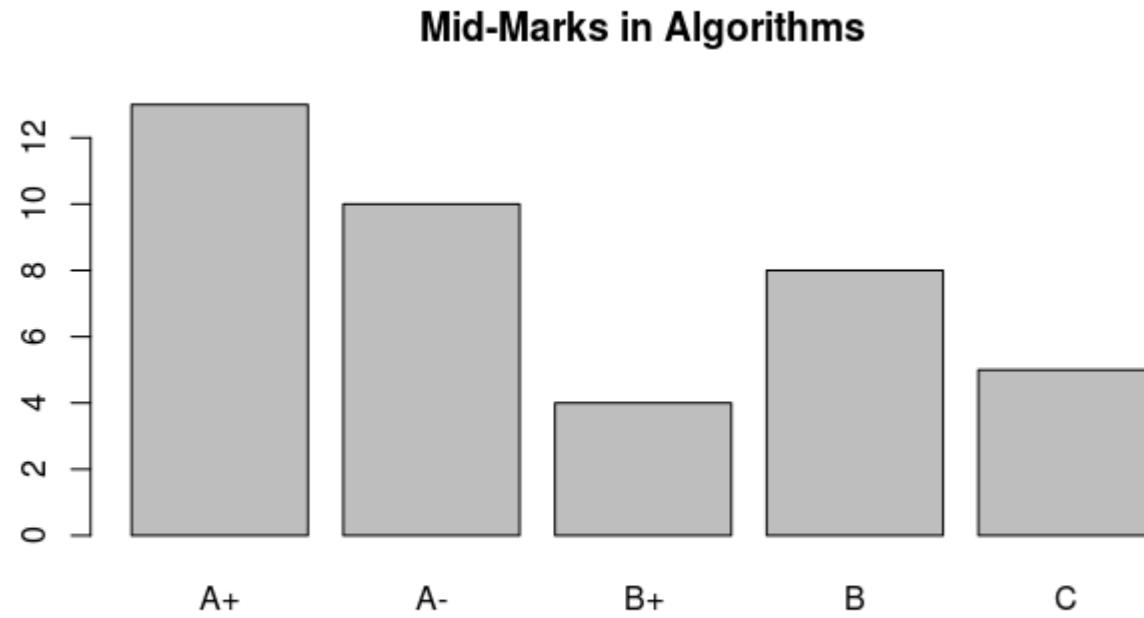
```
> barplot(table(Marks),main="Mid-Marks in Algorithms")
```

Mid-Marks in Algorithms

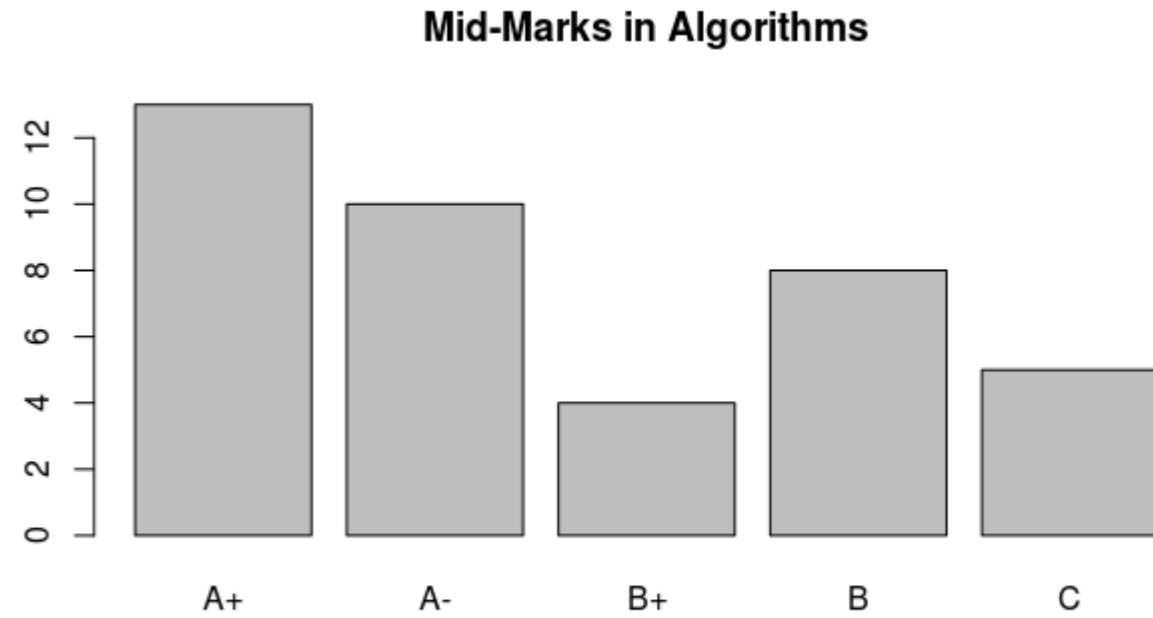


Notice that, the barplot() function places the factor levels on the x-axis in the lexicographical `order` of the levels. Using the parameter `names.arg`, the bars in plot can be placed in the order as stated in the vector, `grades`.

```
# 绘制所需的横轴标签
> barplot(table(Marks),names.arg=grades ,main="算法中的中期成绩")
```

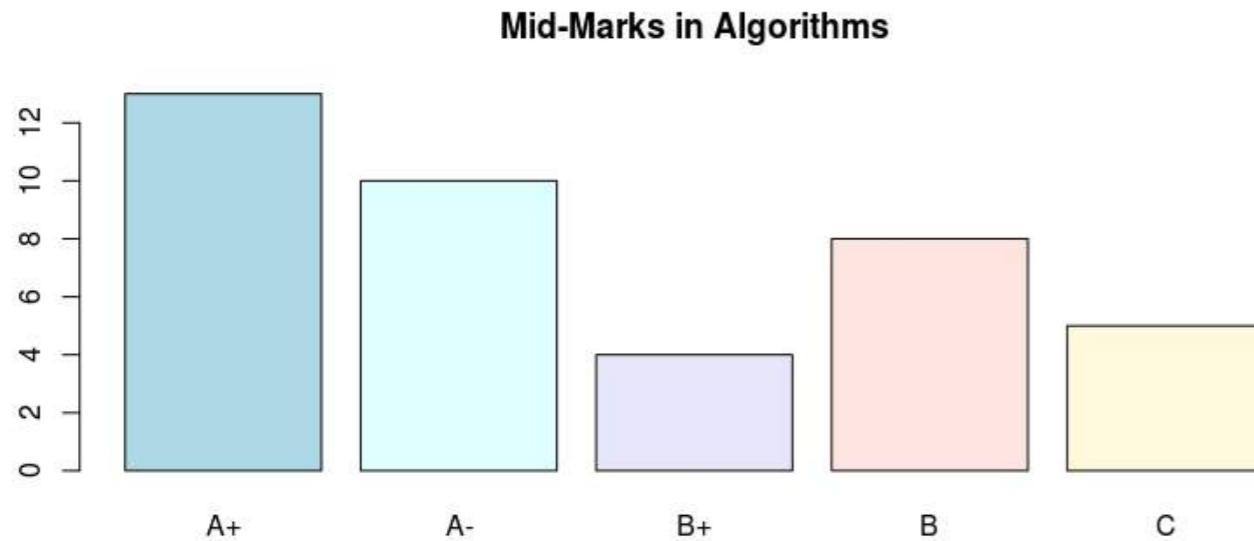


```
# plot to the desired horizontal axis labels
> barplot(table(Marks),names.arg=grades ,main="Mid-Marks in Algorithms")
```



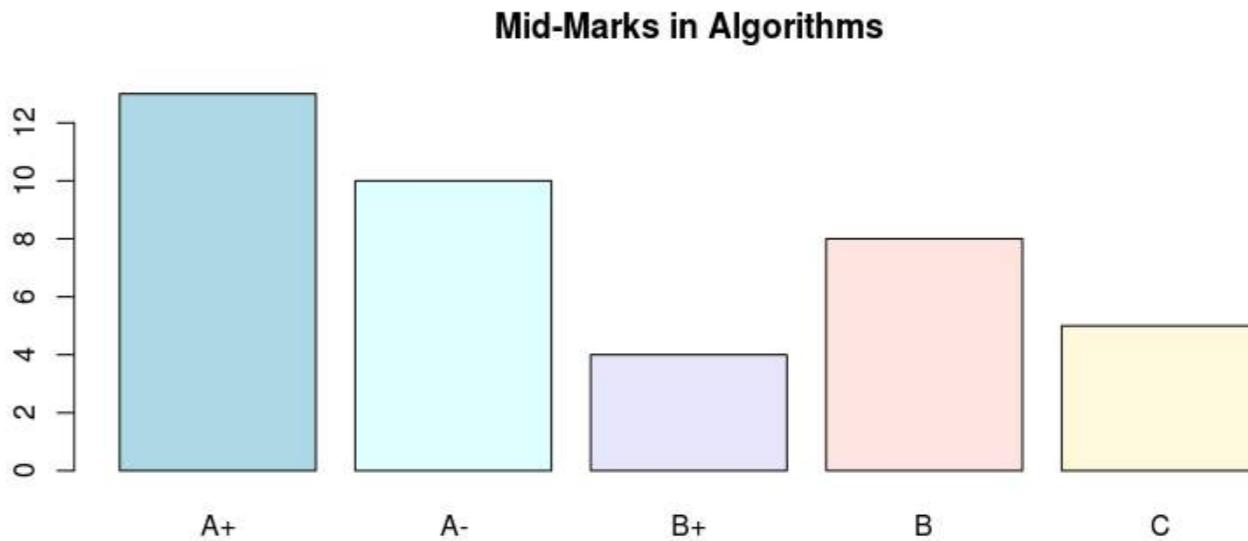
可以使用参数 `col=` 绘制彩色条形。

```
> barplot(table(Marks),names.arg=grades,col = c("lightblue",
  "lightcyan", "lavender", "mistyrose", "cornsilk"),
  main="算法中的中期成绩")
```



Colored bars can be drawn using the `col=` parameter.

```
> barplot(table(Marks),names.arg=grades,col = c("lightblue",
  "lightcyan", "lavender", "mistyrose", "cornsilk"),
  main="Mid-Marks in Algorithms")
```



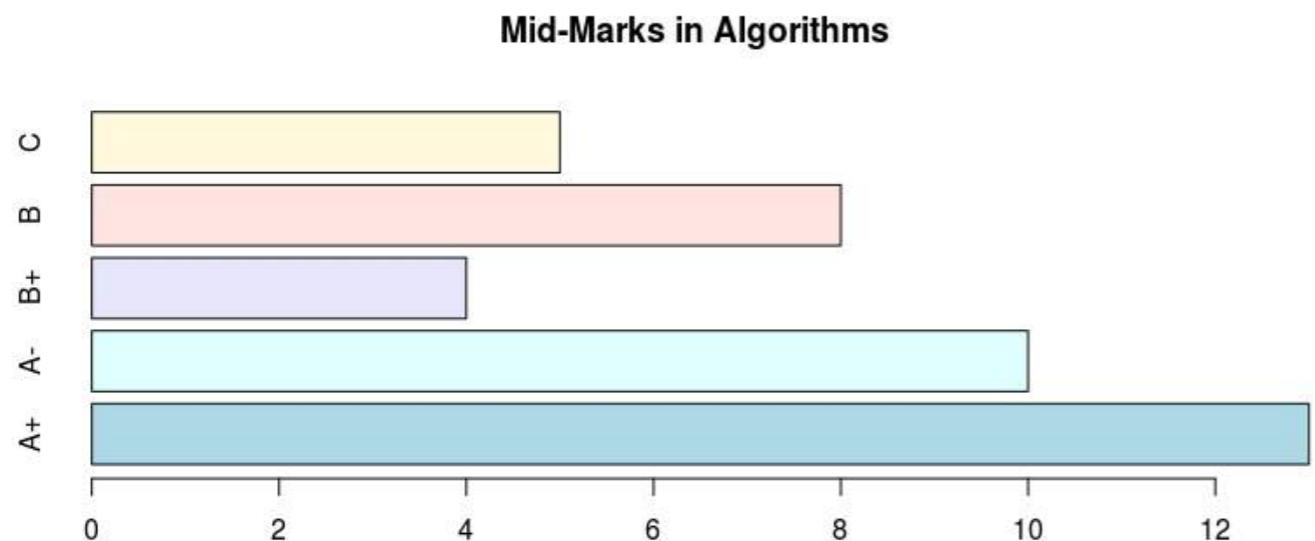
可以按如下方式获得带有水平条形的条形图：

```
> barplot(table(Marks),names.arg=grades,horiz=TRUE,col = c("浅蓝色",
  "浅青色", "薰衣草色", "薄玫瑰色", "玉米丝色"),
```

A bar chart with *horizontal bars* can be obtained as follows:

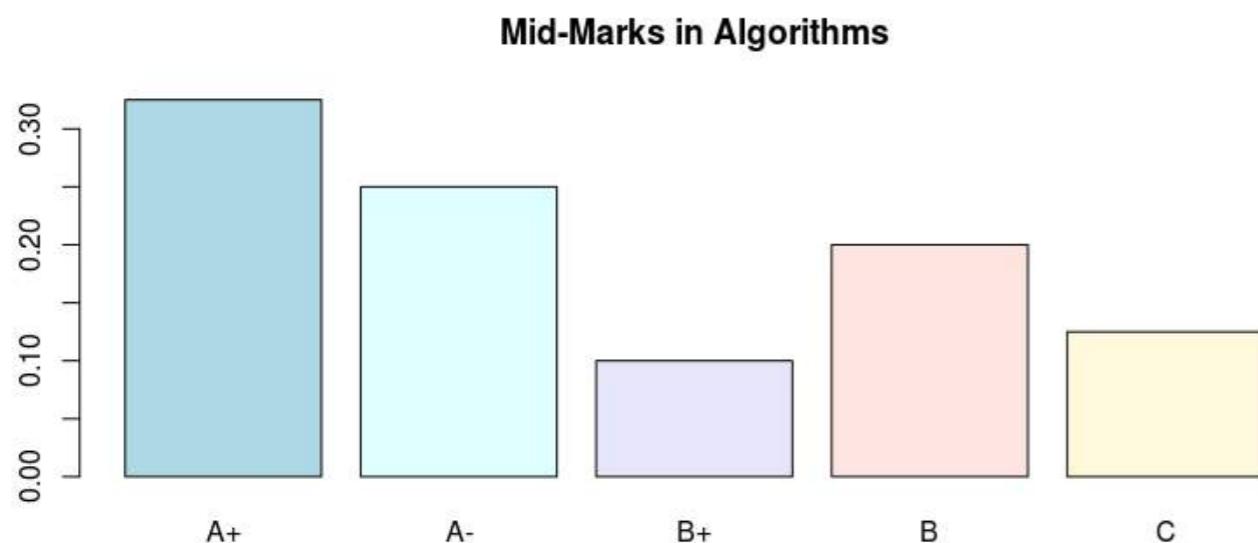
```
> barplot(table(Marks),names.arg=grades,horiz=TRUE,col = c("lightblue",
  "lightcyan", "lavender", "mistyrose", "cornsilk"),
```

```
main="算法中的期中成绩")
```



可以通过以下方式获得y轴为比例的条形图：

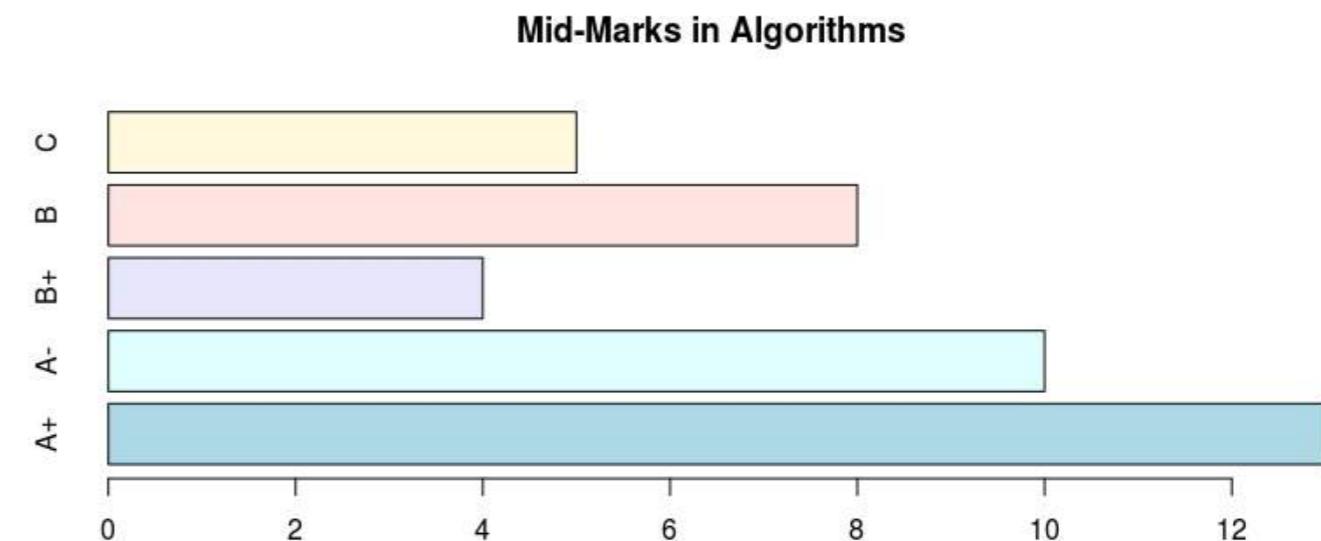
```
> barplot(prop.table(table(Marks)),names.arg=grades,col = c("浅蓝色",
  "浅青色", "薰衣草色", "薄玫瑰色", "玉米丝色"),
main="算法中的期中成绩")
```



可以使用cex.names参数来增大x轴上因子水平名称的大小。

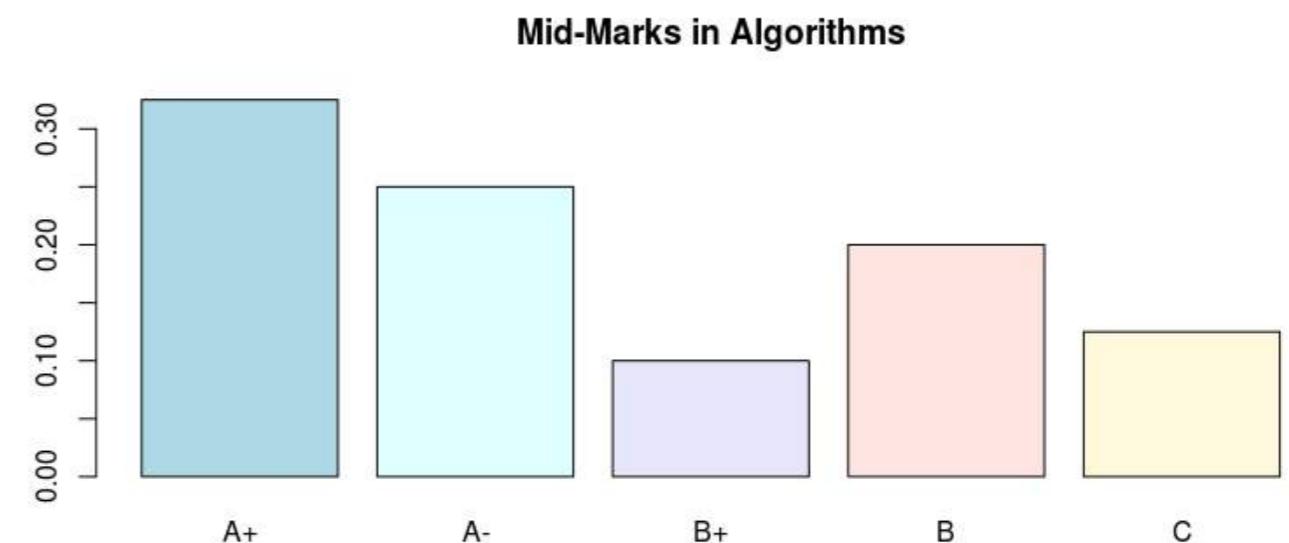
```
> barplot(prop.table(table(Marks)),names.arg=grades,col = c("浅蓝色",
  "浅青色", "薰衣草色", "薄玫瑰色", "玉米丝色"),
main="算法中的期中成绩",cex.names=2)
```

```
main="Mid-Marks in Algorithms")
```



A bar chart with *proportions* on the y-axis can be obtained as follows:

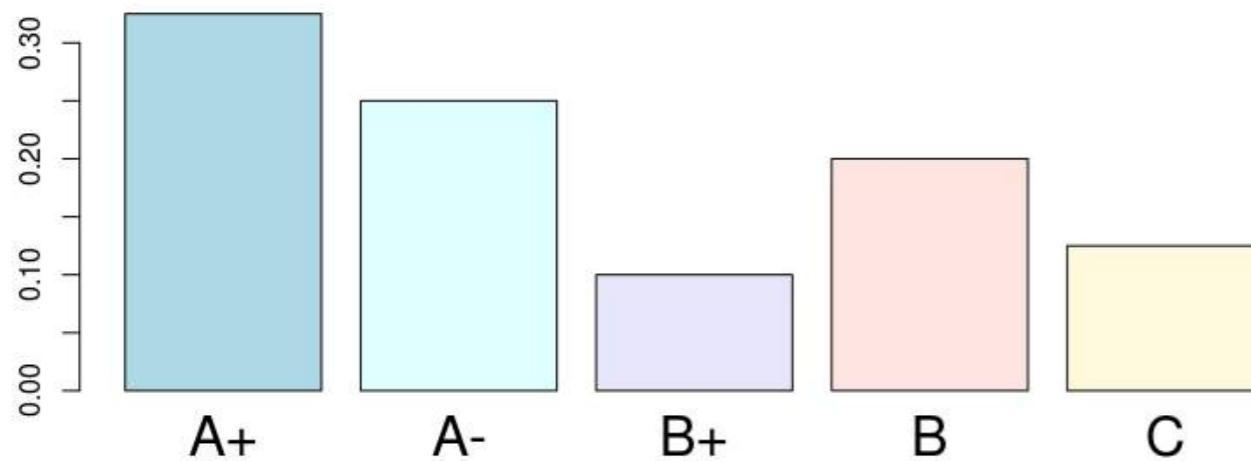
```
> barplot(prop.table(table(Marks)),names.arg=grades,col = c("lightblue",
  "lightcyan", "lavender", "mistyrose", "cornsilk"),
main="Mid-Marks in Algorithms")
```



The sizes of the factor-level names on the x-axis can be increased using cex.names parameter.

```
> barplot(prop.table(table(Marks)),names.arg=grades,col = c("lightblue",
  "lightcyan", "lavender", "mistyrose", "cornsilk"),
main="Mid-Marks in Algorithms",cex.names=2)
```

Mid-Marks in Algorithms



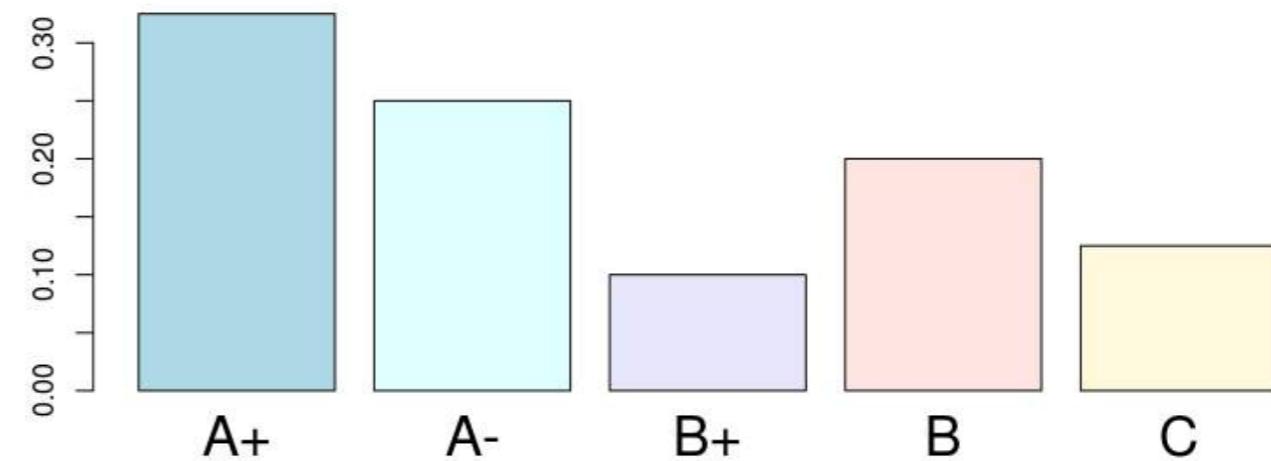
`barplot()`的`heights`参数可以是一个矩阵。例如，它可以是一个矩阵，其中列表示课程中选修的各个科目，行表示成绩的标签。考虑以下矩阵：

```
> gradTab
算法 操作系统 离散数学
A-      13      10      7
A+      10       7      2
B        4       2     14
B+      8      19     12
C        5       2      5
```

要绘制堆积条形图，只需使用以下命令：

```
> barplot(gradTab,col = c("lightblue","lightcyan",
  "lavender", "mistyrose", "cornsilk"),legend.text = grades,
  main="算法中的期中成绩")
```

Mid-Marks in Algorithms

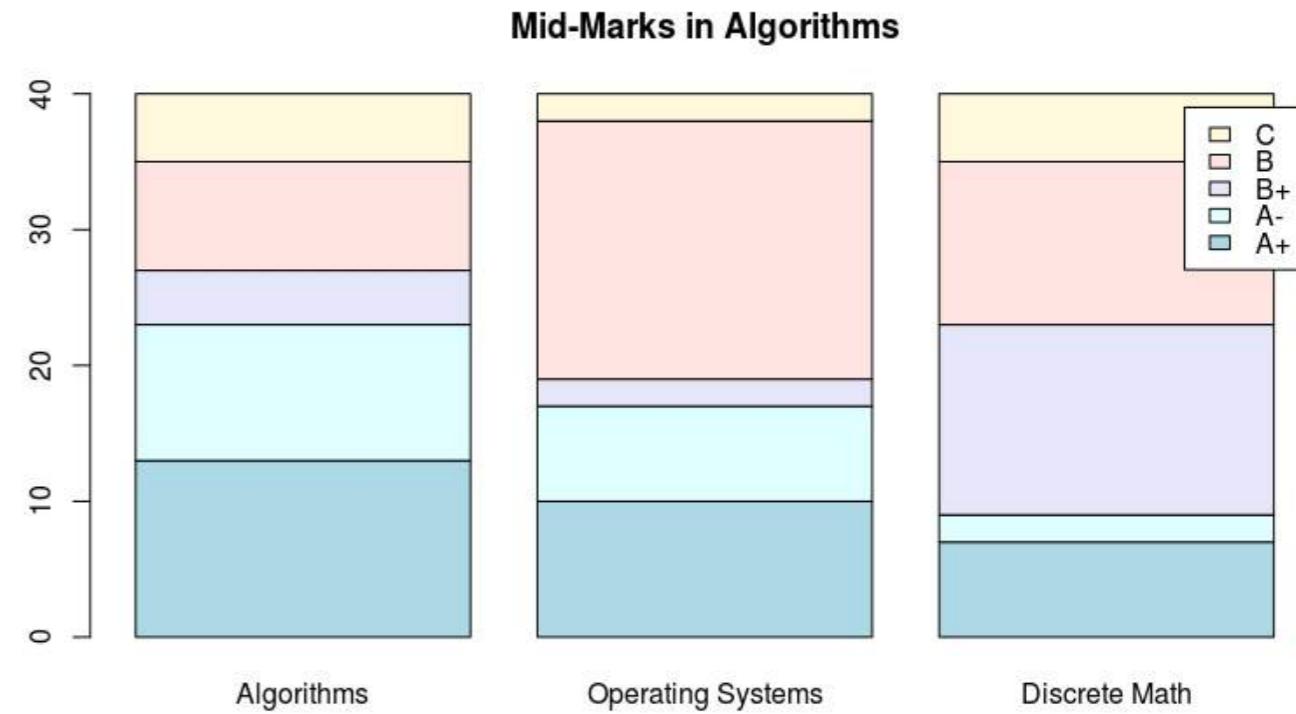
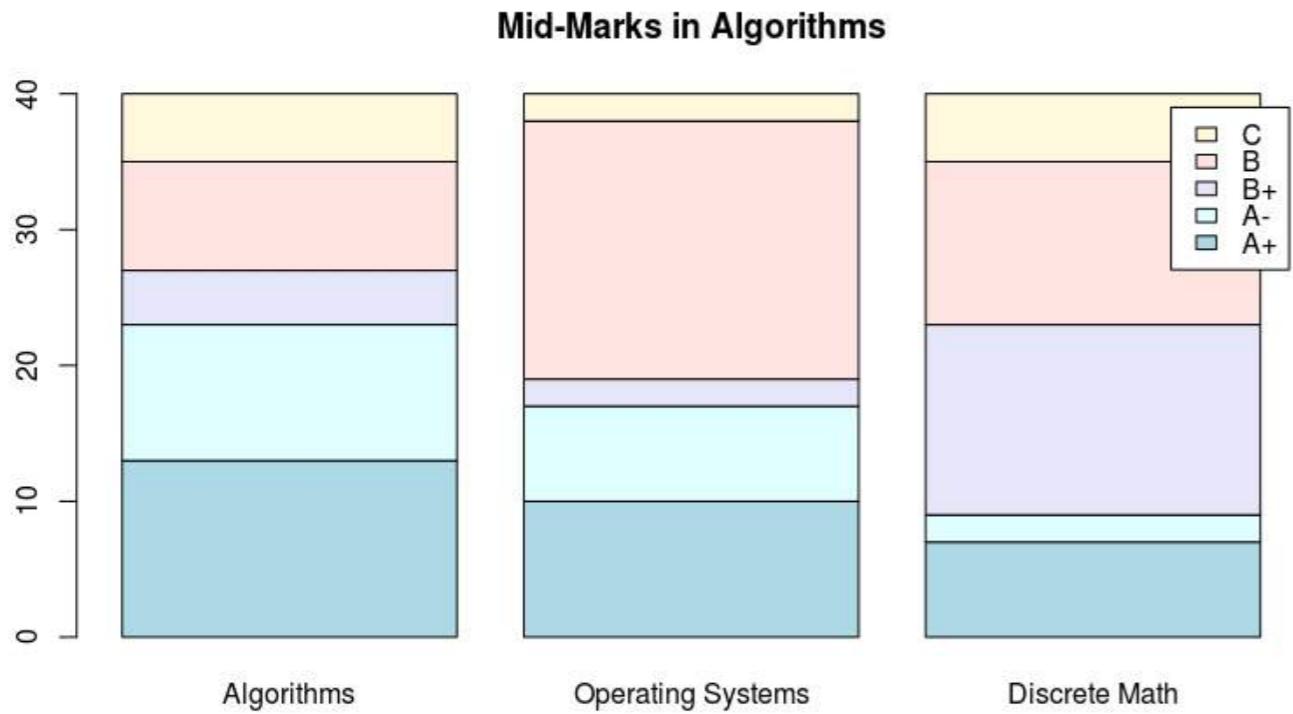


The `heights` parameter of the `barplot()` could be a matrix. For example it could be matrix, where the columns are the various subjects taken in a course, the rows could be the labels of the grades. Consider the following matrix:

```
> gradTab
          Algorithms Operating Systems Discrete Math
A-          13            10            7
A+          10            7            2
B            4            2           14
B+          8            19           12
C            5            2            5
```

To draw a stacked bar, simply use the command:

```
> barplot(gradTab,col = c("lightblue","lightcyan",
  "lavender", "mistyrose", "cornsilk"),legend.text = grades,
  main="Mid-Marks in Algorithms")
```

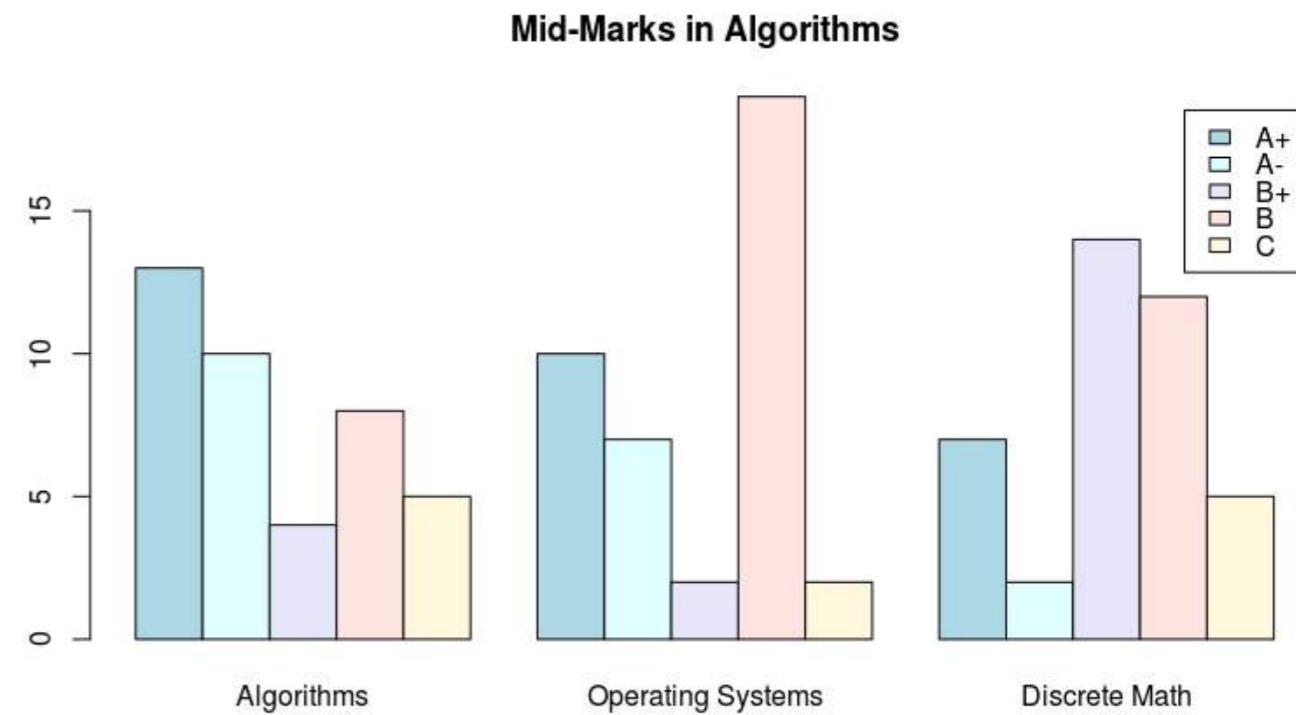
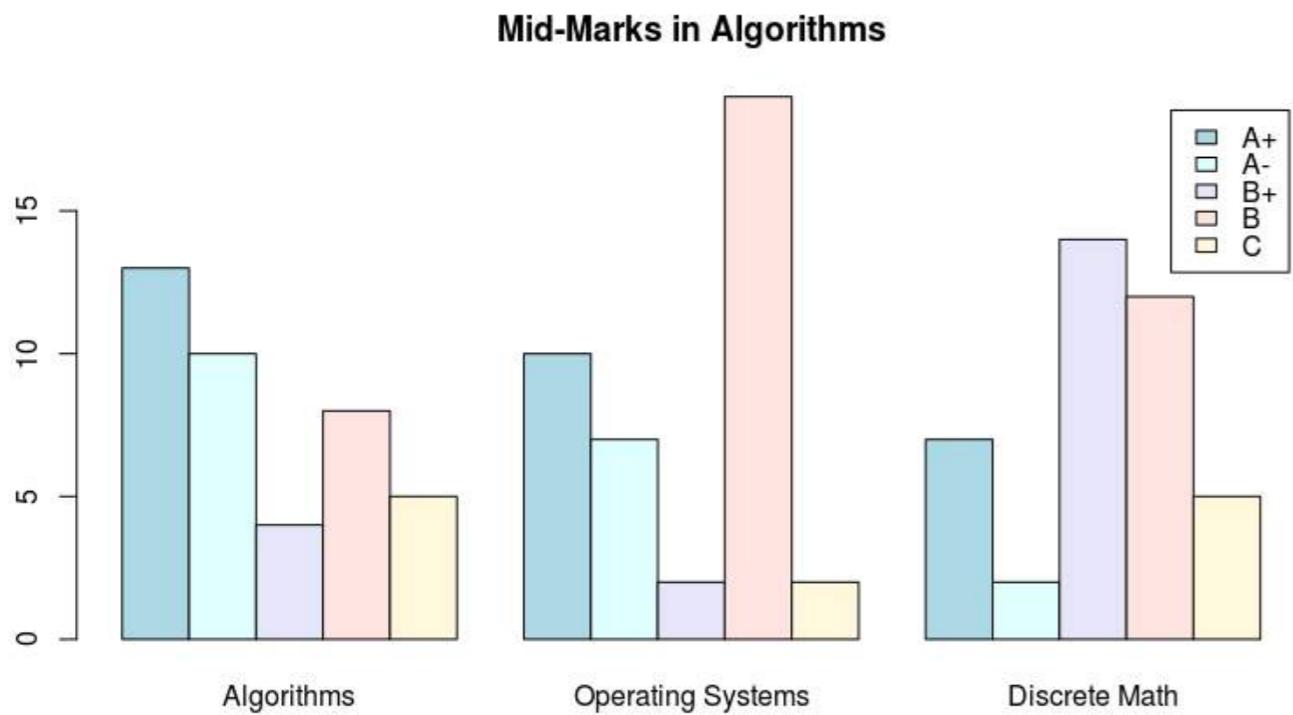


要绘制并列条形图，使用beside参数，如下所示：

```
> barplot(gradTab,beside = T,col = c("lightblue","lightcyan",
  "lavender", "mistyrose", "cornsilk"),legend.text = grades,
  main="算法中的期中成绩")
```

To draw a juxtaposed bars, use the beside parameter, as given under:

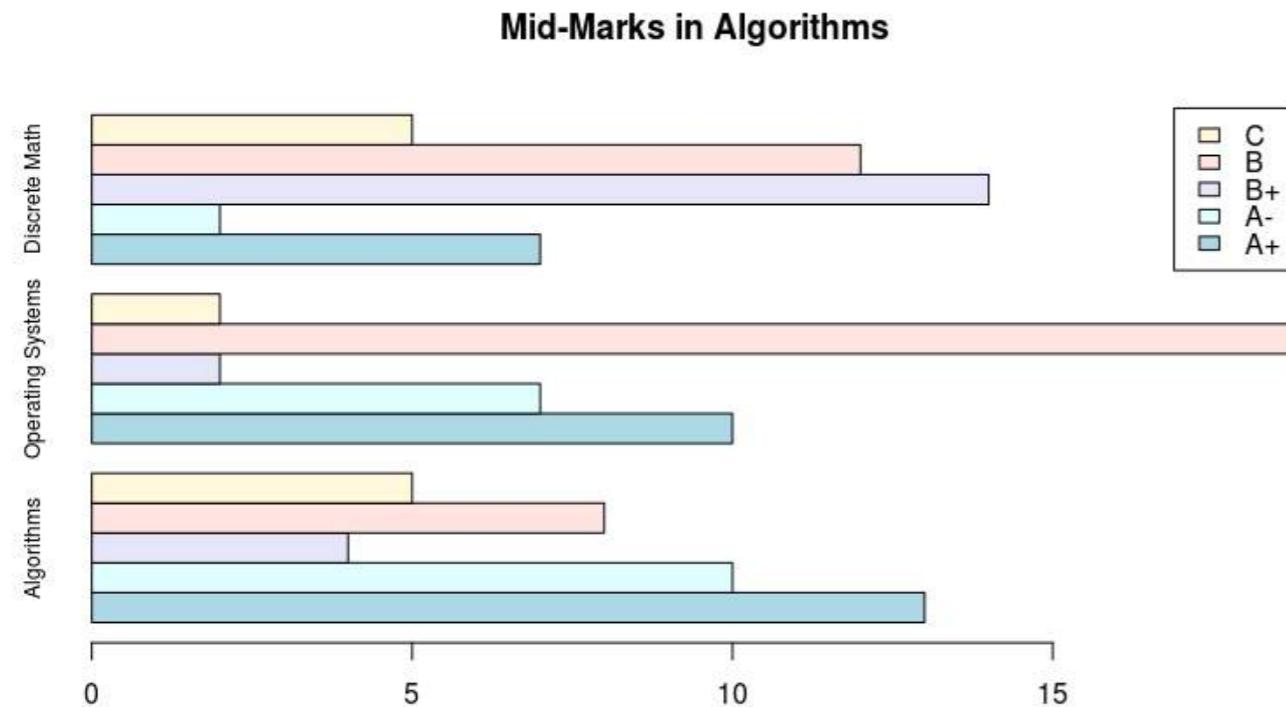
```
> barplot(gradTab,beside = T,col = c("lightblue","lightcyan",
  "lavender", "mistyrose", "cornsilk"),legend.text = grades,
  main="Mid-Marks in Algorithms")
```



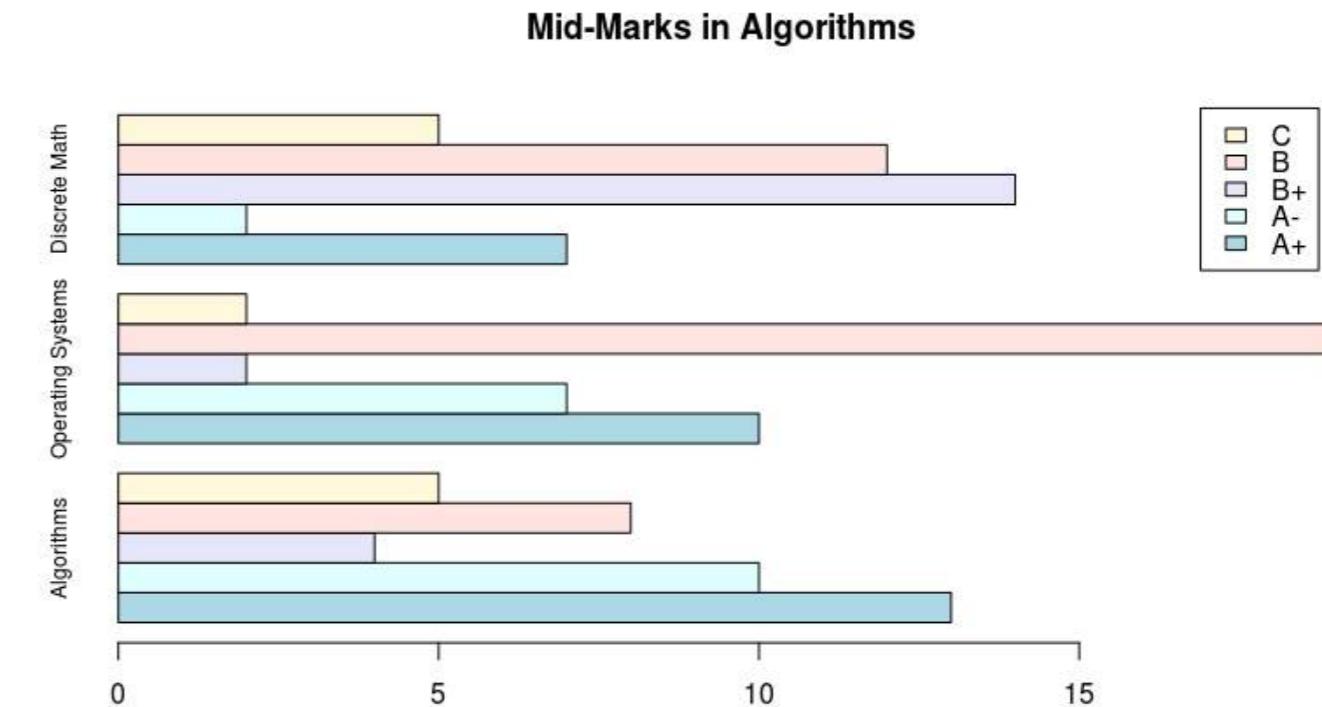
使用horiz=T参数可以获得水平条形图：

A horizontal bar chart can be obtained using horiz=T parameter:

```
> barplot(gradTab,beside = T,horiz=T,col = c("lightblue","lightcyan",
  "lavender", "mistyrose", "cornsilk"),legend.text = grades,
  cex.names=.75,main="算法中的期中成绩")
```



```
> barplot(gradTab,beside = T,horiz=T,col = c("lightblue","lightcyan",
  "lavender", "mistyrose", "cornsilk"),legend.text = grades,
  cex.names=.75,main="Mid-Marks in Algorithms")
```



第26章：基础绘图

参数

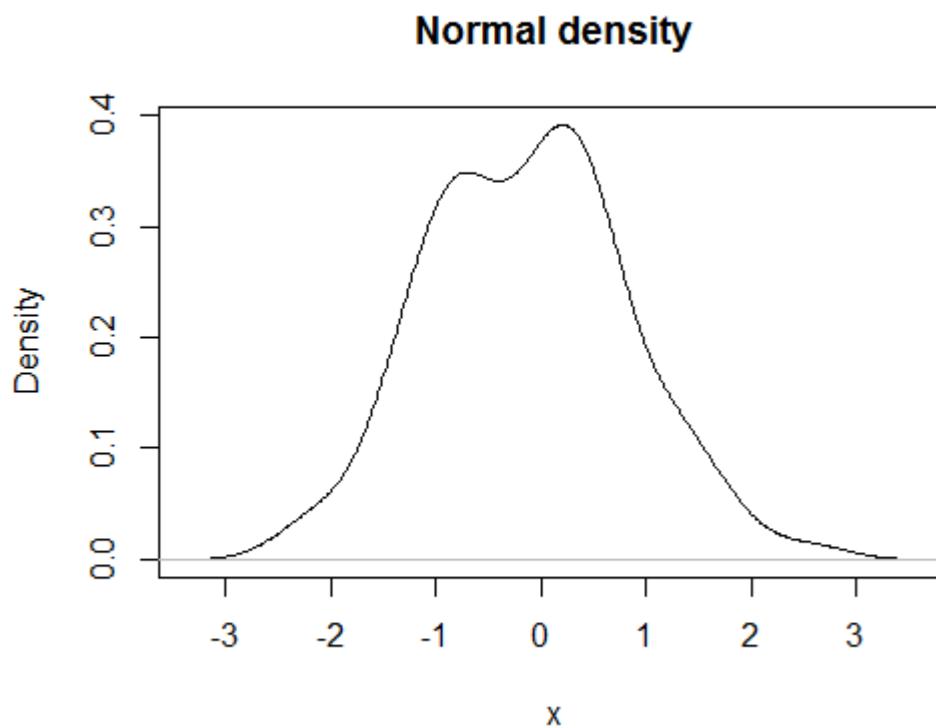
	详细信息
x	x轴变量。可以提供data\$variablex或data[,x]
y	y轴变量。可以提供data\$variabley或data[,y]
主	图表主标题
副	图表可选副标题
x轴标签	x轴的标签
ylab	y轴标签
pch	表示绘图符号的整数或字符
col	表示颜色的整数或字符串
绘图类型	绘图类型。"p"表示点，"l"表示线，"b"表示"b"中仅线的部分，"o"表示点和线的叠加'overplotted'，"h"表示类似于'histogram'（或'high-density'）的垂直线，"s"表示阶梯，"S"表示其他阶梯，"n"表示不绘图
type	绘图类型。"p"表示点，"l"表示线，"b"表示"b"中仅线的部分，"o"表示点和线的叠加'overplotted'，"h"表示类似于'histogram'（或'high-density'）的垂直线，"s"表示阶梯，"S"表示其他阶梯，"n"表示不绘图

第26.1节：密度图

直方图的一个非常有用且合乎逻辑的后续步骤是绘制随机变量的平滑密度函数。一个基本的绘图命令是

```
plot(density(rnorm(100)),main="Normal density",xlab="x")
```

看起来像



你可以叠加直方图和密度曲线

```
x=rnorm(100)  
hist(x,prob=TRUE,main="正态密度 + 直方图")  
lines(density(x),lty="点状",col="红色")
```

Chapter 26: Base Plotting

Parameter

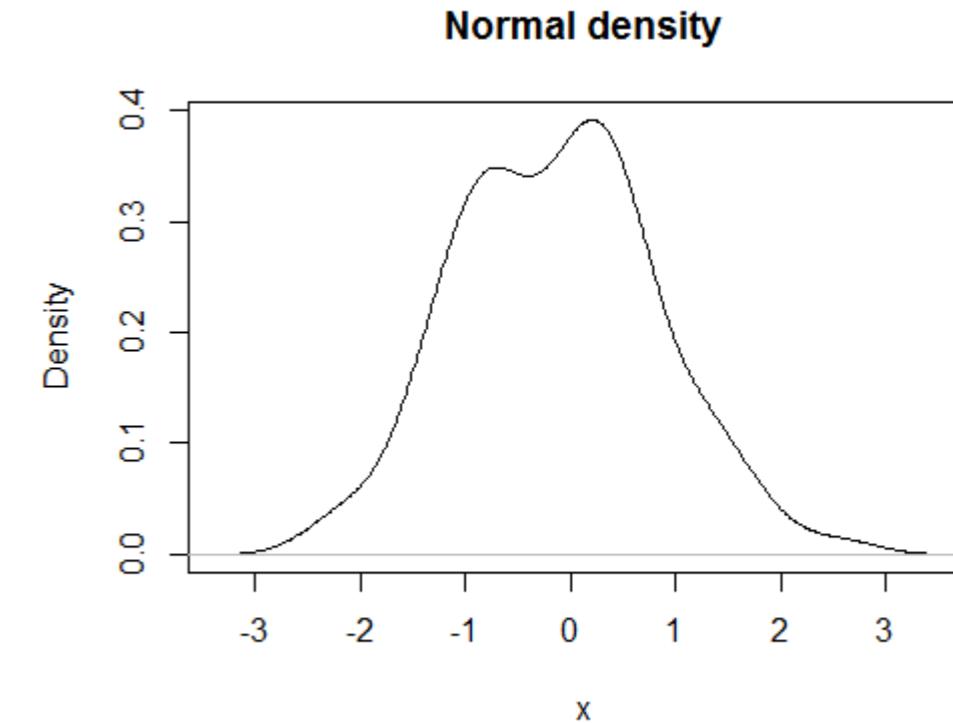
	Details
x	x-axis variable. May supply either <code>data\$variablex</code> or <code>data[,x]</code>
y	y-axis variable. May supply either <code>data\$variabley</code> or <code>data[,y]</code>
main	Main title of plot
sub	Optional subtitle of plot
xlab	Label for x-axis
ylab	Label for y-axis
pch	Integer or character indicating plotting symbol
col	Integer or string indicating color
type	Type of plot. "p" for points, "l" for lines, "b" for both, "c" for the lines part alone of "b", "o" for both 'overplotted', "h" for 'histogram'-like (or 'high-density') vertical lines, "s" for stair steps, "S" for other steps, "n" for no plotting

Section 26.1: Density plot

A very useful and logical follow-up to histograms would be to plot the smoothed density function of a random variable. A basic plot produced by the command

```
plot(density(rnorm(100)),main="Normal density",xlab="x")
```

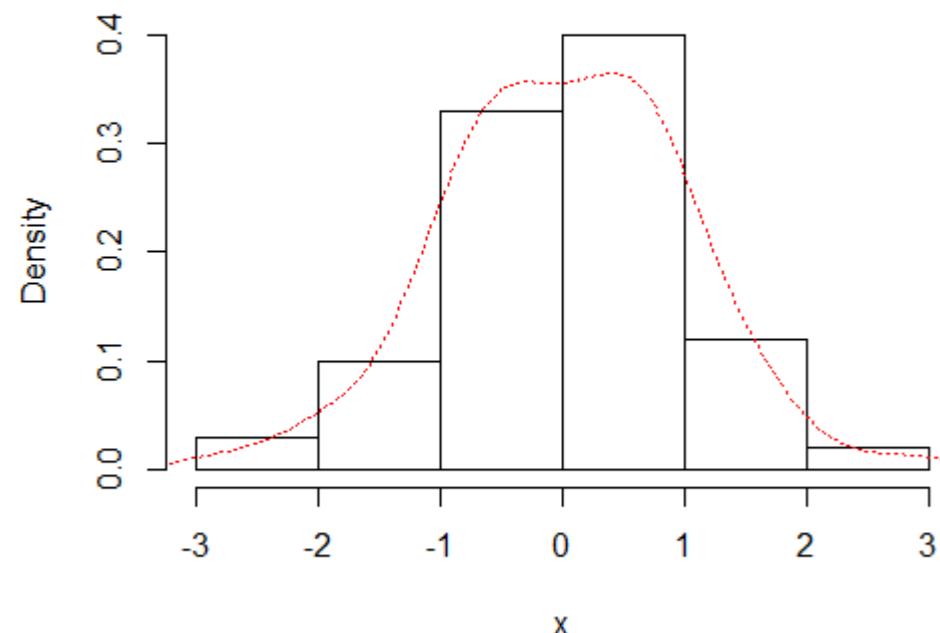
would look like



You can overlay a histogram and a density curve with

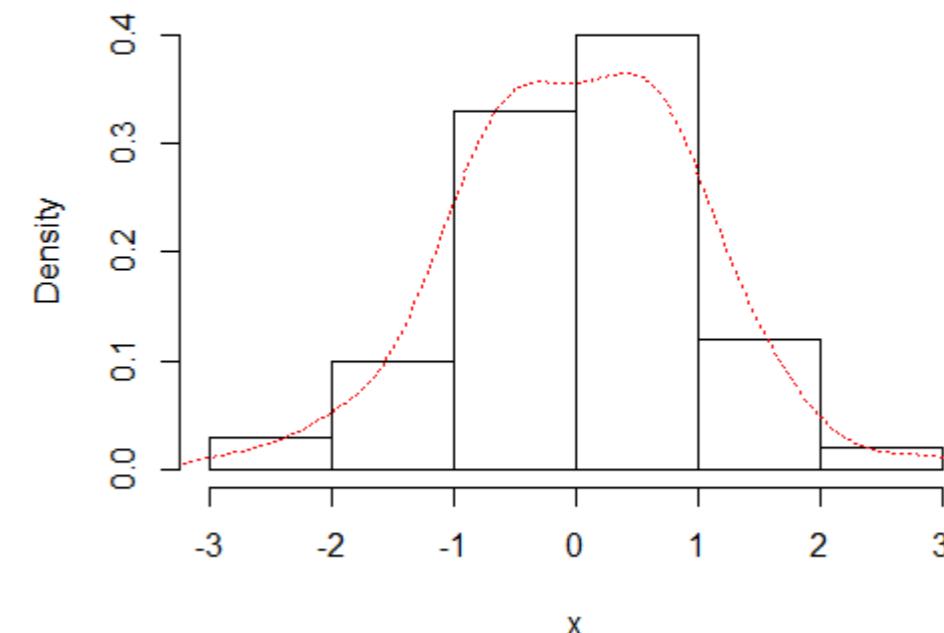
```
x=rnorm(100)  
hist(x,prob=TRUE,main="Normal density + histogram")  
lines(density(x),lty="dotted",col="red")
```

Normal density + histogram



which gives

Normal density + histogram



第26.2节：组合图形

在一个图中组合多种图形类型（例如柱状图和散点图并排）通常很有用。R通过函数par()和layout()使这变得简单。

par()

par使用参数mfrow或mfcol来创建一个由nrows和ncols组成的矩阵c(nrows, ncols)，作为绘图的网格。以下示例展示了如何在一个图中组合四个图形：

```
par(mfrow=c(2,2))
plot(cars, main="速度与距离")
hist(cars$speed, main="速度直方图")
boxplot(cars$dist, main="距离箱线图")
boxplot(cars$speed, main="速度箱线图")
```

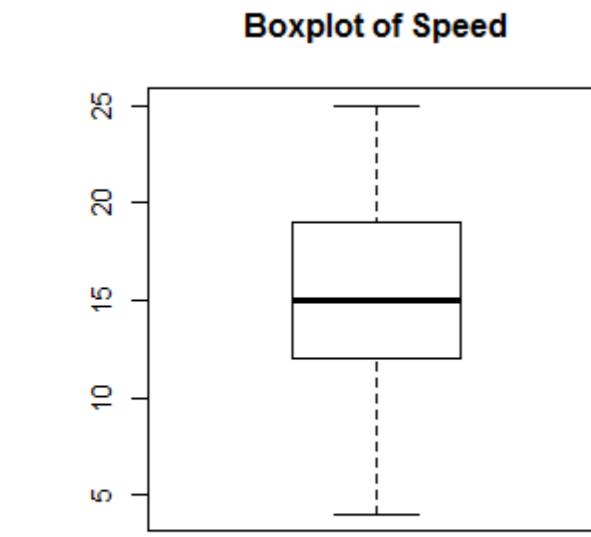
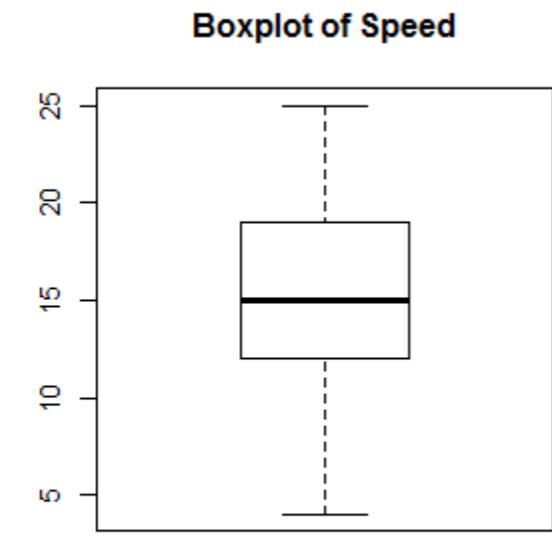
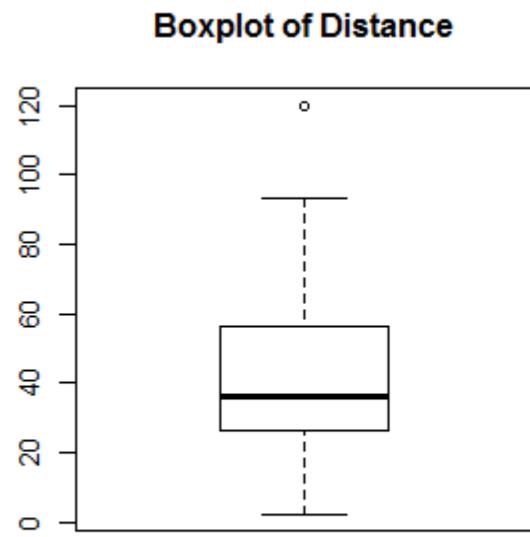
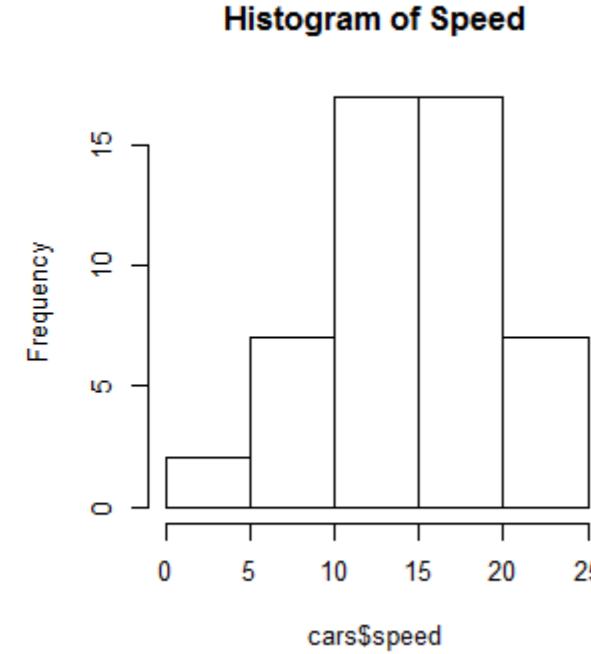
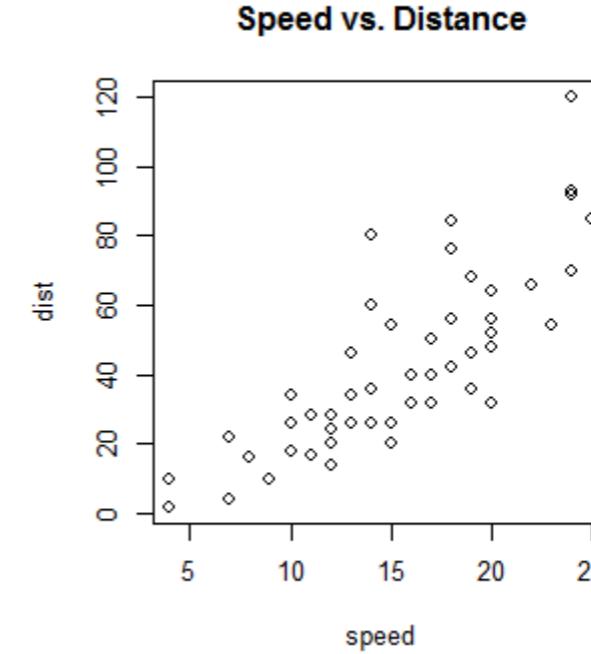
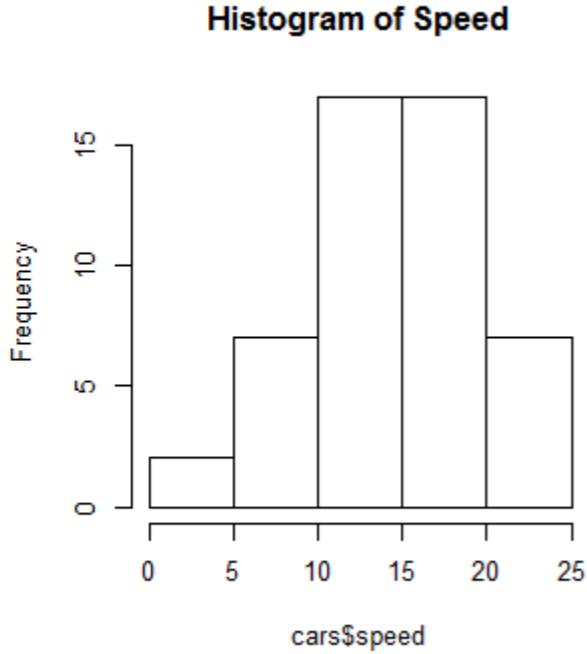
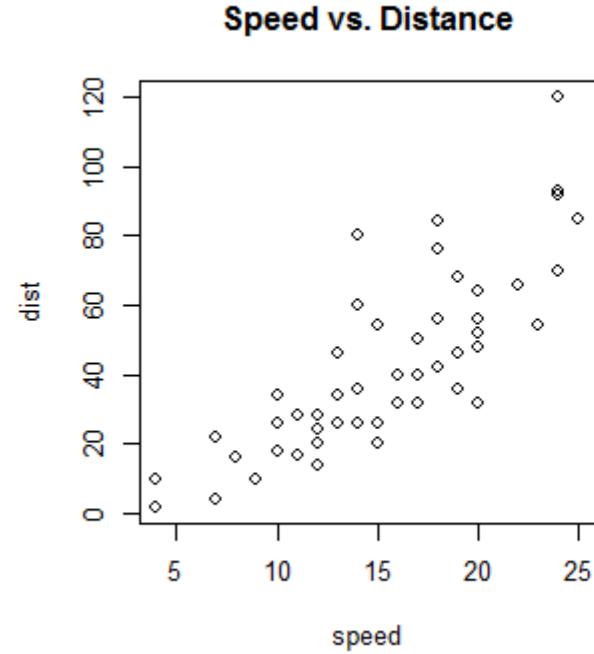
Section 26.2: Combining Plots

It's often useful to combine multiple plot types in one graph (for example a Barplot next to a Scatterplot.) R makes this easy with the help of the functions `par()` and `layout()`.

par()

`par` uses the arguments `mfrow` or `mfcol` to create a matrix of nrows and ncols `c(nrows, ncols)` which will serve as a grid for your plots. The following example shows how to combine four plots in one graph:

```
par(mfrow=c(2,2))
plot(cars, main="Speed vs. Distance")
hist(cars$speed, main="Histogram of Speed")
boxplot(cars$dist, main="Boxplot of Distance")
boxplot(cars$speed, main="Boxplot of Speed")
```



layout()

layout() 更加灵活，允许你指定最终合并图中每个图的位置和范围。该函数期望输入一个矩阵对象：

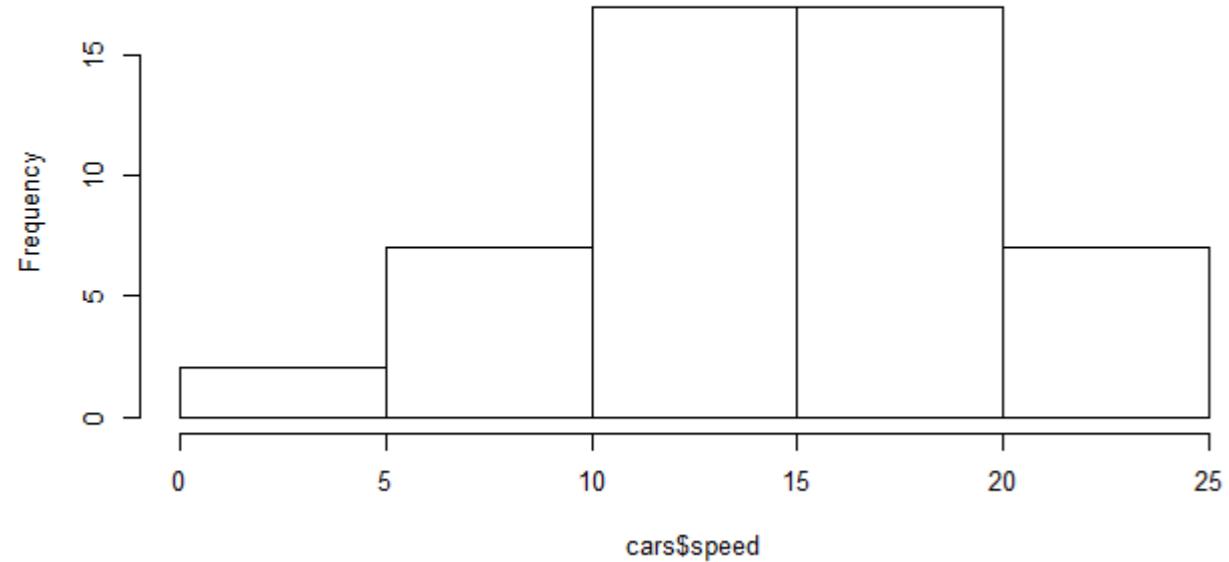
```
layout(matrix(c(1,1,2,3), 2,2, byrow=T))
hist(cars$speed, main="速度的直方图")
boxplot(cars$dist, main="距离的箱线图")
boxplot(cars$speed, main="速度的箱线图")
```

layout()

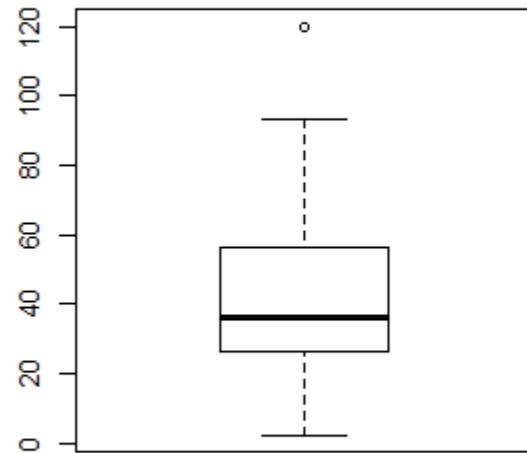
The `layout()` is more flexible and allows you to specify the location and the extent of each plot within the final combined graph. This function expects a matrix object as an input:

```
layout(matrix(c(1,1,2,3), 2,2, byrow=T))
hist(cars$speed, main="Histogram of Speed")
boxplot(cars$dist, main="Boxplot of Distance")
boxplot(cars$speed, main="Boxplot of Speed")
```

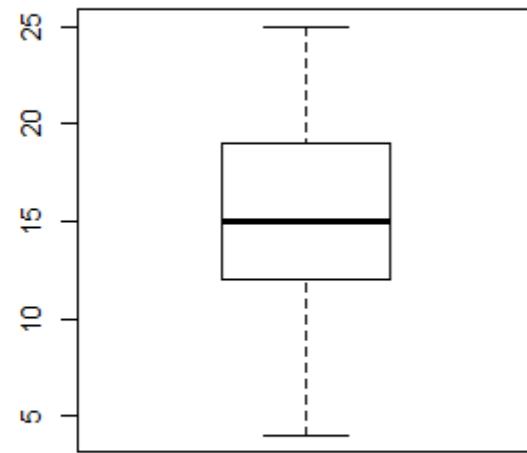
Histogram of Speed



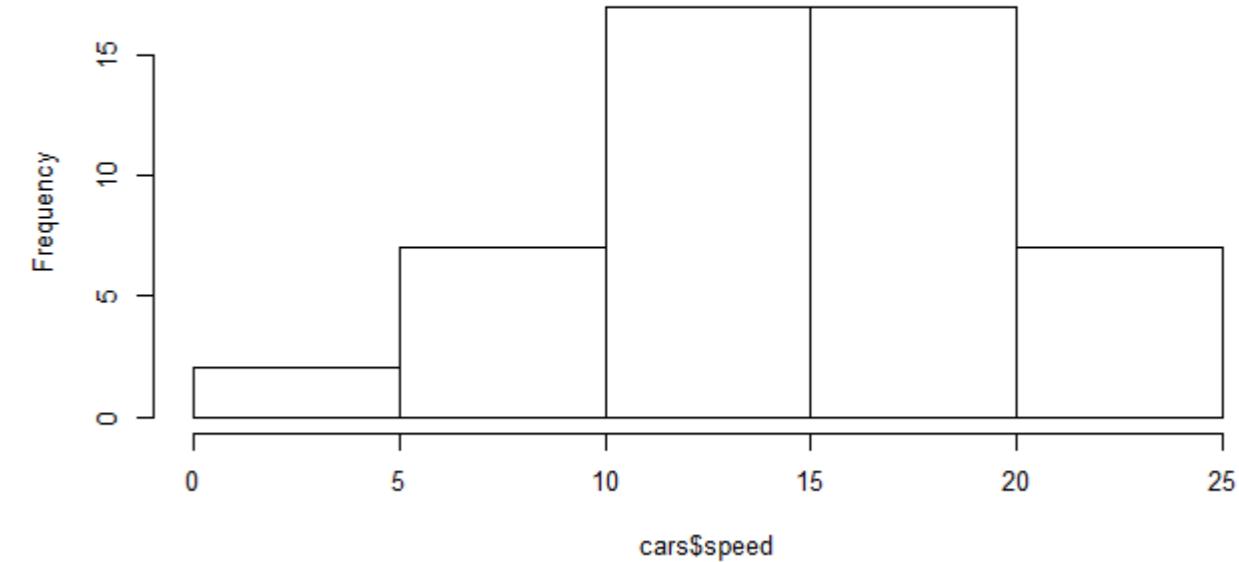
Boxplot of Distance



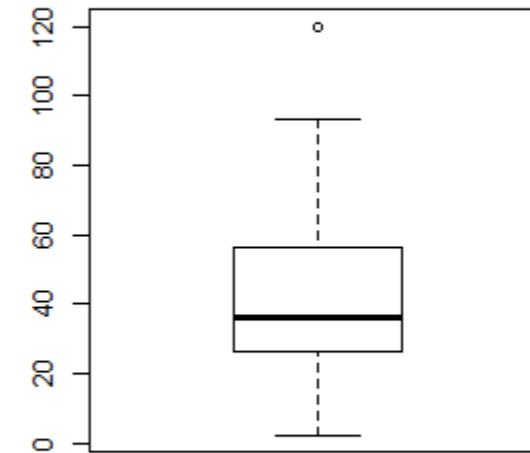
Boxplot of Speed



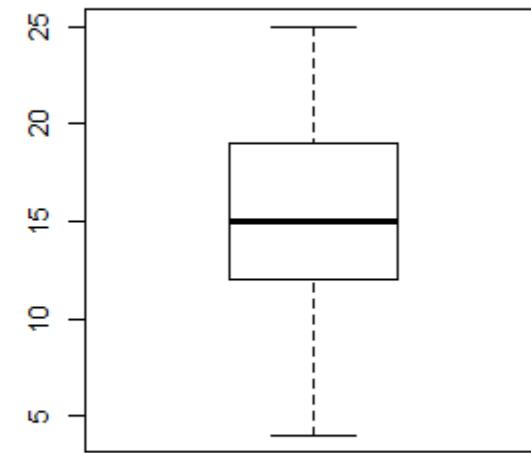
Histogram of Speed



Boxplot of Distance



Boxplot of Speed



第26.3节：R绘图入门

- 散点图

你有两个向量，想要绘制它们。

```
x_values <- rnorm(n = 20, mean = 5, sd = 8) #从正态分布(5,8)生成的20个值  
y_values <- rbeta(n = 20, shape1 = 500, shape2 = 10) #从Beta分布(500,10)生成的20个值
```

如果你想绘制一个图，纵轴为y_values，横轴为x_values，可以使用以下命令：

```
plot(x = x_values, y = y_values, type = "p") #标准散点图  
plot(x = x_values, y = y_values, type = "l") #用线条绘制图形
```

Section 26.3: Getting Started with R_Plots

- Scatterplot

You have two vectors and you want to plot them.

```
x_values <- rnorm(n = 20, mean = 5, sd = 8) #20 values generated from Normal(5,8)  
y_values <- rbeta(n = 20, shape1 = 500, shape2 = 10) #20 values generated from Beta(500,10)
```

If you want to make a plot which has the y_values in vertical axis and the x_values in horizontal axis, you can use the following commands:

```
plot(x = x_values, y = y_values, type = "p") #standard scatter-plot  
plot(x = x_values, y = y_values, type = "l") # plot with lines
```

```
plot(x = x_values, y = y_values, type = "n") # 空图
```

你可以在控制台输入 `?plot()` 来了解更多选项。

• 箱线图

你有一些变量，想要检查它们的分布情况

#箱线图是查看数据中是否存在异常值的简单方法。

```
z<- rbeta(20 , 500 , 10) # 从贝塔分布生成值  
z[c(19 , 20)] <- c(0.97 , 1.05) # 用异常值替换最后两个值  
boxplot(z) # 这两个点是变量 z 的异常值。
```

• 直方图

绘制直方图的简单方法

```
hist(x = x_values) # x向量的直方图  
hist(x = x_values, breaks = 3) #使用 breaks 设置你想要的条形数量
```

• 饼图

如果你想可视化一个变量的频率，只需绘制饼图

首先我们必须生成带有频率的数据，例如：

```
P <- c(rep('A' , 3) , rep('B' , 10) , rep('C' , 7) )  
t <- table(P) # 这是变量 P 的频率矩阵  
pie(t) # 这是上述矩阵的可视化版本
```

第26.4节：基本绘图

基本绘图是通过调用 `plot()` 创建的。这里我们使用内置的 `cars` 数据框，包含了1920年代汽车的速度和停车距离。（想了解更多数据集信息，请使用 `help(cars)`）。

```
plot(x = cars$speed, y = cars$dist, pch = 1, col = 1,  
     main = "汽车速度与距离",  
     xlab = "速度", ylab = "距离")
```

```
plot(x = x_values, y = y_values, type = "n") # empty plot
```

You can type `?plot()` in the console to read about more options.

• Boxplot

You have some variables and you want to examine their Distributions

`#boxplot` is an easy way to see if we have some outliers in the data.

```
z<- rbeta(20 , 500 , 10) #generating values from beta distribution  
z[c(19 , 20)] <- c(0.97 , 1.05) # replace the two last values with outliers  
boxplot(z) # the two points are the outliers of variable z.
```

• Histograms

Easy way to draw histograms

```
hist(x = x_values) # Histogram for x vector  
hist(x = x_values, breaks = 3) #use breaks to set the numbers of bars you want
```

• Pie_charts

If you want to visualize the frequencies of a variable just draw pie

First we have to generate data with frequencies, for example :

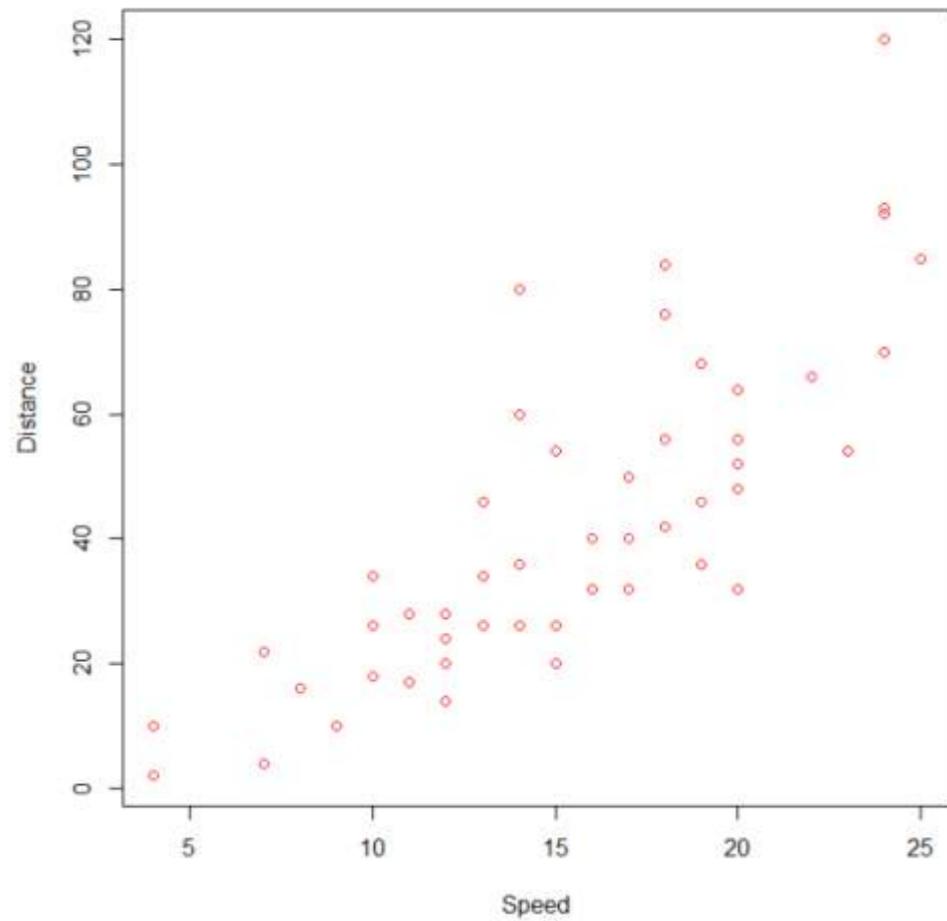
```
P <- c(rep('A' , 3) , rep('B' , 10) , rep('C' , 7) )  
t <- table(P) # this is a frequency matrix of variable P  
pie(t) # And this is a visual version of the matrix above
```

Section 26.4: Basic Plot

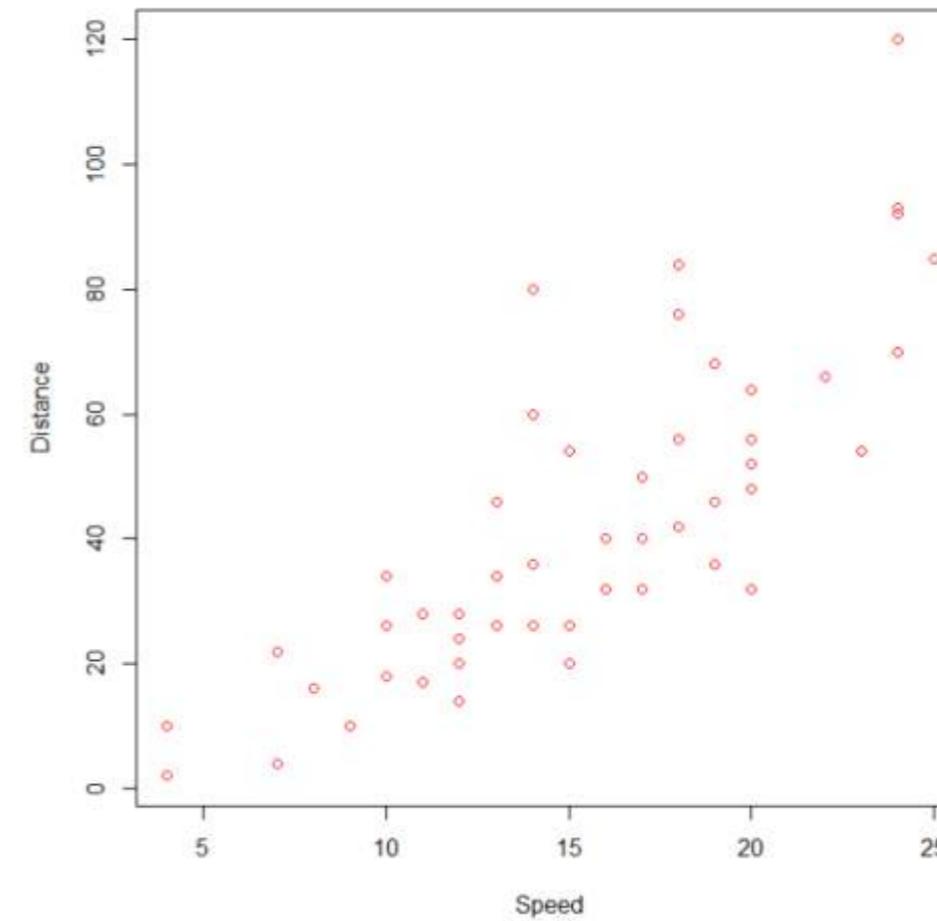
A basic plot is created by calling `plot()`. Here we use the built-in `cars` data frame that contains the speed of cars and the distances taken to stop in the 1920s. (To find out more about the dataset, use `help(cars)`).

```
plot(x = cars$speed, y = cars$dist, pch = 1, col = 1,  
     main = "Distance vs Speed of Cars",  
     xlab = "Speed", ylab = "Distance")
```

Distance to stop vs Speed of Cars



Distance to stop vs Speed of Cars



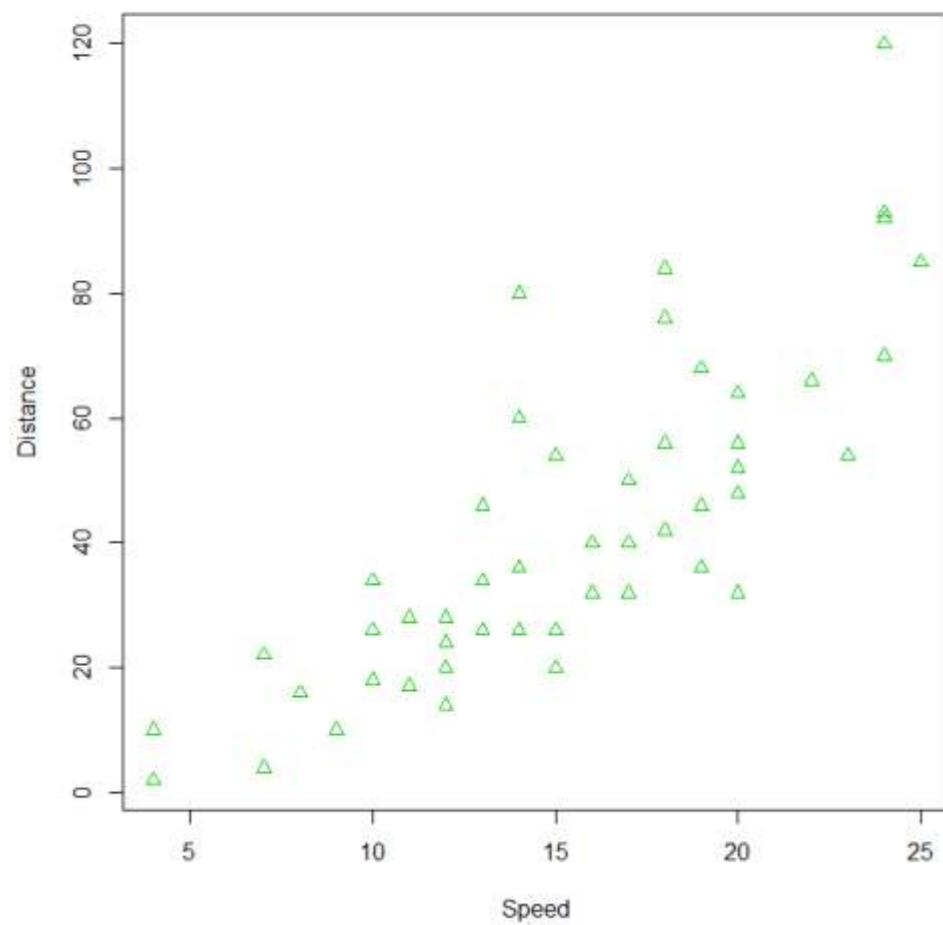
我们可以在代码中使用许多其他变体来获得相同的结果。我们也可以更改参数以获得不同的结果。

```
with(cars, plot(dist~speed, pch = 2, col = 3,  
main = "汽车速度与停车距离",  
xlab = "速度", ylab = "距离"))
```

We can use many other variations in the code to get the same result. We can also change the parameters to obtain different results.

```
with(cars, plot(dist~speed, pch = 2, col = 3,  
main = "Distance to stop vs Speed of Cars",  
xlab = "Speed", ylab = "Distance"))
```

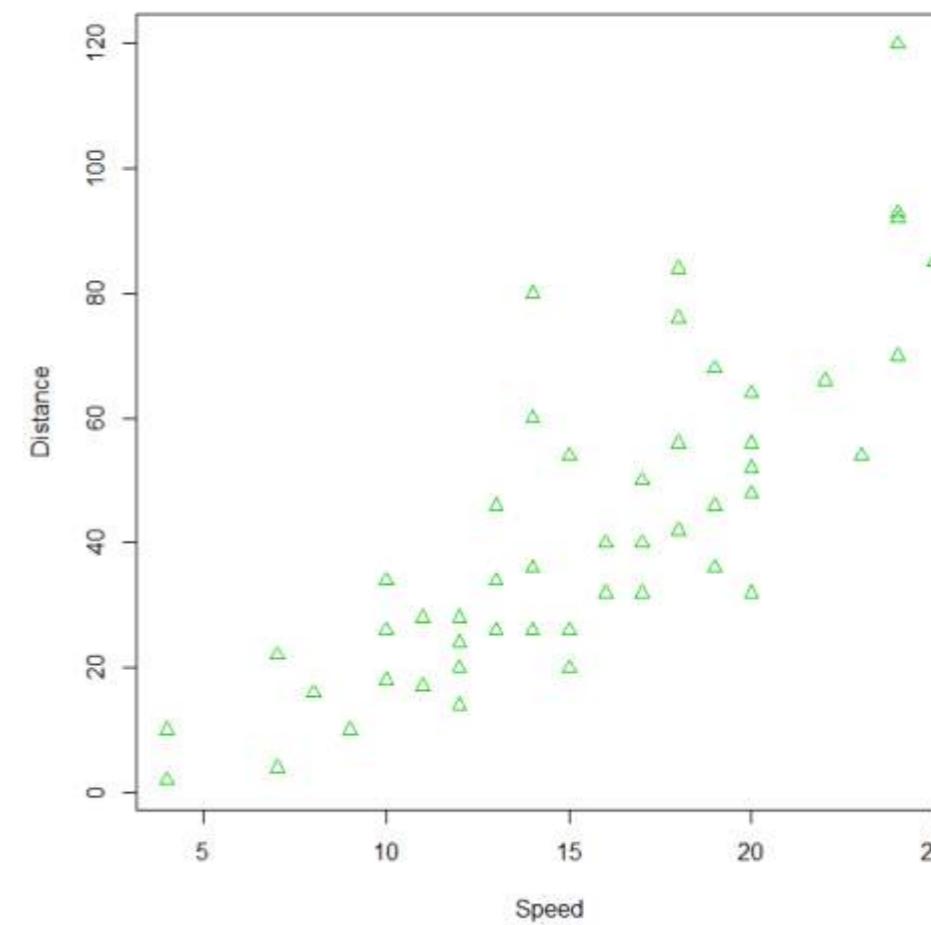
Distance to stop vs Speed of Cars



可以通过调用points()、text()、mtext()、lines()、grid()等函数为此图添加更多功能。

```
plot(dist~speed, pch = "*", col = "品红色", data=cars,
      main = "汽车速度与停车距离",
      xlab = "速度", ylab = "距离")
mtext("1920年代。")
grid(), col="浅蓝色")
```

Distance to stop vs Speed of Cars

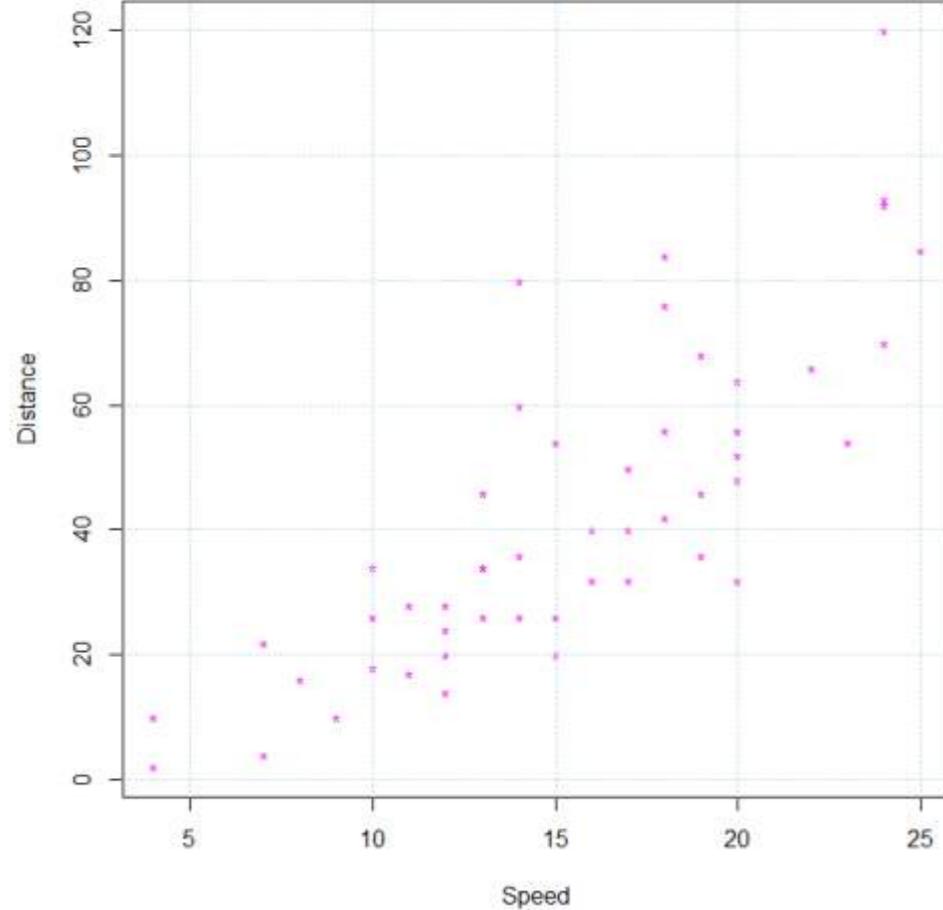


Additional features can be added to this plot by calling `points()`, `text()`, `mtext()`, `lines()`, `grid()`, etc.

```
plot(dist~speed, pch = "*", col = "magenta", data=cars,
      main = "Distance to stop vs Speed of Cars",
      xlab = "Speed", ylab = "Distance")
mtext("In the 1920s.")
grid(), col="lightblue")
```

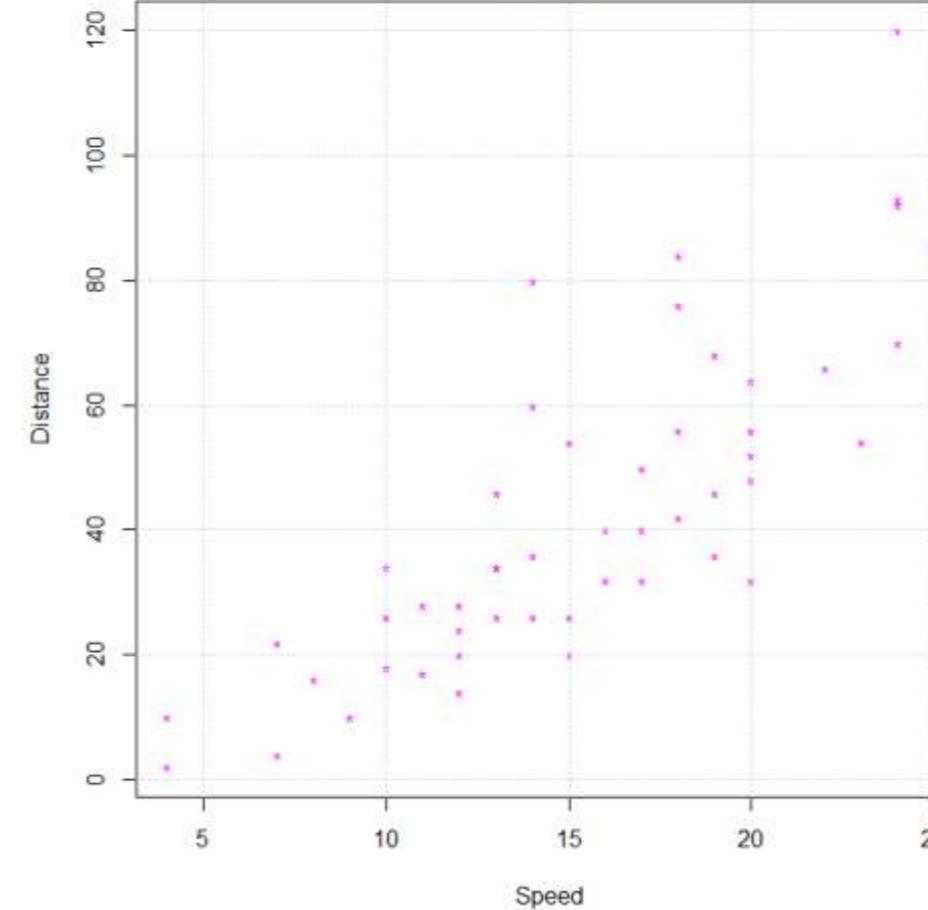
Distance to stop vs Speed of Cars

In the 1920s.



Distance to stop vs Speed of Cars

In the 1920s.



第26.5节：直方图

直方图允许对数据的潜在分布进行伪绘图。

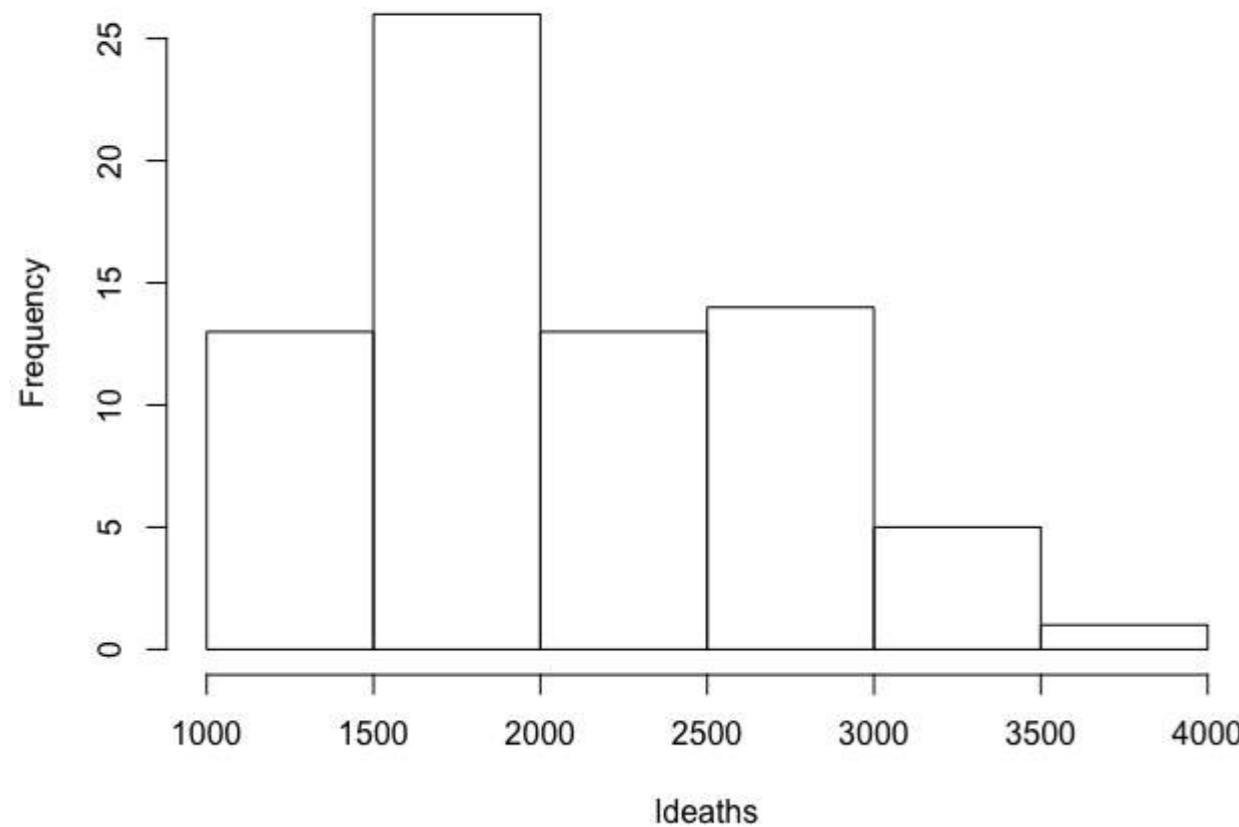
`hist(ldeaths)`

Section 26.5: Histograms

Histograms allow for a pseudo-plot of the underlying distribution of the data.

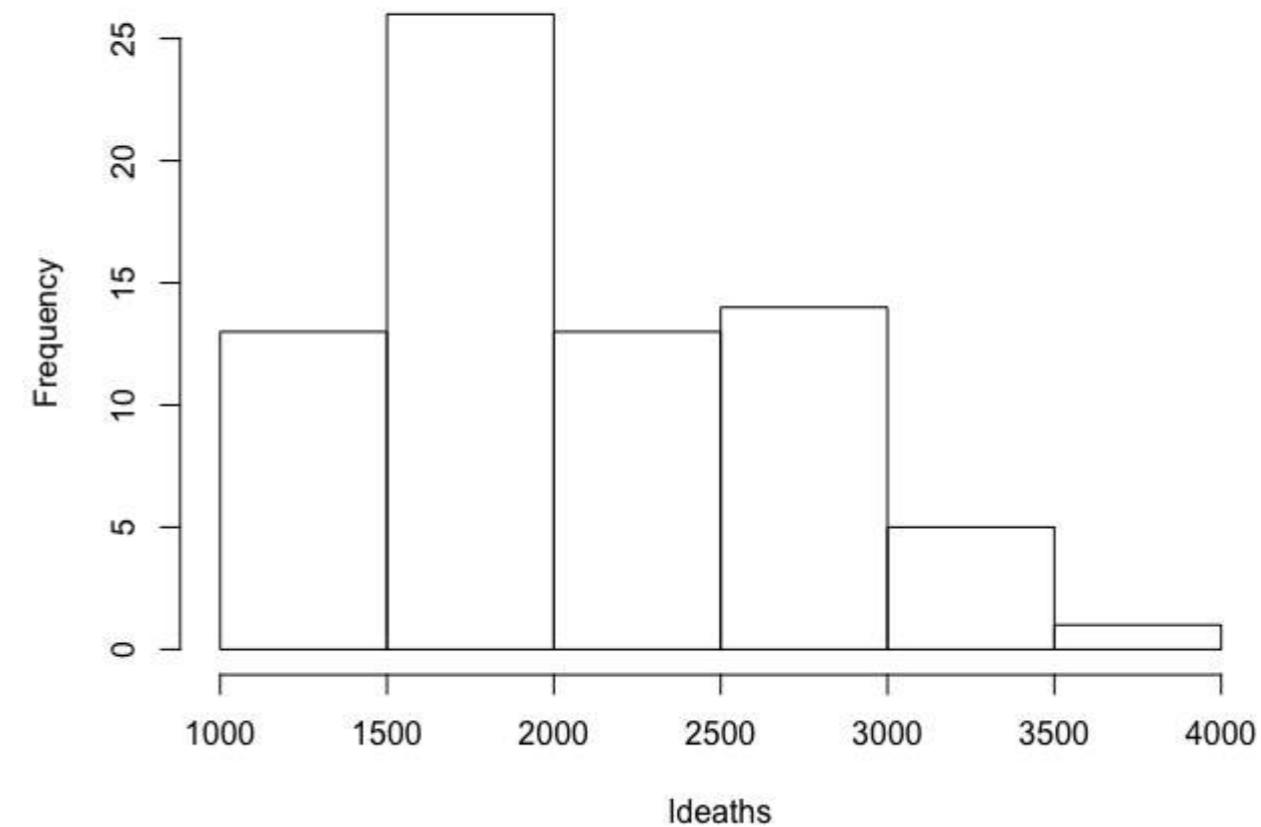
`hist(ldeaths)`

Histogram of Ideaths



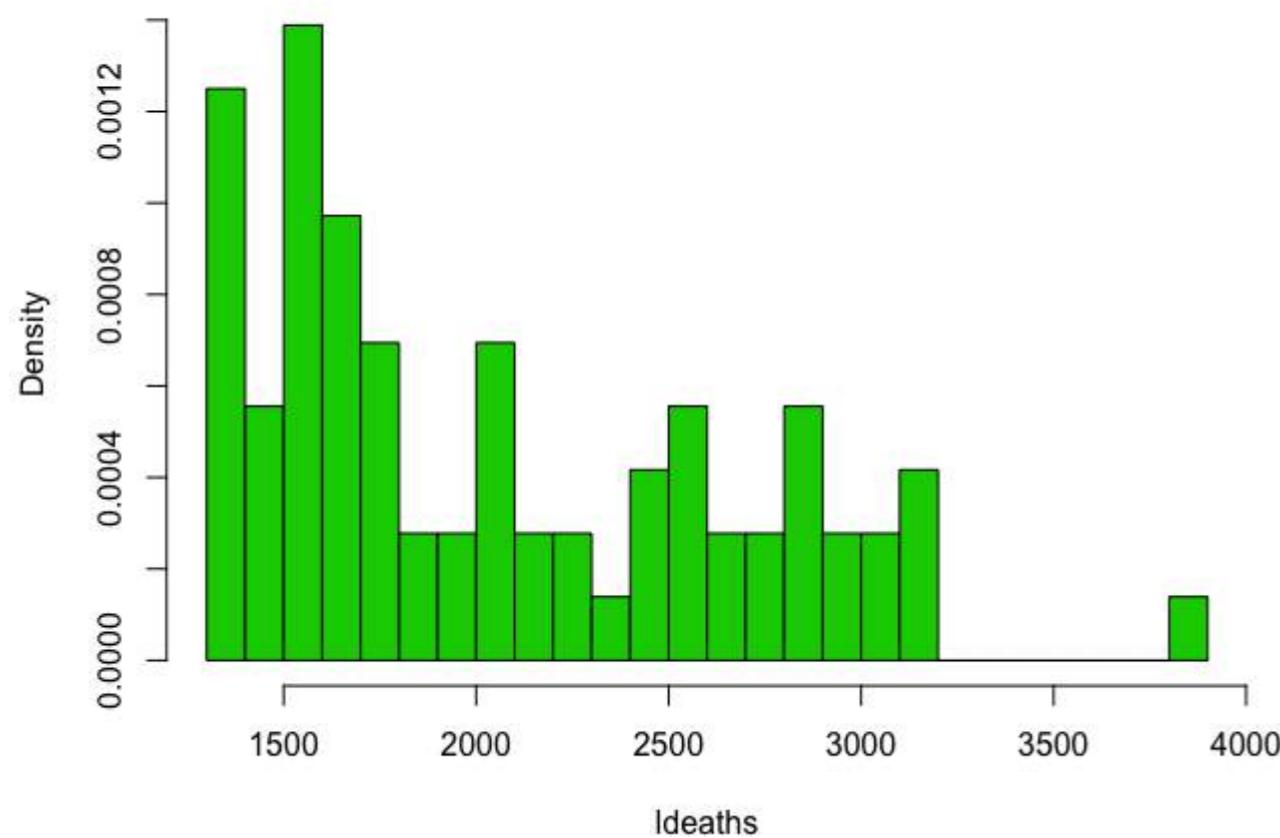
```
hist(ldeaths, breaks = 20, freq = F, col = 3)
```

Histogram of Ideaths

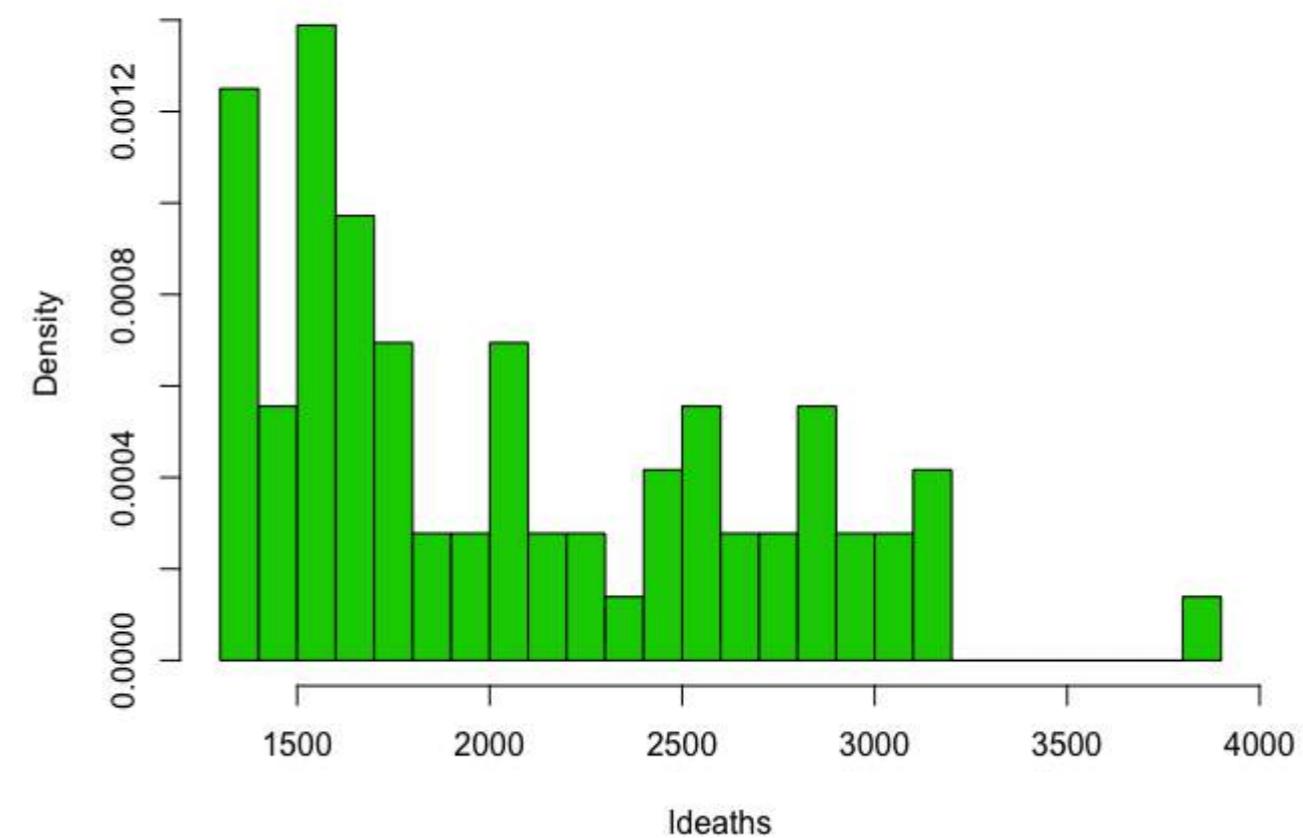


```
hist(ldeaths, breaks = 20, freq = F, col = 3)
```

Histogram of Deaths



Histogram of Deaths



第26.6节：Matplot

matplot 对于快速绘制同一对象的多组观测值特别有用，尤其是来自矩阵的观测值，可以在同一图表上绘制。

下面是一个包含四组随机抽样的矩阵示例，每组的均值不同。

```
xmat <- cbind(rnorm(100, -3), rnorm(100, -1), rnorm(100, 1), rnorm(100, 3))
head(xmat)
# [1] [2] [3] [4]
# [1,] -3.072793 -2.53111494 0.6168063 3.780465
# [2,] -3.702545 -1.42789347 -0.2197196 2.478416
# [3,] -2.890698 -1.88476126 1.9586467 5.268474
# [4,] -3.431133 -2.02626870 1.1153643 3.170689
# [5,] -4.532925 0.02164187 0.9783948 3.162121
# [6,] -2.169391 -1.42699116 0.3214854 4.480305
```

将所有这些观测值绘制在同一图表上的一种方法是先调用一次`plot`，随后再调用三次`points`或`lines`。

```
plot(xmat[,1], type = 'l')
lines(xmat[,2], col = 'red')
lines(xmat[,3], col = 'green')
lines(xmat[,4], col = 'blue')
```

Section 26.6: Matplot

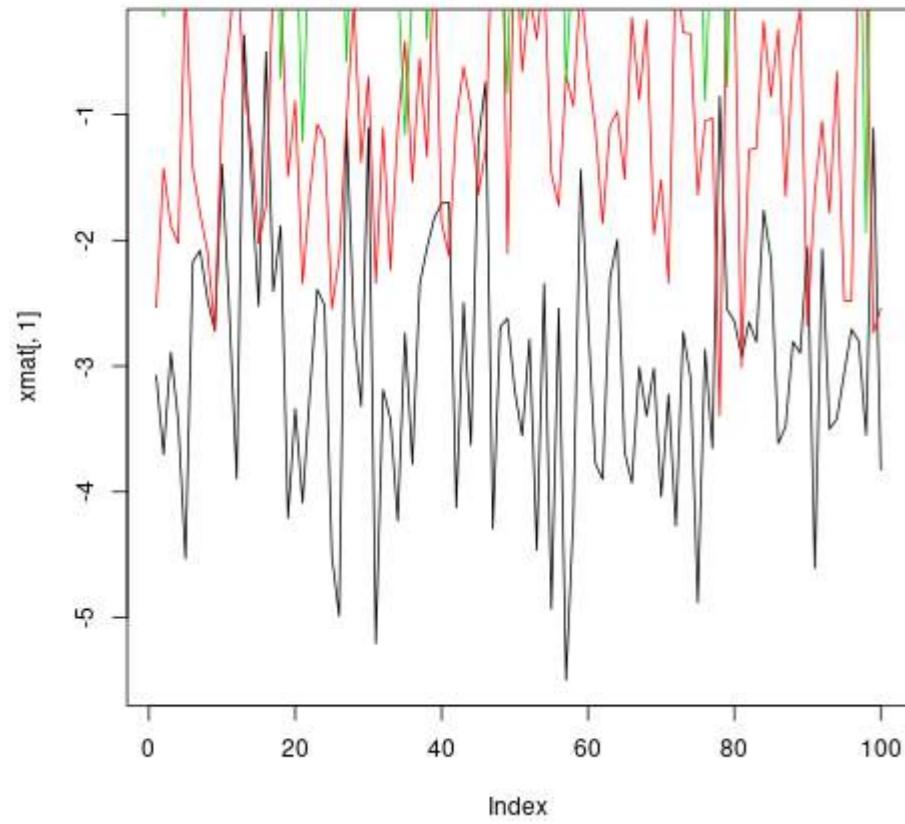
matplot is useful for quickly plotting multiple sets of observations from the same object, particularly from a matrix, on the same graph.

Here is an example of a matrix containing four sets of random draws, each with a different mean.

```
xmat <- cbind(rnorm(100, -3), rnorm(100, -1), rnorm(100, 1), rnorm(100, 3))
head(xmat)
# [1] [2] [3] [4]
# [1,] -3.072793 -2.53111494 0.6168063 3.780465
# [2,] -3.702545 -1.42789347 -0.2197196 2.478416
# [3,] -2.890698 -1.88476126 1.9586467 5.268474
# [4,] -3.431133 -2.02626870 1.1153643 3.170689
# [5,] -4.532925 0.02164187 0.9783948 3.162121
# [6,] -2.169391 -1.42699116 0.3214854 4.480305
```

One way to plot all of these observations on the same graph is to do one `plot` call followed by three more `points` or `lines` calls.

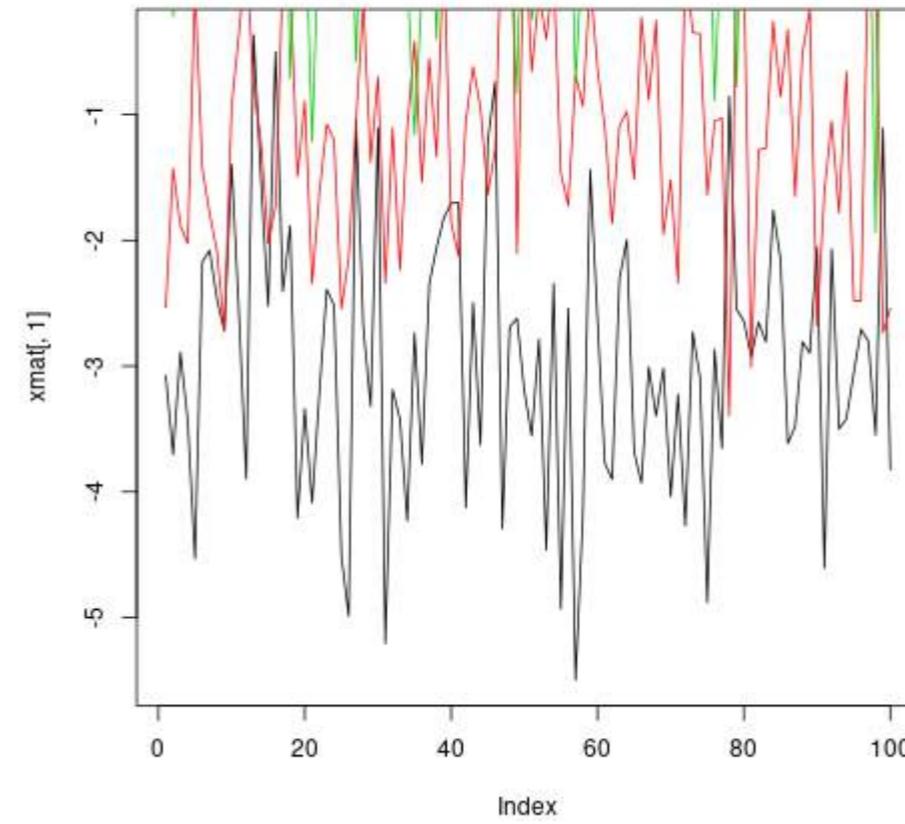
```
plot(xmat[,1], type = 'l')
lines(xmat[,2], col = 'red')
lines(xmat[,3], col = 'green')
lines(xmat[,4], col = 'blue')
```



然而，这种方法既繁琐，又会带来问题，因为默认情况下，`plot`函数会将坐标轴范围固定为仅适合第一列数据。

在这种情况下，更方便的方法是使用`matplotlib`函数，它只需调用一次，且会自动处理坐标轴范围和为每列更改美观设置，使其易于区分。

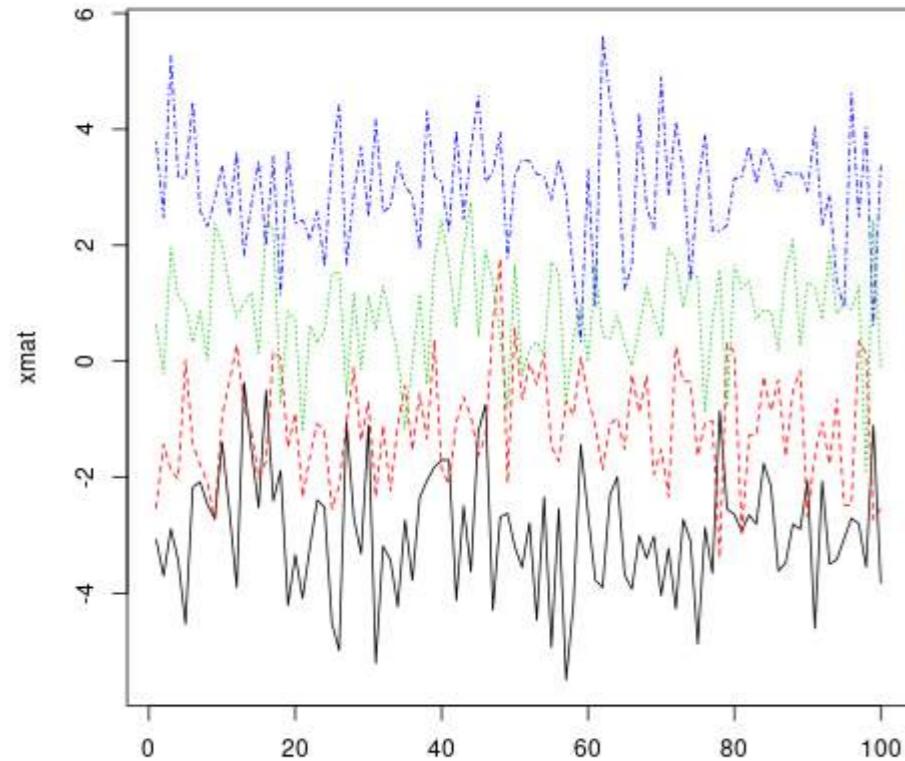
```
matplotlib(xmat, type = 'l')
```



However, this is both tedious, and causes problems because, among other things, by default the axis limits are fixed by `plot` to fit only the first column.

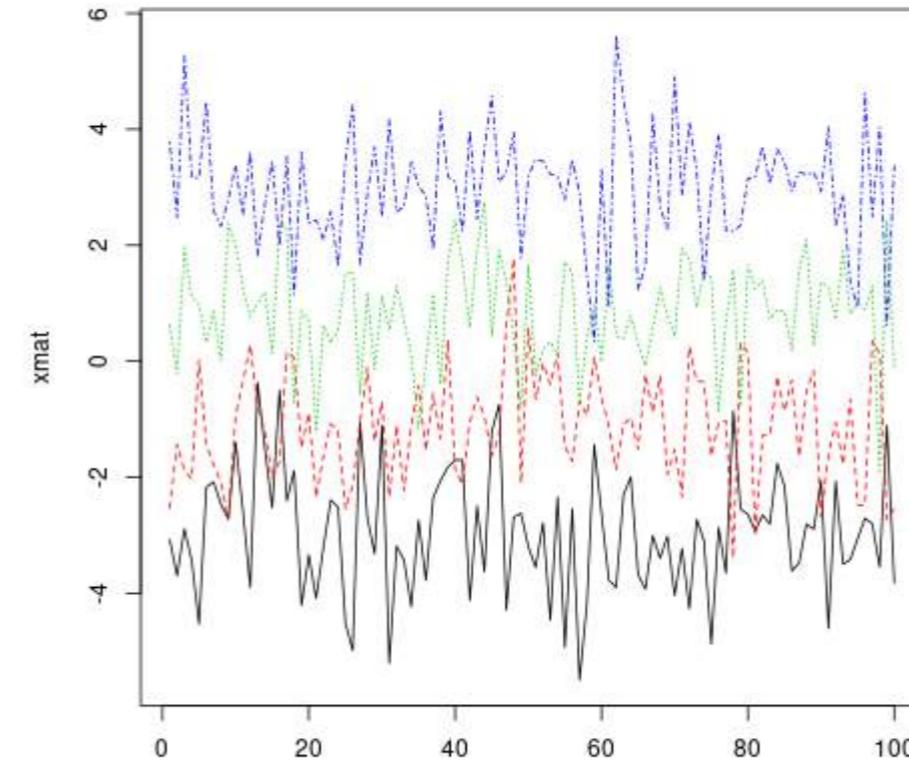
Much more convenient in this situation is to use the `matplotlib` function, which only requires one call and automatically takes care of axis limits *and* changing the aesthetics for each column to make them distinguishable.

```
matplotlib(xmat, type = 'l')
```



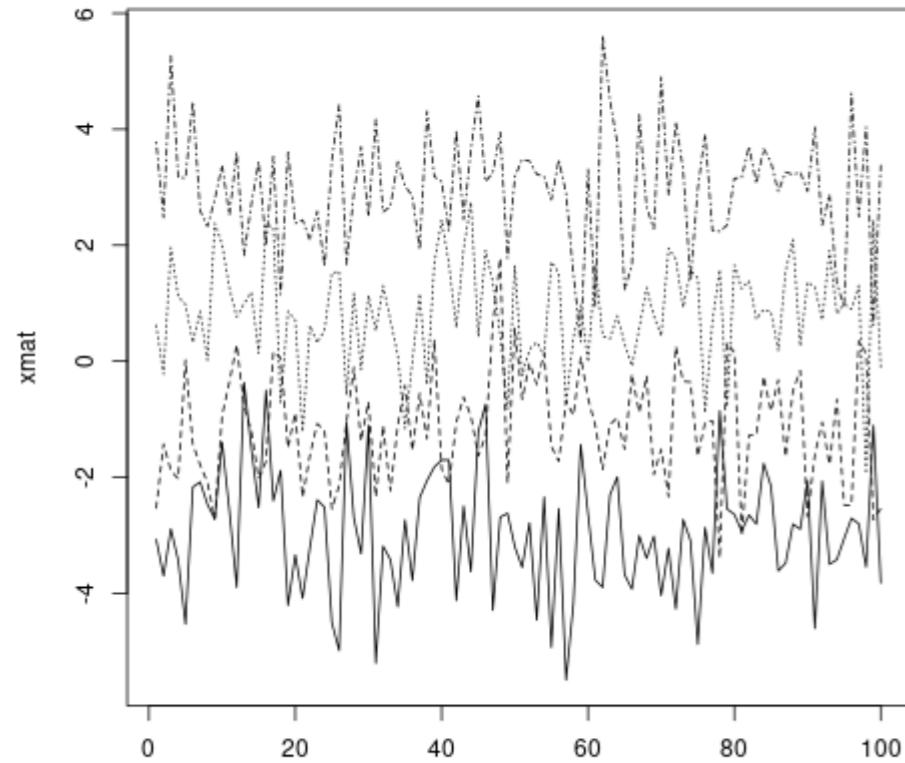
请注意，默认情况下，`matplotlib` 会同时改变颜色（`col`）和线型（`lty`），因为这样可以增加可能的组合数量，避免重复。然而，这些美学属性中的任意一个（或两个）都可以固定为单一值...

```
matplotlib(xmat, type = 'l', col = 'black')
```



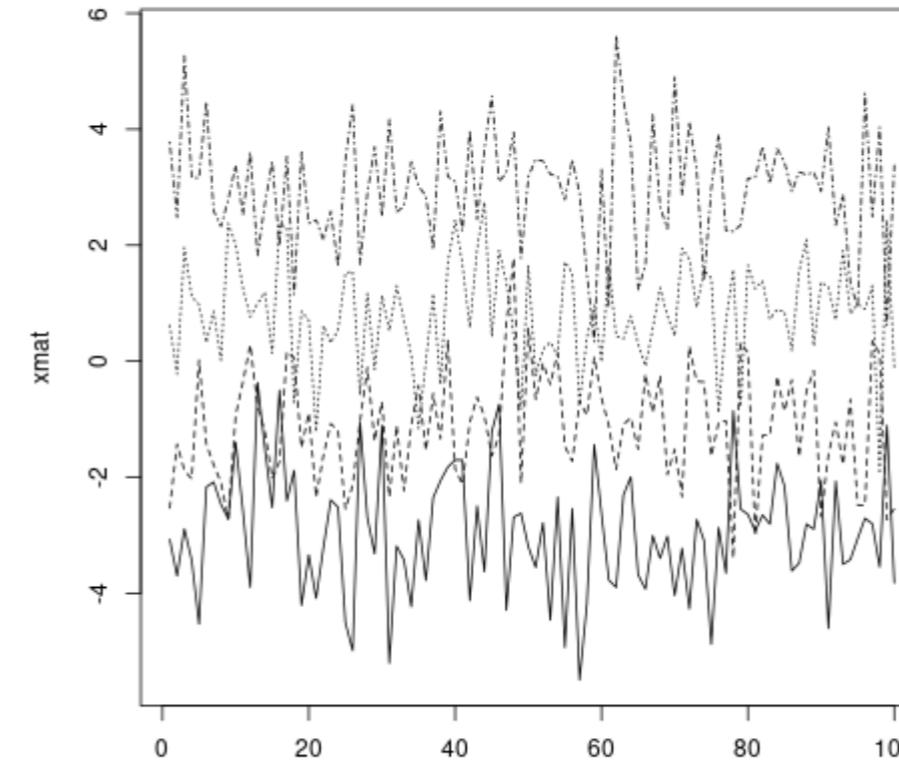
Note that, by default, `matplotlib` varies both color (`col`) and linetype (`lty`) because this increases the number of possible combinations before they get repeated. However, any (or both) of these aesthetics can be fixed to a single value...

```
matplotlib(xmat, type = 'l', col = 'black')
```



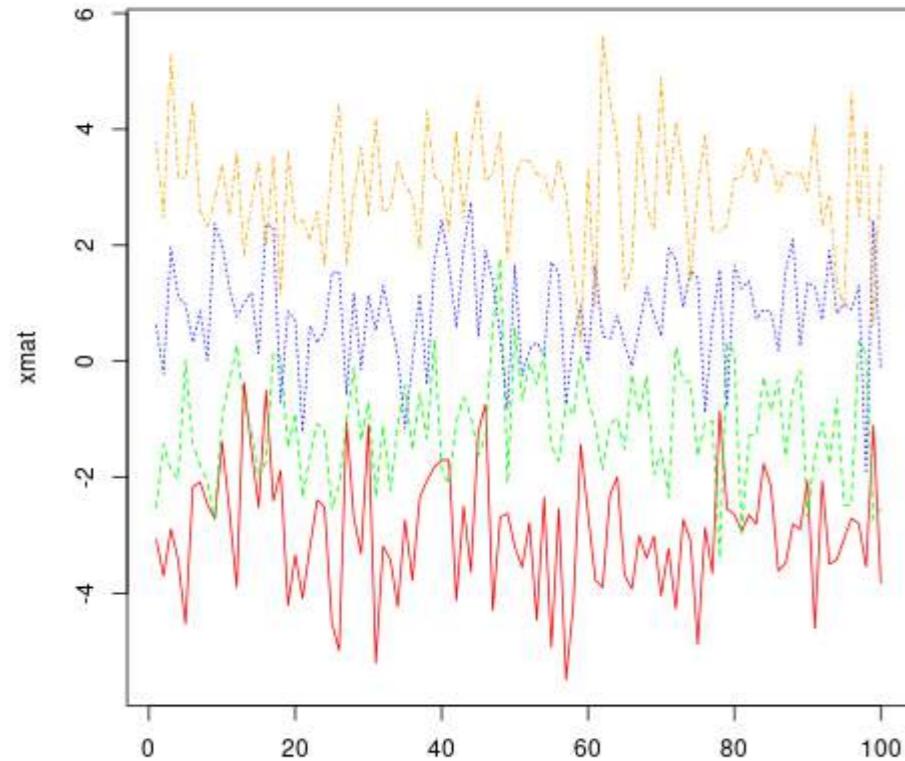
...或者一个自定义向量（该向量会根据标准的R向量回收规则循环至列数）。

```
matplot(xmat, type = 'l', col = c('red', 'green', 'blue', 'orange'))
```



...or a custom vector (which will recycle to the number of columns, following standard R vector recycling rules).

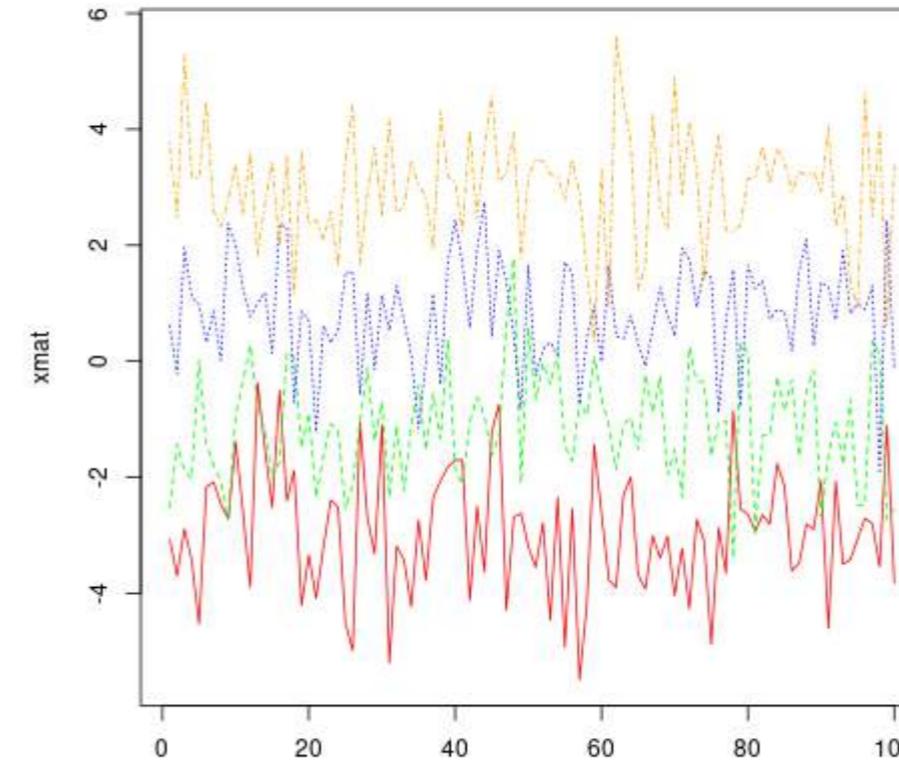
```
matplot(xmat, type = 'l', col = c('red', 'green', 'blue', 'orange'))
```



标准图形参数，包括main、xlab、xmin，使用方式与plot完全相同。更多相关内容请参见?par。

与plot类似，如果只给出一个对象，matplot会假设它是y变量，并使用索引作为x。然而，x和y也可以被显式指定。

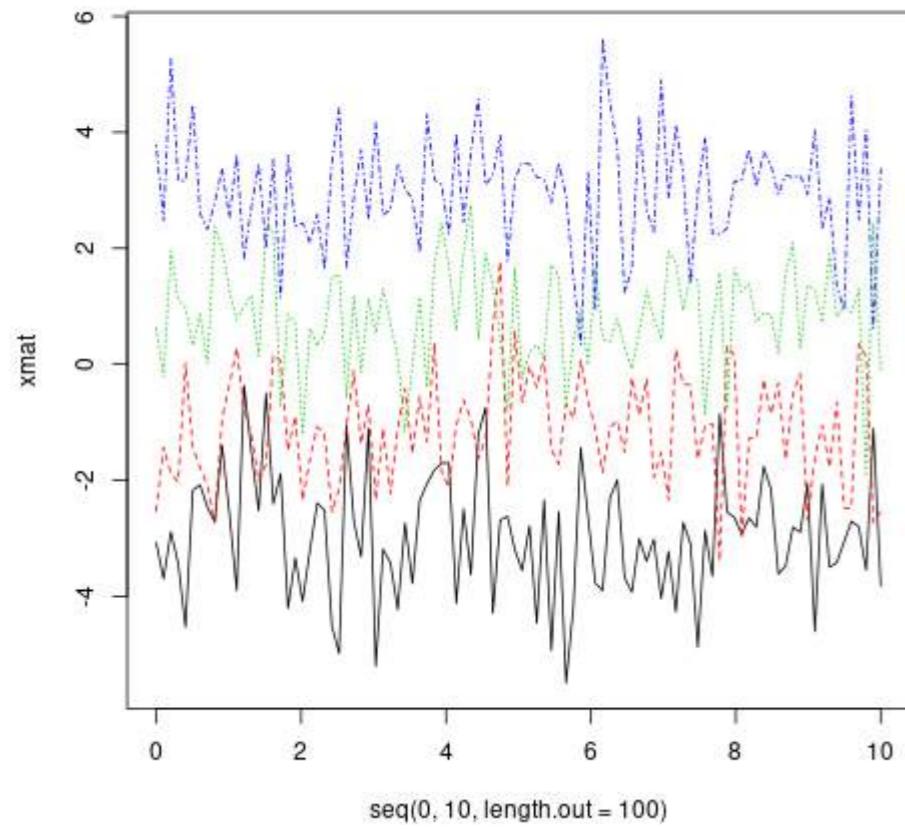
```
matplot(x = seq(0, 10, length.out = 100), y = xmat, type='l')
```



Standard graphical parameters, including main, xlab, xmin, work exactly the same way as for **plot**. For more on those, see [?par](#).

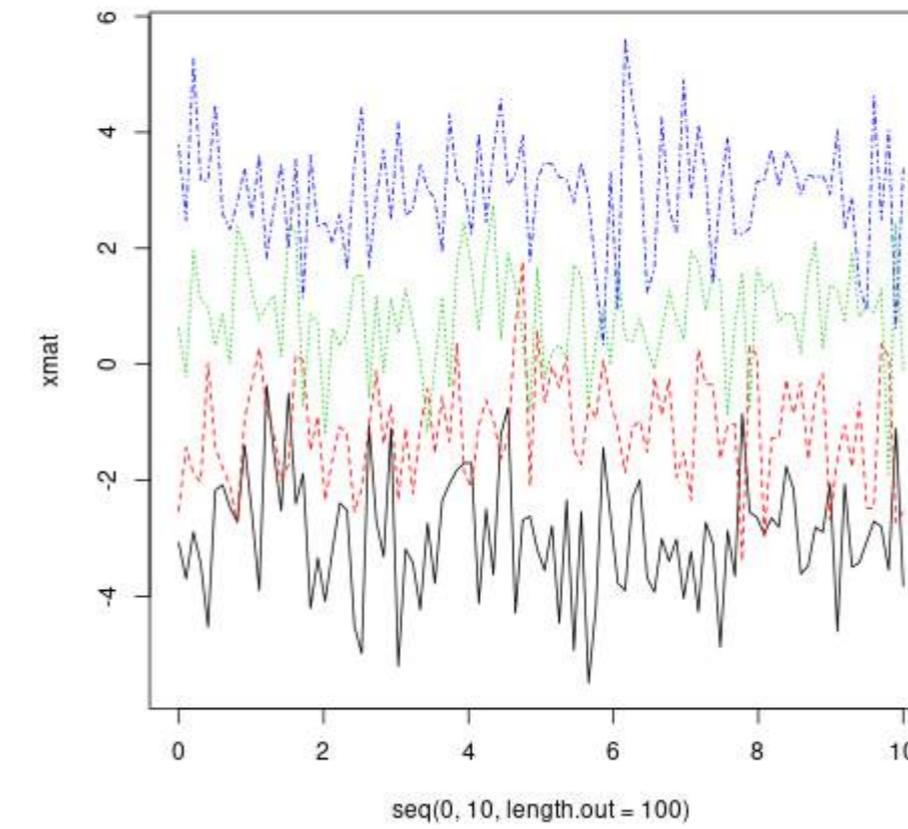
Like **plot**, if given only one object, **matplot** assumes it's the y variable and uses the indices for x. However, x and y can be specified explicitly.

```
matplot(x = seq(0, 10, length.out = 100), y = xmat, type='l')
```



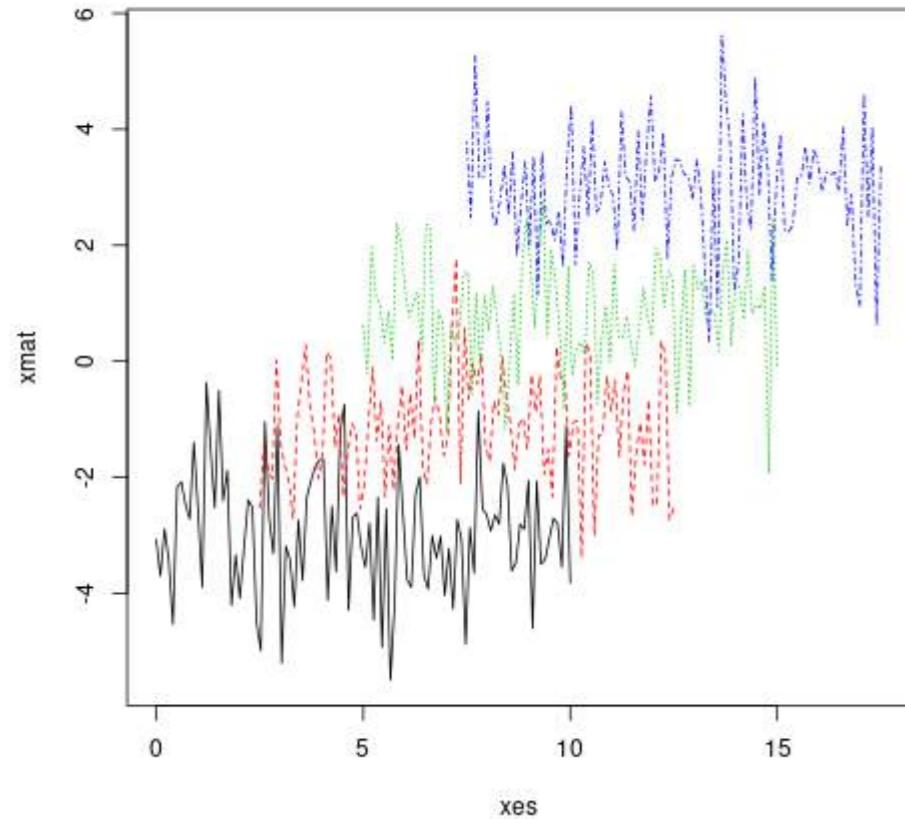
事实上，`x` 和 `y` 都可以是矩阵。

```
xes <- cbind(seq(0, 10, length.out = 100),
             seq(2.5, 12.5, length.out = 100),
             seq(5, 15, length.out = 100),
             seq(7.5, 17.5, length.out = 100))
matplot(x = xes, y = xmat, type = 'l')
```



In fact, both `x` and `y` can be matrices.

```
xes <- cbind(seq(0, 10, length.out = 100),
             seq(2.5, 12.5, length.out = 100),
             seq(5, 15, length.out = 100),
             seq(7.5, 17.5, length.out = 100))
matplot(x = xes, y = xmat, type = 'l')
```

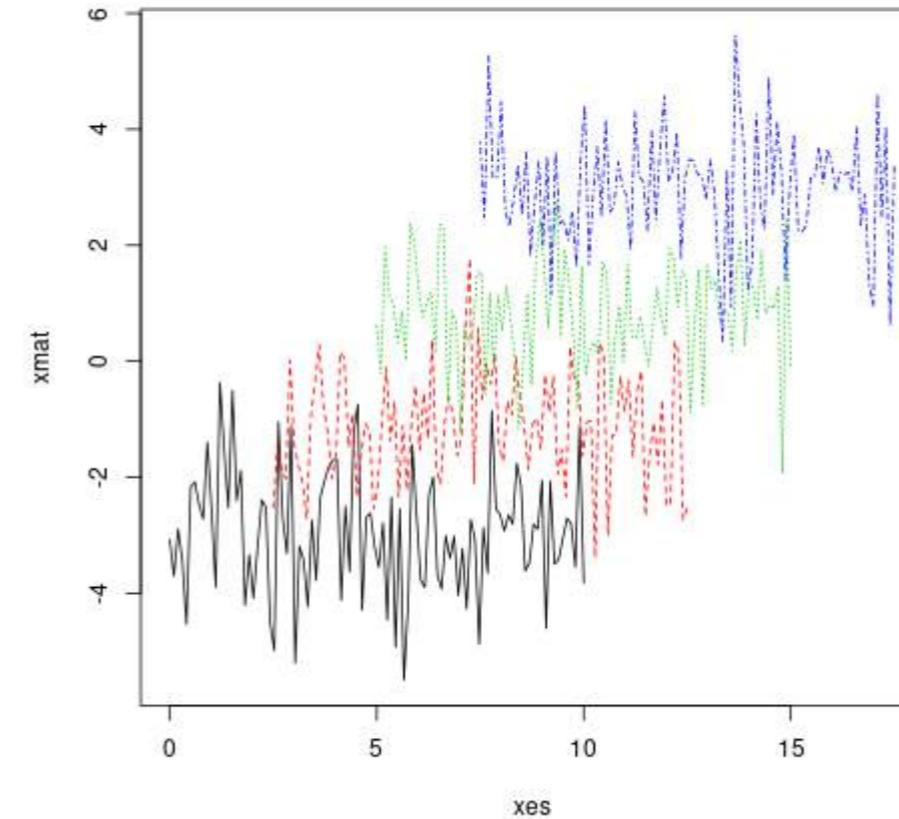


第26.7节：经验累积分布函数

经验累积分布函数是直方图和密度图的一个非常有用且合乎逻辑的后续内容。我们可以使用函数 `ecdf()` 来实现这一目的。通过以下命令可以生成一个基本的图形

```
plot(ecdf(rnorm(100)),main="累积分布",xlab="x")
```

看起来像



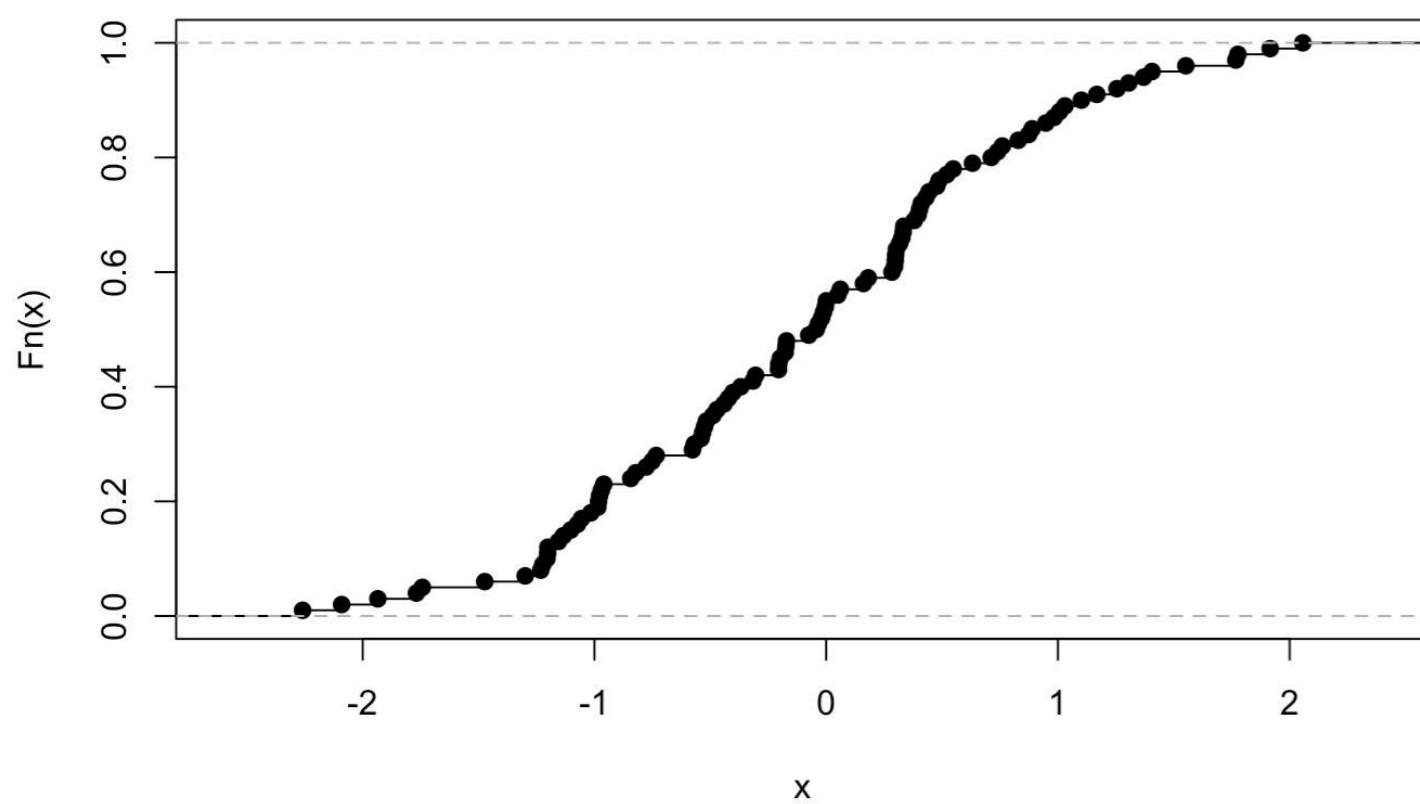
Section 26.7: Empirical Cumulative Distribution Function

A very useful and logical follow-up to histograms and density plots would be the Empirical Cumulative Distribution Function. We can use the function `ecdf()` for this purpose. A basic plot produced by the command

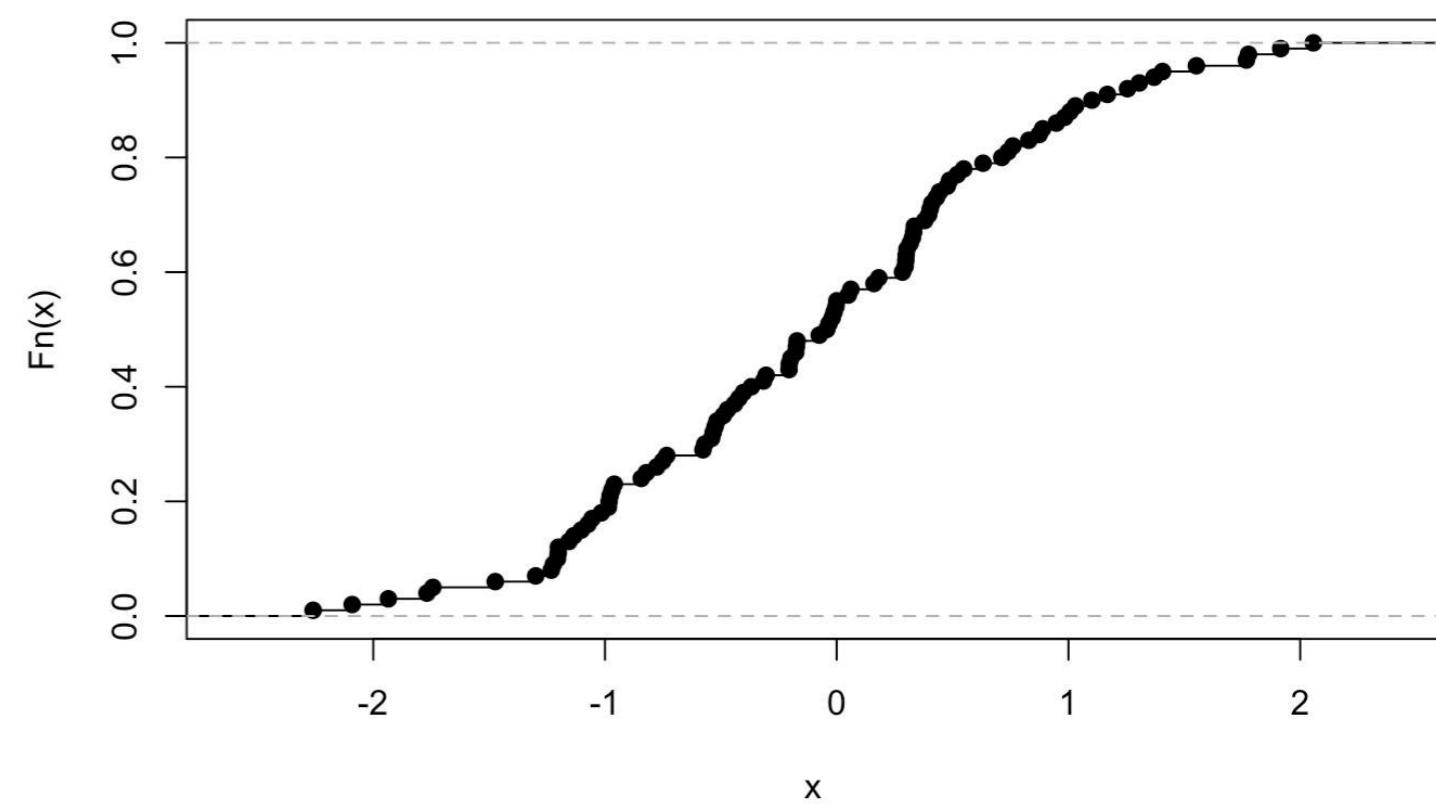
```
plot(ecdf(rnorm(100)),main="Cumulative distribution",xlab="x")
```

would look like

Cumulative distribution



Cumulative distribution



第27章：箱线图

	详细信息（来源 R 文档）
参数	
公式	一个公式，例如 $y \sim grp$ ，其中 y 是一个数值向量，数据值将根据分组变量 grp （通常是因子）进行分组。
数据	一个 <code>data.frame</code> （或列表），公式中的变量应从中获取。
subset	一个可选向量，指定用于绘图的观测子集。
na.action	一个函数，指示当数据包含缺失值（NA）时应执行的操作。默认是忽略响应变量或分组变量中的缺失值。
boxwex	应用于所有箱体的缩放因子。当组数较少时，通过使箱体更窄可以改善图形的外观。
plot	如果为 <code>TRUE</code> （默认），则生成箱线图。如果不是，则返回箱线图所基于的汇总数据。
col	如果 <code>col</code> 非空，则假定其包含用于给箱体着色的颜色。默认情况下，颜色为背景色。

第27.1节：使用 `boxplot()` 创建箱线图 {graphics}

此示例使用默认的`boxplot()`函数和`iris`数据框。

```
> head(iris)
Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5         1.4         0.2   setosa
2          4.9         3.0         1.4         0.2   setosa
3          4.7         3.2         1.3         0.2   setosa
4          4.6         3.1         1.5         0.2  versicolor
5          5.0         3.6         1.4         0.2  versicolor
6          5.4         3.9         1.7         0.4  versicolor
```

简单箱线图（萼片长度）

创建数值变量的箱线图

```
boxplot(iris[, 1], xlab="萼片长度", ylab="长度 (厘米) ",
        main="萼片长度的摘要特征 (鸢尾花数据) ")
```

Chapter 27: boxplot

	Parameters	Details (source R Documentation)
formula	a formula, such as $y \sim grp$, where y is a numeric vector of data values to be split into groups according to the grouping variable grp (usually a factor).	
data	a <code>data.frame</code> (or list) from which the variables in formula should be taken.	
subset	an optional vector specifying a subset of observations to be used for plotting.	
na.action	a function which indicates what should happen when the data contain NAs. The default is to ignore missing values in either the response or the group.	
boxwex	a scale factor to be applied to all boxes. When there are only a few groups, the appearance of the plot can be improved by making the boxes narrower.	
plot	if <code>TRUE</code> (the default) then a boxplot is produced. If not, the summaries which the boxplots are based on are returned.	
col	if <code>col</code> is non-null it is assumed to contain colors to be used to colour the bodies of the box plots. By default they are in the background colour.	

Section 27.1: Create a box-and-whisker plot with `boxplot()` {graphics}

This example use the default `boxplot()` function and the `iris` data frame.

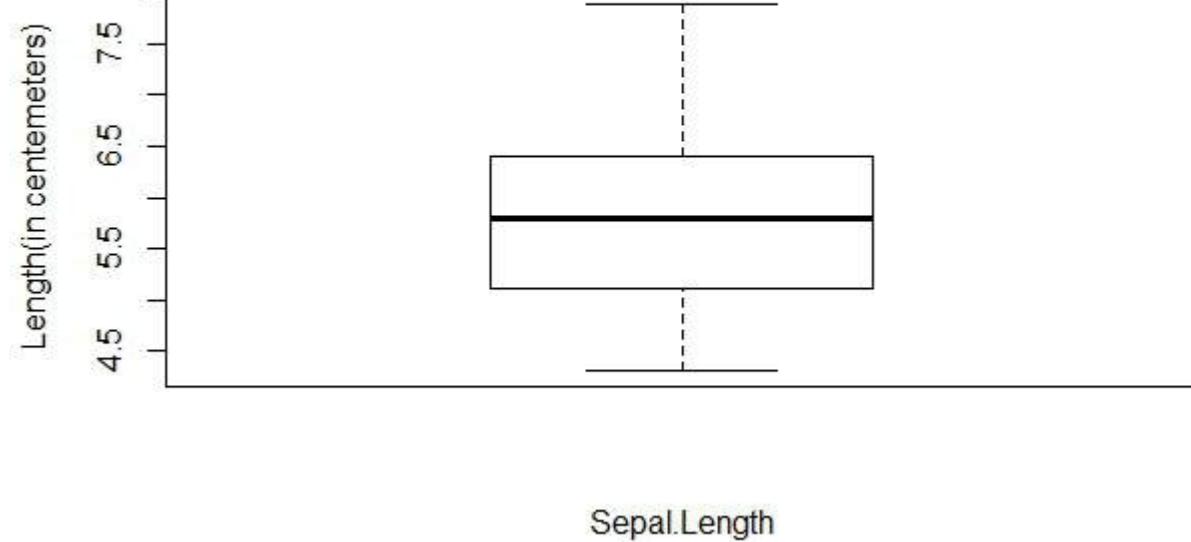
```
> head(iris)
Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5         1.4         0.2   setosa
2          4.9         3.0         1.4         0.2   setosa
3          4.7         3.2         1.3         0.2   setosa
4          4.6         3.1         1.5         0.2  versicolor
5          5.0         3.6         1.4         0.2  versicolor
6          5.4         3.9         1.7         0.4  versicolor
```

Simple boxplot (Sepal.Length)

Create a box-and-whisker graph of a numerical variable

```
boxplot(iris[, 1], xlab="Sepal.Length", ylab="Length(in centimeters)",
        main="Summary Charateristics of Sepal.Length(Iris Data)")
```

Summary Characteristics of Sepal.Length(Iris Data)



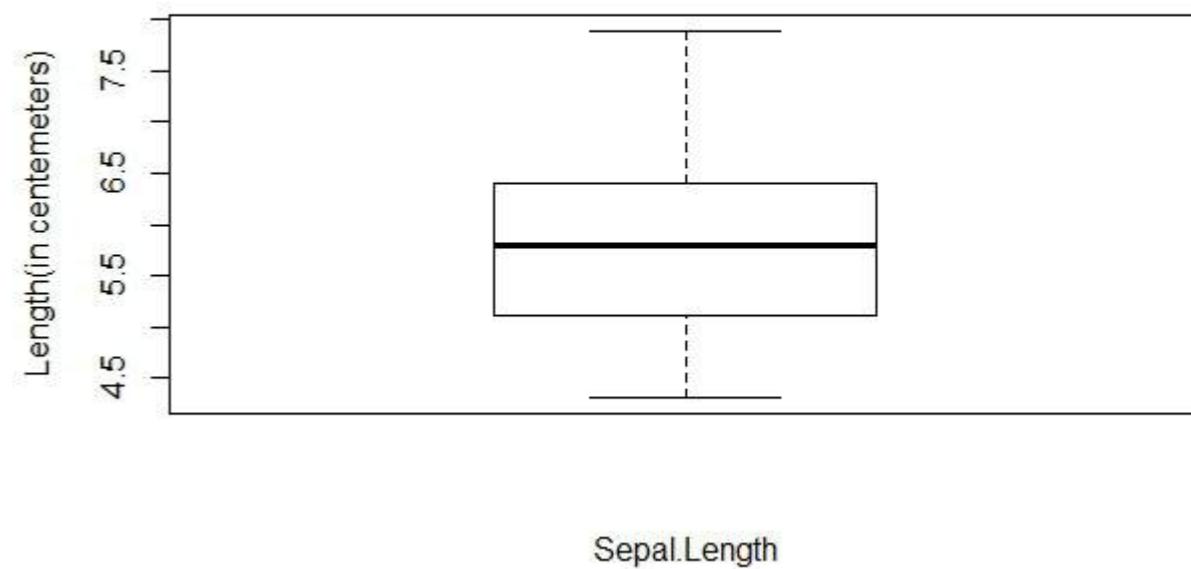
Sepal.Length

按物种分组的萼片长度箱线图

创建按分类变量分组的数值变量箱线图

```
boxplot(萼片长度~物种,data = iris)
```

Summary Characteristics of Sepal.Length(Iris Data)

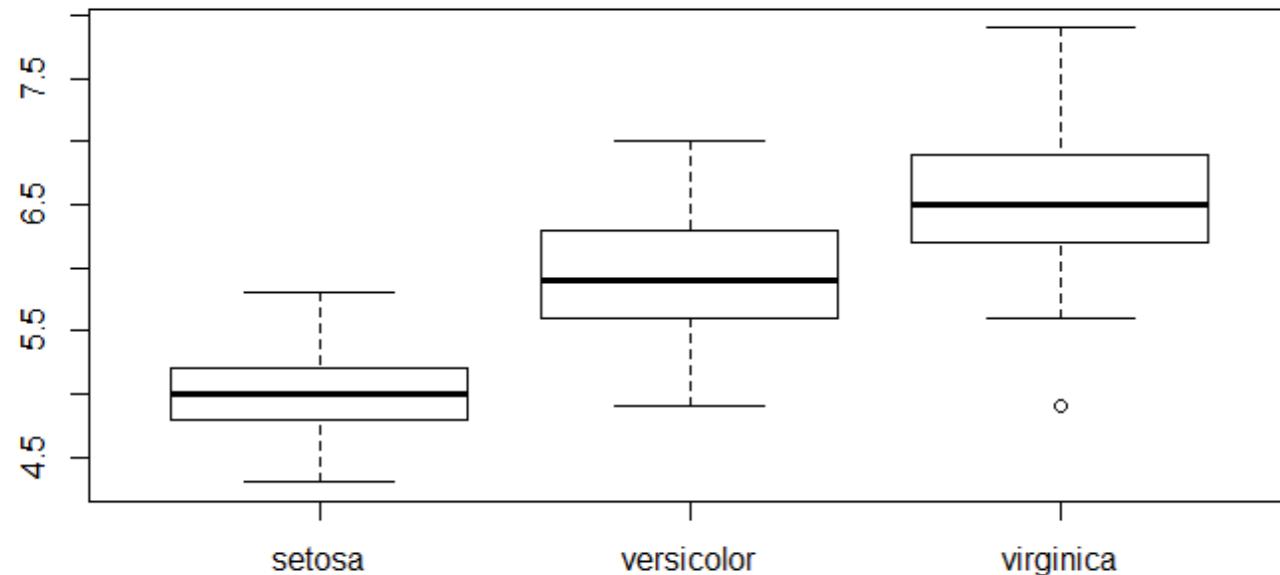


Sepal.Length

Boxplot of sepal length grouped by species

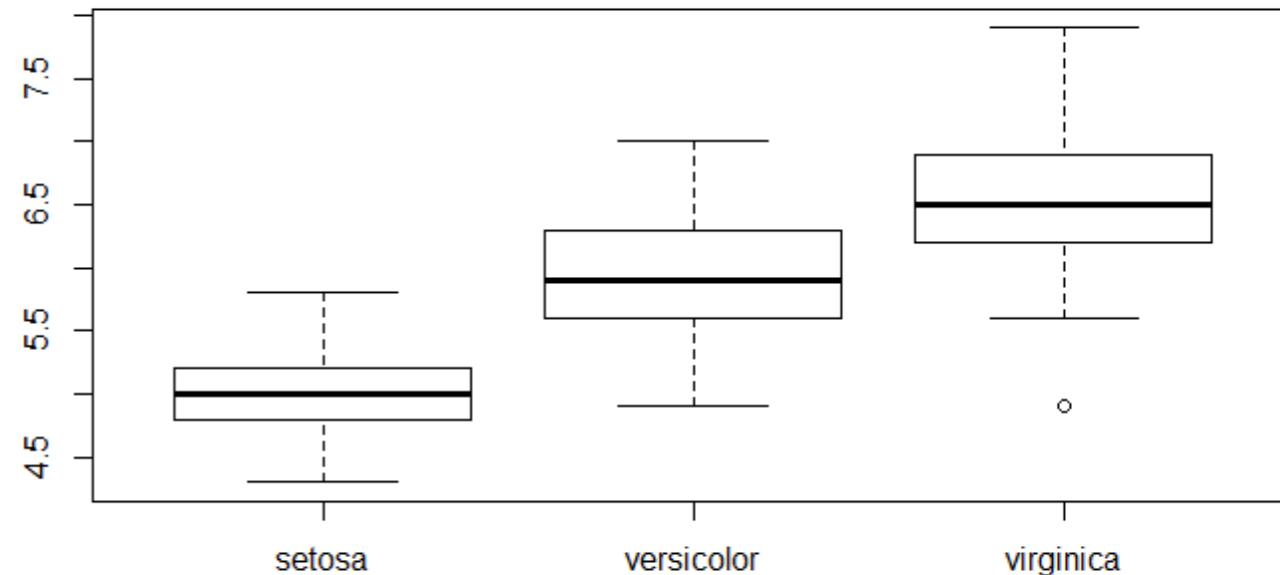
Create a boxplot of a numerical variable grouped by a categorical variable

```
boxplot(Sepal.Length~Species,data = iris)
```



调整顺序

要改变图中箱线的顺序，必须更改分类变量的水平顺序。

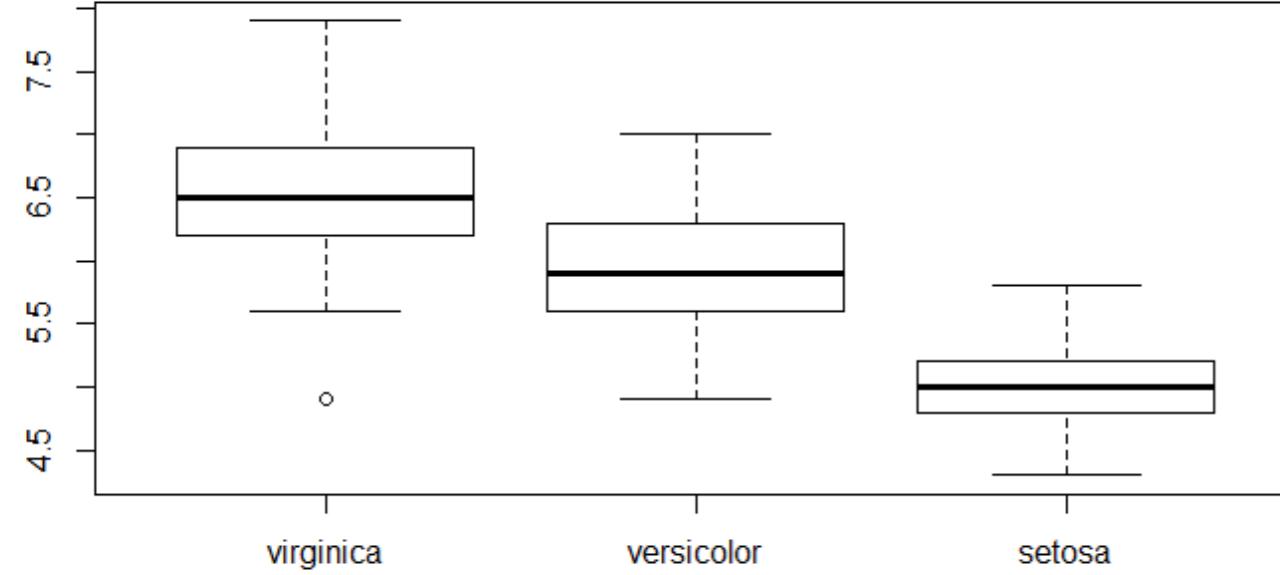


Bring order

To change order of the box in the plot you have to change the order of the categorical variable's levels.

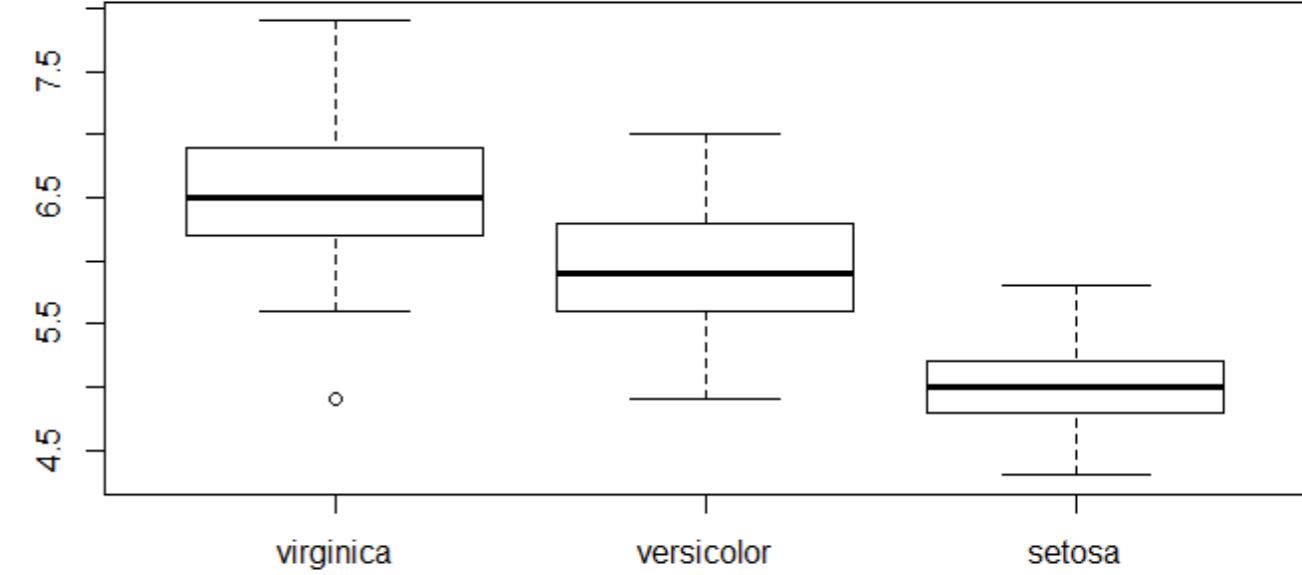
例如，如果我们想要顺序为山鸢尾-变色鸢尾-杂色鸢尾

```
newSpeciesOrder <- factor(iris$Species, levels=c("virginica", "versicolor", "setosa"))
boxplot(Sepal.Length~newSpeciesOrder, data = iris)
```



For example if we want to have the order virginica - versicolor - setosa

```
newSpeciesOrder <- factor(iris$Species, levels=c("virginica", "versicolor", "setosa"))
boxplot(Sepal.Length~newSpeciesOrder, data = iris)
```



更改组名

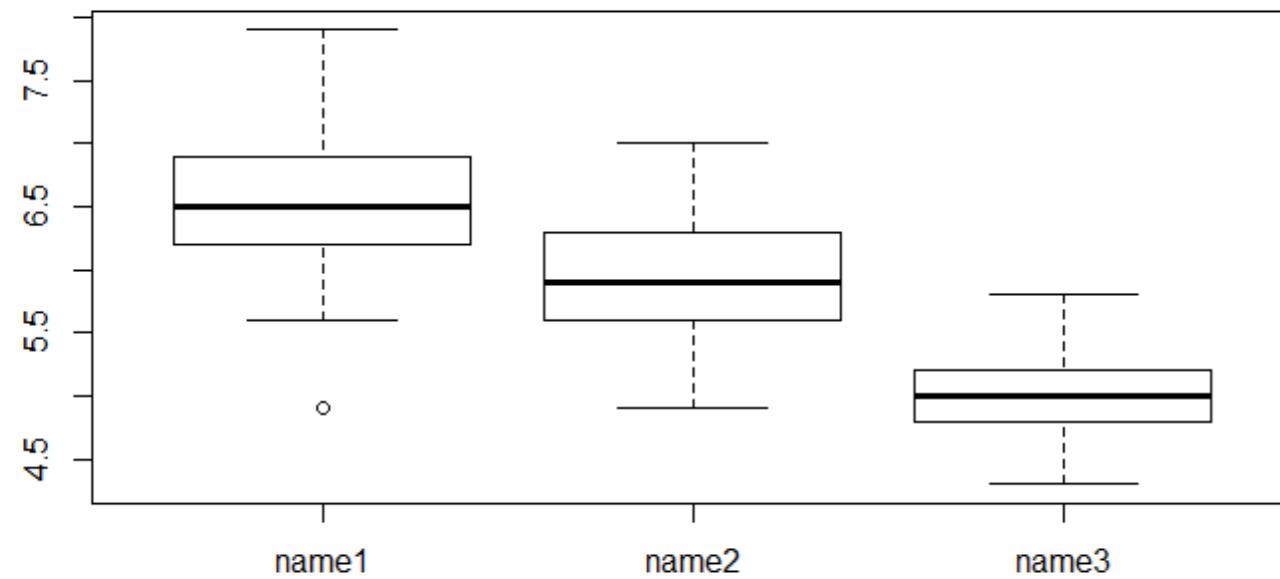
如果你想为你的组指定更好的名称，可以使用Names参数。它接受一个与分类变量水平数量相同的向量

```
boxplot(Sepal.Length~newSpeciesOrder, data = iris, names= c("name1", "name2", "name3"))
```

Change groups names

If you want to specify a better name to your groups you can use the Names parameter. It take a vector of the size of the levels of categorical variable

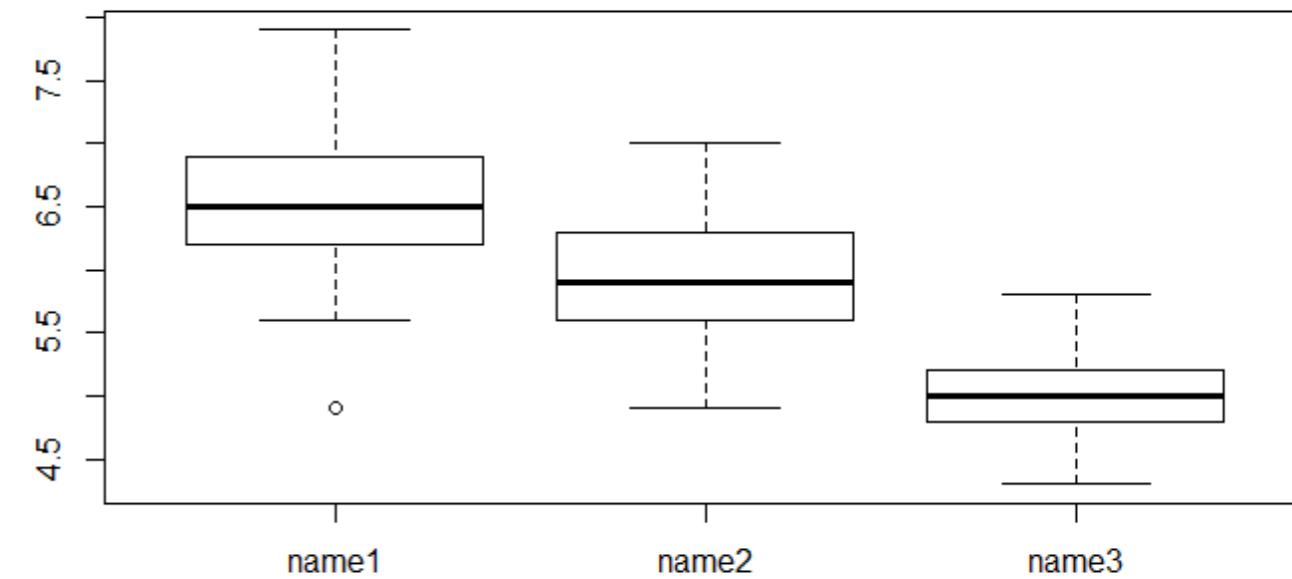
```
boxplot(Sepal.Length~newSpeciesOrder, data = iris, names= c("name1", "name2", "name3"))
```



小改进
颜色

col : 添加一个与分类变量水平数量相同的向量

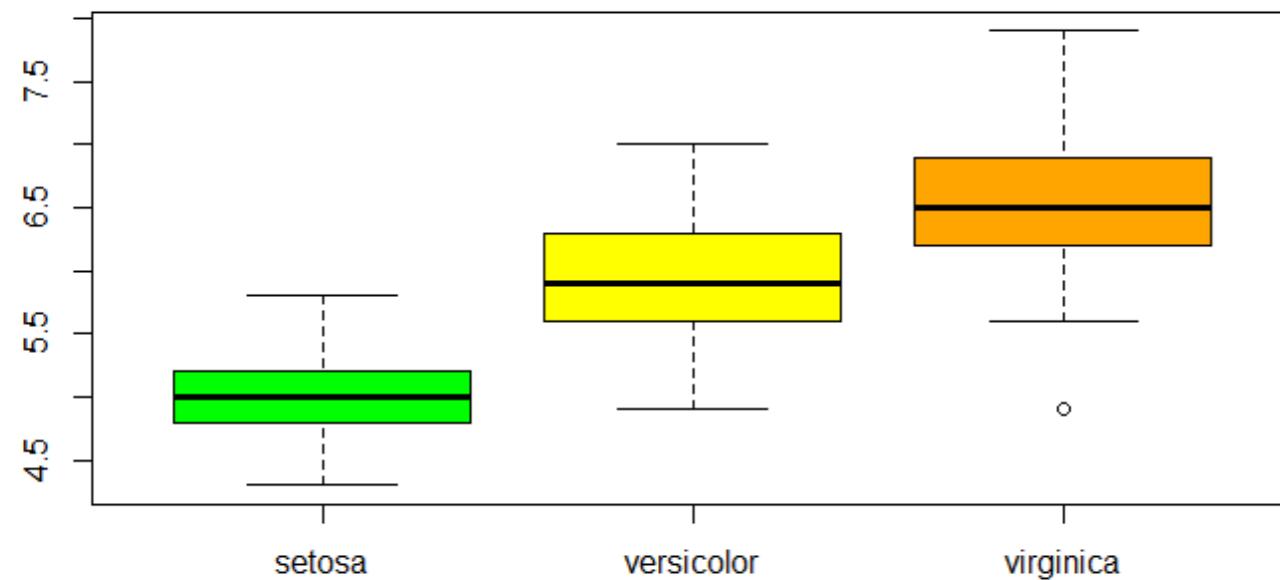
```
boxplot(Sepal.Length~Species,data = iris,col=c("green","yellow","orange"))
```



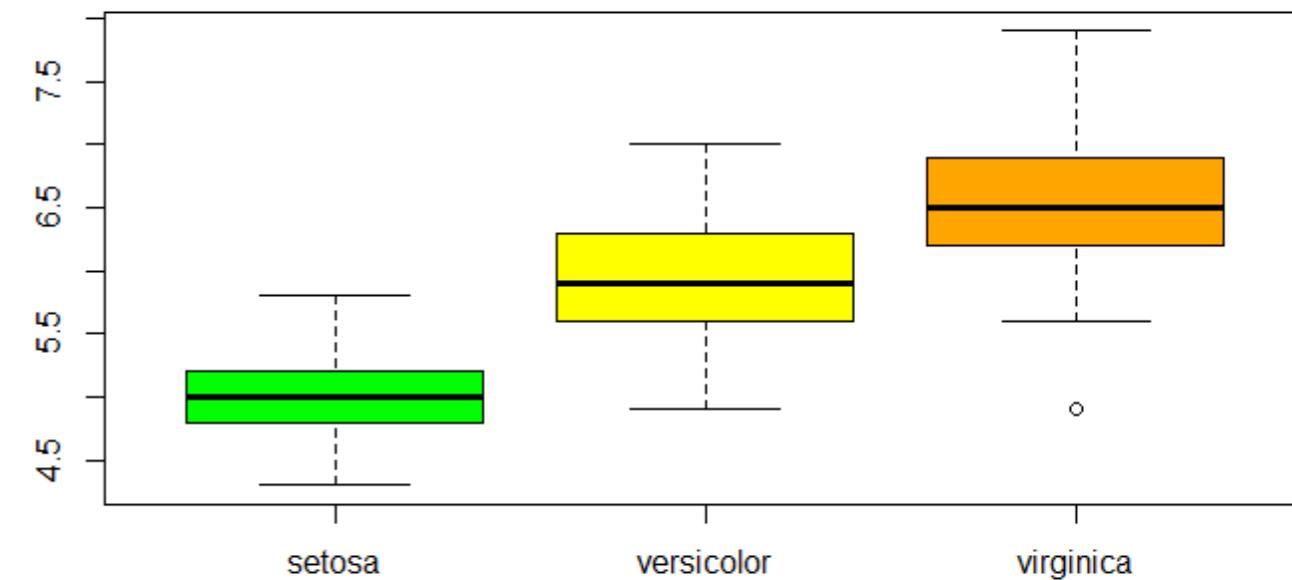
Small improvements
Color

col : add a vector of the size of the levels of categorical variable

```
boxplot(Sepal.Length~Species,data = iris,col=c("green","yellow","orange"))
```



盒子的接近度

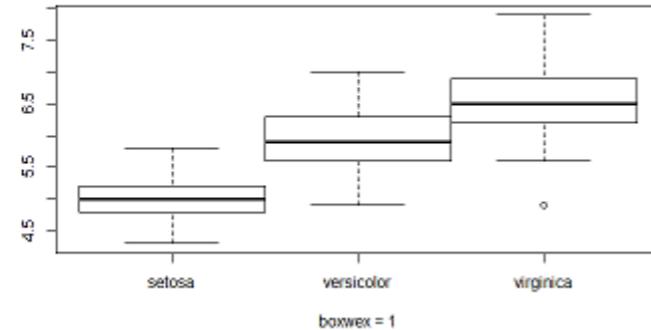
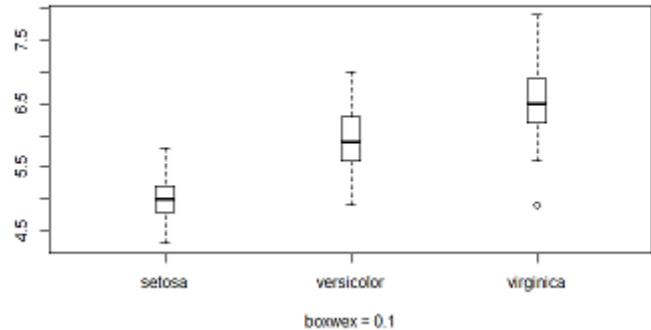


Proximity of the box

boxwex : 设置箱体之间的间距。

左侧箱线图(萼片长度~物种,数据= 鸢尾花,boxwex = 0.1)

右侧箱线图(萼片长度~物种,数据= 鸢尾花,boxwex = 1)



查看箱线图所基于的汇总 `plot=FALSE`

要查看汇总，必须将参数 `plot` 设置为 `FALSE`。

给出各种结果

```
> 箱线图(萼片长度~新物种顺序,数据= 鸢尾花,plot=FALSE)
```

```
$stats #三个组数值变量的汇总
```

```
[,1] [,2] [,3]
```

```
[1,] 5.6 4.9 4.3 # 极值
```

```
[2,] 6.2 5.6 4.8 # 第一四分位数界限
```

```
[3,] 6.5 5.9 5.0 # 中位数界限
```

```
[4,] 6.9 6.3 5.2 # 第三四分位数界限
```

```
[5,] 7.9 7.0 5.8 # 极值
```

```
$n #每组的观测数量
```

```
[1] 50 50 50
```

```
$conf #缺口的极值
```

```
[,1] [,2] [,3]
```

```
[1,] 6.343588 5.743588 4.910622
```

```
[2,] 6.656412 6.056412 5.089378
```

```
$out #极值
```

```
[1] 4.9
```

```
$group #包含极值的组
```

```
[1] 1
```

```
$names #组名
```

```
[1] "virginica" "versicolor" "setosa"
```

第27.2节：附加箱线图样式参数

箱体

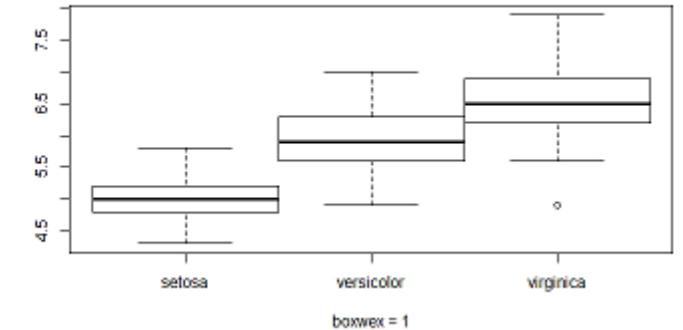
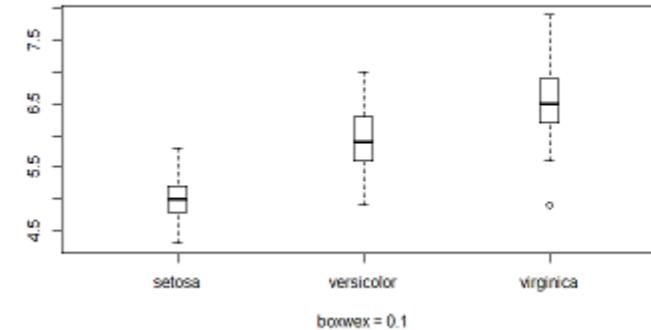
- `boxty` - 箱线类型
- `boxlwd` - 箱线宽度
- `boxcol` - 箱线颜色
- `boxfill` - 箱体填充颜色

中位数

boxwex: set the margin between boxes.

Left `boxplot`(Sepal.Length~Species, data = iris, boxwex = 0.1)

Right `boxplot`(Sepal.Length~Species, data = iris, boxwex = 1)



See the summaries which the boxplots are based `plot=FALSE`

To see a summary you have to put the parameter `plot` to `FALSE`.

Various results are given

```
> boxplot(Sepal.Length~newSpeciesOrder, data = iris, plot=FALSE)
```

```
$stats #summary of the numerical variable for the 3 groups
```

```
[,1] [,2] [,3]
```

```
[1,] 5.6 4.9 4.3 # extreme value
```

```
[2,] 6.2 5.6 4.8 # first quartile limit
```

```
[3,] 6.5 5.9 5.0 # median limit
```

```
[4,] 6.9 6.3 5.2 # third quartile limit
```

```
[5,] 7.9 7.0 5.8 # extreme value
```

```
$n #number of observations in each groups
```

```
[1] 50 50 50
```

```
$conf #extreme value of the notches
```

```
[,1] [,2] [,3]
```

```
[1,] 6.343588 5.743588 4.910622
```

```
[2,] 6.656412 6.056412 5.089378
```

```
$out #extreme value
```

```
[1] 4.9
```

```
$group #group in which are the extreme value
```

```
[1] 1
```

```
$names #groups names
```

```
[1] "virginica" "versicolor" "setosa"
```

Section 27.2: Additional boxplot style parameters

Box

- `boxty` - box line type
- `boxlwd` - box line width
- `boxcol` - box line color
- `boxfill` - box fill colors

Median

- medlty - 中位数线类型 ("blank"表示无线)
- mediwd - 中位数线宽度
- medcol - 中位数线颜色
- medpch - 中位数点符号 (NA表示无符号)
- medcex - 中位数点大小
- medbg - 中位点背景色

须状线

- whisklty - 须状线类型
- whisklwd - 须状线宽度
- whiskcol - 须状线颜色

订书钉

- staplelty - 订书钉线类型
- staplelwd - 订书钉线宽度
- staplecol - 订书钉线颜色

异常值

- outlty - 异常值线类型 ("blank"表示无线)
- outlwd - 异常值线宽度
- outcol - 异常值线颜色
- outpch - 异常值点类型 (NA表示无符号)
- outcex - 异常值点大小
- outbg - 异常值点背景颜色

示例

默认图和大幅修改图并排显示

```
par(mfrow=c(1,2))
# 默认
boxplot(Sepal.Length ~ Species, data=iris)
# 修改后
boxplot(Sepal.Length ~ Species, data=iris,
        boxlty=2, boxlwd=3, boxfill="cornflowerblue", boxcol="darkblue",
        mediwd=2, medcol="red", medpch=21, medcex=1, medbg="white",
        whisklty=2, whisklwd=3, whiskcol="darkblue",
        staplelty=2, staplelwd=2, staplecol="red",
        outlty=3, outlwd=3, outcol="grey", outpch=NA
      )
```

- medlty - median line type ("blank" for no line)
- mediwd - median line width
- medcol - median line color
- medpch - median point (NA for no symbol)
- medcex - median point size
- medbg - median point background color

Whisker

- whisklty - whisker line type
- whisklwd - whisker line width
- whiskcol - whisker line color

Staple

- staplelty - staple line type
- staplelwd - staple line width
- staplecol - staple line color

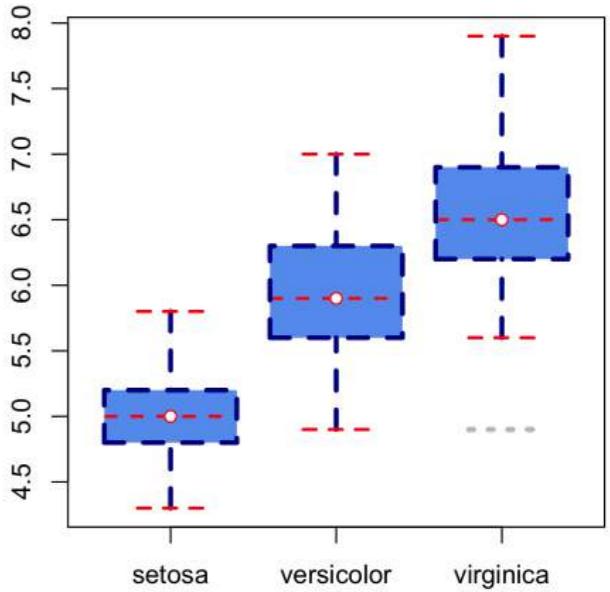
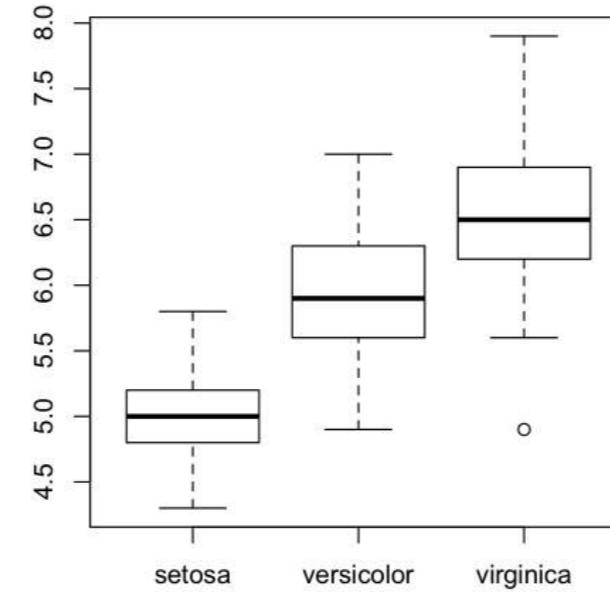
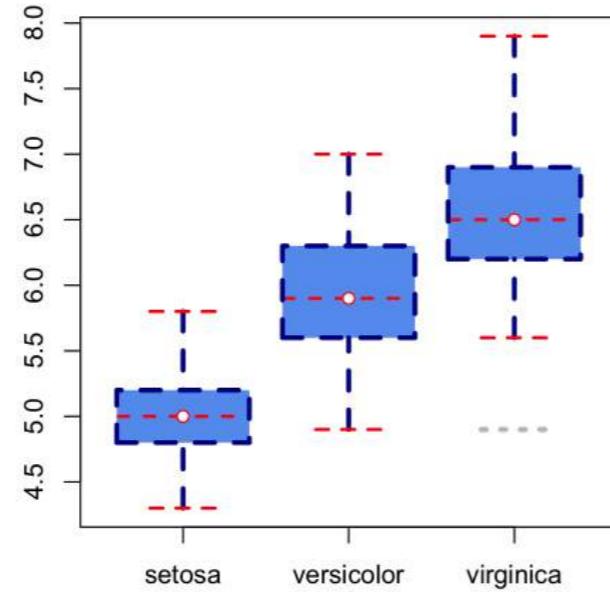
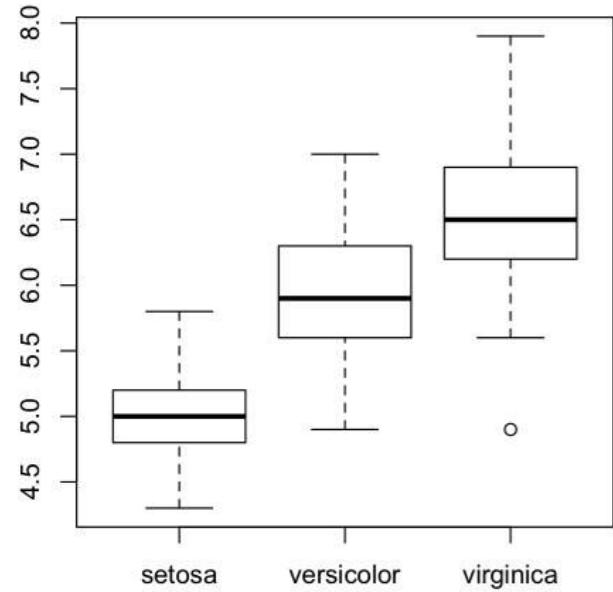
Outliers

- outlty - outlier line type ("blank" for no line)
- outlwd - outlier line width
- outcol - outlier line color
- outpch - outlier point type (NA for no symbol)
- outcex - outlier point size
- outbg - outlier point background color

Example

Default and heavily modified plots side by side

```
par(mfrow=c(1,2))
# Default
boxplot(Sepal.Length ~ Species, data=iris)
# Modified
boxplot(Sepal.Length ~ Species, data=iris,
        boxlty=2, boxlwd=3, boxfill="cornflowerblue", boxcol="darkblue",
        mediwd=2, medcol="red", medpch=21, medcex=1, medbg="white",
        whisklty=2, whisklwd=3, whiskcol="darkblue",
        staplelty=2, staplelwd=2, staplecol="red",
        outlty=3, outlwd=3, outcol="grey", outpch=NA
      )
```



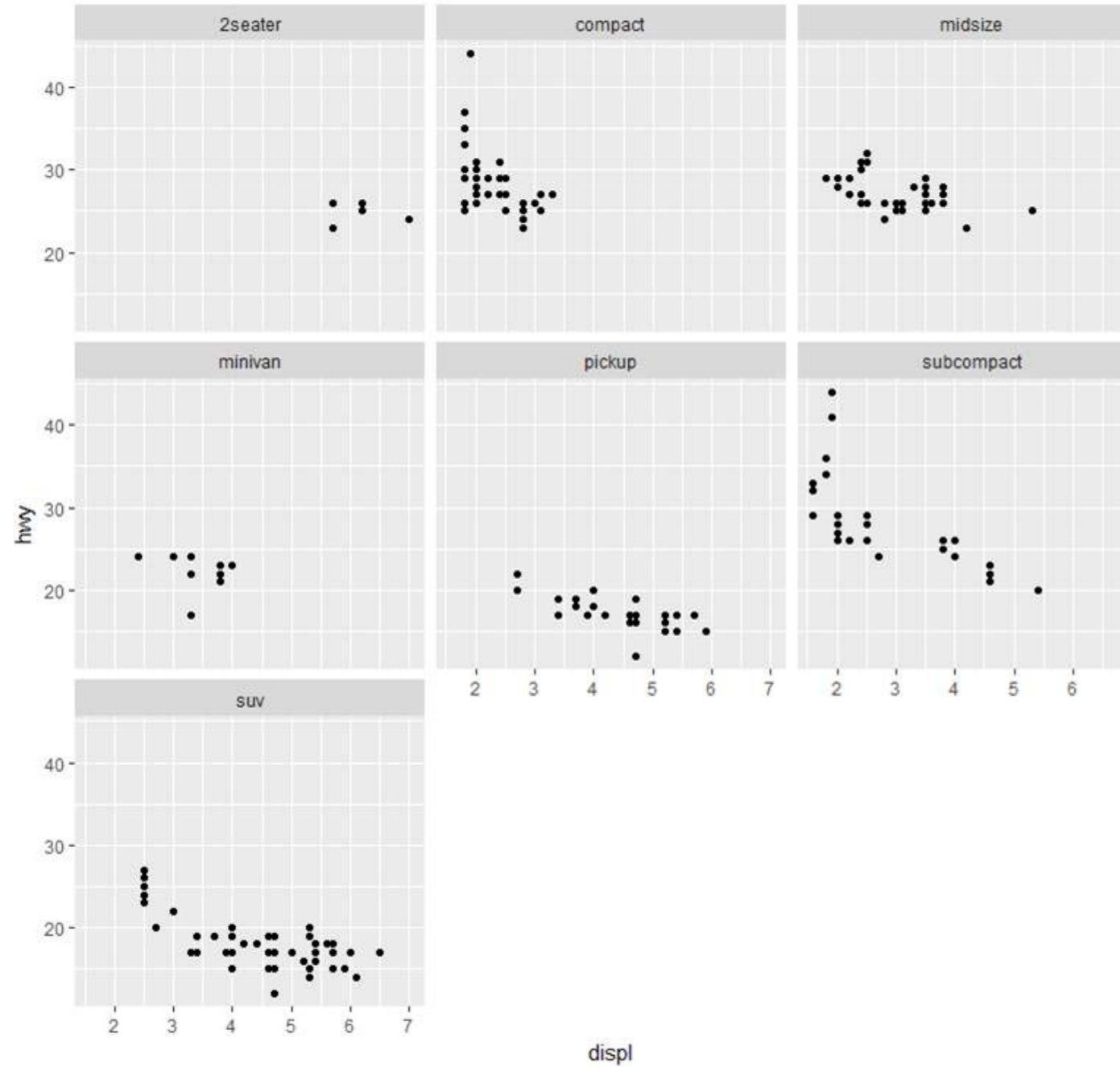
第28章：ggplot2

第28.1节：显示多个图表

使用不同的facet函数在一张图中显示多个图表。这种方法的一个优点是所有坐标轴在各图表间共享相同的刻度，便于一目了然地进行比较。我们将使用包含在ggplot2中的mpg数据集。

按行换行排列图表（尝试创建方形布局）：

```
ggplot(mpg, aes(x = displ, y = hwy)) +  
  geom_point() +  
  facet_wrap(~class)
```



在一行多列显示多个图表：

```
ggplot(mpg, aes(x = displ, y = hwy)) +
```

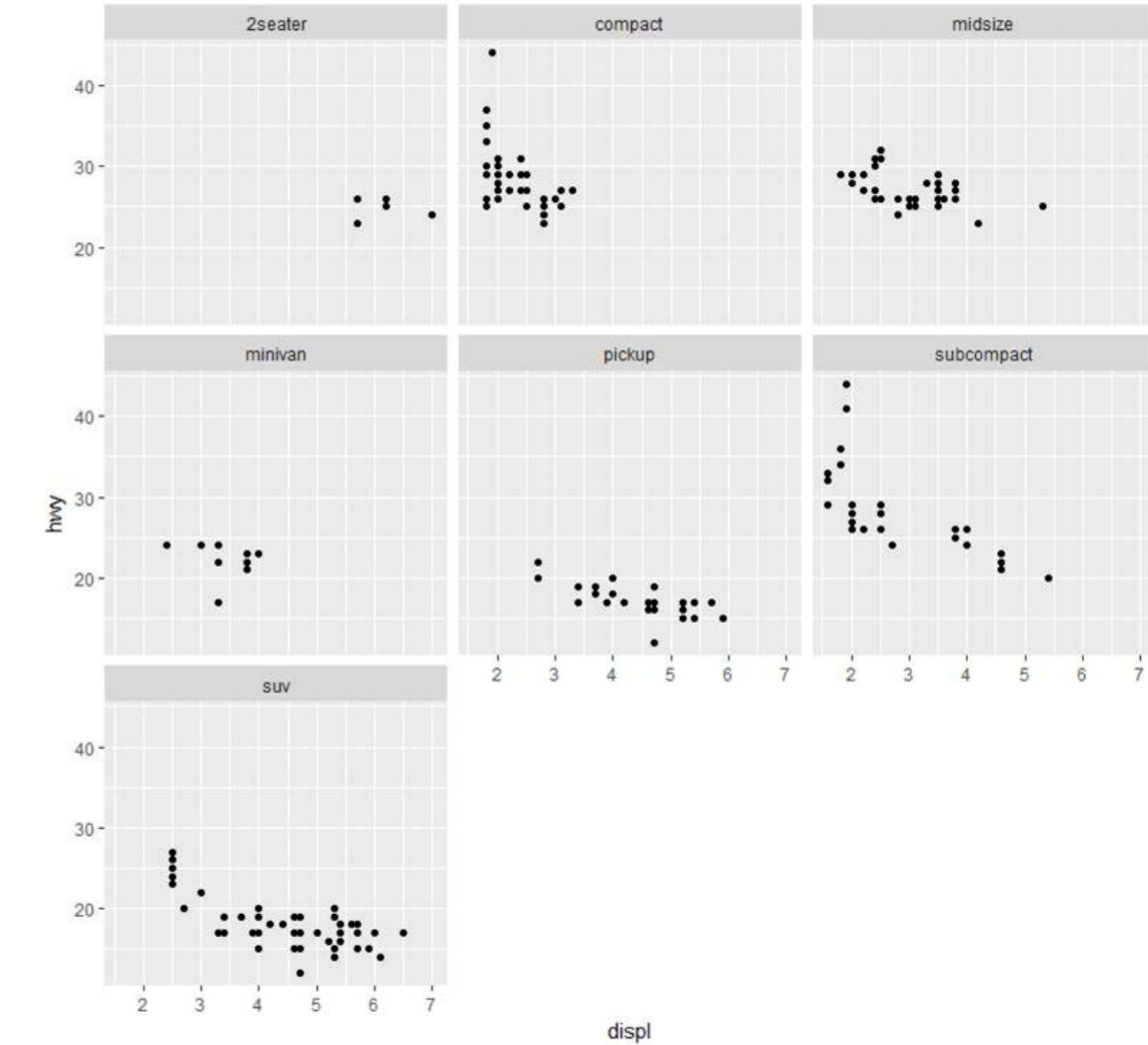
Chapter 28: ggplot2

Section 28.1: Displaying multiple plots

Display multiple plots in one image with the different facet functions. An advantage of this method is that all axes share the same scale across charts, making it easy to compare them at a glance. We'll use the mpg dataset included in ggplot2.

Wrap charts line by line (attempts to create a square layout):

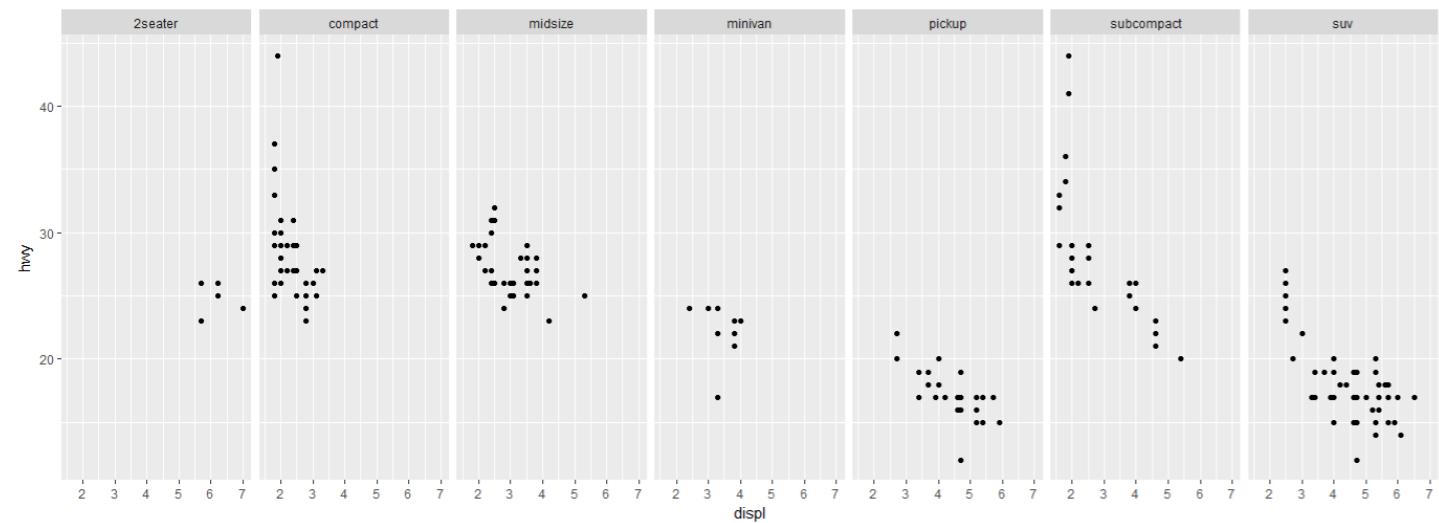
```
ggplot(mpg, aes(x = displ, y = hwy)) +  
  geom_point() +  
  facet_wrap(~class)
```



Display multiple charts on one row, multiple columns:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
```

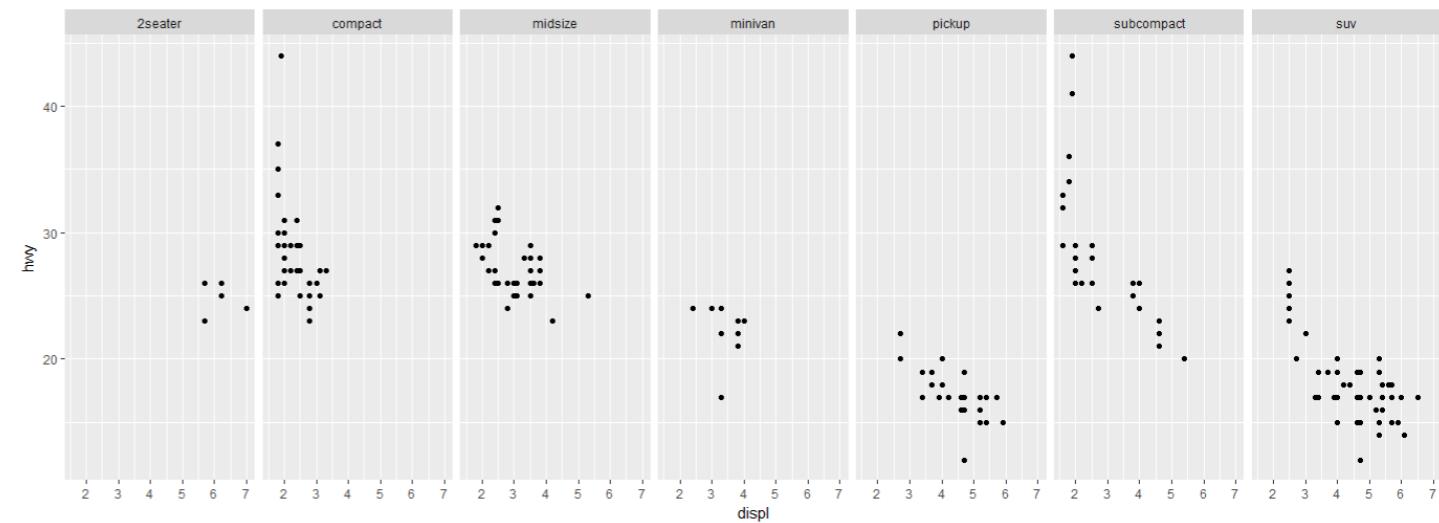
```
geom_point() +  
facet_grid(.~class)
```



在一列多行显示多个图表：

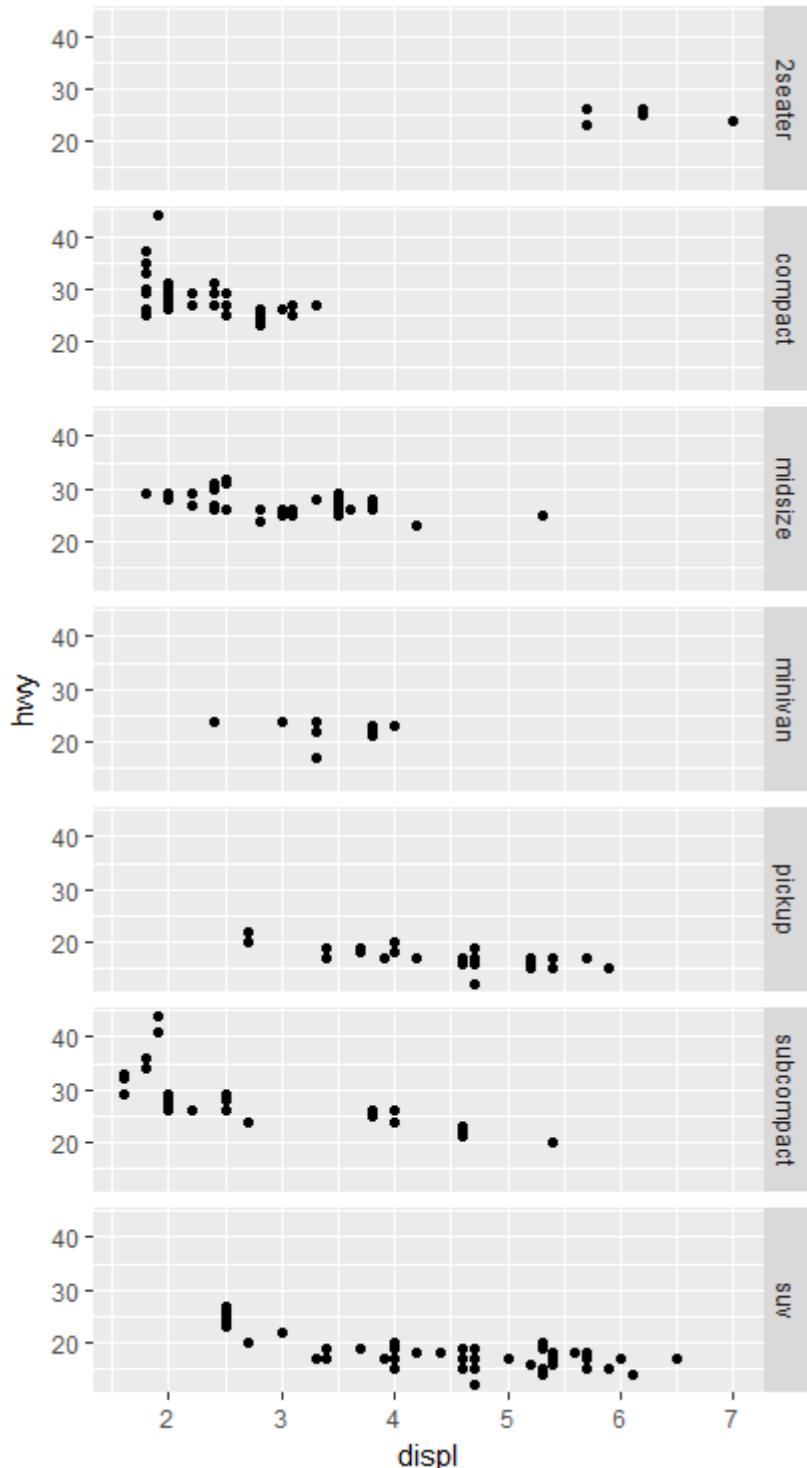
```
ggplot(mpg, aes(x = displ, y = hwy)) +  
geom_point() +  
facet_grid(class~.)
```

```
geom_point() +  
facet_grid(.~class)
```



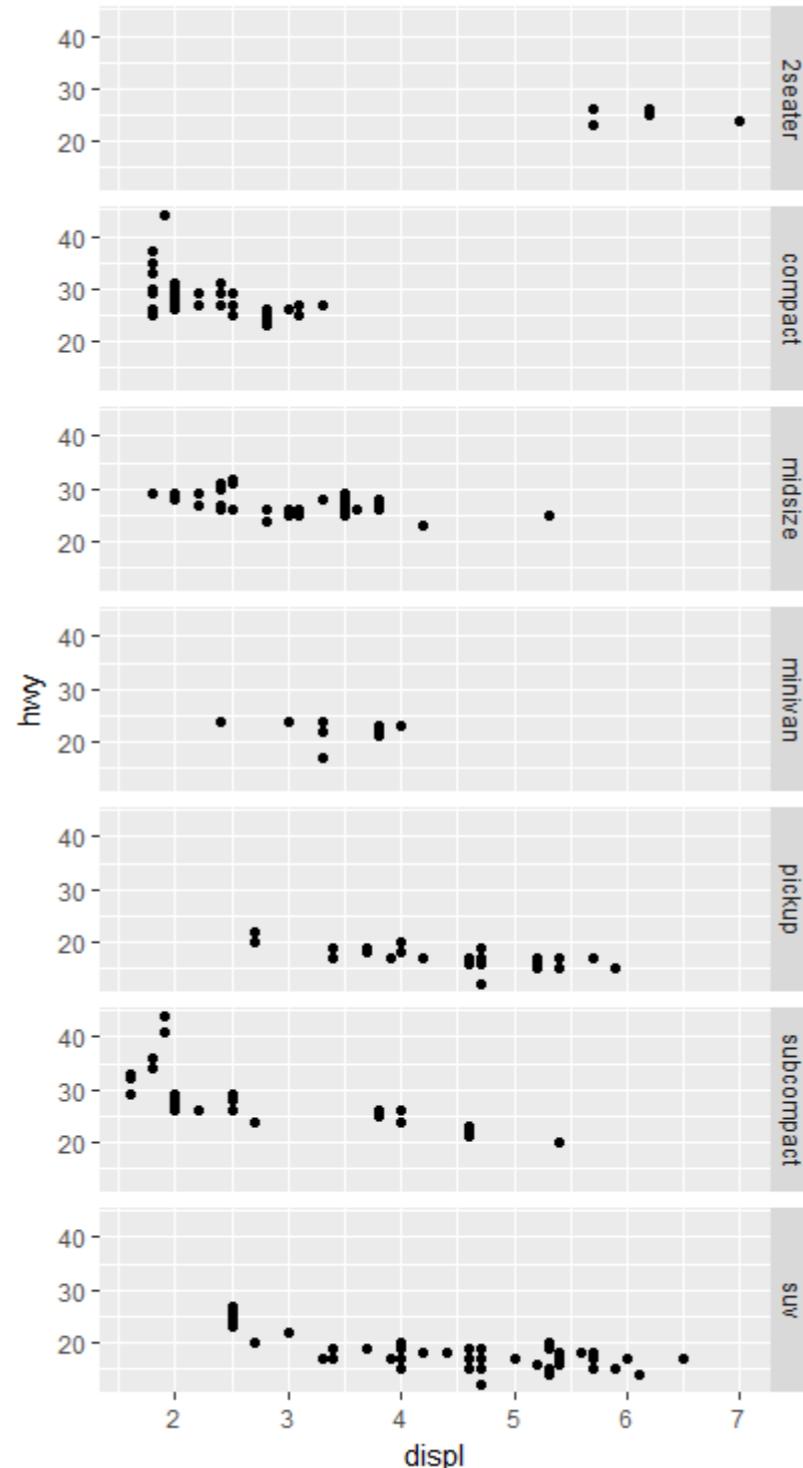
Display multiple charts on one column, multiple rows:

```
ggplot(mpg, aes(x = displ, y = hwy)) +  
geom_point() +  
facet_grid(class~.)
```



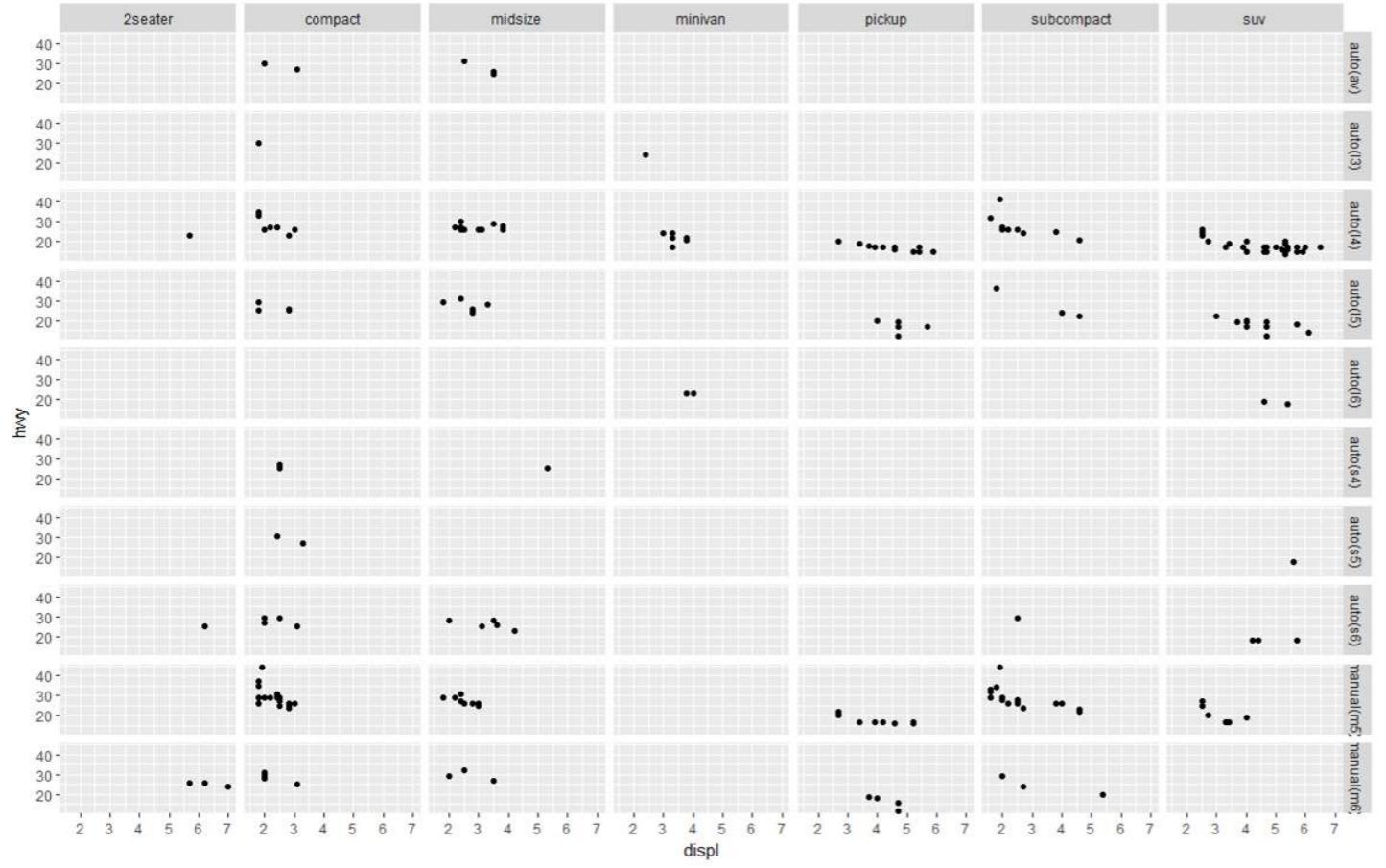
按两个变量在网格中显示多个图表：

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(trans~class) #“行”参数，然后是“列”参数
```



Display multiple charts in a grid by 2 variables:

```
ggplot(mpg, aes(x = displ, y = hwy)) +
  geom_point() +
  facet_grid(trans~class) #“row” parameter, then “column” parameter
```



第28.2节：准备绘图数据

ggplot2 最适合使用长格式数据框。以下示例数据表示20个不同日期的糖果价格，格式称为宽格式，因为每个类别都有一列。

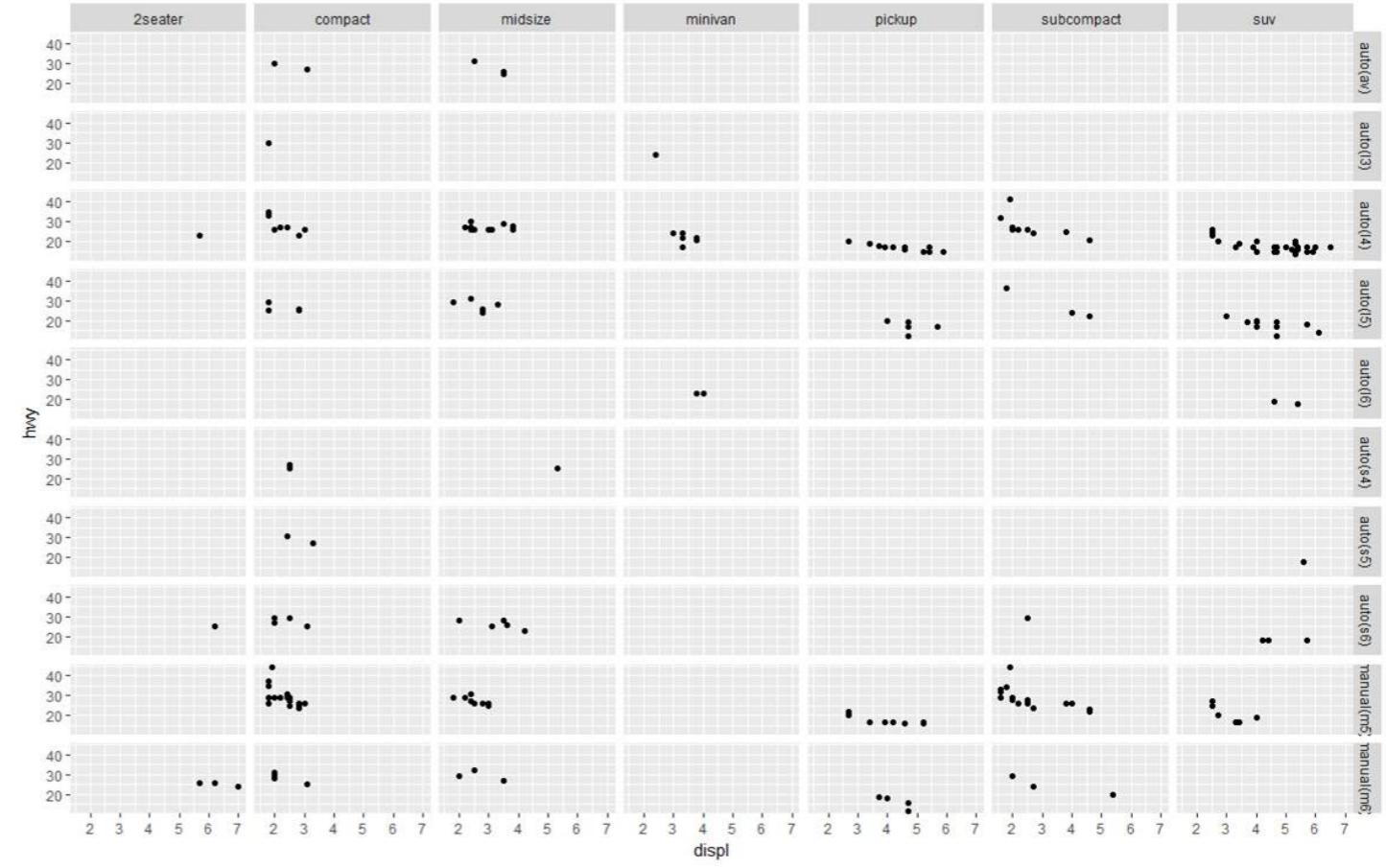
```
set.seed(47)
sweetsWide <- data.frame(date = 1:20,
                           chocolate = runif(20, min = 2, max = 4),
                           iceCream = runif(20, min = 0.5, max = 1),
                           candy = runif(20, min = 1, max = 3))
```

```
head(sweetsWide)
##   date chocolate iceCream candy
## 1    1  3.953924 0.5890727 1.117311
## 2    2  2.747832 0.7783982 1.740851
## 3    3  3.523004 0.7578975 2.196754
## 4    4  3.644983 0.5667152 2.875028
## 5    5  3.147089 0.8446417 1.733543
## 6    6  3.382825 0.6900125 1.405674
```

要将 sweetsWide 转换为适用于 ggplot2 的长格式，可以使用base R中的几个有用函数，以及按时间顺序排列的包reshape2、data.table和 tidyR：

```
# base R中的reshape函数
sweetsLong <- reshape(sweetsWide, idvar = 'date', direction = 'long',
                      varying = list(2:4), new.row.names = NULL, times = names(sweetsWide)[-1])

# reshape2包中的melt函数
library(reshape2)
sweetsLong <- melt(sweetsWide, id.vars = 'date')
```



Section 28.2: Prepare your data for plotting

ggplot2 works best with a long data frame. The following sample data which represents the prices for sweets on 20 different days, in a format described as wide, because each category has a column.

```
set.seed(47)
sweetsWide <- data.frame(date = 1:20,
                           chocolate = runif(20, min = 2, max = 4),
                           iceCream = runif(20, min = 0.5, max = 1),
                           candy = runif(20, min = 1, max = 3))
```

```
head(sweetsWide)
##   date chocolate iceCream candy
## 1    1  3.953924 0.5890727 1.117311
## 2    2  2.747832 0.7783982 1.740851
## 3    3  3.523004 0.7578975 2.196754
## 4    4  3.644983 0.5667152 2.875028
## 5    5  3.147089 0.8446417 1.733543
## 6    6  3.382825 0.6900125 1.405674
```

To convert sweetsWide to long format for use with ggplot2, several useful functions from base R, and the packages reshape2, data.table and tidyR (in chronological order) can be used:

```
# reshape from base R
sweetsLong <- reshape(sweetsWide, idvar = 'date', direction = 'long',
                      varying = list(2:4), new.row.names = NULL, times = names(sweetsWide)[-1])

# melt from 'reshape2'
library(reshape2)
sweetsLong <- melt(sweetsWide, id.vars = 'date')
```

```
# 来自 'data.table' 的 melt
# 这是 'reshape2' 包中 'melt' 函数的优化和扩展版本
library(data.table)
sweetsLong <- melt(setDT(sweetsWide), id.vars = 'date')

# 来自 'tidy' 包的 gather 函数
library(tidy)
sweetsLong <- gather(sweetsWide, sweet, price, chocolate:candy)
```

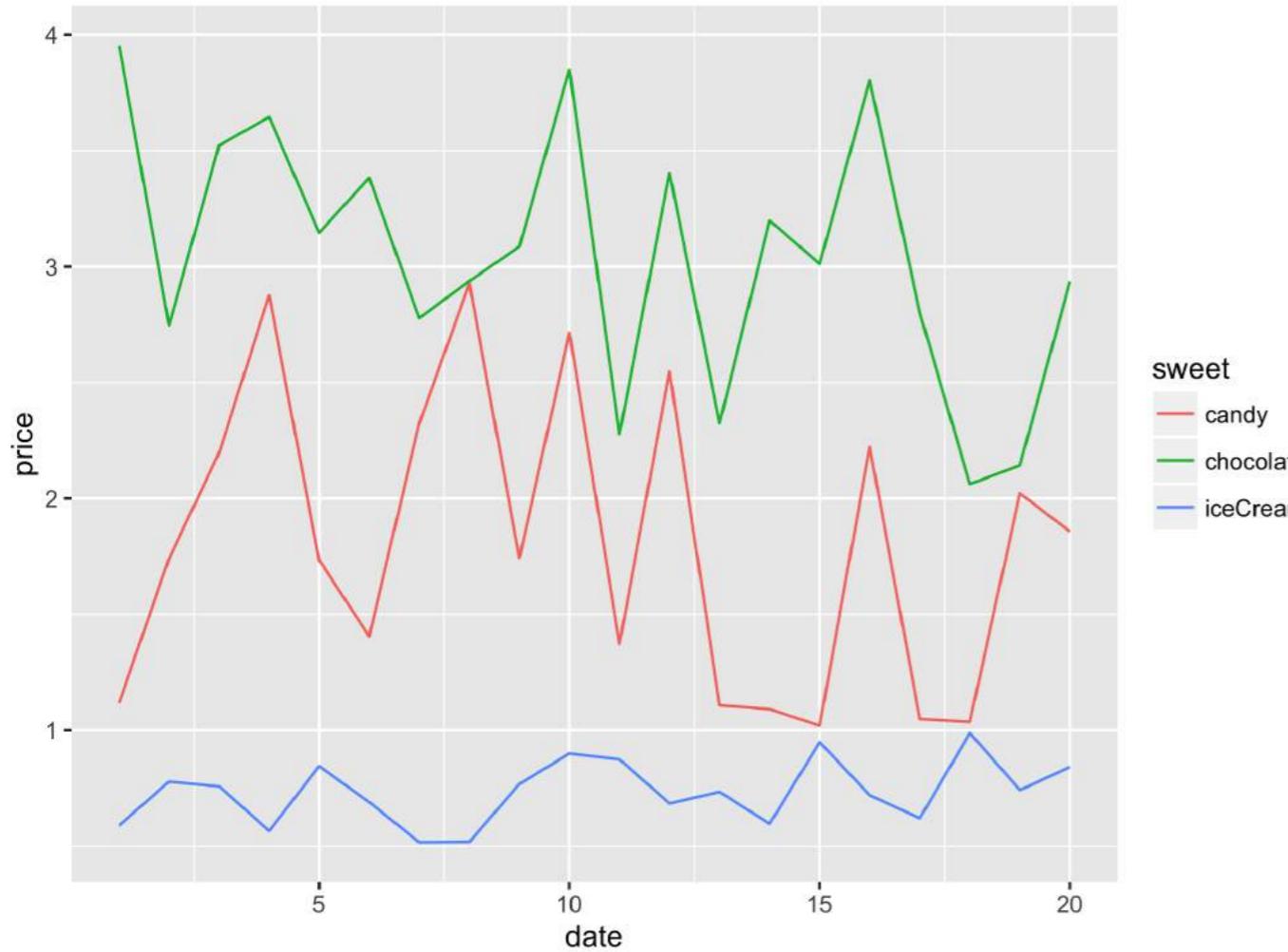
它们的结果都类似：

```
head(sweetsLong)
## # date sweet price
## 1 1 chocolate 3.953924
## 2 2 chocolate 2.747832
## 3 3 chocolate 3.523004
## 4 4 chocolate 3.644983
## 5 5 chocolate 3.147089
## 6 6 chocolate 3.382825
```

另请参见“在长格式和宽格式之间重塑数据”，了解如何在长格式和宽格式之间转换数据的详细信息。

生成的 sweetsLong 有一列价格和一列描述甜点类型的列。现在绘图变得更加简单：

```
library(ggplot2)
ggplot(sweetsLong, aes(x = date, y = price, colour = sweet)) + geom_line()
```



```
# melt from 'data.table'
# which is an optimized & extended version of 'melt' from 'reshape2'
library(data.table)
sweetsLong <- melt(setDT(sweetsWide), id.vars = 'date')

# gather from 'tidy'
library(tidy)
sweetsLong <- gather(sweetsWide, sweet, price, chocolate:candy)
```

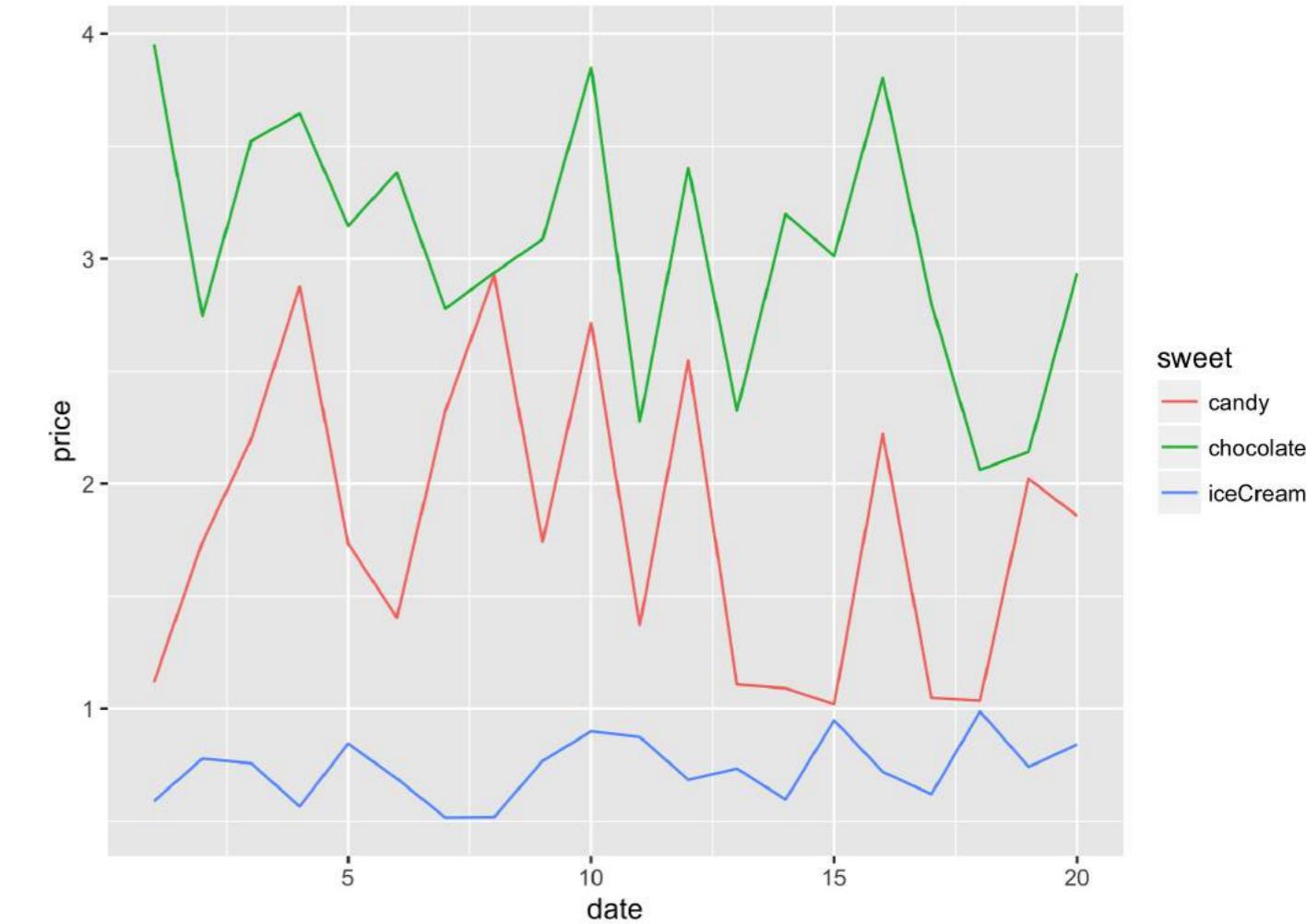
The all give a similar result:

```
head(sweetsLong)
## # date sweet price
## 1 1 chocolate 3.953924
## 2 2 chocolate 2.747832
## 3 3 chocolate 3.523004
## 4 4 chocolate 3.644983
## 5 5 chocolate 3.147089
## 6 6 chocolate 3.382825
```

See also Reshaping data between long and wide forms for details on converting data between *long* and *wide* format.

The resulting sweetsLong has one column of prices and one column describing the type of sweet. Now plotting is much simpler:

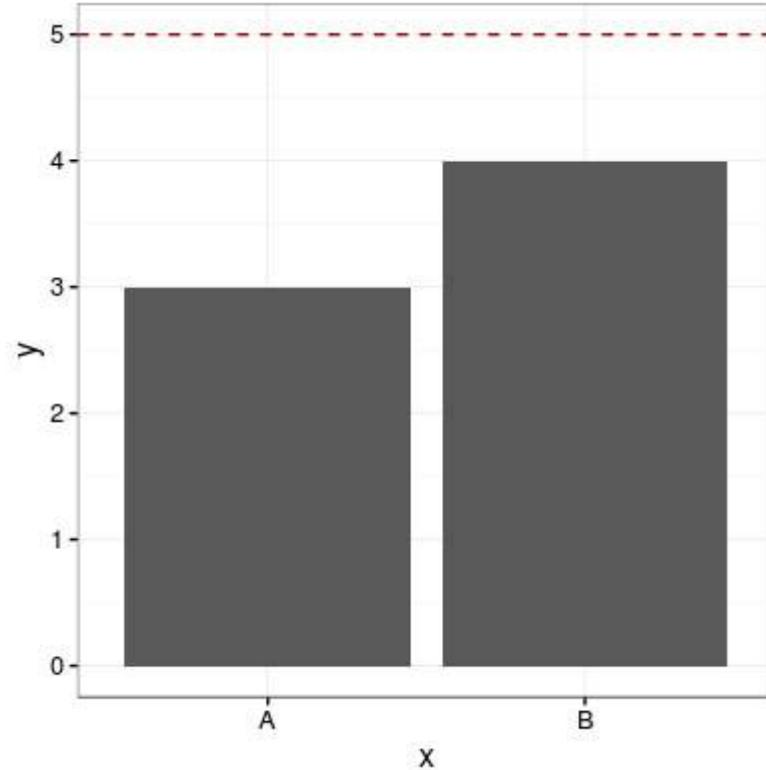
```
library(ggplot2)
ggplot(sweetsLong, aes(x = date, y = price, colour = sweet)) + geom_line()
```



第28.3节：向图中添加水平线和垂直线

为所有分类变量添加一条公共水平线

```
# 例数据  
df <- data.frame(x=c('A', 'B'), y = c(3, 4))  
  
p1 <- ggplot(df, aes(x=x, y=y))  
+geom_bar(position = "dodge", stat = 'identity')  
+theme_bw()  
  
p1 + geom_hline(aes(yintercept=5), colour="#990000", linetype="dashed")
```



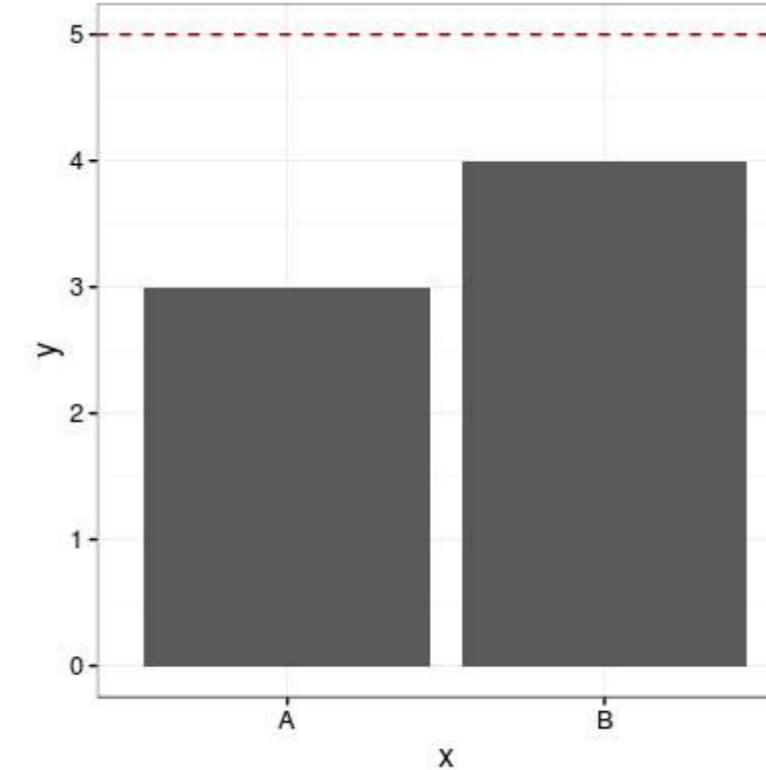
为每个分类变量添加一条水平线

```
# 例数据  
df <- data.frame(x=c('A', 'B'), y = c(3, 4))  
  
# 添加用于绘制线条的水平值  
df$hval <- df$y + 2  
  
p1 <- ggplot(df, aes(x=x, y=y))  
+geom_bar(position = "dodge", stat = 'identity')  
+theme_bw()  
  
p1 + geom_errorbar(aes(y=hval, ymax=hval, ymin=hval), colour="#990000", width=0.75)
```

Section 28.3: Add horizontal and vertical lines to plot

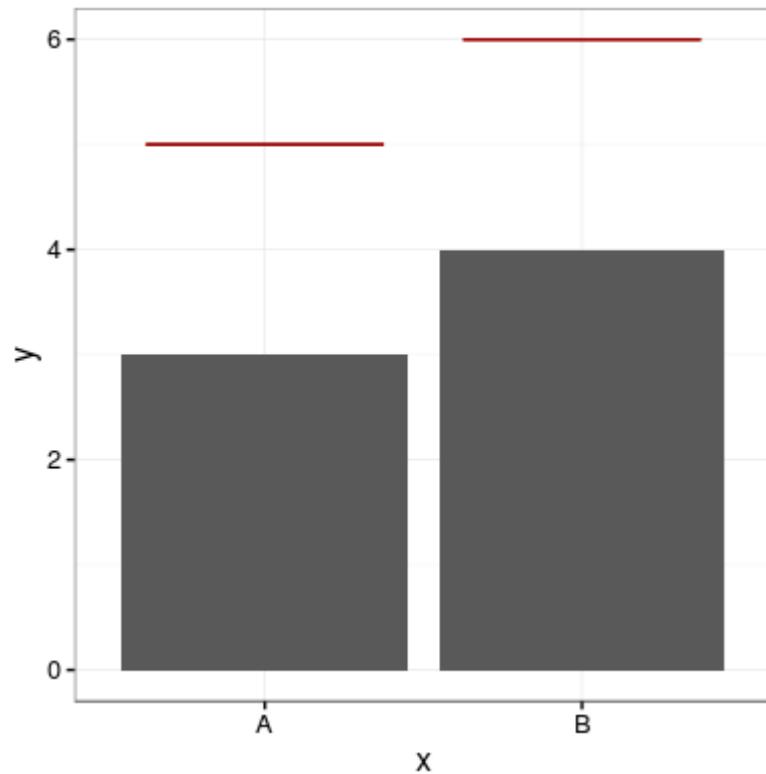
Add one common horizontal line for all categorical variables

```
# sample data  
df <- data.frame(x=c('A', 'B'), y = c(3, 4))  
  
p1 <- ggplot(df, aes(x=x, y=y))  
+geom_bar(position = "dodge", stat = 'identity')  
+theme_bw()  
  
p1 + geom_hline(aes(yintercept=5), colour="#990000", linetype="dashed")
```



Add one horizontal line for each categorical variable

```
# sample data  
df <- data.frame(x=c('A', 'B'), y = c(3, 4))  
  
# add horizontal levels for drawing lines  
df$hval <- df$y + 2  
  
p1 <- ggplot(df, aes(x=x, y=y))  
+geom_bar(position = "dodge", stat = 'identity')  
+theme_bw()  
  
p1 + geom_errorbar(aes(y=hval, ymax=hval, ymin=hval), colour="#990000", width=0.75)
```

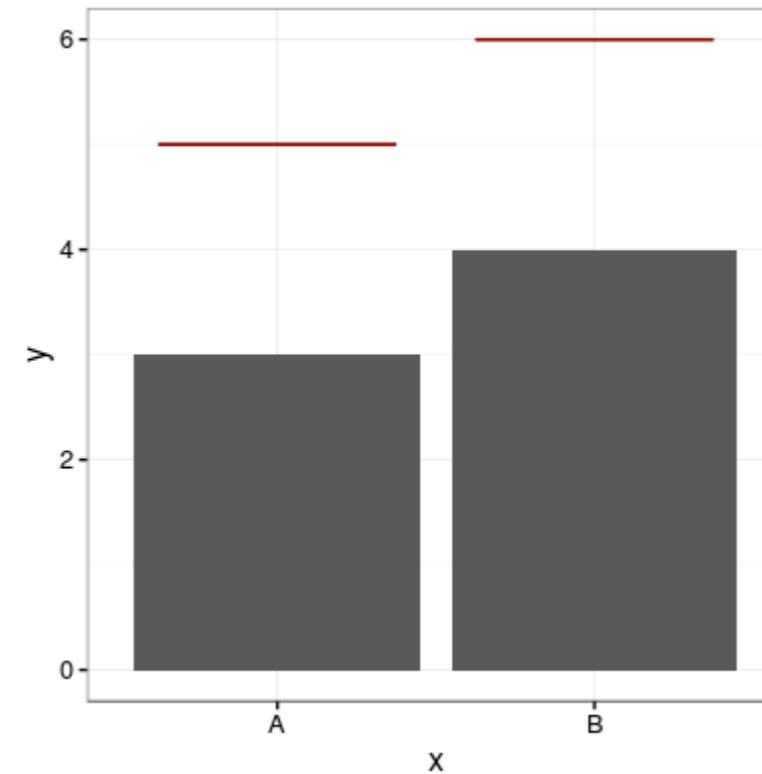


在分组柱状图上添加水平线

```
# 读取数据
df <- data.frame(x = rep(c('A', 'B'), times=2),
                  group = rep(c('G1', 'G2'), each=2),
                  y = c(3, 4, 5, 6),
                  hval = c(5, 6, 7, 8))

p1 <- ggplot(df, aes(x=x, y=y, fill=group))
  +geom_bar(position="dodge", stat="identity")

p1 + geom_errorbar(aes(y=hval, ymax=hval, ymin=hval),
                    colour="#990000",
                    position = "dodge",
                    linetype = "dashed")
```

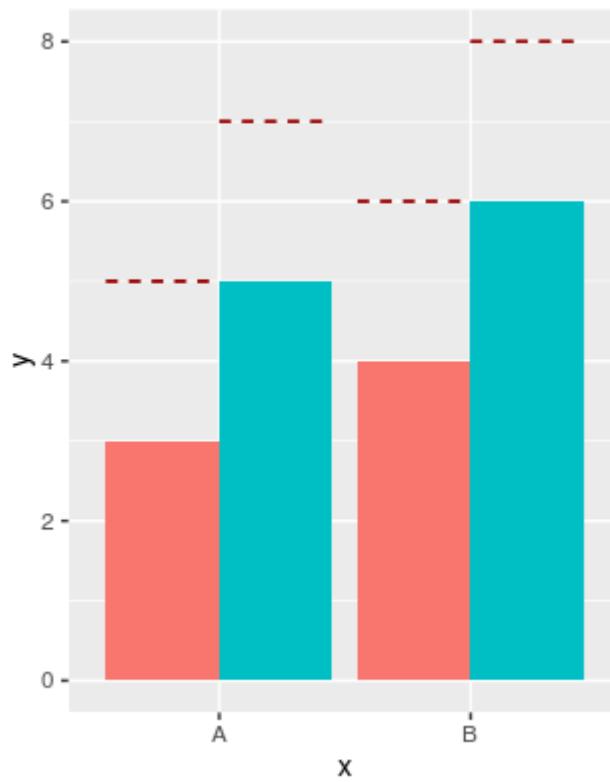


Add horizontal line over grouped bars

```
# 读取数据
df <- data.frame(x = rep(c('A', 'B'), times=2),
                  group = rep(c('G1', 'G2'), each=2),
                  y = c(3, 4, 5, 6),
                  hval = c(5, 6, 7, 8))

p1 <- ggplot(df, aes(x=x, y=y, fill=group))
  +geom_bar(position="dodge", stat="identity")

p1 + geom_errorbar(aes(y=hval, ymax=hval, ymin=hval),
                    colour="#990000",
                    position = "dodge",
                    linetype = "dashed")
```

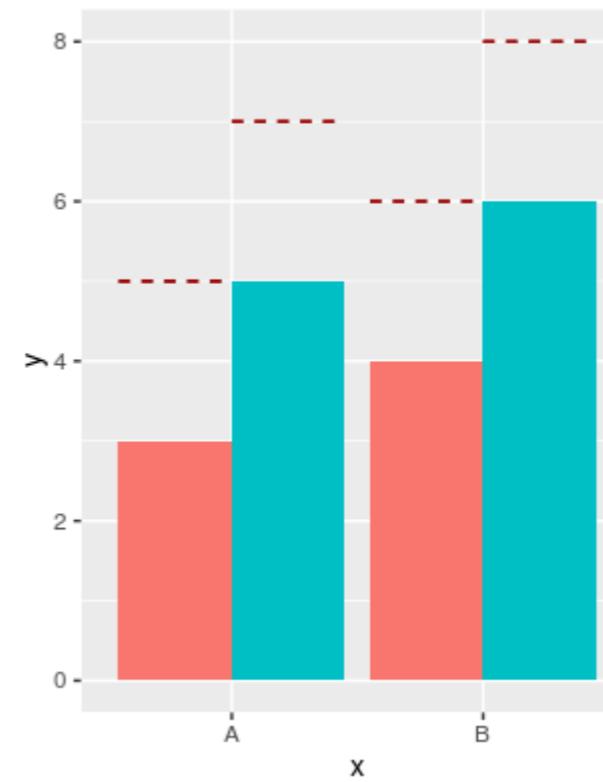
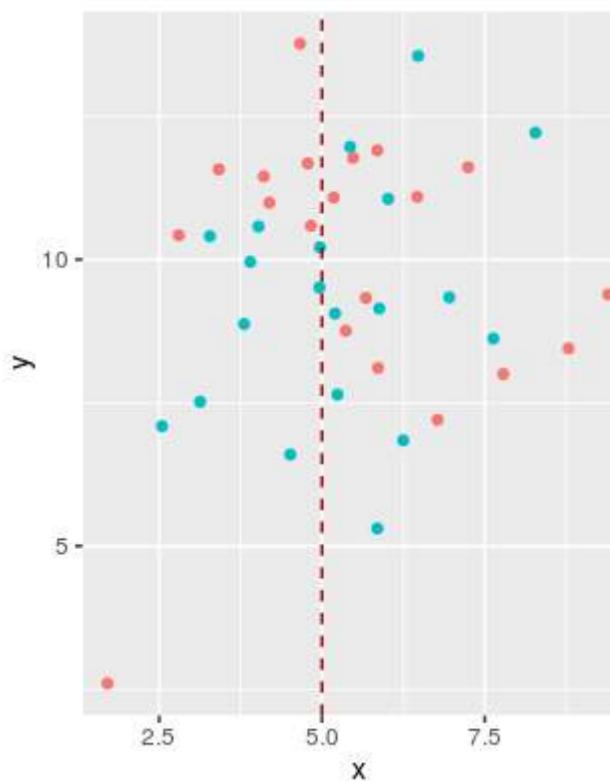


添加垂直线

```
# 生成示例数据
df <- data.frame(group=rep(c('A', 'B'), each=20),
                  x = rnorm(40, 5, 2),
                  y = rnorm(40, 10, 2))

p1 <- ggplot(df, aes(x=x, y=y, colour=group)) + geom_point()

p1 + geom_vline(aes(xintercept=5), color="#990000", linetype="dashed")
```

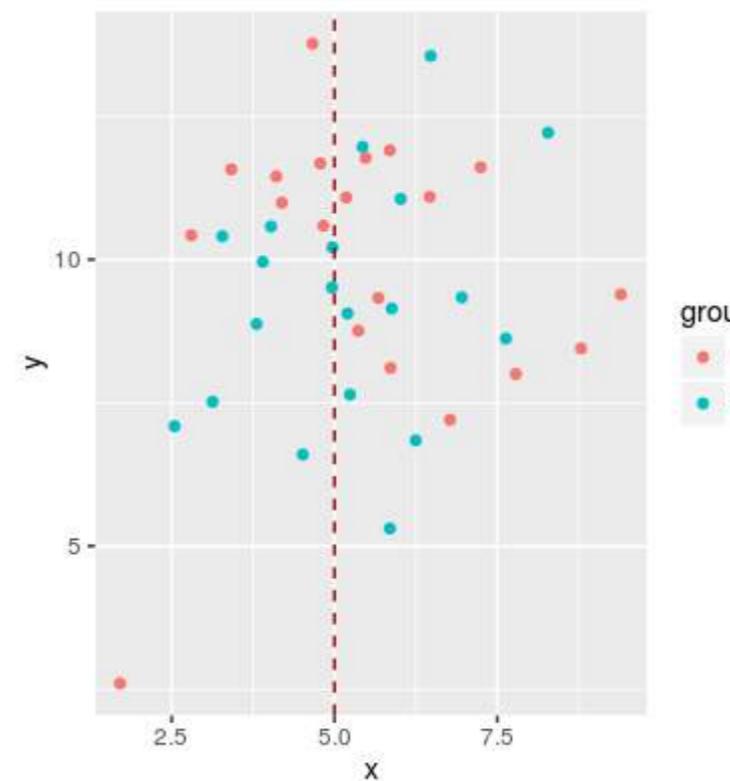


Add vertical line

```
# 生成示例数据
df <- data.frame(group=rep(c('A', 'B'), each=20),
                  x = rnorm(40, 5, 2),
                  y = rnorm(40, 10, 2))

p1 <- ggplot(df, aes(x=x, y=y, colour=group)) + geom_point()

p1 + geom_vline(aes(xintercept=5), color="#990000", linetype="dashed")
```

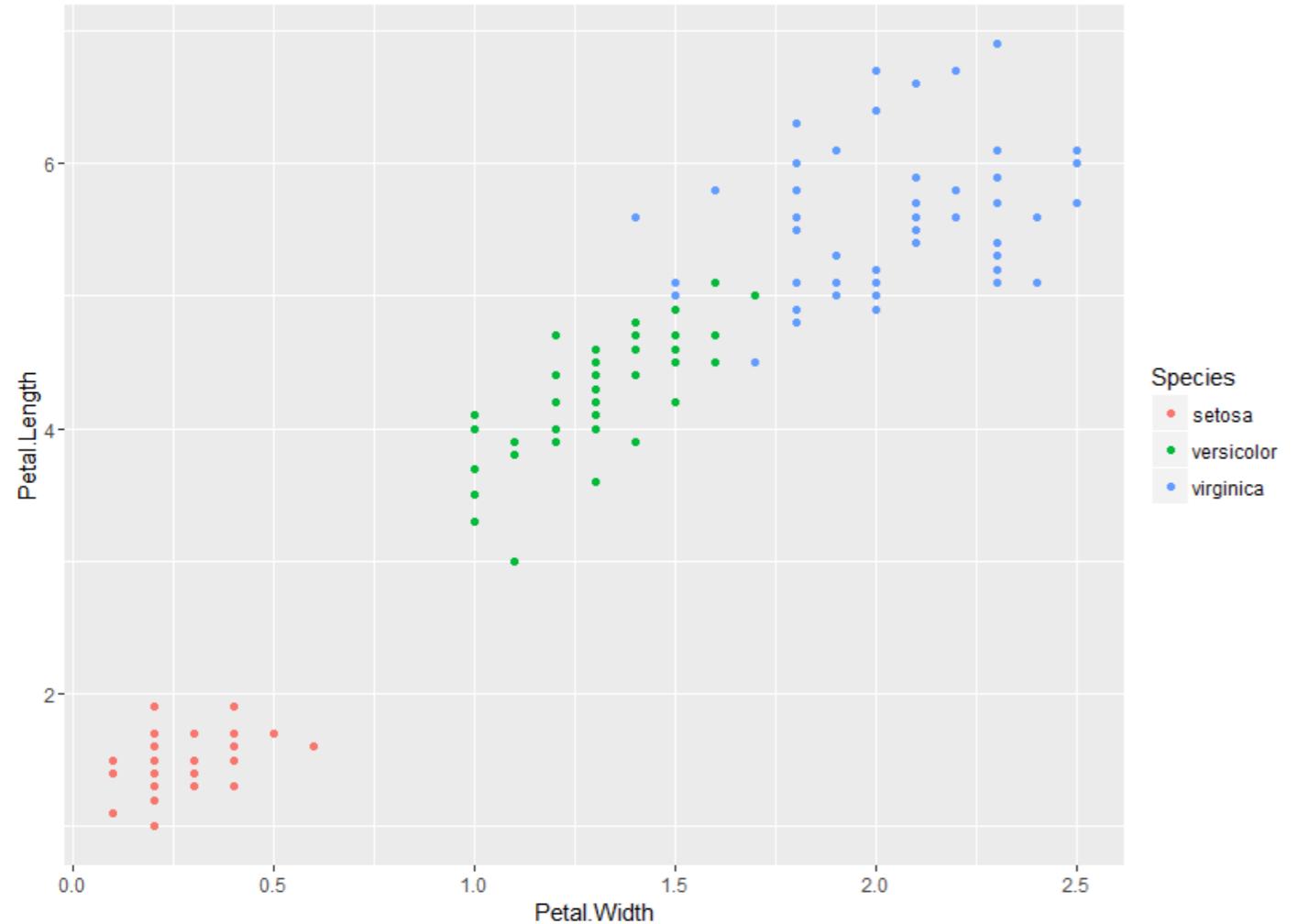


第28.4节：散点图

我们使用内置的鸢尾花（iris）数据集绘制一个简单的散点图，方法如下：

```
library(ggplot2)
ggplot(iris, aes(x = Petal.Width, y = Petal.Length, color = Species)) +
  geom_point()
```

结果如下：



第28.5节：使用qplot生成基本图形

qplot旨在类似于基础R的plot()函数，尽量在不需要过多参数的情况下直接绘制数据。

基本的qplot

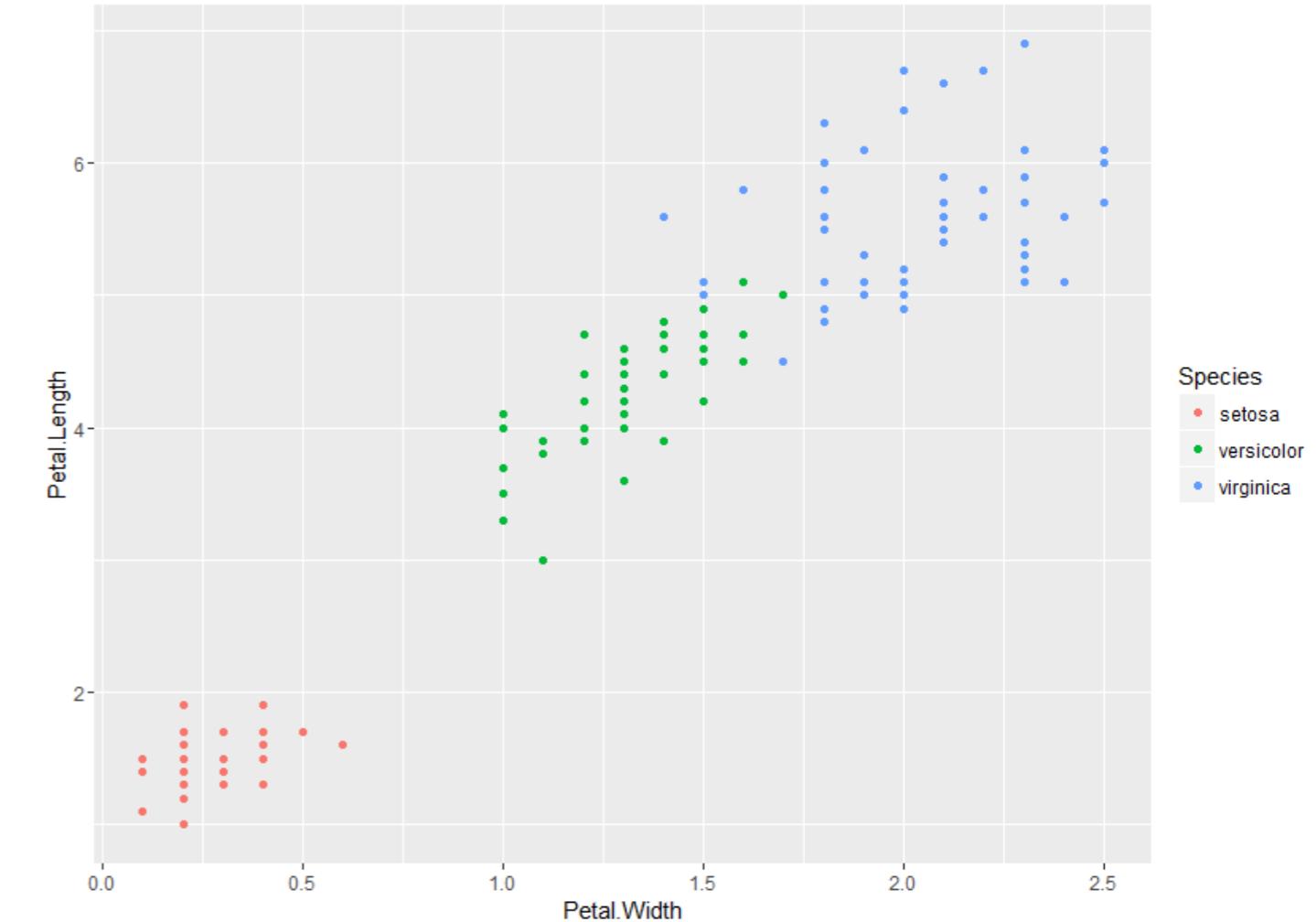
```
qplot(x = disp, y = mpg, data = mtcars)
```

Section 28.4: Scatter Plots

We plot a simple scatter plot using the builtin iris data set as follows:

```
library(ggplot2)
ggplot(iris, aes(x = Petal.Width, y = Petal.Length, color = Species)) +
  geom_point()
```

This gives:

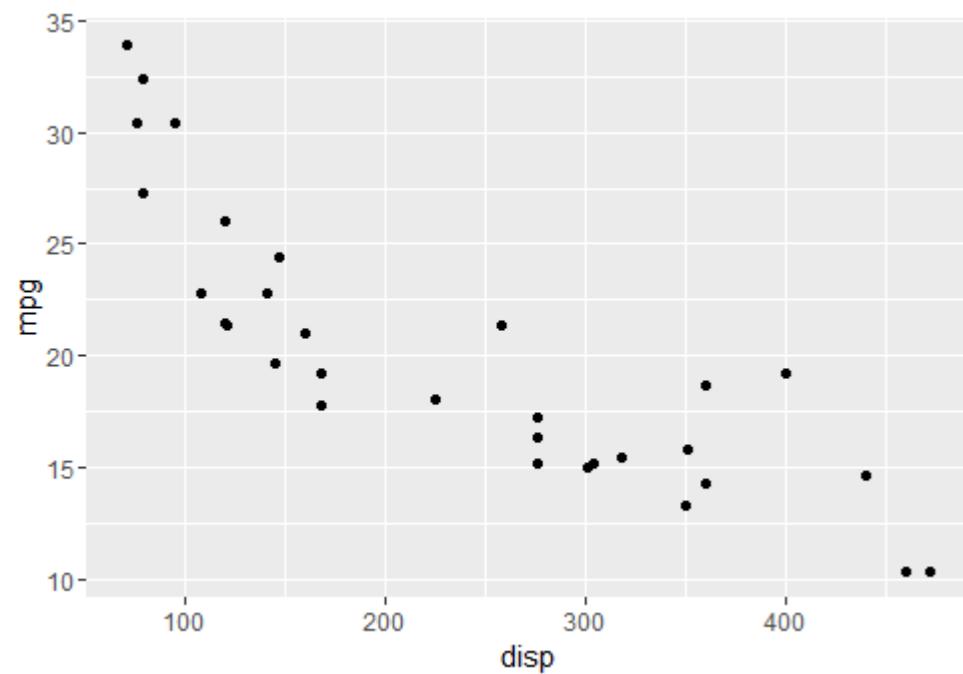


Section 28.5: Produce basic plots with qplot

qplot is intended to be similar to base R `plot()` function, trying to always plot out your data without requiring too much specifications.

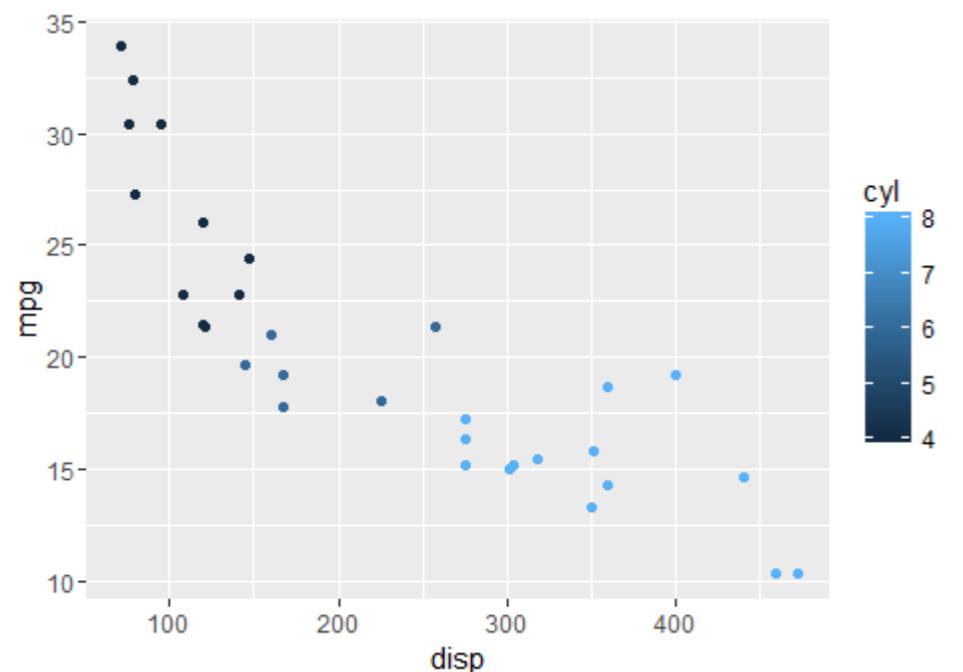
basic qplot

```
qplot(x = disp, y = mpg, data = mtcars)
```



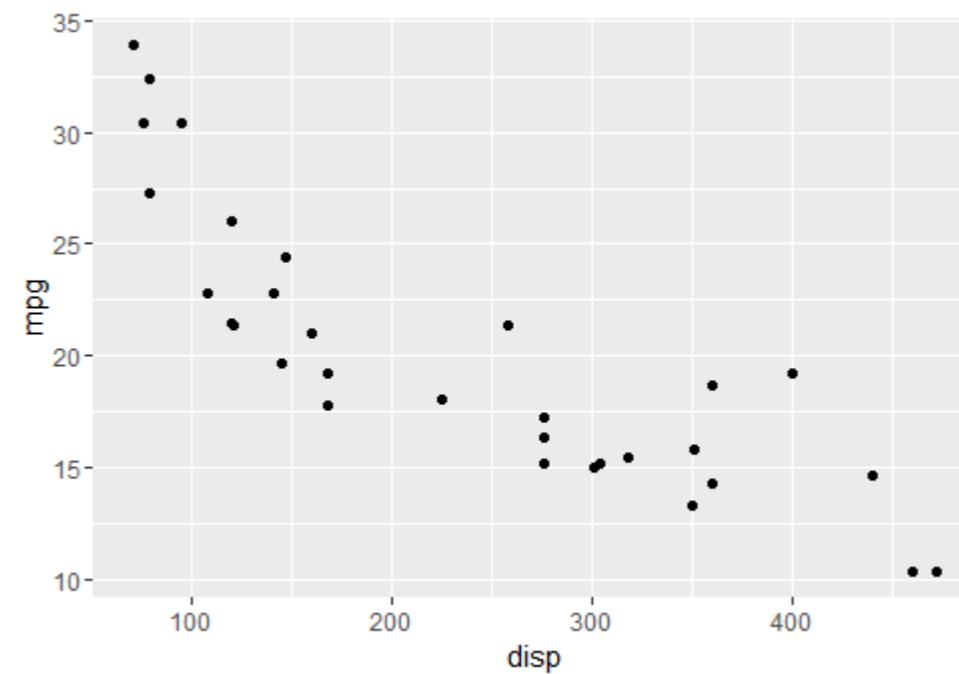
添加颜色

```
qplot(x = 排量, y = 英里每加仑, 颜色 = 气缸数, 数据 = mtcars)
```



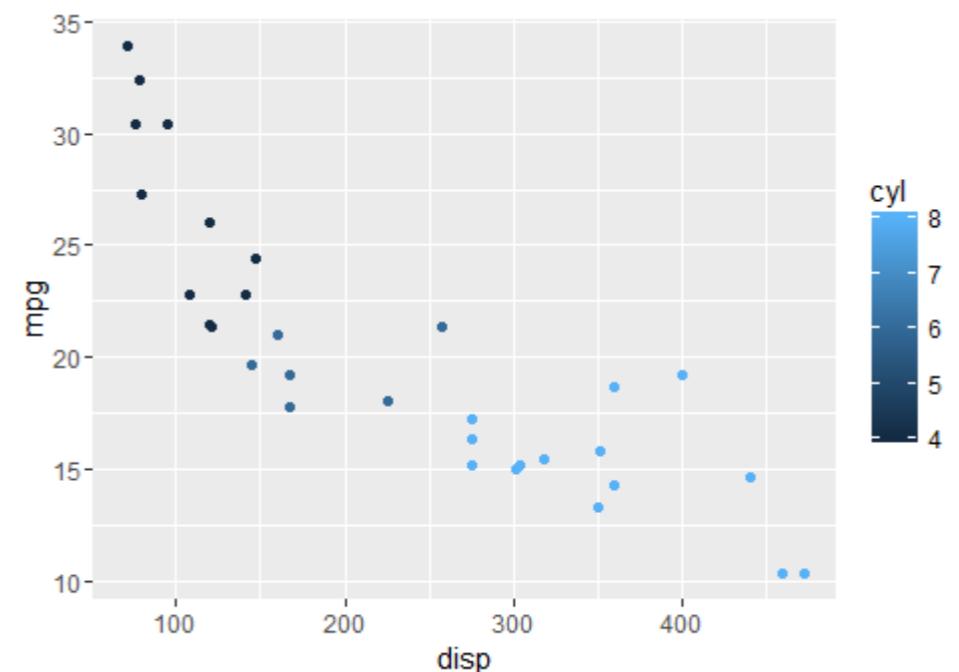
添加平滑曲线

```
qplot(x = 排量, y = 英里每加仑, 图形 = c("点", "平滑"), 数据 = mtcars)
```



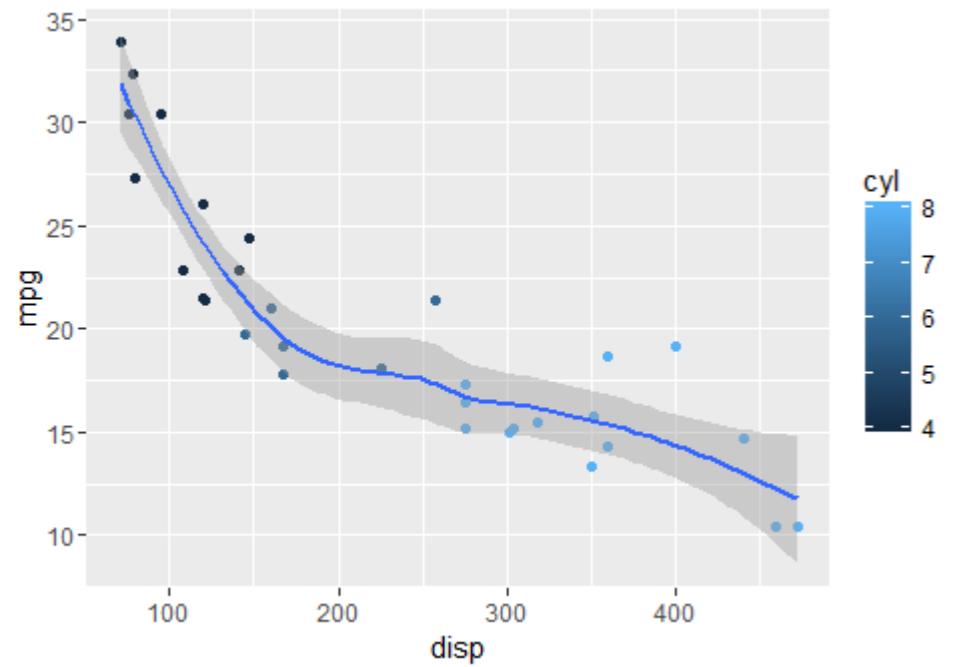
adding colors

```
qplot(x = disp, y = mpg, colour = cyl, data = mtcars)
```



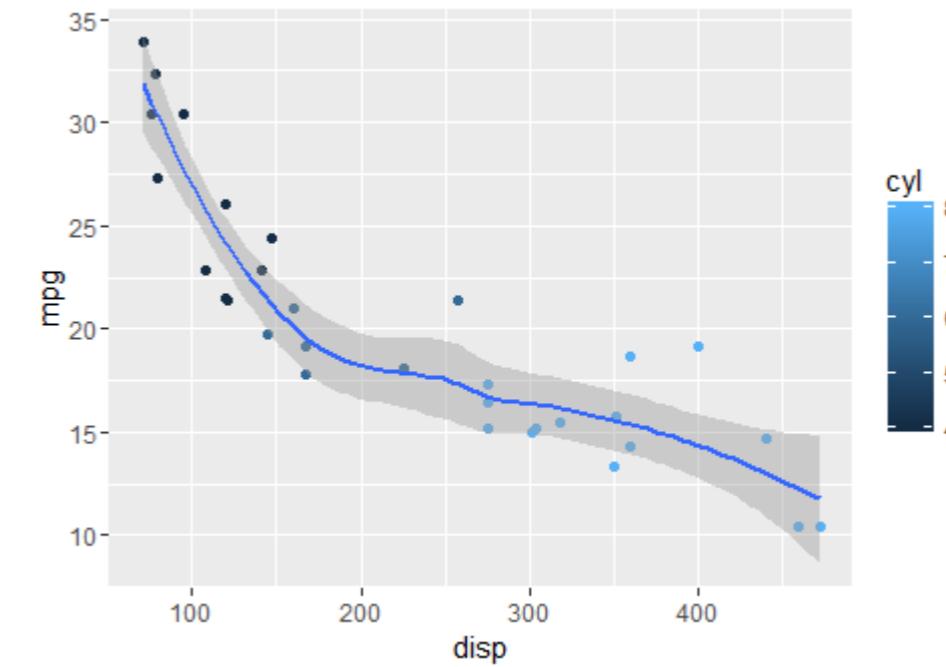
adding a smoother

```
qplot(x = disp, y = mpg, geom = c("point", "smooth"), data = mtcars)
```



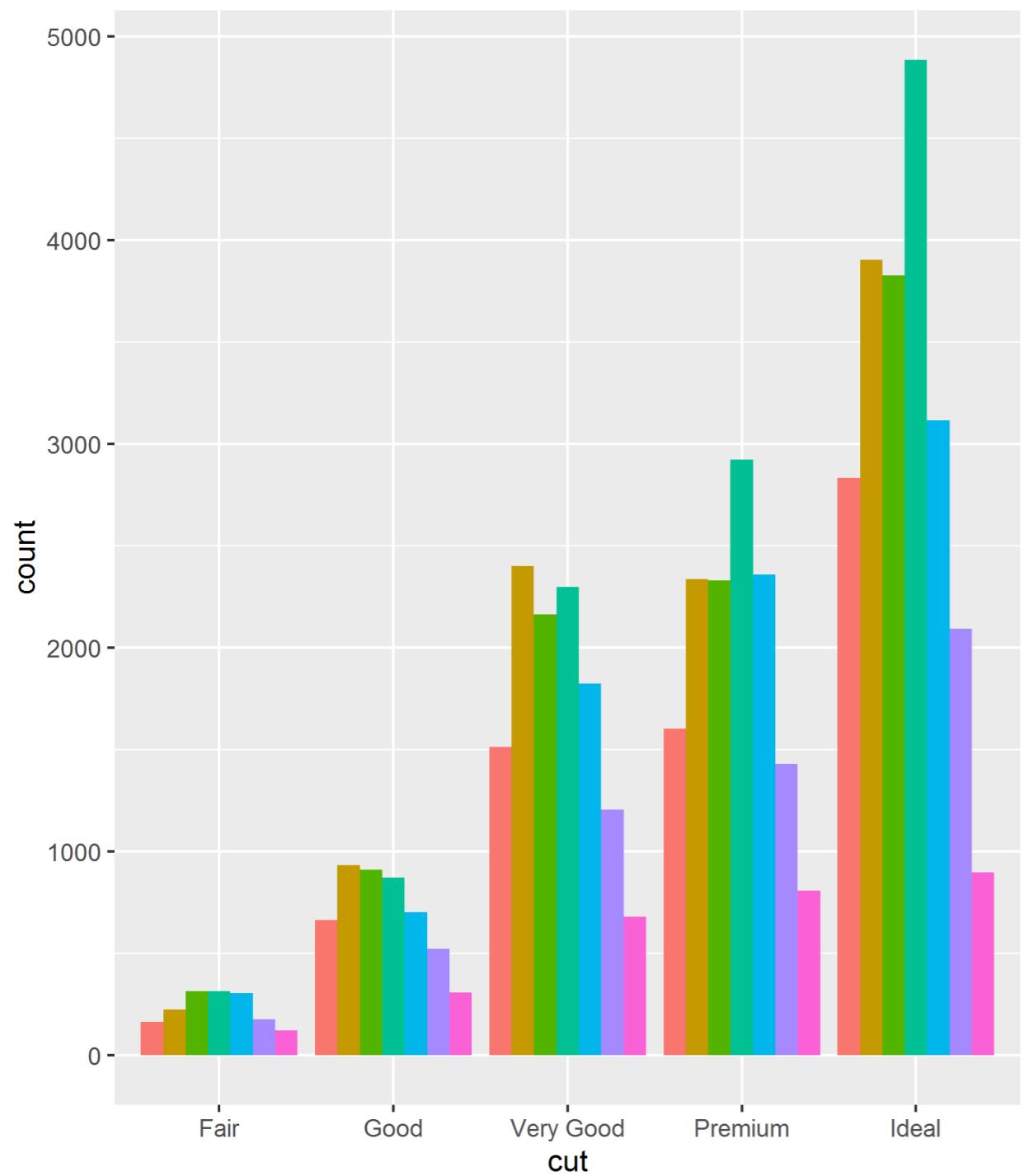
第28.6节：垂直和水平条形图

```
ggplot(数据 = diamonds, aes(x = 切工, 填充色 = 颜色)) +
  geom_bar(统计 = "计数", 位置 = "并列")
```



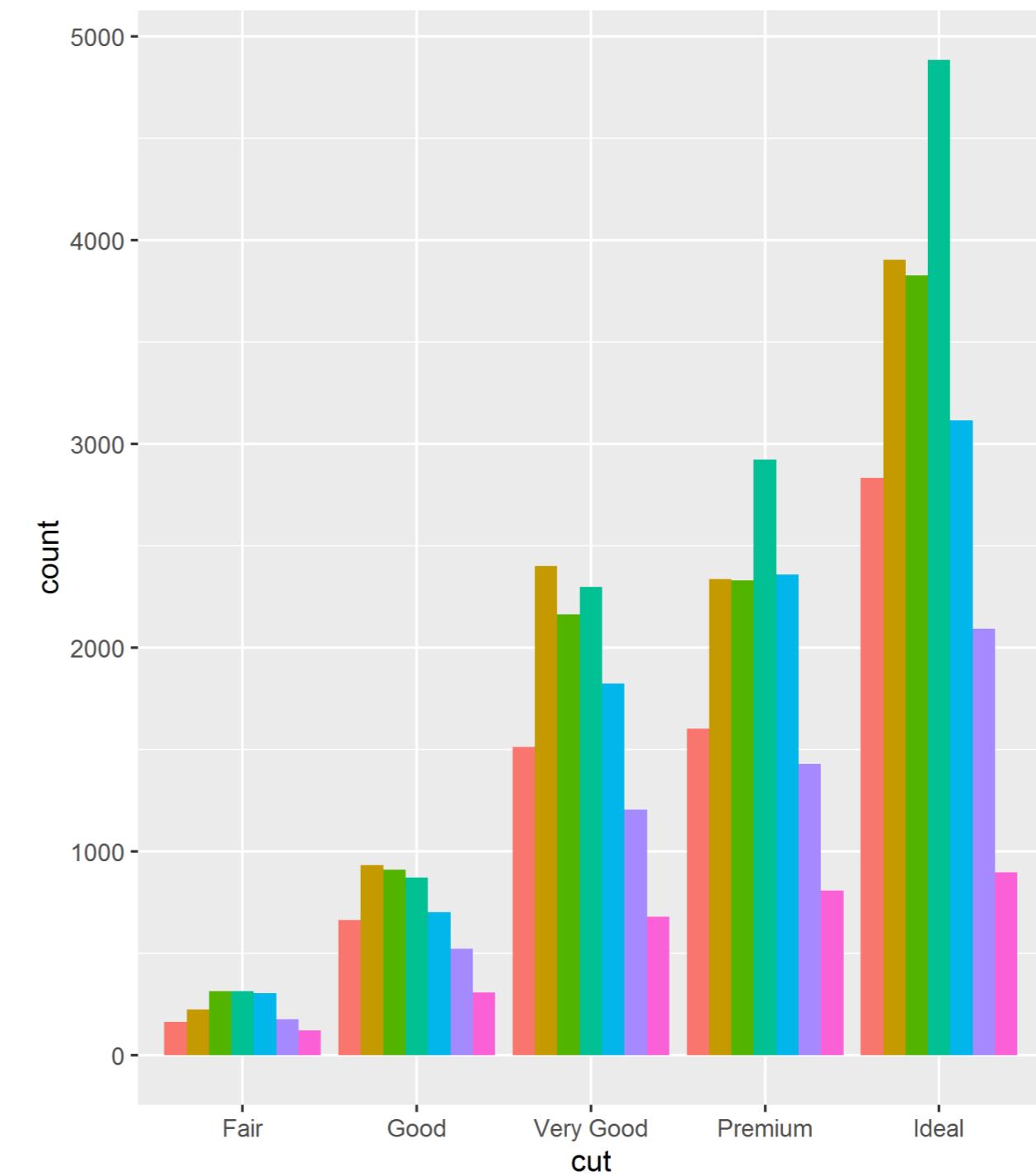
Section 28.6: Vertical and Horizontal Bar Chart

```
ggplot(data = diamonds, aes(x = cut, fill = color)) +
  geom_bar(stat = "count", position = "dodge")
```



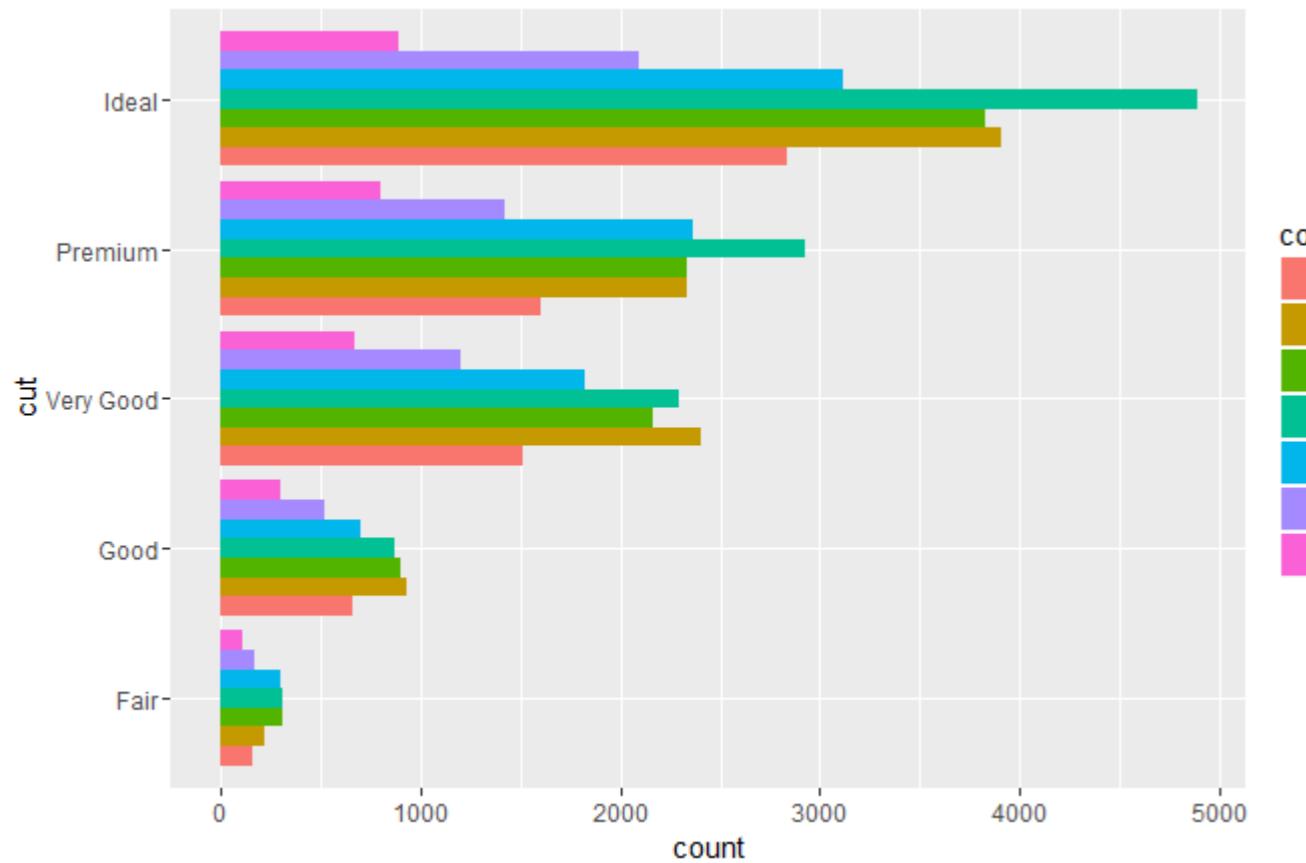
只需向ggplot对象添加coord_flip()美学属性，即可获得水平条形图：

```
ggplot(数据 = diamonds, aes(x = 切工, 填充色 = 颜色)) +
  geom_bar(统计 = "计数", 位置 = "并列")+
  coord_flip()
```

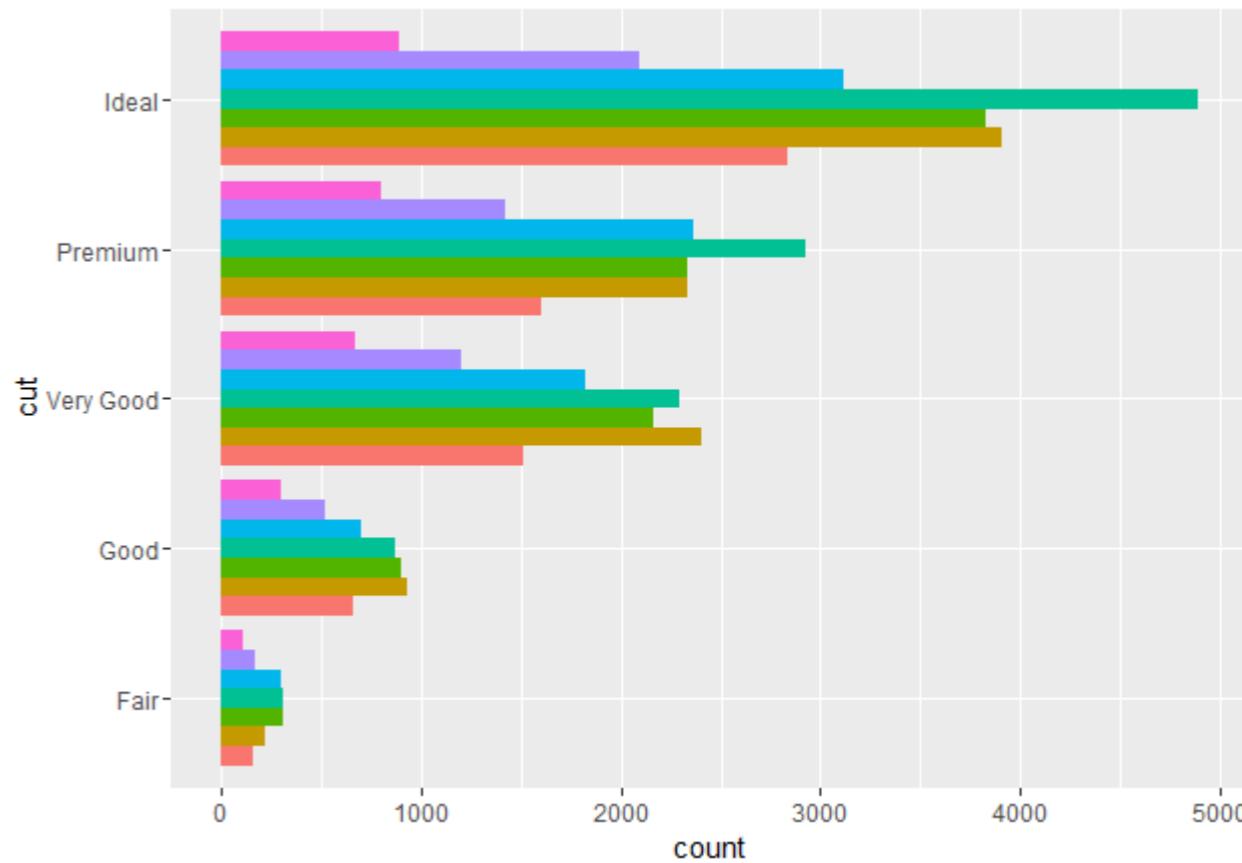


it is possible to obtain an horizontal bar chart simply adding coord_flip() aesthetic to the ggplot object:

```
ggplot(数据 = diamonds, aes(x = cut, fill = color)) +
  geom_bar(stat = "count", position = "dodge")+
  coord_flip()
```



color
D
E
F
G
H
I
J



color
D
E
F
G
H
I
J

第28.7节：小提琴图

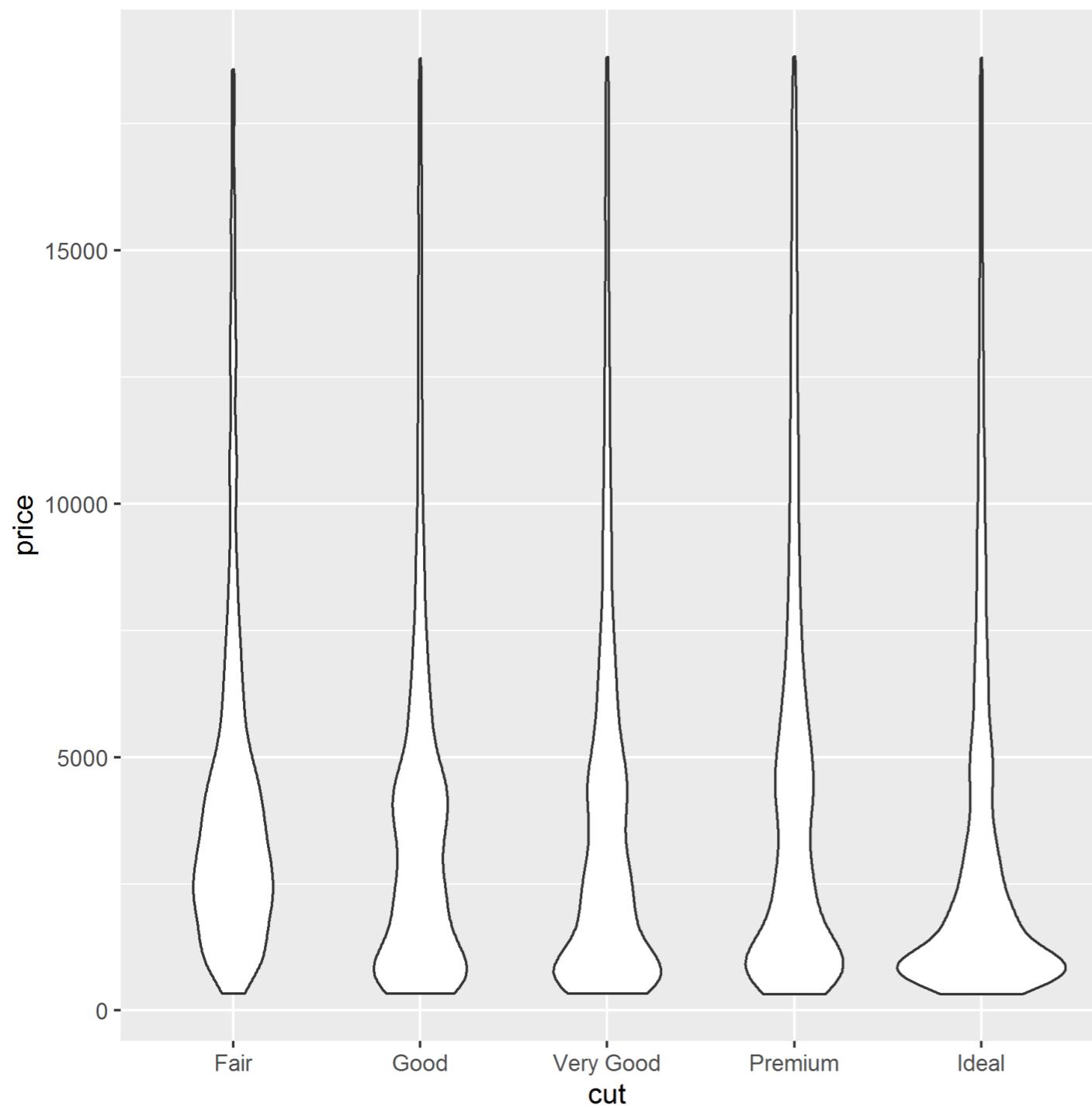
小提琴图是垂直平面中镜像的核密度估计。它们可以用来并排可视化多个分布，镜像效果有助于突出任何差异。

```
ggplot(diamonds, aes(cut, price)) +  
  geom_violin()
```

Section 28.7: Violin plot

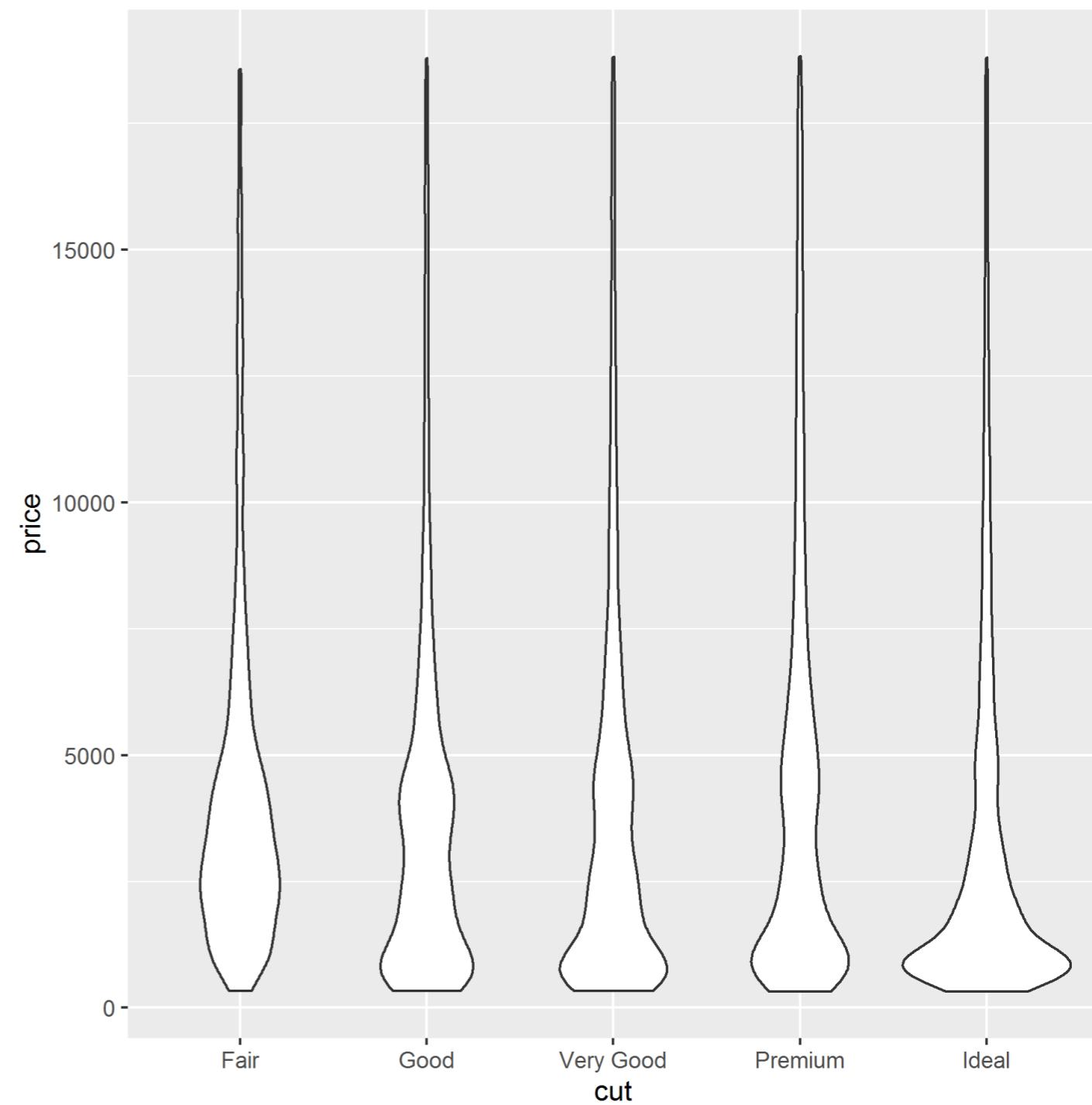
Violin plots are kernel density estimates mirrored in the vertical plane. They can be used to visualize several distributions side-by-side, with the mirroring helping to highlight any differences.

```
ggplot(diamonds, aes(cut, price)) +  
  geom_violin()
```



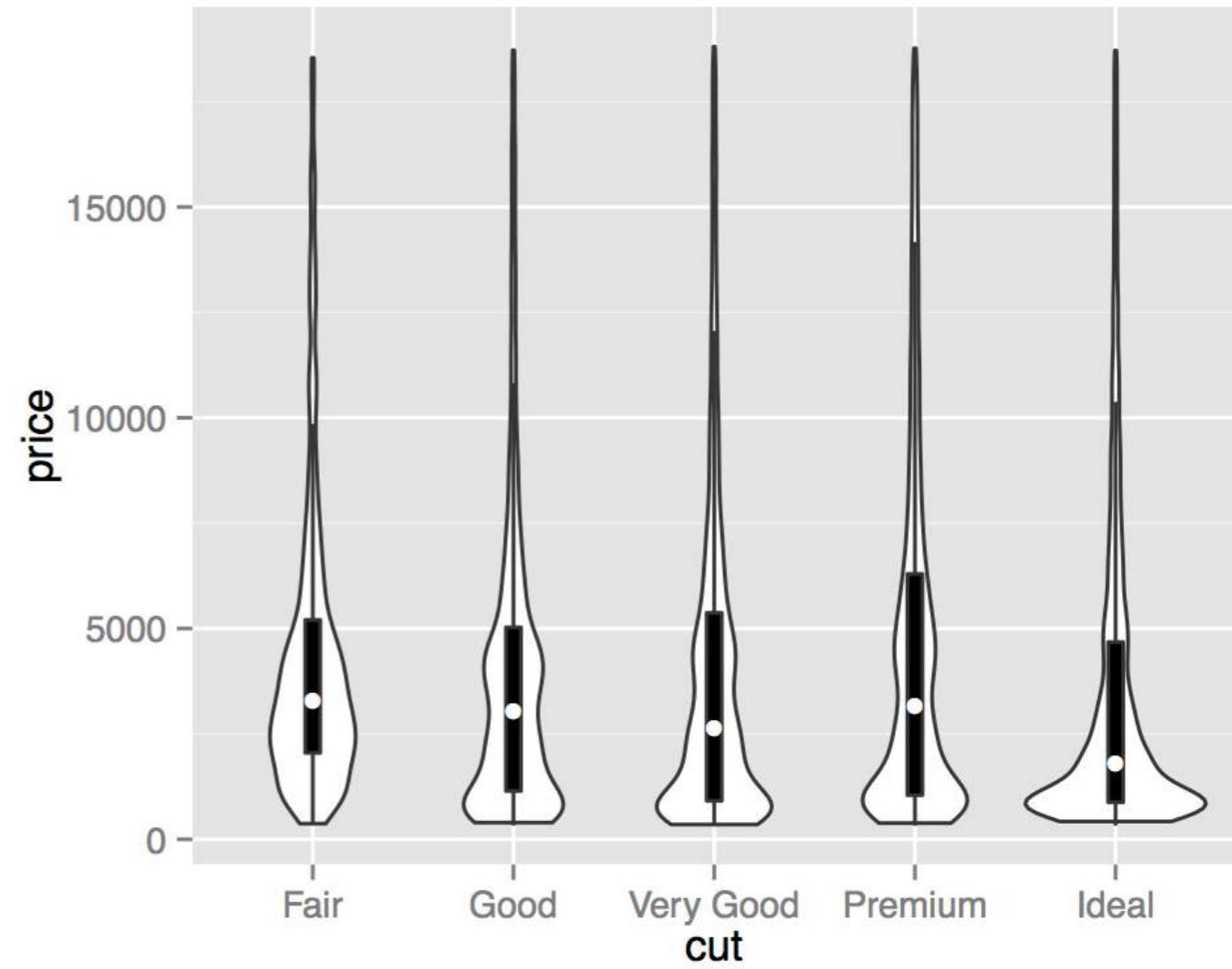
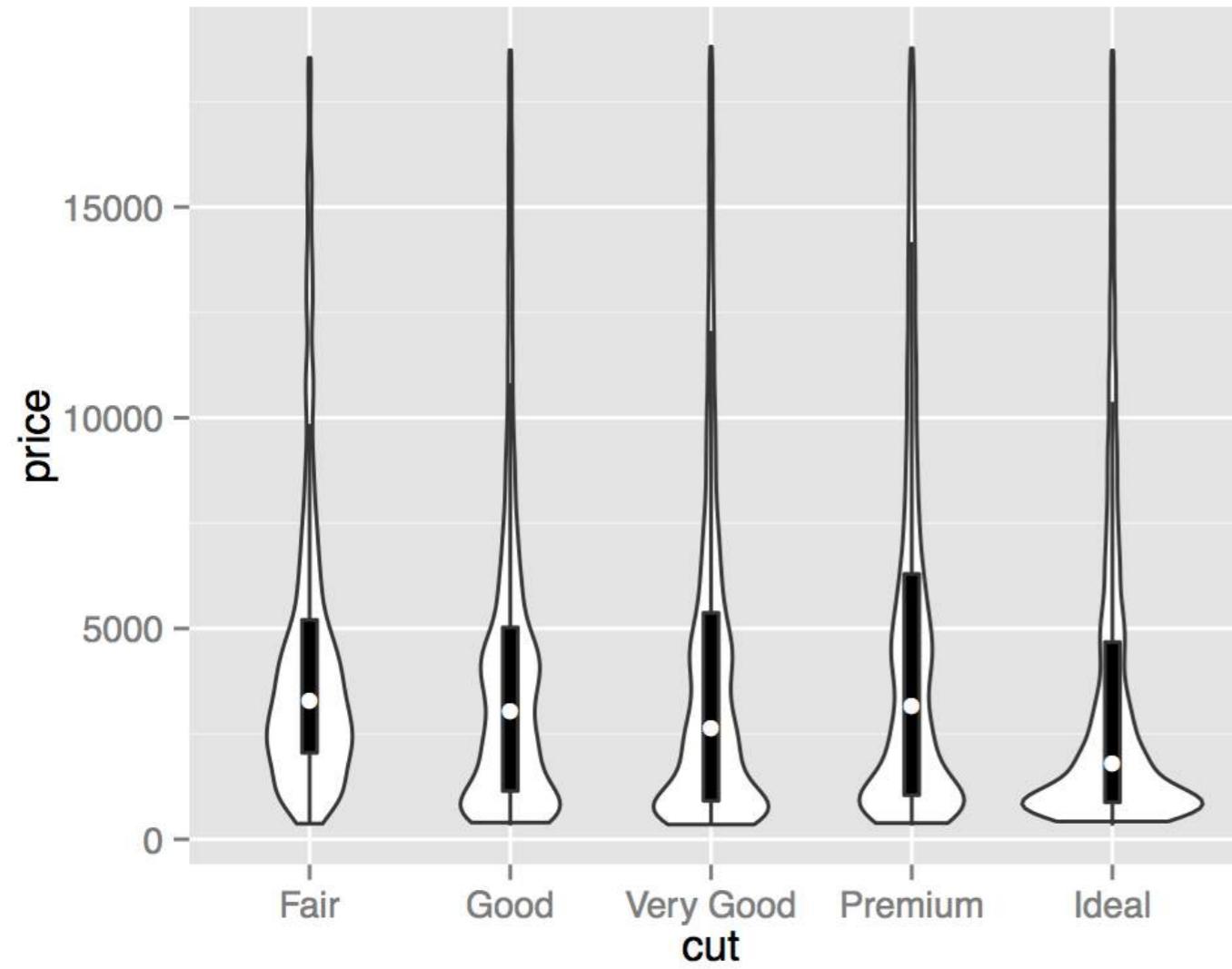
小提琴图因其形状类似于乐器小提琴而得名，当与叠加的箱线图结合时，这一点尤为明显。该可视化同时描述了基础分布的Tukey五数概括（以箱线图形式）和完整的连续密度估计（小提琴图）。

```
ggplot(diamonds, aes(cut, price)) +
  geom_violin() +
  geom_boxplot(width = .1, fill = "black", outlier.shape = NA) +
  stat_summary(fun.y = "median", geom = "point", col = "white")
```



Violin plots are named for their resemblance to the musical instrument, this is particularly visible when they are coupled with an overlaid boxplot. This visualisation then describes the underlying distributions both in terms of Tukey's 5 number summary (as boxplots) and full continuous density estimates (violins).

```
ggplot(diamonds, aes(cut, price)) +
  geom_violin() +
  geom_boxplot(width = .1, fill = "black", outlier.shape = NA) +
  stat_summary(fun.y = "median", geom = "point", col = "white")
```



第29章：因子

第29.1节：使用列表合并因子水平

有时希望将因子水平合并为更少的组，可能是因为某个类别中的数据稀疏。也可能是因为类别名称的拼写或大小写不一致。以因子为例

```
set.seed(1)
colorful <- 样本(c("red", "Red", "RED", "blue", "Blue", "BLUE", "green", "gren"),
                 size = 20,
                 replace = TRUE)
colorful <- factor(colorful)
```

由于R区分大小写，该向量的频率表如下所示。

```
table(colorful)
```

colorful	blue	Blue	BLUE	green	gren	red	Red	RED
3	1	4	2	4	1	3	3	2

然而，该表并未反映数据的真实分布，类别实际上可以归结为三种类型：蓝色、绿色和红色。提供了三个示例。第一个展示了看似明显的解决方案，但实际上无法解决问题。第二个给出了可行的解决方案，但冗长且计算开销大。第三个不是显而易见的解决方案，但相对简洁且计算效率高。

使用factor合并水平 (factor_approach)

```
factor(as.character(colorful),
       levels = c("blue", "Blue", "BLUE", "green", "gren", "red", "Red", "RED"),
       labels = c("蓝色", "蓝色", "蓝色", "绿色", "绿色", "红色", "红色"))
```

```
[1] 绿色 蓝色 红色 红色 蓝色 红色 红色 蓝色 红色 绿色 绿色 绿色 蓝色  
红色 绿色  
[17] 红色 绿色 绿色 红色  
级别: 蓝色 蓝色 蓝色 绿色 绿色 红色 红色  
警告 message:  
在 `levels<-`(*tmp*), value = if (nl == nL) as.character(labels) else paste0(labels, :  
因子中的重复级别已被弃用
```

注意存在重复的级别。我们仍然有三个“蓝色”类别，这并未完成合并级别的任务。此外，有一个警告提示重复级别已被弃用，这意味着这段代码将来可能会产生错误。

使用 ifelse 合并级别 (ifelse_方法)

```
factor(ifelse(colorful %in% c("blue", "Blue", "BLUE"),
              "蓝色",
              ifelse(colorful %in% c("green", "gren"),
                     "绿色",
                     "红色")))
```

```
[1] 绿色 蓝色 红色 红色 蓝色 红色 红色 蓝色 红色 绿色 绿色 绿色 蓝色  
红色 绿色
```

Chapter 29: Factors

Section 29.1: Consolidating Factor Levels with a List

There are times in which it is desirable to consolidate factor levels into fewer groups, perhaps because of sparse data in one of the categories. It may also occur when you have varying spellings or capitalization of the category names. Consider as an example the factor

```
set.seed(1)
colorful <- sample(c("red", "Red", "RED", "blue", "Blue", "BLUE", "green", "gren"),
                  size = 20,
                  replace = TRUE)
colorful <- factor(colorful)
```

Since R is case-sensitive, a frequency table of this vector would appear as below.

```
table(colorful)
```

colorful	blue	Blue	BLUE	green	gren	red	Red	RED
3	1	4	2	4	1	3	3	2

This table, however, doesn't represent the true distribution of the data, and the categories may effectively be reduced to three types: Blue, Green, and Red. Three examples are provided. The first illustrates what seems like an obvious solution, but won't actually provide a solution. The second gives a working solution, but is verbose and computationally expensive. The third is not an obvious solution, but is relatively compact and computationally efficient.

Consolidating levels using factor (factor_approach)

```
factor(as.character(colorful),
       levels = c("blue", "Blue", "BLUE", "green", "gren", "red", "Red", "RED"),
       labels = c("Blue", "Blue", "Blue", "Green", "Green", "Red", "Red"))
```

```
[1] Green Blue Red Red Blue Red Red Red Blue Red Green Green Green Blue  
Red Green  
[17] Red Green Green Red  
Levels: Blue Blue Blue Green Green Red Red Red  
Warning message:  
In `levels<-`(*tmp*), value = if (nl == nL) as.character(labels) else paste0(labels, :  
duplicated levels in factors are deprecated
```

Notice that there are duplicated levels. We still have three categories for "Blue", which doesn't complete our task of consolidating levels. Additionally, there is a warning that duplicated levels are deprecated, meaning that this code may generate an error in the future.

Consolidating levels using ifelse (ifelse_approach)

```
factor(ifelse(colorful %in% c("blue", "Blue", "BLUE"),
              "Blue",
              ifelse(colorful %in% c("green", "gren"),
                     "Green",
                     "Red")))
```

```
[1] Green Blue Red Red Blue Red Red Red Blue Red Green Green Green Blue  
Red Green
```

```
[17] 红色 绿色 绿色 红色  
级别: 蓝色 绿色 红色
```

这段代码能够生成所需的结果，但需要使用嵌套的ifelse语句。虽然这种方法没有问题，但管理嵌套的ifelse语句可能是一项繁琐的工作，且必须小心进行。

使用列表 (list_approach) 合并因子水平合并水平的一个不太

明显的方法是使用列表，其中每个元素的名称是所需的类别名称，元素是因子中应映射到所需类别的水平的字符向量。这种方法的额外优点是直接作用于因子的levels属性，无需分配新的对象。

```
levels(colorful) <-  
  list("Blue" = c("blue", "Blue", "BLUE"),  
       "Green" = c("green", "gren"),  
       "Red" = c("red", "Red", "RED"))
```

```
[1] 绿色 蓝色 红色 红色 蓝色 红色 红色 蓝色 红色 绿色 绿色 绿色 蓝色  
红色 绿色  
[17] 红色 绿色 绿色 红色  
级别: 蓝色 绿色 红色
```

各方法的基准测试

执行每种方法所需的时间总结如下。（为节省空间，生成此摘要的代码未显示）

expr	min	lq	mean	median	uq	max	neval	cld
factor	78.725	83.256	93.26023	87.5030	97.131	218.899	100	b
ifelse	104.494	107.609	123.53793	113.4145	128.281	254.580	100	c
list_approach	49.557	52.955	60.50756	54.9370	65.132	138.193	100	a

列表方法的运行速度约为ifelse方法的两倍。然而，除非数据量非常非常大，否则执行时间的差异很可能仅以微秒或毫秒计。

在此如此微小的时间差异下，效率不必成为选择使用哪种方法的决定因素。相反，应选择一种熟悉且舒适的方法，并且您和您的合作者在未来复查时能够理解。

第29.2节：因子的基本创建

因子是表示R中分类变量的一种方式。因子在内部存储为一个整数向量。所提供的字符向量的唯一元素被称为因子的水平。默认情况下，如果用户未提供水平，则R会生成向量中的唯一值集合，对这些值进行字母数字排序，并将其用作水平。

```
charvar <- rep(c("n", "c"), each = 3)  
f <- factor(charvar)  
f  
levels(f)  
  
> f  
[1] n n n c c c  
级别: c n  
> levels(f)
```

```
[17] Red Green Green Red  
Levels: Blue Green Red
```

This code generates the desired result, but requires the use of nested `ifelse` statements. While there is nothing wrong with this approach, managing nested `ifelse` statements can be a tedious task and must be done carefully.

Consolidating Factors Levels with a List (list_approach)

A less obvious way of consolidating levels is to use a list where the name of each element is the desired category name, and the element is a character vector of the levels in the factor that should map to the desired category. This has the added advantage of working directly on the `levels` attribute of the factor, without having to assign new objects.

```
levels(colorful) <-  
  list("Blue" = c("blue", "Blue", "BLUE"),  
       "Green" = c("green", "gren"),  
       "Red" = c("red", "Red", "RED"))
```

```
[1] Green Blue Red Red Blue Red Red Red Blue Red Green Green Green Blue  
Red Green  
[17] Red Green Green Red  
Levels: Blue Green Red
```

Benchmarking each approach

The time required to execute each of these approaches is summarized below. (For the sake of space, the code to generate this summary is not shown)

expr	min	lq	mean	median	uq	max	neval	cld
factor	78.725	83.256	93.26023	87.5030	97.131	218.899	100	b
ifelse	104.494	107.609	123.53793	113.4145	128.281	254.580	100	c
list_approach	49.557	52.955	60.50756	54.9370	65.132	138.193	100	a

The list approach runs about twice as fast as the `ifelse` approach. However, except in times of very, very large amounts of data, the differences in execution time will likely be measured in either microseconds or milliseconds. With such small time differences, efficiency need not guide the decision of which approach to use. Instead, use an approach that is familiar and comfortable, and which you and your collaborators will understand on future review.

Section 29.2: Basic creation of factors

Factors are one way to represent categorical variables in R. A factor is stored internally as a **vector of integers**. The unique elements of the supplied character vector are known as the *levels* of the factor. By default, if the levels are not supplied by the user, then R will generate the set of unique values in the vector, sort these values alphabetically, and use them as the levels.

```
charvar <- rep(c("n", "c"), each = 3)  
f <- factor(charvar)  
f  
levels(f)  
  
> f  
[1] n n n c c c  
Levels: c n  
> levels(f)
```

```
[1] "c" "n"
```

如果您想更改级别的顺序，那么一种选择是手动指定级别：

```
levels(因子(charvar, levels = c("n", "c")))
> levels(因子(charvar, levels = c("n", "c")))
[1] "n" "c"
```

因素具有多种属性。例如，水平可以被赋予标签：

```
> f <- factor(charvar, levels=c("n", "c"), labels=c("Newt", "Capybara"))
> f
[1] Newt      Newt      Newt      Capybara  Capybara  Capybara
Levels: Newt Capybara
```

另一个可以指定的属性是因子是否有序：

```
> Weekdays <- factor(c("Monday", "Wednesday", "Thursday", "Tuesday", "Friday", "Sunday",
  "Saturday"))
> Weekdays
[1] Monday      Wednesday Thursday Tuesday Friday     Sunday Saturday
Levels: Friday Monday Saturday Sunday Thursday Tuesday Wednesday
> Weekdays <- factor(Weekdays, levels=c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
  "Saturday", "Sunday"), ordered=TRUE)
> Weekdays
[1] Monday      Wednesday Thursday Tuesday Friday     Sunday Saturday
Levels: Monday < Tuesday < Wednesday < Thursday < Friday < Saturday < Sunday
```

当因子的某个水平不再使用时，可以使用 droplevels() 函数将其删除：

```
> Weekend <- subset(Weekdays, Weekdays == "Saturday" | Weekdays == "Sunday")
> Weekend
[1] 星期日 星期六
级别: 星期一 < 星期二 < 星期三 < 星期四 < 星期五 < 星期六 < 星期日
> 周末 <- droplevels(周末)
> 周末
[1] 星期日 星期六
级别: 星期六 < 星期日
```

第29.3节：更改和重新排序因子

当因子使用默认值创建时，级别 是通过对输入应用 as.character 形成的，并按字母顺序排列。

```
charvar <- rep(c("W", "n", "c"), 次数=c(17,20,14))
f <- factor(charvar)
级别(f)
# [1] "c" "n" "W"
```

在某些情况下，默认的级别排序（字母/词汇顺序）的处理是可以接受的。例如，如果只是想绘制频率，结果将是：

```
plot(f,col=1:length(级别(f)))
```

```
[1] "c" "n"
```

If you want to change the ordering of the levels, then one option is to specify the levels manually:

```
levels(factor(charvar, levels = c("n", "c")))
> levels(factor(charvar, levels = c("n", "c")))
[1] "n" "c"
```

Factors have a number of properties. For example, levels can be given labels:

```
> f <- factor(charvar, levels=c("n", "c"), labels=c("Newt", "Capybara"))
> f
[1] Newt      Newt      Newt      Capybara  Capybara  Capybara
Levels: Newt Capybara
```

Another property that can be assigned is whether the factor is ordered:

```
> Weekdays <- factor(c("Monday", "Wednesday", "Thursday", "Tuesday", "Friday", "Sunday",
  "Saturday"))
> Weekdays
[1] Monday      Wednesday Thursday Tuesday Friday     Sunday Saturday
Levels: Friday Monday Saturday Sunday Thursday Tuesday Wednesday
> Weekdays <- factor(Weekdays, levels=c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
  "Saturday", "Sunday"), ordered=TRUE)
> Weekdays
[1] Monday      Wednesday Thursday Tuesday Friday     Sunday Saturday
Levels: Monday < Tuesday < Wednesday < Thursday < Friday < Saturday < Sunday
```

When a level of the factor is no longer used, you can drop it using the droplevels() function:

```
> Weekend <- subset(Weekdays, Weekdays == "Saturday" | Weekdays == "Sunday")
> Weekend
[1] Sunday Saturday
Levels: Monday < Tuesday < Wednesday < Thursday < Friday < Saturday < Sunday
> Weekend <- droplevels(Weekend)
> Weekend
[1] Sunday Saturday
Levels: Saturday < Sunday
```

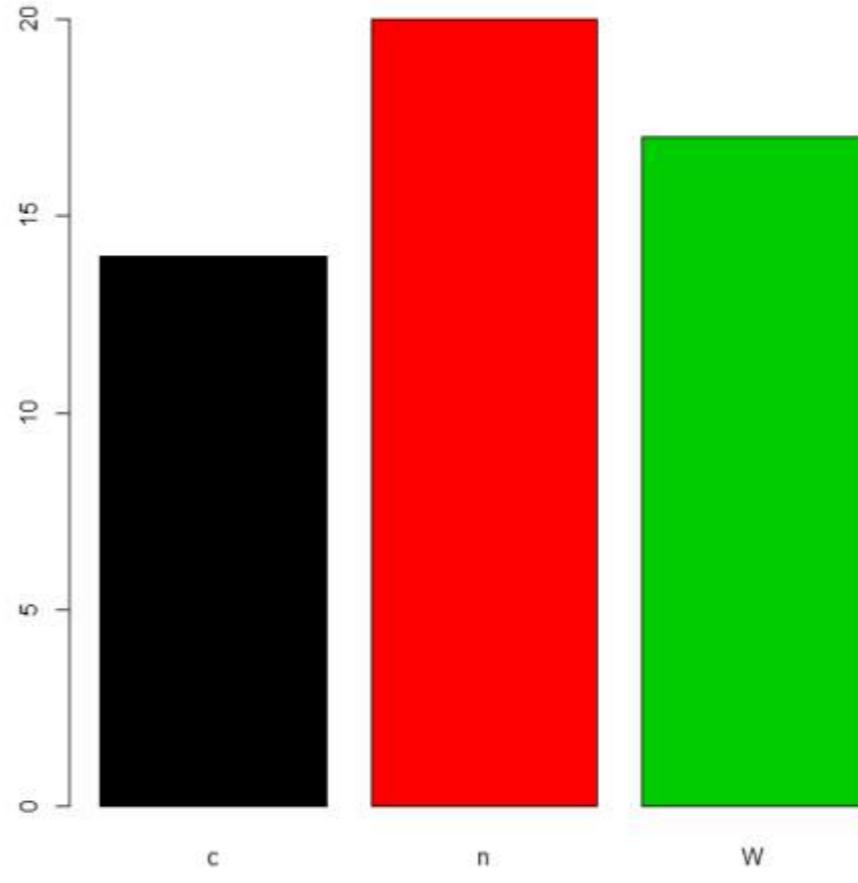
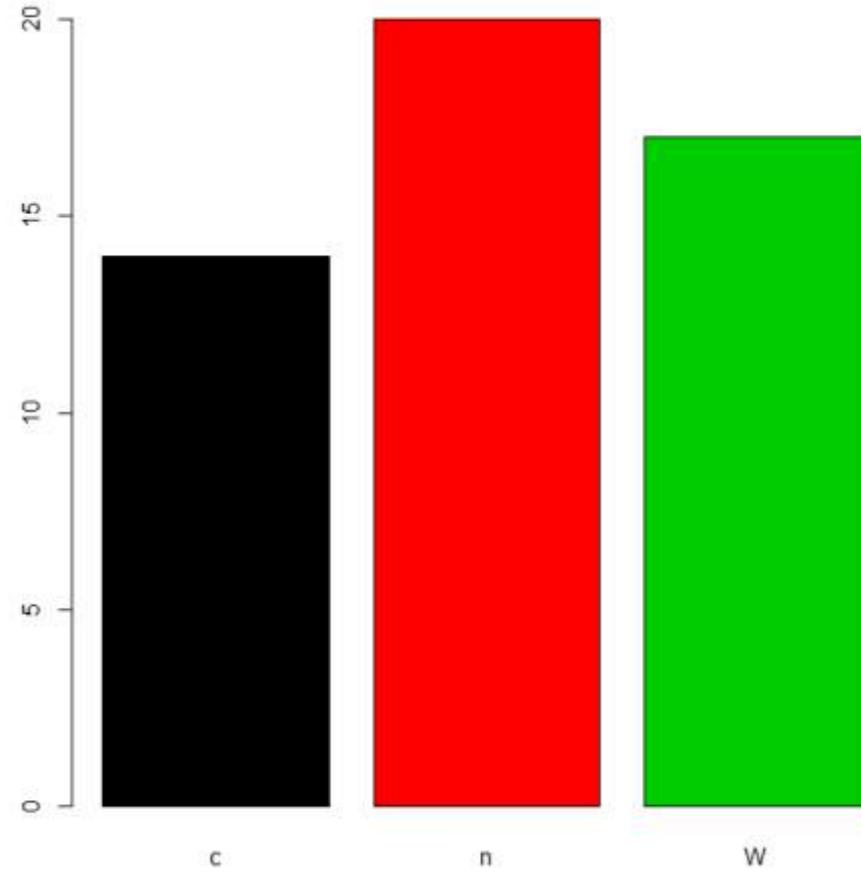
Section 29.3: Changing and reordering factors

When factors are created with defaults, **levels** are formed by **as.character** applied to the inputs and are ordered alphabetically.

```
charvar <- rep(c("W", "n", "c"), times=c(17,20,14))
f <- factor(charvar)
levels(f)
# [1] "c" "n" "W"
```

In some situations the treatment of the default ordering of **levels** (alphabetic/lexical order) will be acceptable. For example, if one just wants to **plot** the frequencies, this will be the result:

```
plot(f,col=1:length(levels(f)))
```



但如果我们想要不同的levels排序，我们需要在levels或labels参数中指定（注意这里“order”的含义不同于ordered因子，见下文）。根据具体情况，有多种方法可以实现这一任务。

1. 重新定义因子

当可能时，我们可以使用levels参数以想要的顺序重新创建因子。

```
ff <- factor(charvar, levels = c("n", "W", "c"))
levels(ff)
# [1] "n" "W" "c"

gg <- factor(charvar, levels = c("W", "c", "n"))
levels(gg)
# [1] "W" "c" "n"
```

当输入的levels与期望的输出levels不同，我们使用labels参数，这会使levels参数成为可接受输入值的“过滤器”，但因子向量最终的“levels”值由labels参数决定：

```
fm <- factor(as.numeric(f), levels = c(2,3,1),
             labels = c("nn", "WW", "cc"))
levels(fm)
# [1] "nn" "WW" "cc"

fm <- factor(LETTERS[1:6], levels = LETTERS[1:4], # 仅将 'A'-'D' 作为输入
             labels = letters[1:4]) # 但赋值为 'a'-d'
```

But if we want a different ordering of `levels`, we need to specify this in the `levels` or `labels` parameter (taking care that the meaning of "order" here is different from `ordered` factors, see below). There are many alternatives to accomplish that task depending on the situation.

1. Redefine the factor

When it is possible, we can recreate the factor using the `levels` parameter with the order we want.

```
ff <- factor(charvar, levels = c("n", "W", "c"))
levels(ff)
# [1] "n" "W" "c"

gg <- factor(charvar, levels = c("W", "c", "n"))
levels(gg)
# [1] "W" "c" "n"
```

When the input levels are different than the desired output levels, we use the `labels` parameter which causes the `levels` parameter to become a "filter" for acceptable input values, but leaves the final values of "levels" for the factor vector as the argument to `labels`:

```
fm <- factor(as.numeric(f), levels = c(2,3,1),
             labels = c("nn", "WW", "cc"))
levels(fm)
# [1] "nn" "WW" "cc"

fm <- factor(LETTERS[1:6], levels = LETTERS[1:4], # only 'A'-'D' as input
             labels = letters[1:4]) # but assigned to 'a'-'d'
```

```
fm  
#[1] a b c d <NA> <NA>  
#Levels: a b c d
```

2. 使用 relevel 函数

当有一个特定的 level 需要作为第一个时，我们可以使用 `relevel`。比如在统计分析的情境中，当需要一个 base 类别来进行假设检验时会用到。

```
g<-relevel(f, "n") # 将 n 移动为第一个水平  
levels(g)  
# [1] "n" "c" "W"
```

可以验证 f 和 g 是相同的

```
all.equal(f, g)  
# [1] "属性: < 组件 “级别”: 2 个字符串不匹配 >"  
all.equal(f, g, check.attributes = F)  
# [1] TRUE
```

3. 重新排序因子

有时我们需要根据数字、部分结果、计算统计量或之前的计算结果来重新排序因子的级别。让我们根据级别的频率来重新排序

```
table(g)  
# g  
# n c W  
# 20 14 17
```

函数reorder是通用的（参见`help(reordered)`），但在此上下文中需要：x，即因子；X，与|x|长度相同的数值；以及 FUN，一个应用于X并按|x|的级别计算的函数，FUN决定了级别的顺序，默认是递增。结果是同一个因子，但其级别被重新排序。

```
g.ord <- reorder(g, rep(1,length(g)), FUN=sum) #递增  
levels(g.ord)  
# [1] "c" "W" "n"
```

要获得递减顺序，我们考虑负值 (-1)

```
g.ord.d <- reorder(g, rep(-1,length(g)), FUN=sum)  
levels(g.ord.d)  
# [1] "n" "W" "c"
```

因子与其他因子相同。

```
data.frame(f,g,g.ord,g.ord.d)[seq(1,length(g),by=5),] #仅相同行  
# f g g.ord g.ord.d  
# 1 W W W W  
# 6 W W W W  
# 11 W W W W  
# 16 W W W W  
# 21 n n n n  
# 26 n n n n  
# 31 n n n n  
# 36 n n n n  
# 41 c c c c
```

```
fm  
#[1] a b c d <NA> <NA>  
#Levels: a b c d
```

2. Use `relevel` function

When there is one specific level that needs to be the first we can use `relevel`. This happens, for example, in the context of statistical analysis, when a base category is necessary for testing hypothesis.

```
g<-relevel(f, "n") # moves n to be the first level  
levels(g)  
# [1] "n" "c" "W"
```

As can be verified f and g are the same

```
all.equal(f, g)  
# [1] "Attributes: < Component “levels”: 2 string mismatches >"  
all.equal(f, g, check.attributes = F)  
# [1] TRUE
```

3. Reordering factors

There are cases when we need to `reorder` the `levels` based on a number, a partial result, a computed statistic, or previous calculations. Let's reorder based on the `frequencies` of the `levels`

```
table(g)  
# g  
# n c W  
# 20 14 17
```

The `reorder` function is generic (see `help(reordered)`), but in this context needs: x, in this case the factor; X, a numeric value of the same length as x; and FUN, a function to be applied to X and computed by level of the x, which determines the `levels` order, by default increasing. The result is the same factor with its levels reordered.

```
g.ord <- reorder(g, rep(1,length(g)), FUN=sum) #increasing  
levels(g.ord)  
# [1] "c" "W" "n"
```

To get de decreasing order we consider negative values (-1)

```
g.ord.d <- reorder(g, rep(-1,length(g)), FUN=sum)  
levels(g.ord.d)  
# [1] "n" "W" "c"
```

Again the factor is the same as the others.

```
data.frame(f,g,g.ord,g.ord.d)[seq(1,length(g),by=5),] #just same lines  
# f g g.ord g.ord.d  
# 1 W W W W  
# 6 W W W W  
# 11 W W W W  
# 16 W W W W  
# 21 n n n n  
# 26 n n n n  
# 31 n n n n  
# 36 n n n n  
# 41 c c c c
```

```
# 46 c c      c      c  
# 51 c c      c      c
```

当存在与因子变量相关的定量变量时，我们可以使用其他函数重新排序
levels。以iris数据为例（更多信息请参见[help\("iris"\)](#)），通过其

平均萼片宽度 (Sepal.Width) 重新排序Species因子。

```
miris <- iris #help("iris") # 复制数据  
with(miris, tapply(Sepal.Width, Species, mean))  
#   setosa versicolor virginica  
#   3.428     2.770     2.974  
  
miris$Species.o<-with(miris, reorder(Species, -Sepal.Width))  
levels(miris$Species.o)  
# [1] "山鸢尾"    "维吉尼亚鸢尾" "变色鸢尾"
```

通常的**箱线图**（例如：[使用\(miris, boxplot\(Petal.Width~Species\)\)](#)）会按以下顺序显示物种：山鸢尾，变色鸢尾，和维吉尼亚鸢尾。但使用有序因子，我们可以按平均萼片宽度对物种进行排序：

```
boxplot(Petal.Width~Species.o, data = miris,  
       xlab = "物种", ylab = "花瓣宽度",  
       main = "鸢尾数据，按平均萼片宽度排序", varwidth = TRUE,  
       col = 2:4)
```

```
# 46 c c      c      c  
# 51 c c      c      c
```

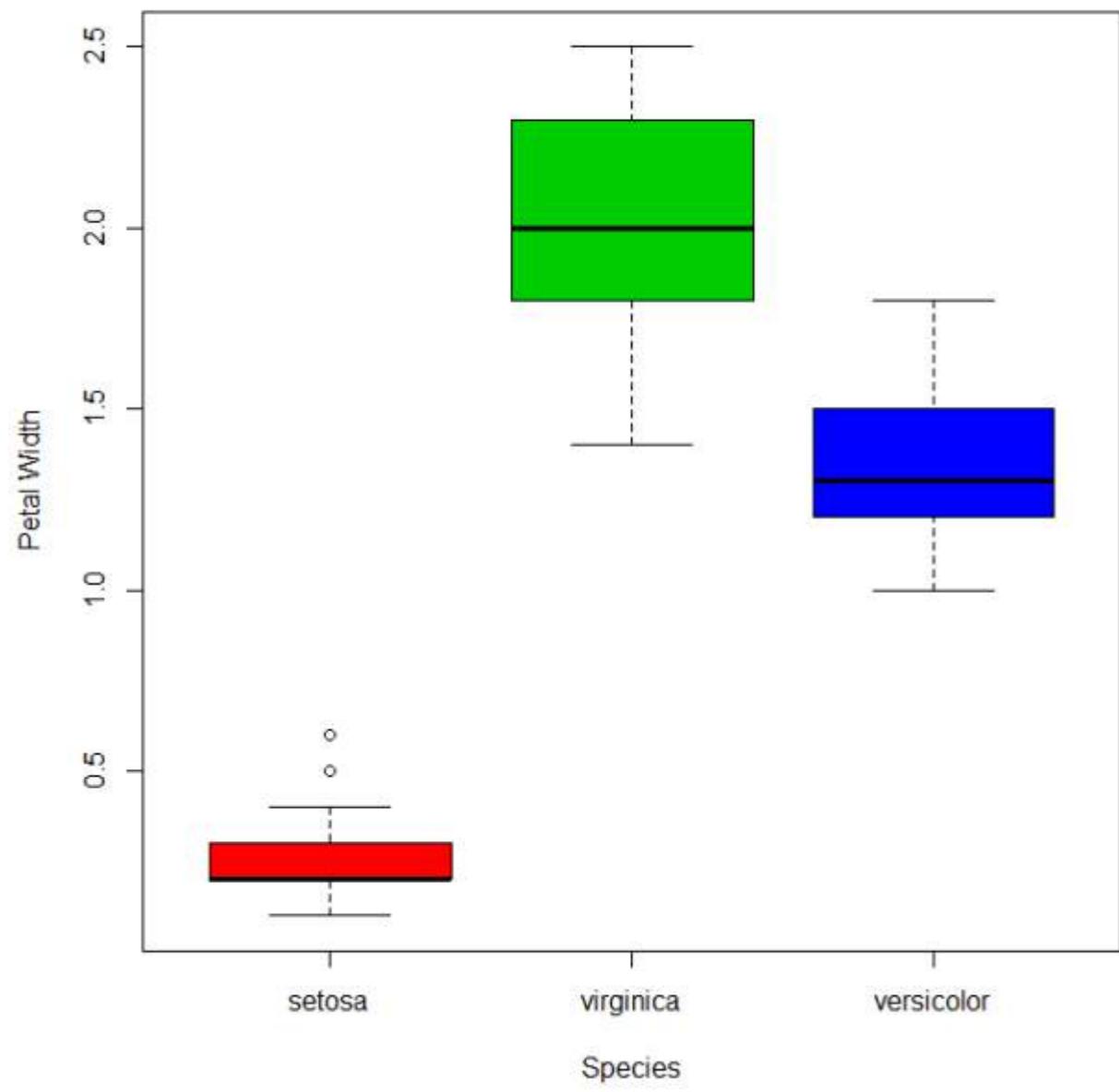
When there is a **quantitative variable** related to the factor variable, we could use other functions to reorder the **levels**. Lets take the **iris** data ([help\("iris"\)](#) for more information), for reordering the Species factor by using its mean Sepal.Width.

```
miris <- iris #help("iris") # copy the data  
with(miris, tapply(Sepal.Width, Species, mean))  
#   setosa versicolor virginica  
#   3.428     2.770     2.974  
  
miris$Species.o<-with(miris, reorder(Species, -Sepal.Width))  
levels(miris$Species.o)  
# [1] "setosa"    "virginica"  "versicolor"
```

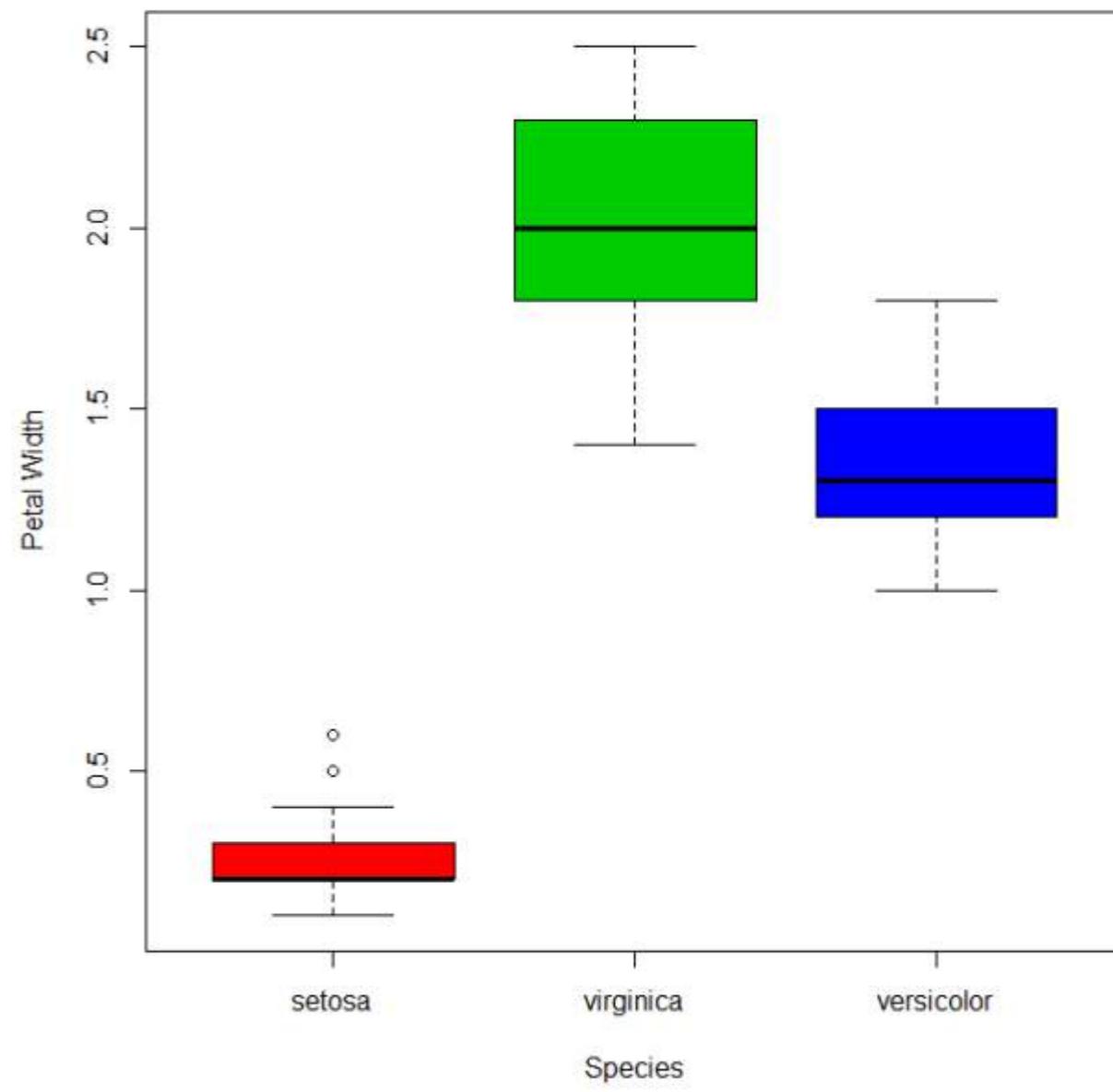
The usual **boxplot** (say: [with\(miris, boxplot\(Petal.Width~Species\)\)](#)) will show the species in this order: *setosa*, *versicolor*, and *virginica*. But using the ordered factor we get the species ordered by its mean Sepal.Width:

```
boxplot(Petal.Width~Species.o, data = miris,  
       xlab = "Species", ylab = "Petal Width",  
       main = "Iris Data, ordered by mean sepal width", varwidth = TRUE,  
       col = 2:4)
```

Iris Data, ordered by mean sepal width



Iris Data, ordered by mean sepal width



此外，也可以更改levels的名称，将它们合并成组，或添加新的levels。为此，我们使用同名函数levels。

```
f1<-f
levels(f1)
# [1] "c" "n" "W"
levels(f1) <- c("upper","upper","CAP") #重命名和分组
levels(f1)
# [1] "upper" "CAP"

f2<-f1
levels(f2) <- c("upper","CAP", "Number") #添加空的Number级别
levels(f2)
# [1] "upper" "CAP"      "Number"
f2[length(f2):(length(f2)+5)]<- "Number" # 为新级别添加案例
table(f2)
# f2
#   upper     CAP Number
#   33       17      6
```

Additionally, it is also possible to change the names of `levels`, combine them into groups, or add new `levels`. For that we use the function of the same name `levels`.

```
f1<-f
levels(f1)
# [1] "c" "n" "W"
levels(f1) <- c("upper","upper","CAP") #rename and grouping
levels(f1)
# [1] "upper" "CAP"

f2<-f1
levels(f2) <- c("upper","CAP", "Number") #add Number level, which is empty
levels(f2)
# [1] "upper" "CAP"      "Number"
f2[length(f2):(length(f2)+5)]<- "Number" # add cases for the new level
table(f2)
# f2
#   upper     CAP Number
#   33       17      6
```

```

f3<-f1
levels(f3) <- list(G1 = "upper", G2 = "CAP", G3 = "Number") # 使用列表的同样方法
levels(f3)
# [1] "G1" "G2" "G3"
f3[length(f3):(length(f3)+6)]<-"G3" ## 为新级别添加案例
table(f3)
# f3
# G1 G2 G3
# 33 17 7

```

- 有序因子

最后，我们知道**有序**因子不同于因子，前者用于表示**有序数据**，后者用于处理**名义数据**。起初，改变有序因子的**水平**顺序没有意义，但我们可以更改其**标签**。

```

ordvar<-rep(c("低", "中", "高"), times=c(7,2,4))

of<-ordered(ordvar, levels=c("低", "中", "高"))
levels(of)
# [1] "低" "中" "高"

of1<-of
levels(of1)<- c("低", "中", "高")
levels(of1)
# [1] "低" "中" "高"
is.ordered(of1)
# [1] TRUE
of1
# [1] 低 低 低 低 低 中 中 高 高 高
# 等级：低 < 中 < 高

```

第29.4节：从零重建因子

问题

因子用于表示取自一组类别的变量，这些类别在R中称为等级。例如，某个实验可以用电池的能量等级来描述，分为四个等级：空、低、正常和满。然后，对于5个不同的采样点，这些等级可以用以下方式表示：

满, 满, 正常, 空, 低

通常，在数据库或其他信息源中，这些数据是通过与类别或等级相关联的任意整数索引来处理的。如果我们假设，对于给定的例子，我们将索引分配如下：1 = 空，2 = 低，3 = 正常，4 = 满，那么这5个样本可以编码为：

4, 4, 3, 1, 2

可能会出现这样的情况：从你的信息来源，例如数据库，你只有编码后的整数列表，以及将每个整数与每个等级关键词关联的目录。如何从这些信息中重建R的因子？

解决方案

```

f3<-f1
levels(f3) <- list(G1 = "upper", G2 = "CAP", G3 = "Number") # The same using list
levels(f3)
# [1] "G1" "G2" "G3"
f3[length(f3):(length(f3)+6)]<-"G3" ## add cases for the new level
table(f3)
# f3
# G1 G2 G3
# 33 17 7

```

- Ordered factors

Finally, we know that **ordered** factors are different from factors, the first one are used to represent *ordinal data*, and the second one to work with *nominal data*. At first, it does not make sense to change the order of **levels** for ordered factors, but we can change its **labels**.

```

ordvar<-rep(c("Low", "Medium", "High"), times=c(7,2,4))

of<-ordered(ordvar, levels=c("Low", "Medium", "High"))
levels(of)
# [1] "Low" "Medium" "High"

of1<-of
levels(of1)<- c("LOW", "MEDIUM", "HIGH")
levels(of1)
# [1] "LOW" "MEDIUM" "HIGH"
is.ordered(of1)
# [1] TRUE
of1
# [1] LOW LOW LOW LOW LOW LOW MEDIUM MEDIUM HIGH HIGH HIGH HIGH
# Levels: LOW < MEDIUM < HIGH

```

Section 29.4: Rebuilding factors from zero

Problem

Factors are used to represent variables that take values from a set of categories, known as Levels in R. For example, some experiment could be characterized by the energy level of a battery, with four levels: empty, low, normal, and full. Then, for 5 different sampling sites, those levels could be identified, in those terms, as follows:

full, full, normal, empty, low

Typically, in databases or other information sources, the handling of these data is by arbitrary integer indices associated with the categories or levels. If we assume that, for the given example, we would assign, the indices as follows: 1 = empty, 2 = low, 3 = normal, 4 = full, then the 5 samples could be coded as:

4, 4, 3, 1, 2

It could happen that, from your source of information, e.g. a database, you only have the encoded list of integers, and the catalog associating each integer with each level-keyword. How can a factor of R be reconstructed from that information?

Solution

我们将模拟一个包含20个整数的向量，表示样本，每个样本可能具有四种不同的值之一：

```
set.seed(18)
ii <- sample(1:4, 20, replace=T)
ii
```

```
[1] 4 3 4 1 1 3 2 3 2 1 3 4 1 2 4 1 3 1 4 1
```

第一步是从之前的序列中创建一个因子，其水平或类别正好是从1到4的数字。

```
fii <- factor(ii, levels=1:4) # 需要指明数值水平
fii
```

```
[1] 4 3 4 1 1 3 2 3 2 1 3 4 1 2 4 1 3 1 4 1
Levels: 1 2 3 4
```

现在只需用索引标签为已创建的因子“穿衣”即可：

```
levels(fii) <- c("empty", "low", "normal", "full")
fii
```

```
[1] full normal full empty empty normal low empty
[11] normal full empty low full empty normal empty full empty
Levels: empty low normal full
```

We will simulate a vector of 20 integers that represents the samples, each of which may have one of four different values:

```
set.seed(18)
ii <- sample(1:4, 20, replace=T)
ii
```

```
[1] 4 3 4 1 1 3 2 3 2 1 3 4 1 2 4 1 3 1 4 1
```

The first step is to make a factor, from the previous sequence, in which the levels or categories are exactly the numbers from 1 to 4.

```
fii <- factor(ii, levels=1:4) # it is necessary to indicate the numeric levels
fii
```

```
[1] 4 3 4 1 1 3 2 3 2 1 3 4 1 2 4 1 3 1 4 1
Levels: 1 2 3 4
```

Now simply, you have to *dress* the factor already created with the index tags:

```
levels(fii) <- c("empty", "low", "normal", "full")
fii
```

```
[1] full normal full empty empty normal low normal low empty
[11] normal full empty low full empty normal empty full empty
Levels: empty low normal full
```

第30章：模式匹配与替换

本主题涵盖匹配字符串模式，以及提取或替换它们。有关定义复杂模式的详细信息，请参见正则表达式。

第30.1节：查找匹配项

```
# 示例数据  
test_sentences <- c("快速的棕色狐狸", "跳过懒狗")
```

是否有匹配？

`grepl()` 用于检查字符串或字符向量中是否存在某个单词或正则表达式。该函数返回一个 TRUE/FALSE（或“布尔”）向量。

注意，我们可以检查每个字符串中是否包含单词“fox”，并返回一个布尔向量。

```
grepl("fox", test_sentences)  
#[1] TRUE FALSE
```

匹配位置

`grep` 接受一个字符字符串和一个正则表达式。它返回一个索引的数字向量。该函数将返回包含单词“fox”的句子索引。

```
grep("fox", test_sentences)  
#[1] 1
```

匹配的值

选择匹配某个模式的句子：

```
# 以下每一行都能完成该任务：  
test_sentences[grep("fox", test_sentences)]  
test_sentences[grepl("fox", test_sentences)]  
grep("fox", test_sentences, value = TRUE)  
# [1] "The quick brown fox"
```

详细信息

由于“fox”模式只是一个单词，而不是正则表达式，我们可以通过指定`fixed = TRUE`来提升性能（无论是使用`grep`还是`grepl`）。

```
grep("fox", test_sentences, fixed = TRUE)  
#[1] 1
```

要选择不匹配某个模式的句子，可以使用带有`invert = TRUE`参数的`grep`；或者按照子集规则使用`-grep(...)`或`!grepl(...)`。

在`grepl(pattern, x)`和`grep(pattern, x)`中，参数 `x`是向量化的，而`pattern`参数不是。因此，不能直接用它们来匹配`pattern[1]`与 `x[1]`，`pattern[2]`与 `x[2]`，依此类推。

匹配总结

执行例如`grepl`命令后，您可能想要了解匹配的数量概况 TRUE 或 FALSE。这在处理大数据集时非常有用。例如，为此可以运行 `summary` 命令：

Chapter 30: Pattern Matching and Replacement

This topic covers matching string patterns, as well as extracting or replacing them. For details on defining complicated patterns see Regular Expressions.

Section 30.1: Finding Matches

```
# example data  
test_sentences <- c("The quick brown fox", "jumps over the lazy dog")
```

Is there a match?

`grepl()` 是用于检查字符串或字符向量中是否存在某个单词或正则表达式的函数。该函数返回一个 TRUE/FALSE（或“布尔”）向量。

注意，我们可以检查每个字符串中是否包含单词“fox”，并返回一个布尔向量。

```
grepl("fox", test_sentences)  
#[1] TRUE FALSE
```

Match locations

`grep` 接受一个字符字符串和一个正则表达式。它返回一个数字向量。该函数将返回包含单词“fox”的句子索引。

```
grep("fox", test_sentences)  
#[1] 1
```

Matched values

To select sentences that match a pattern:

```
# each of the following lines does the job:  
test_sentences[grep("fox", test_sentences)]  
test_sentences[grepl("fox", test_sentences)]  
grep("fox", test_sentences, value = TRUE)  
# [1] "The quick brown fox"
```

Details

由于“fox”模式只是一个单词，而不是正则表达式，我们可以通过指定`fixed = TRUE`来提升性能（无论是使用`grep`还是`grepl`）。

```
grep("fox", test_sentences, fixed = TRUE)  
#[1] 1
```

To select sentences that don't match a pattern, one can use `grep` with `invert = TRUE`; or follow subsetting rules with `-grep(...)` or `!grepl(...)`。

In both `grepl(pattern, x)` and `grep(pattern, x)`, the `x` parameter is vectorized, the `pattern` parameter is not. As a result, you cannot use these directly to match `pattern[1]` against `x[1]`, `pattern[2]` against `x[2]`, and so on.

Summary of matches

After performing the e.g. the `grepl` command, maybe you want to get an overview about how many matches where TRUE or FALSE. This is useful e.g. in case of big data sets. In order to do so run the `summary` command:

```
# 示例数据
test_sentences <- c("快速的棕色狐狸", "跳过懒狗")

# 查找匹配项
matches <- grep("狐狸", test_sentences)

# 概览
summary(matches)
```

第30.2节：单次匹配与全局匹配

在使用正则表达式时，PCRE的一个修饰符是 `g`，表示全局匹配。

在R中，匹配和替换函数有两个版本：首次匹配和全局匹配：

- `sub(pattern, replacement, text)` 会将文本中模式的第一个匹配项替换为替换内容
- `gsub(pattern, replacement, text)` 与`sub`相同，但会替换模式的每个匹配项
- `regexpr(pattern, text)` 会返回模式首次匹配的位置
- `gregexpr(pattern, text)` 会返回所有匹配项的位置。

一些随机数据：

```
set.seed(123)
teststring <- paste0(sample(letters, 20), collapse="")

# teststring
#[1] "htjuwakqxzpgrsbncvyo"
```

让我们看看如果想用其他内容替换元音会怎样：

```
sub("[aeiou]", " ** 这里是一个元音** ",teststring)
#[1] "htj ** 这里是一个元音** wakqxzpgrsbncvyo"

gsub("[aeiou]", " ** 这里是一个元音** ",teststring)
#[1] "htj ** 这里是一个元音** w ** 这里是一个元音** kqxzpgrsbncv ** 这里是一个元音**  ** 这里
是一个元音** "
```

现在让我们看看如何找到一个辅音后面紧跟一个或多个元音：

```
regexpr("[^aeiou][aeiou]+",teststring)
#[1] 3
#attr("match.length")
#[1] 2
#attr("useBytes")
#[1] TRUE
```

我们在字符串的第3个位置找到了一个长度为2的匹配，即：`ju`

现在如果我们想获取所有匹配项：

```
gregexpr("[^aeiou][aeiou]+",teststring)
#[[1]]
#[1] 3 5 19
#attr("match.length")
#[1] 2 2 2
```

```
# example data
test_sentences <- c("The quick brown fox", "jumps over the lazy dog")

# find matches
matches <- grep("fox", test_sentences)

# overview
summary(matches)
```

Section 30.2: Single and Global match

When working with regular expressions one modifier for PCRE is `g` for global match.

In R matching and replacement functions have two version: first match and global match:

- `sub(pattern, replacement, text)` will replace the first occurrence of pattern by replacement in text
- `gsub(pattern, replacement, text)` will do the same as `sub` but for each occurrence of pattern
- `regexpr(pattern, text)` will return the position of match for the first instance of pattern
- `gregexpr(pattern, text)` will return all matches.

Some random data:

```
set.seed(123)
teststring <- paste0(sample(letters, 20), collapse="")

# teststring
#[1] "htjuwakqxzpgrsbncvyo"
```

Let's see how this works if we want to replace vowels by something else:

```
sub("[aeiou]", " ** HERE WAS A VOWEL** ",teststring)
#[1] "htj ** HERE WAS A VOWEL** wakqxzpgrsbncvyo"

gsub("[aeiou]", " ** HERE WAS A VOWEL** ",teststring)
#[1] "htj ** HERE WAS A VOWEL** w ** HERE WAS A VOWEL** kqxzpgrsbncv ** HERE WAS A VOWEL**  ** HERE
WAS A VOWEL** "
```

Now let's see how we can find a consonant immediately followed by one or more vowel:

```
regexpr("[^aeiou][aeiou]+",teststring)
#[1] 3
#attr("match.length")
#[1] 2
#attr("useBytes")
#[1] TRUE
```

We have a match on position 3 of the string of length 2, i.e: `ju`

Now if we want to get all matches:

```
gregexpr("[^aeiou][aeiou]+",teststring)
#[[1]]
#[1] 3 5 19
#attr("match.length")
#[1] 2 2 2
```

```
#attr("useBytes")
#[1] TRUE
```

这一切都非常棒，但这只给出了匹配的位置，而要知道匹配的内容并不那么容易，这时就用到了`regmatches`，它的唯一目的是从`regexpr`中提取匹配的字符串，但语法有所不同。

让我们将匹配结果保存到一个变量中，然后从原始字符串中提取它们：

```
matches <- gregexpr("[^aeiou][aeiou]+", teststring)
regmatches(teststring, matches)
#[[1]]
#[1] "ju" "wa" "yo"
```

这听起来可能有些奇怪，没有快捷方式，但这允许通过第一个字符串的匹配结果从另一个字符串中提取（想象比较两个长向量，你知道第一个有共同模式但第二个没有，这样可以轻松比较）：

```
teststring2 <- "this is another string to match against"
regmatches(teststring2, matches)
#[[1]]
#[1] "is" " i" "ri"
```

注意：默认情况下，模式不是Perl兼容正则表达式，一些功能如前瞻后顾不被支持，但这里介绍的每个函数都允许通过`perl=TRUE`参数启用它们。

第30.3节：进行替换

```
# 示例数据
test_sentences <- c("快速的棕色狐狸快速地", "跳过懒狗")
```

让我们把棕色狐狸变成红色：

```
sub("brown", "red", test_sentences)
#[1] "快速的红色狐狸快速地"      "跳过懒狗"
```

现在，让“快速”的狐狸表现得“快速地”。这不行：

```
sub("quick", "fast", test_sentences)
#[1] "快速的红色狐狸快速地"      "跳过懒狗"
```

`sub`只进行第一个可用替换，我们需要`gsub`来进行全局替换：

```
gsub("quick", "fast", test_sentences)
#[1] "快速的红色狐狸快速地"      "跳过懒狗"
```

更多示例请参见通过替换修改字符串。

第30.4节：在大数据集中查找匹配项

在大数据集的情况下，调用`grep("fox", test_sentences)`表现不佳。大数据集例如抓取的网站或数百万条推文等。

第一个加速方法是使用`perl=TRUE`选项。更快的是选项`fixed=TRUE`。一个完整的示例如下：

```
#attr("useBytes")
#[1] TRUE
```

All this is really great, but this only give use positions of match and that's not so easy to get what is matched, and here comes `regmatches` it's sole purpose is to extract the string matched from `regexpr`, but it has a different syntax.

Let's save our matches in a variable and then extract them from original string:

```
matches <- gregexpr("[^aeiou][aeiou]+", teststring)
regmatches(teststring, matches)
#[[1]]
#[1] "ju" "wa" "yo"
```

This may sound strange to not have a shortcut, but this allow extraction from another string by the matches of our first one (think comparing two long vector where you know there's a common pattern for the first but not for the second, this allow an easy comparison):

```
teststring2 <- "this is another string to match against"
regmatches(teststring2, matches)
#[[1]]
#[1] "is" " i" "ri"
```

Attention note: by default the pattern is not Perl Compatible Regular Expression, some things like lookarounds are not supported, but each function presented here allow for `perl=TRUE` argument to enable them.

Section 30.3: Making substitutions

```
# example data
test_sentences <- c("The quick brown fox quickly", "jumps over the lazy dog")
```

Let's make the brown fox red:

```
sub("brown", "red", test_sentences)
#[1] "The quick red fox quickly"      "jumps over the lazy dog"
```

Now, let's make the “`fast`” fox act “`fastly`”. This won't do it:

```
sub("quick", "fast", test_sentences)
#[1] "The fast red fox quickly"      "jumps over the lazy dog"
```

`sub` only makes the first available replacement, we need `gsub` for global replacement:

```
gsub("quick", "fast", test_sentences)
#[1] "The fast red fox fastly"      "jumps over the lazy dog"
```

See Modifying strings by substitution for more examples.

Section 30.4: Find matches in big data sets

In case of big data sets, the call of `grep("fox", test_sentences)` does not perform well. Big data sets are e.g. crawled websites or million of Tweets, etc.

The first acceleration is the usage of the `perl = TRUE` option. Even faster is the option `fixed = TRUE`. A complete example would be:

```
# 示例数据
test_sentences<-c("快速的棕色狐狸","跳过懒狗")

grep("fox", test_sentences, perl=TRUE)
#[1] TRUE FALSE
```

在文本挖掘中，通常会使用语料库。语料库不能直接用于`grep`。因此，请考虑这个函数：

```
searchCorpus<-function(corpus, pattern) {
  return(tm_index(corpus, FUN=function(x) {
    grep(pattern, x, ignore.case=TRUE, perl=TRUE)
  }))
}
```

```
# example data
test_sentences <- c("The quick brown fox", "jumps over the lazy dog")

grep("fox", test_sentences, perl = TRUE)
#[1] TRUE FALSE
```

In case of text mining, often a corpus gets used. A corpus cannot be used directly with `grep`. Therefore, consider this function:

```
searchCorpus <- function(corpus, pattern) {
  return(tm_index(corpus, FUN = function(x) {
    grep(pattern, x, ignore.case = TRUE, perl = TRUE)
  }))
}
```

第31章：游程编码

第31.1节：使用`rle`的游程编码

游程编码记录向量中连续元素的长度。考虑以下示例向量：

```
dat <- c(1, 2, 2, 2, 3, 1, 4, 4, 1, 1)
```

rle函数提取每个连续值及其长度：

```
r <- rle(dat)  
r  
# 运行长度编码  
#   lengths: int [1:6] 1 3 1 1 2 2  
#   values : num [1:6] 1 2 3 1 4 1
```

每个连续值的数值保存在 r\$values 中：

```
r$values  
# [1] 1 2 3 1 4 1
```

这表示我们首先看到一段1的连续值，然后是一段2的连续值，再是一段3的连续值，接着是一段1的连续值，依此类推。

每段连续值的长度保存在 r\$lengths 中：

```
r$lengths  
# [1] 1 3 1 1 2 2
```

我们看到最初的1的连续段长度为1，随后2的连续段长度为3，依此类推。

第31.2节：在基础R中识别并按连续段分组

有人可能想根据变量的连续段对数据进行分组并执行某种分析。考虑以下简单的数据集：

```
(dat <- data.frame(x = c(1, 1, 2, 2, 2, 1), y = 1:6))  
#   x y  
# 1 1 1  
# 2 1 2  
# 3 2 3  
# 4 2 4  
# 5 2 5  
# 6 1 6
```

变量 x 有三个连续段：一个值为1的长度为2的连续段，一个值为2的长度为3的连续段，以及一个值为1的长度为1的连续段。我们可能想计算变量 y 在变量 x 每个连续段中的平均值（这些平均值分别是1.5、4和6）。

在基础R中，我们首先使用 rle 计算 x 变量的连续段编码：

```
(r <- rle(dat$x))  
# 连续段编码  
#   lengths: int [1:3] 2 3 1  
#   values : num [1:3] 1 2 1
```

Chapter 31: Run-length encoding

Section 31.1: Run-length Encoding with `rle`

Run-length encoding captures the lengths of runs of consecutive elements in a vector. Consider an example vector:

```
dat <- c(1, 2, 2, 2, 3, 1, 4, 4, 1, 1)
```

The **rle** function extracts each run and its length:

```
r <- rle(dat)  
r  
# Run Length Encoding  
#   lengths: int [1:6] 1 3 1 1 2 2  
#   values : num [1:6] 1 2 3 1 4 1
```

The values for each run are captured in r\$values:

```
r$values  
# [1] 1 2 3 1 4 1
```

This captures that we first saw a run of 1's, then a run of 2's, then a run of 3's, then a run of 1's, and so on.

The lengths of each run are captured in r\$lengths:

```
r$lengths  
# [1] 1 3 1 1 2 2
```

We see that the initial run of 1's was of length 1, the run of 2's that followed was of length 3, and so on.

Section 31.2: Identifying and grouping by runs in base R

One might want to group their data by the runs of a variable and perform some sort of analysis. Consider the following simple dataset:

```
(dat <- data.frame(x = c(1, 1, 2, 2, 2, 1), y = 1:6))  
#   x y  
# 1 1 1  
# 2 1 2  
# 3 2 3  
# 4 2 4  
# 5 2 5  
# 6 1 6
```

The variable x has three runs: a run of length 2 with value 1, a run of length 3 with value 2, and a run of length 1 with value 1. We might want to compute the mean value of variable y in each of the runs of variable x (these mean values are 1.5, 4, and 6).

In base R, we would first compute the run-length encoding of the x variable using **rle**:

```
(r <- rle(dat$x))  
# Run Length Encoding  
#   lengths: int [1:3] 2 3 1  
#   values : num [1:3] 1 2 1
```

下一步是计算数据集中每一行所属的连续段编号。我们知道连续段的总数是 `length(r$lengths)`, 且每个连续段的长度是 `r$lengths`, 因此我们可以用 `rep` 计算每个连续段的编号：

```
(run.id <- rep(seq_along(r$lengths), r$lengths))  
# [1] 1 1 2 2 2 3
```

现在我们可以使用 `apply` 通过按运行ID分组来计算每次运行的平均 `y` 值：

```
data.frame(x=r$values, meanY=tapply(dat$y, run.id, mean))  
#   x meanY  
# 1 1 1.5  
# 2 2 4.0  
# 3 1 6.0
```

第31.3节：运行长度编码用于压缩和解压向量

具有大量相同值连续出现的长向量，可以通过存储其运行长度编码（每个连续段的值及该值重复的次数）来显著压缩。举例来说，考虑一个长度为1000万的向量，其中包含大量的1和少量的0：

```
set.seed(144)  
dat <- sample(rep(0:1, c(1, 1e5)), 1e7, replace=TRUE)  
table(dat)  
# 0 1  
# 103 9999897
```

存储1000万个条目将需要大量空间，但我们可以创建一个包含该向量运行长度编码的数据框：

```
rle.df <- with(rle(dat), data.frame(values, lengths))  
dim(rle.df)  
# [1] 207 2  
head(rle.df)  
#  值 长度  
# 1 1 52818  
# 2 0 1  
# 3 1 219329  
# 4 0 1  
# 5 1 318306  
# 6 0 1
```

从游程编码中，我们看到向量的前52,818个值是1，接着是一个0，然后是219,329个连续的1，接着是一个0，依此类推。游程编码只有207条目，只需存储414个值，而不是1000万个值。由于 `rle.df` 是一个数据框，可以使用标准函数如 `write.csv` 进行存储。

解压缩游程编码向量可以通过两种方式完成。第一种方法是简单调用 `rep`，将游程编码的 `values` 元素作为第一个参数，游程编码的 `lengths` 元素作为第二个参数传入：

```
decompressed <- rep(rle.df$values, rle.df$lengths)
```

我们可以确认解压后的数据与原始数据完全相同：

The next step is to compute the run number of each row of our dataset. We know that the total number of runs is `length(r$lengths)`, and the length of each run is `r$lengths`, so we can compute the run number of each of our runs with `rep`:

```
(run.id <- rep(seq_along(r$lengths), r$lengths))  
# [1] 1 1 2 2 2 3
```

Now we can use `tapply` to compute the mean `y` value for each run by grouping on the run id:

```
data.frame(x=r$values, meanY=tapply(dat$y, run.id, mean))  
#   x meanY  
# 1 1 1.5  
# 2 2 4.0  
# 3 1 6.0
```

Section 31.3: Run-length encoding to compress and decompress vectors

Long vectors with long runs of the same value can be significantly compressed by storing them in their run-length encoding (the value of each run and the number of times that value is repeated). As an example, consider a vector of length 10 million with a huge number of 1's and only a small number of 0's:

```
set.seed(144)  
dat <- sample(rep(0:1, c(1, 1e5)), 1e7, replace=TRUE)  
table(dat)  
# 0 1  
# 103 9999897
```

Storing 10 million entries will require significant space, but we can instead create a data frame with the run-length encoding of this vector:

```
rle.df <- with(rle(dat), data.frame(values, lengths))  
dim(rle.df)  
# [1] 207 2  
head(rle.df)  
#  values lengths  
# 1 1 52818  
# 2 0 1  
# 3 1 219329  
# 4 0 1  
# 5 1 318306  
# 6 0 1
```

From the run-length encoding, we see that the first 52,818 values in the vector are 1's, followed by a single 0, followed by 219,329 consecutive 1's, followed by a 0, and so on. The run-length encoding only has 207 entries, requiring us to store only 414 values instead of 10 million values. As `rle.df` is a data frame, it can be stored using standard functions like `write.csv`.

Decompressing a vector in run-length encoding can be accomplished in two ways. The first method is to simply call `rep`, passing the `values` element of the run-length encoding as the first argument and the `lengths` element of the run-length encoding as the second argument:

```
decompressed <- rep(rle.df$values, rle.df$lengths)
```

We can confirm that our decompressed data is identical to our original data:

```
identical(decompressed, dat)
# [1] TRUE
```

第二种方法是对 rle对象使用R内置的inverse.rle函数，例如：

```
rle.obj <- rle(dat)          # 在这里创建一个rle对象
class(rle.obj)
# [1] "rle"

dat.inv <- inverse.rle(rle.obj)    # 对rle对象应用inverse.rle
```

我们可以再次确认这确实产生了原始的dat：

```
identical(dat.inv, dat)
# [1] TRUE
```

第31.4节：在data.table中识别并按连续段分组

data.table包提供了一种方便的方法来按数据中的连续段进行分组。考虑以下示例数据：

```
library(data.table)
(DT <- data.table(x = c(1, 1, 2, 2, 2, 1), y = 1:6))
#   x y
# 1: 1 1
# 2: 1 2
# 3: 2 3
# 4: 2 4
# 5: 2 5
# 6: 1 6
```

变量x有三个连续段：一个值为1的长度为2的连续段，一个值为2的长度为3的连续段，以及一个值为1的长度为1的连续段。我们可能想计算变量y在变量x的每个连续段中的平均值（这些平均值分别是1.5、4和6）。

data.table的rleid函数提供了一个标识符，指示向量中每个元素所属的连续段ID：

```
rleid(DT$x)
# [1] 1 1 2 2 2 3
```

然后可以很容易地根据该运行ID进行分组并汇总y数据：

```
DT[,mean(y),by=(x, rleid(x))]

# 1: 1     1 1.5
# 2: 2     2 4.0
# 3: 1     3 6.0
```

```
identical(decompressed, dat)
# [1] TRUE
```

The second method is to use R's built-in **inverse.rle** function on the **rle** object, for instance:

```
rle.obj <- rle(dat)          # create a rle object here
class(rle.obj)
# [1] "rle"

dat.inv <- inverse.rle(rle.obj)    # apply the inverse.rle on the rle object
```

We can confirm again that this produces exactly the original dat:

```
identical(dat.inv, dat)
# [1] TRUE
```

Section 31.4: Identifying and grouping by runs in data.table

The data.table package provides a convenient way to group by runs in data. Consider the following example data:

```
library(data.table)
(DT <- data.table(x = c(1, 1, 2, 2, 2, 1), y = 1:6))
#   x y
# 1: 1 1
# 2: 1 2
# 3: 2 3
# 4: 2 4
# 5: 2 5
# 6: 1 6
```

The variable x has three runs: a run of length 2 with value 1, a run of length 3 with value 2, and a run of length 1 with value 1. We might want to compute the mean value of variable y in each of the runs of variable x (these mean values are 1.5, 4, and 6).

The data.table rleid function provides an id indicating the run id of each element of a vector:

```
rleid(DT$x)
# [1] 1 1 2 2 2 3
```

One can then easily group on this run ID and summarize the y data:

```
DT[,mean(y),by=(x, rleid(x))]
#   x rleid V1
# 1: 1     1 1.5
# 2: 2     2 4.0
# 3: 1     3 6.0
```

第32章：加速难以向量化的代码

第32.1节：使用Rcpp加速难以向量化的for循环

考虑以下难以向量化的for循环，它创建一个长度为len的向量，其中第一个元素为指定的first，每个元素 x_i 等于 $\cos(x_{i-1} + 1)$ ：

```
repeatedCosPlusOne <- function(first, len) {  
  x <- numeric(len)  
  x[1] <- first  
  for (i in 2:len) {  
    x[i] <- cos(x[i-1] + 1)  
  }  
  return(x)  
}
```

这段代码包含一个带有快速运算的for循环 ($\cos(x[i-1]+1)$)，通常可以通过向量化来提升性能。然而，使用基础R对该操作进行向量化并非易事，因为R没有“x+1的累积余弦”函数。

加速此函数的一种可能方法是使用Rcpp包用C++实现：

```
library(Rcpp)  
cppFunction("NumericVector repeatedCosPlusOneRcpp(double first, int len) {  
  NumericVector x(len);  
  x[0] = first;  
  for (int i=1; i < len; ++i) {  
    x[i] = cos(x[i-1]+1);  
  }  
  return x;  
}")
```

这通常能为大型计算带来显著的加速，同时产生完全相同的结果：

```
all.equal(repeatedCosPlusOne(1, 1e6), repeatedCosPlusOneRcpp(1, 1e6))  
# [1] TRUE  
system.time(repeatedCosPlusOne(1, 1e6))  
#   user   system elapsed  
# 1.274  0.015  1.310  
system.time(repeatedCosPlusOneRcpp(1, 1e6))  
#   user   system elapsed  
# 0.028  0.001  0.030
```

在这种情况下，Rcpp 代码生成一个长度为 100 万的向量只需 0.03 秒，而使用基础 R 方法则需 1.31 秒。

第 32.2 节：通过字节编译加速难以向量化的 for 循环

参考本文档条目中的 Rcpp 示例，考虑以下难以向量化的函数，该函数创建一个长度为len的向量，其中第一个元素为指定的first，每个元素 x_i 等于 $\cos(x_{i-1} + 1)$ ：

```
repeatedCosPlusOne <- function(first, len) {
```

Chapter 32: Speeding up tough-to-vectorize code

Section 32.1: Speeding tough-to-vectorize for loops with Rcpp

Consider the following tough-to-vectorize for loop, which creates a vector of length len where the first element is specified (first) and each element x_i is equal to $\cos(x_{i-1} + 1)$:

```
repeatedCosPlusOne <- function(first, len) {  
  x <- numeric(len)  
  x[1] <- first  
  for (i in 2:len) {  
    x[i] <- cos(x[i-1] + 1)  
  }  
  return(x)  
}
```

This code involves a for loop with a fast operation ($\cos(x[i-1]+1)$), which often benefit from vectorization. However, it is not trivial to vectorize this operation with base R, since R does not have a "cumulative cosine of x+1" function.

One possible approach to speeding this function would be to implement it in C++, using the Rcpp package:

```
library(Rcpp)  
cppFunction("NumericVector repeatedCosPlusOneRcpp(double first, int len) {  
  NumericVector x(len);  
  x[0] = first;  
  for (int i=1; i < len; ++i) {  
    x[i] = cos(x[i-1]+1);  
  }  
  return x;  
}")
```

This often provides significant speedups for large computations while yielding the exact same results:

```
all.equal(repeatedCosPlusOne(1, 1e6), repeatedCosPlusOneRcpp(1, 1e6))  
# [1] TRUE  
system.time(repeatedCosPlusOne(1, 1e6))  
#   user   system elapsed  
# 1.274  0.015  1.310  
system.time(repeatedCosPlusOneRcpp(1, 1e6))  
#   user   system elapsed  
# 0.028  0.001  0.030
```

In this case, the Rcpp code generates a vector of length 1 million in 0.03 seconds instead of 1.31 seconds with the base R approach.

Section 32.2: Speeding tough-to-vectorize for loops by byte compiling

Following the Rcpp example in this documentation entry, consider the following tough-to-vectorize function, which creates a vector of length len where the first element is specified (first) and each element x_i is equal to $\cos(x_{i-1} + 1)$:

```
repeatedCosPlusOne <- function(first, len) {
```

```

x <- 数值型(长度)
x[1] <- 第一个
for (i in 2:长度) {
  x[i] <- cos(x[i-1] + 1)
}
return(x)
}

```

一种无需重写任何代码即可加速此类函数的简单方法是使用R的compile包对代码进行字节编译：

```

library(compiler)
repeatedCosPlusOneCompiled <- cmpfun(repeatedCosPlusOne)

```

生成的函数通常会显著更快，同时仍返回相同的结果：

```

all.equal(repeatedCosPlusOne(1, 1e6), repeatedCosPlusOneCompiled(1, 1e6))
# [1] TRUE
system.time(repeatedCosPlusOne(1, 1e6))
#   user system elapsed
# 1.175  0.014  1.201
system.time(repeatedCosPlusOneCompiled(1, 1e6))
#   user system elapsed
# 0.339  0.002  0.341

```

在此例中，字节编译将对长度为100万的向量进行难以向量化的操作的运行时间从1.20秒加速到了0.34秒。

备注

repeatedCosPlusOne的本质，作为单一函数的累积应用，可以用Reduce更清晰地表达：

```

iterFunc <- function(init, n, func) {
  funcs <- replicate(n, func)
  Reduce(function(.., f) f(..), funcs, init = init, accumulate = TRUE)
}
repeatedCosPlusOne_vec <- function(first, len) {
  iterFunc(first, len - 1, function(.) cos(. + 1))
}

```

repeatedCosPlusOne_vec可以被视为 repeatedCosPlusOne的“向量化”。然而，预计它会比原函数慢约2倍：

```

library(microbenchmark)
microbenchmark(
  repeatedCosPlusOne(1, 1e4),
  repeatedCosPlusOne_vec(1, 1e4)
)
#> 单位：毫秒
#> expr      min       lq      mean     median       uq      max neval
cld
#>   repeatedCosPlusOne(1, 10000) 8.349261 9.216724 10.22715 10.23095 11.10817 14.33763 100
a
#>   repeatedCosPlusOne_vec(1, 10000) 14.406291 16.236153 17.55571 17.22295 18.59085 24.37059 100
b

```

```

x <- numeric(len)
x[1] <- first
for (i in 2:len) {
  x[i] <- cos(x[i-1] + 1)
}
return(x)
}

```

One simple approach to speeding up such a function without rewriting a single line of code is byte compiling the code using the R compile package:

```

library(compiler)
repeatedCosPlusOneCompiled <- cmpfun(repeatedCosPlusOne)

```

The resulting function will often be significantly faster while still returning the same results:

```

all.equal(repeatedCosPlusOne(1, 1e6), repeatedCosPlusOneCompiled(1, 1e6))
# [1] TRUE
system.time(repeatedCosPlusOne(1, 1e6))
#   user system elapsed
# 1.175  0.014  1.201
system.time(repeatedCosPlusOneCompiled(1, 1e6))
#   user system elapsed
# 0.339  0.002  0.341

```

In this case, byte compiling sped up the tough-to-vectorize operation on a vector of length 1 million from 1.20 seconds to 0.34 seconds.

Remark

The essence of repeatedCosPlusOne, as the cumulative application of a single function, can be expressed more transparently with [Reduce](#):

```

iterFunc <- function(init, n, func) {
  funcs <- replicate(n, func)
  Reduce(function(.., f) f(..), funcs, init = init, accumulate = TRUE)
}
repeatedCosPlusOne_vec <- function(first, len) {
  iterFunc(first, len - 1, function(.) cos(. + 1))
}

```

repeatedCosPlusOne_vec may be regarded as a "vectorization" of repeatedCosPlusOne. However, it can be expected to be *slower* by a factor of 2:

```

library(microbenchmark)
microbenchmark(
  repeatedCosPlusOne(1, 1e4),
  repeatedCosPlusOne_vec(1, 1e4)
)
#> Unit: milliseconds
#> expr      min       lq      mean     median       uq      max neval
cld
#>   repeatedCosPlusOne(1, 10000) 8.349261 9.216724 10.22715 10.23095 11.10817 14.33763 100
a
#>   repeatedCosPlusOne_vec(1, 10000) 14.406291 16.236153 17.55571 17.22295 18.59085 24.37059 100
b

```

第33章：地理地图简介

另见地理数据的输入/输出

第33.1节：使用maps包中的map()进行基本制图

maps包中的函数map()为使用R创建地图提供了一个简单的起点。

可以按如下方式绘制一个基本的世界地图：

```
require(maps)  
map()
```



轮廓的颜色可以通过设置颜色参数col为颜色的字符名称或十六进制值来更改：

```
require(maps)  
map(col = "cornflowerblue")
```

Chapter 33: Introduction to Geographical Maps

See also I/O for geographic data

Section 33.1: Basic map-making with map() from the package maps

The function `map()` from the package `maps` provides a simple starting point for creating maps with R.

A basic world map can be drawn as follows:

```
require(maps)  
map()
```



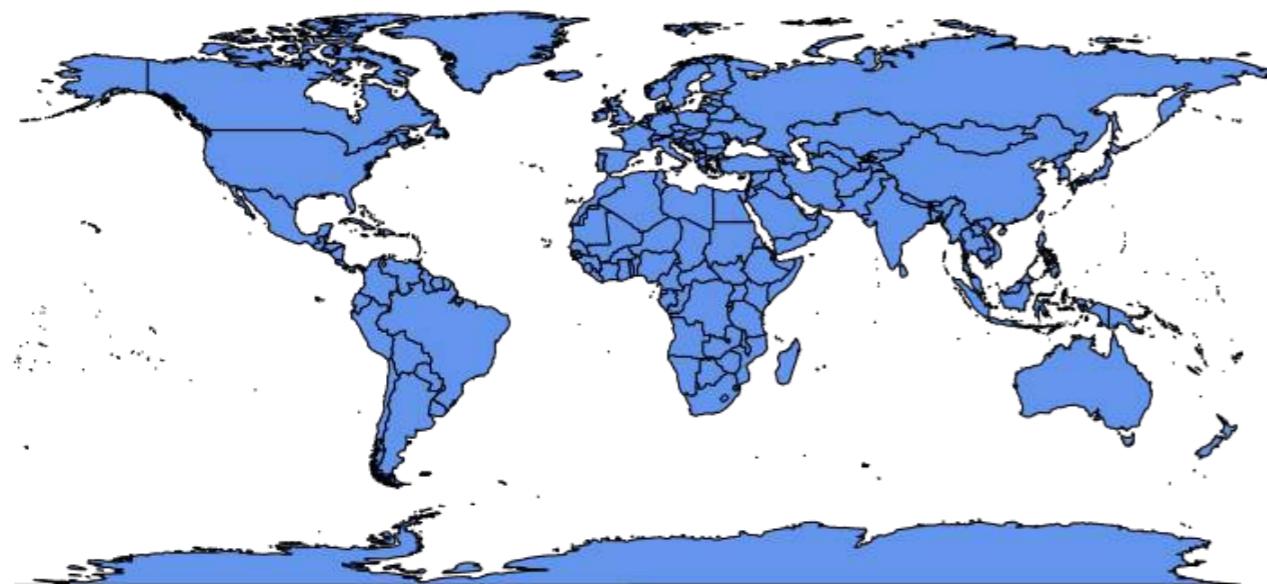
The color of the outline can be changed by setting the color parameter, `col`, to either the character name or hex value of a color:

```
require(maps)  
map(col = "cornflowerblue")
```



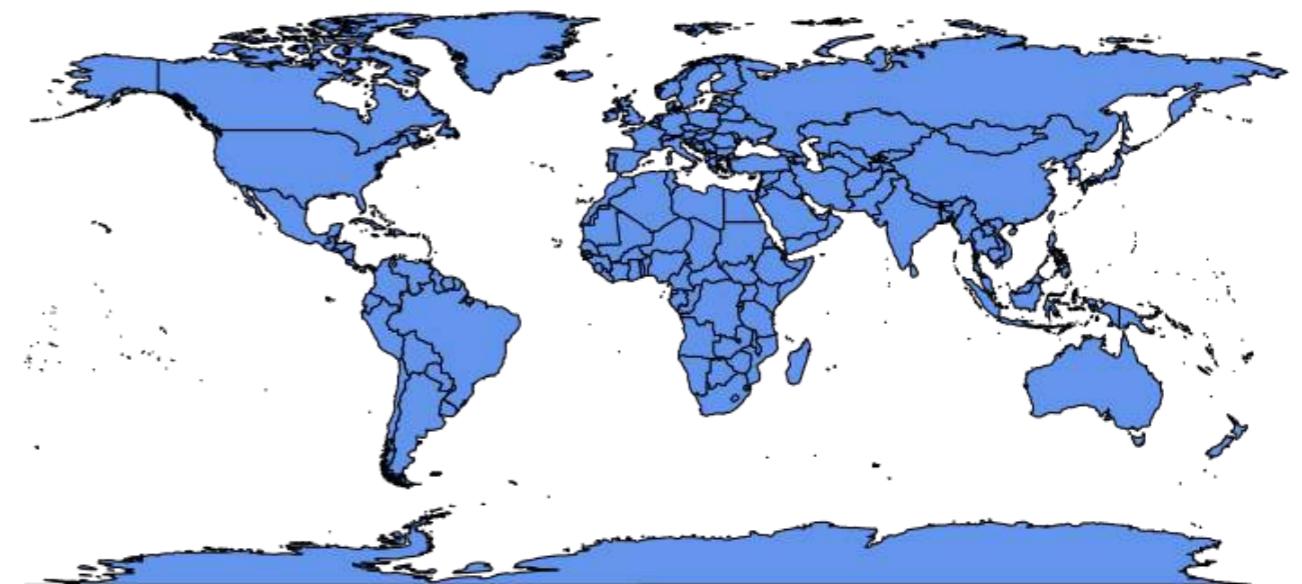
要用col中的颜色填充陆地区域，我们可以设置fill=TRUE：

```
require(maps)  
map(fill = TRUE, col = c("cornflowerblue"))
```



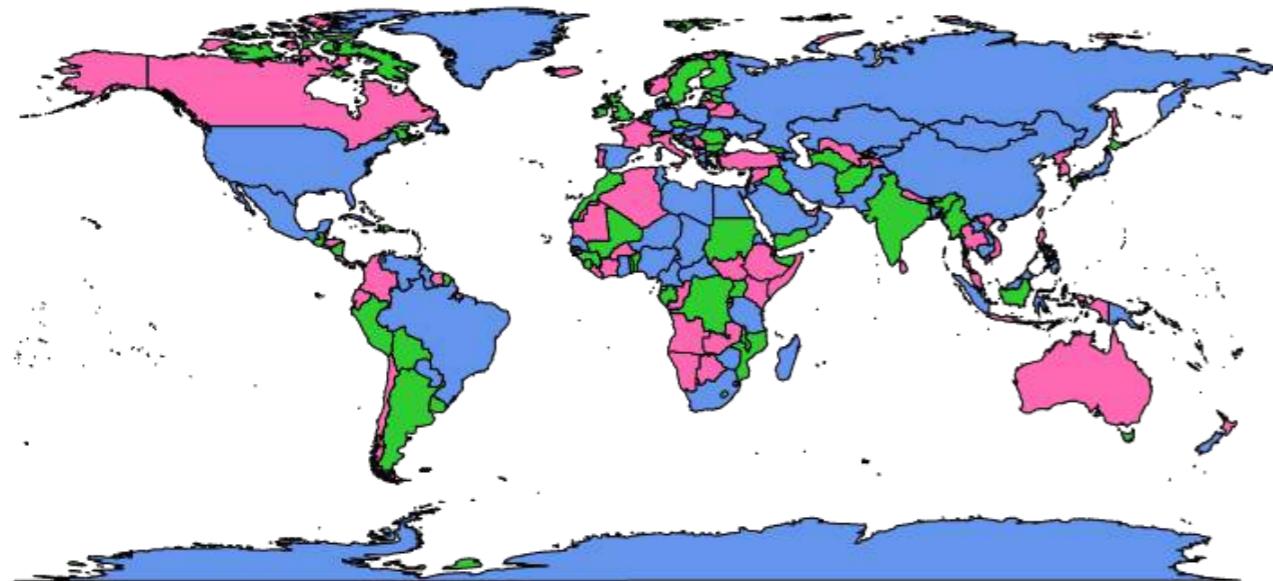
To fill land masses with the color in **col** we can set **fill = TRUE**:

```
require(maps)  
map(fill = TRUE, col = c("cornflowerblue"))
```



当fill = TRUE也被设置时，col可以提供任意长度的向量：

```
require(maps)
map(fill = TRUE, col = c("cornflowerblue", "limegreen", "hotpink"))
```



在上面的例子中，col中的颜色被任意分配给地图中代表区域的多边形，如果颜色数量少于多边形数量，则颜色会被循环使用。

我们也可以使用颜色编码来表示统计变量，图例中可以选择性地描述该变量。这样创建的地图称为“分级色彩图”（choropleth）。

下面的分级色彩图示例将map()的第一个参数 database设置为"county"，将"state"用于颜色编码失业率，使用内置数据集unemp和county.fips的数据，同时用白色叠加州界线：

```
require(maps)
if(require(mapproj)) { # mapproj 用于 projection="polyconic"
  # 按2009年失业率为美国县级地图着色
  # 使用FIPS县代码匹配县与地图
  # 基于对“分级色彩图挑战”的解决方案
  # 代码改进由Hack-R (hack-r.github.io)完成

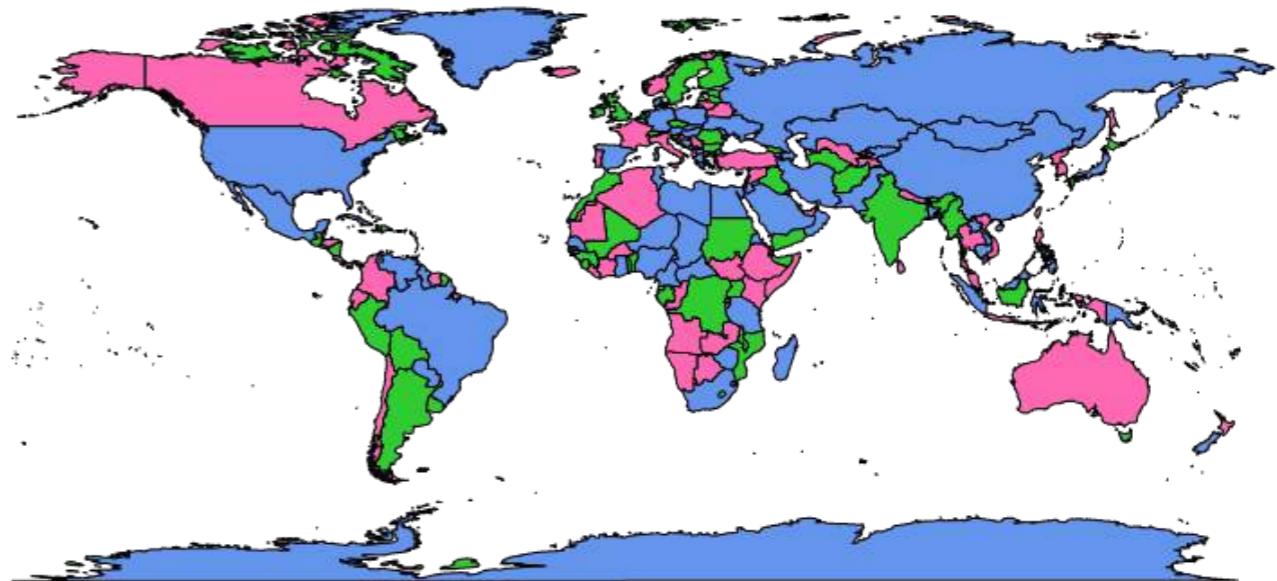
  # 加载数据
  # unemp 包含了一些不在“美国本土48州”县级地图上的县的数据，# 例如阿拉斯加、夏威夷、波多黎各以及弗吉尼亚州的一些小城市

  data(unemp)
  data(county.fips)

  # 定义颜色区间
  colors = c("paleturquoise", "skyblue", "cornflowerblue", "blueviolet", "hotpink", "darkgrey")
  unemp$colorBuckets <- as.numeric(cut(unemp$unemp, c(0, 2, 4, 6, 8, 10, 100)))
```

A vector of any length may be supplied to col when fill = TRUE is also set:

```
require(maps)
map(fill = TRUE, col = c("cornflowerblue", "limegreen", "hotpink"))
```



In the example above colors from col are assigned arbitrarily to polygons in the map representing regions and colors are recycled if there are fewer colors than polygons.

We can also use color coding to represent a statistical variable, which may optionally be described in a legend. A map created as such is known as a "choropleth".

The following choropleth example sets the first argument of map(), which is database to "county" and "state" to color code unemployment using data from the built-in datasets unemp and county.fips while overlaying state lines in white:

```
require(maps)
if(require(mapproj)) { # mapproj is used for projection="polyconic"
  # color US county map by 2009 unemployment rate
  # match counties to map using FIPS county codes
  # Based on J's solution to the "Choropleth Challenge"
  # Code improvements by Hack-R (hack-r.github.io)

  # load data
  # unemp includes data for some counties not on the "lower 48 states" county
  # map, such as those in Alaska, Hawaii, Puerto Rico, and some tiny Virginia
  # cities
  data(unemp)
  data(county.fips)

  # define color buckets
  colors = c("paleturquoise", "skyblue", "cornflowerblue", "blueviolet", "hotpink", "darkgrey")
  unemp$colorBuckets <- as.numeric(cut(unemp$unemp, c(0, 2, 4, 6, 8, 10, 100)))
```

```

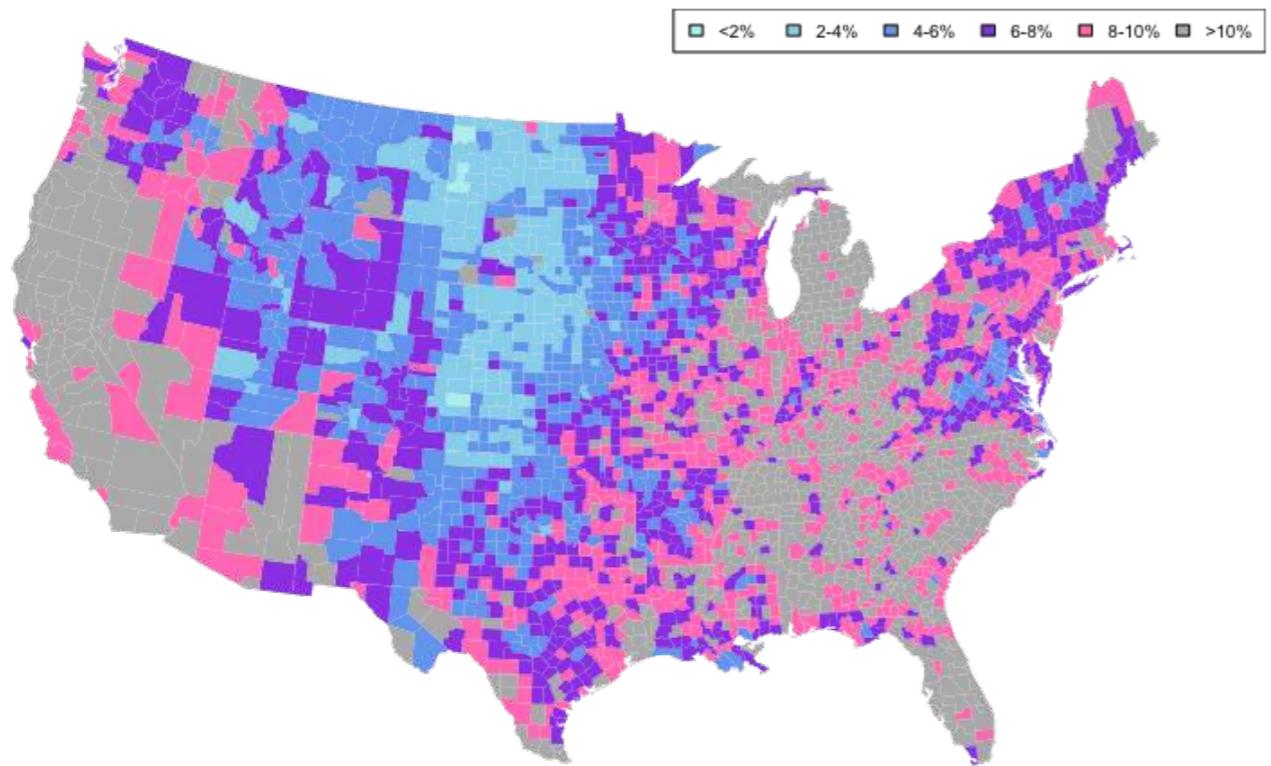
leg.txt <- c("<2%", "2-4%", "4-6%", "6-8%", "8-10%", ">10%")

# 通过(部分)匹配州、县名称对数据与地图定义进行对齐
# 某些县包含多个多边形
cnty.fips <- county.fips$fips[match(map("county", plot=FALSE)$names,
                                         county.fips$polyname)]
colorsmatched <- unemp$colorBuckets[match(cnty.fips, unemp$fips)]

# 绘制地图
par(mar=c(1, 1, 2, 1) + 0.1)
map("county", col = colors[colorsmatched], fill = TRUE, resolution = 0,
    lty = 0, projection = "polyconic")
map("state", col = "white", fill = FALSE, add = TRUE, lty = 1, lwd = 0.1,
     projection="polyconic")
title("2009年各县失业率")
legend("topright", leg.txt, horiz = TRUE, fill = colors, cex=0.6)
}

```

unemployment by county, 2009



```

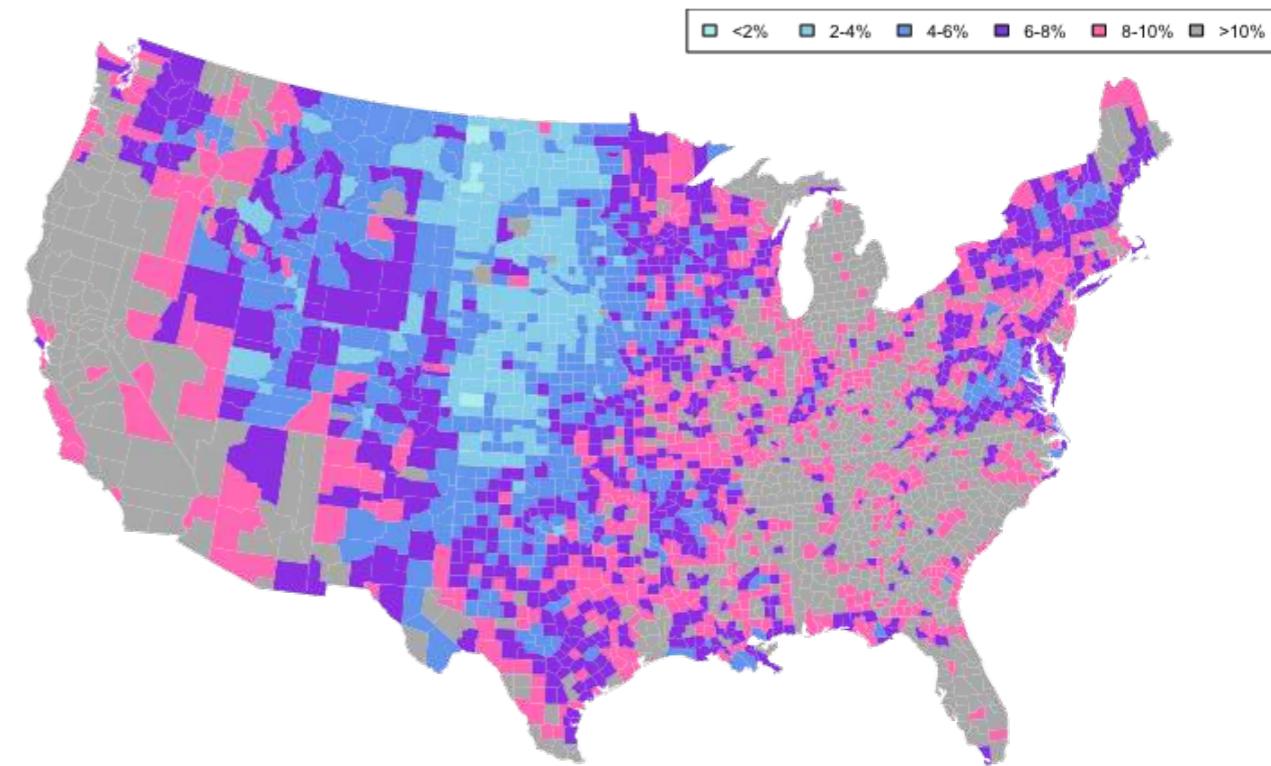
leg.txt <- c("<2%", "2-4%", "4-6%", "6-8%", "8-10%", ">10%")

# align data with map definitions by (partial) matching state, county
# names, which include multiple polygons for some counties
cnty.fips <- county.fips$fips[match(map("county", plot=FALSE)$names,
                                         county.fips$polyname)]
colorsmatched <- unemp$colorBuckets[match(cnty.fips, unemp$fips)]

# draw map
par(mar=c(1, 1, 2, 1) + 0.1)
map("county", col = colors[colorsmatched], fill = TRUE, resolution = 0,
    lty = 0, projection = "polyconic")
map("state", col = "white", fill = FALSE, add = TRUE, lty = 1, lwd = 0.1,
     projection="polyconic")
title("unemployment by county, 2009")
legend("topright", leg.txt, horiz = TRUE, fill = colors, cex=0.6)
}

```

unemployment by county, 2009



第33.2节：50州地图与使用Google Viz的高级分级图

一个常见的问题是如何在同一张地图上并列（组合）物理上分离的地理区域，例如描述所有50个美国州的分级图（大陆与阿拉斯加和夏威夷并列）。

利用Google地图创建一张美观的50州地图非常简单。Google的API接口包括包googleVis、ggmap和RgoogleMaps。

```

require(googleVis)

G4 <- gvisGeoChart(CityPopularity, locationvar='City', colorvar='Popularity',
                     options=list(region='US', height=350,
                               displayMode='markers',

```

Section 33.2: 50 State Maps and Advanced Choropleths with Google Viz

A common [question](#) is how to juxtapose (combine) physically separate geographical regions on the same map, such as in the case of a choropleth describing all 50 American states (The mainland with Alaska and Hawaii juxtaposed).

Creating an attractive 50 state map is simple when leveraging Google Maps. Interfaces to Google's API include the packages googleVis, ggmap, and RgoogleMaps.

```

require(googleVis)

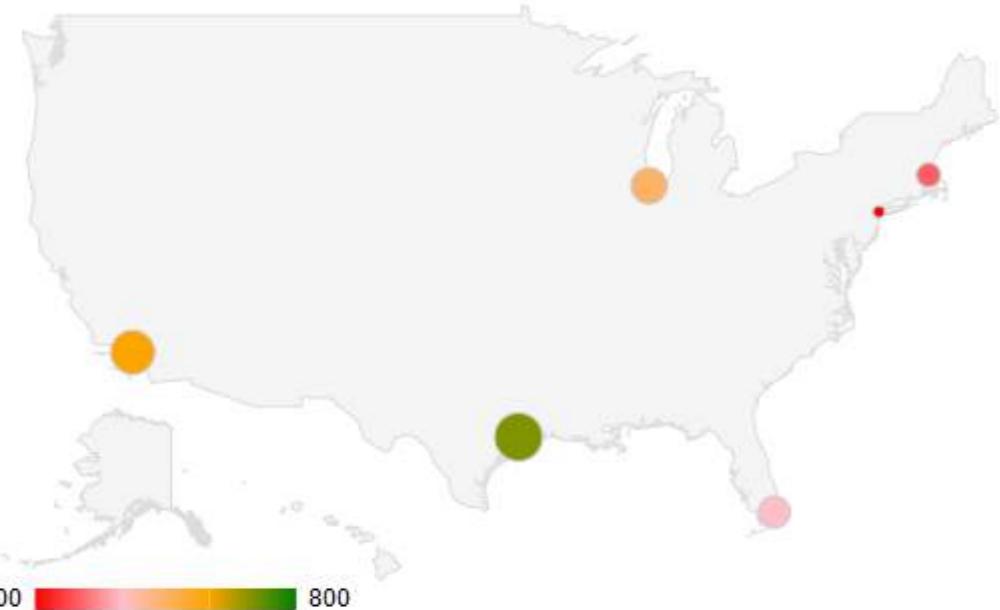
G4 <- gvisGeoChart(CityPopularity, locationvar='City', colorvar='Popularity',
                     options=list(region='US', height=350,
                               displayMode='markers',

```

```

colorAxis="{values:[200,400,600,800],
           colors:['red', 'pink', 'orange','green'])}"
)
绘制(G4)

```

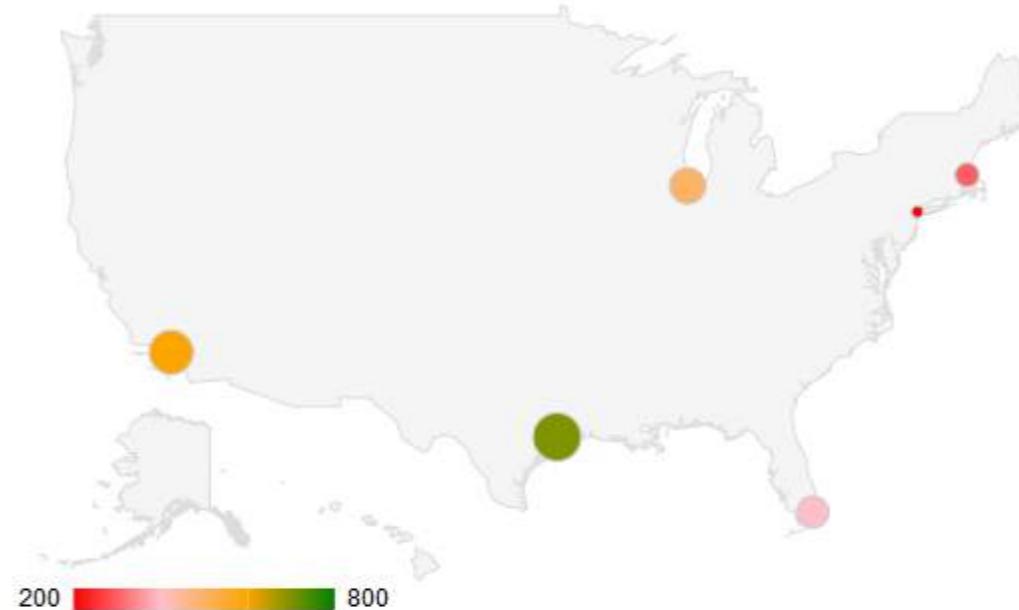


Data: CityPopularity • Chart ID: GeoChartID28504adb439a • googleVis-0.5.2
R version 3.1.0 (2014-04-10) • Google Terms of Use • Documentation and Data Policy

```

colorAxis="{values:[200,400,600,800],
           colors:['red', 'pink', 'orange','green'])}"
)
plot(G4)

```



Data: CityPopularity • Chart ID: GeoChartID28504adb439a • googleVis-0.5.2
R version 3.1.0 (2014-04-10) • Google Terms of Use • Documentation and Data Policy

函数 gvisGeoChart() 相较于旧的映射方法，如包 maps 中的 map()，创建分级着色图所需的编码量要少得多。colorvar 参数允许对统计变量进行简单着色，着色级别由 locationvar 参数指定。传递给 options 的各种选项列表可以自定义地图的细节，如大小 (height)、形状 (markers) 和颜色编码 (colorAxis 和 colors)。

第33.3节：交互式 plotly 地图

plotly 包允许创建多种交互式图表，包括地图。在 plotly 中创建地图有几种方式。可以自己提供地图数据（通过 plot_ly() 或 ggplotly()），使用 plotly 的“原生”映射功能（通过 plot_geo() 或 plot_mapbox()），甚至两者结合。自己提供地图数据的示例为：

```

library(plotly)
map_data("county") %>%
  group_by(group) %>%
  plot_ly(x = ~long, y = ~lat) %>%
  add_polygons() %>%
  layout(
    xaxis = list(title = "", showgrid = FALSE, showticklabels = FALSE),
    yaxis = list(title = "", showgrid = FALSE, showticklabels = FALSE)
  )

```

The function gvisGeoChart() requires far less coding to create a choropleth compared to older mapping methods, such as map() from the package maps. The colorvar parameter allows easy coloring of a statistical variable, at a level specified by the locationvar parameter. The various options passed to options as a list allow customization of the map's details such as size (height), shape (markers), and color coding (colorAxis and colors).

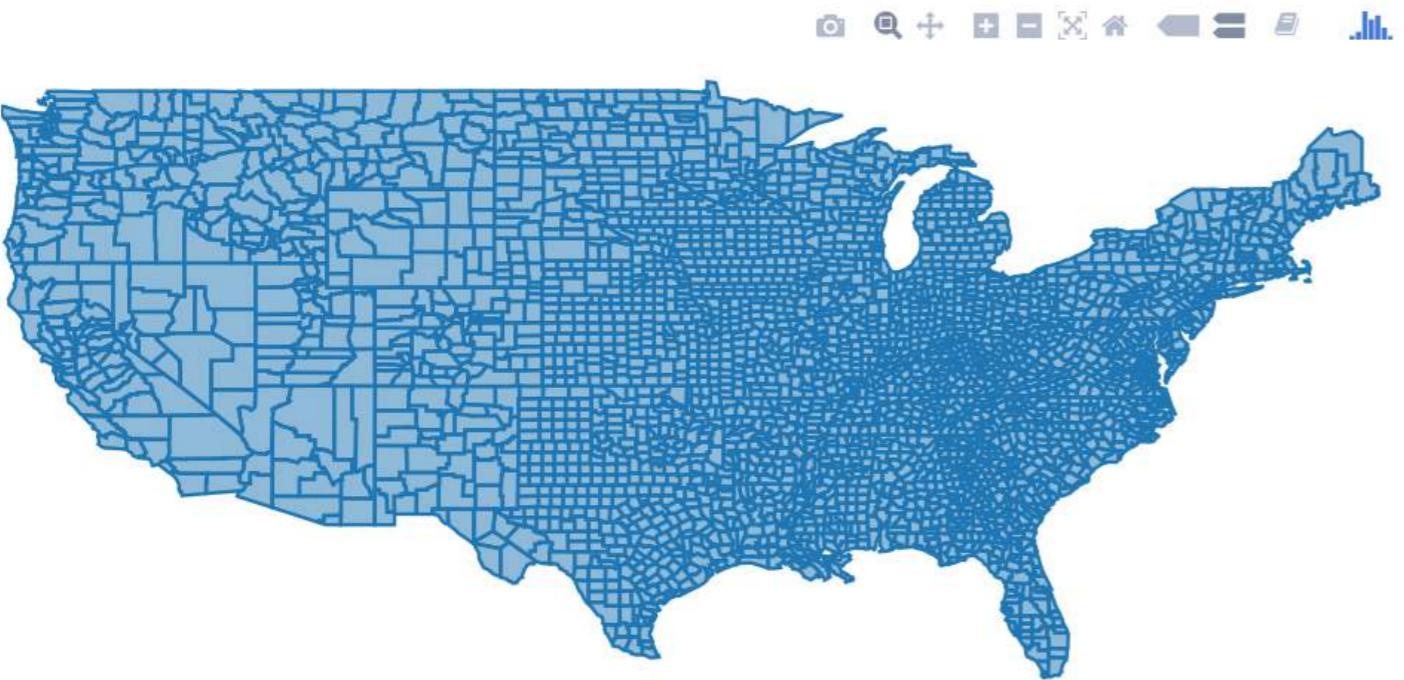
Section 33.3: Interactive plotly maps

The plotly package allows many kind of interactive plots, including maps. There are a few ways to create a map in plotly. Either supply the map data yourself (via plot_ly() or ggplotly()), use plotly's "native" mapping capabilities (via plot_geo() or plot_mapbox()), or even a combination of both. An example of supplying the map yourself would be:

```

library(plotly)
map_data("county") %>%
  group_by(group) %>%
  plot_ly(x = ~long, y = ~lat) %>%
  add_polygons() %>%
  layout(
    xaxis = list(title = "", showgrid = FALSE, showticklabels = FALSE),
    yaxis = list(title = "", showgrid = FALSE, showticklabels = FALSE)
  )

```



对于两种方法的结合，在上述示例中将 `plot_ly()` 替换为 `plot_geo()` 或 `plot_mapbox()` 即可。

更多示例请参见 [plotly book](#)。

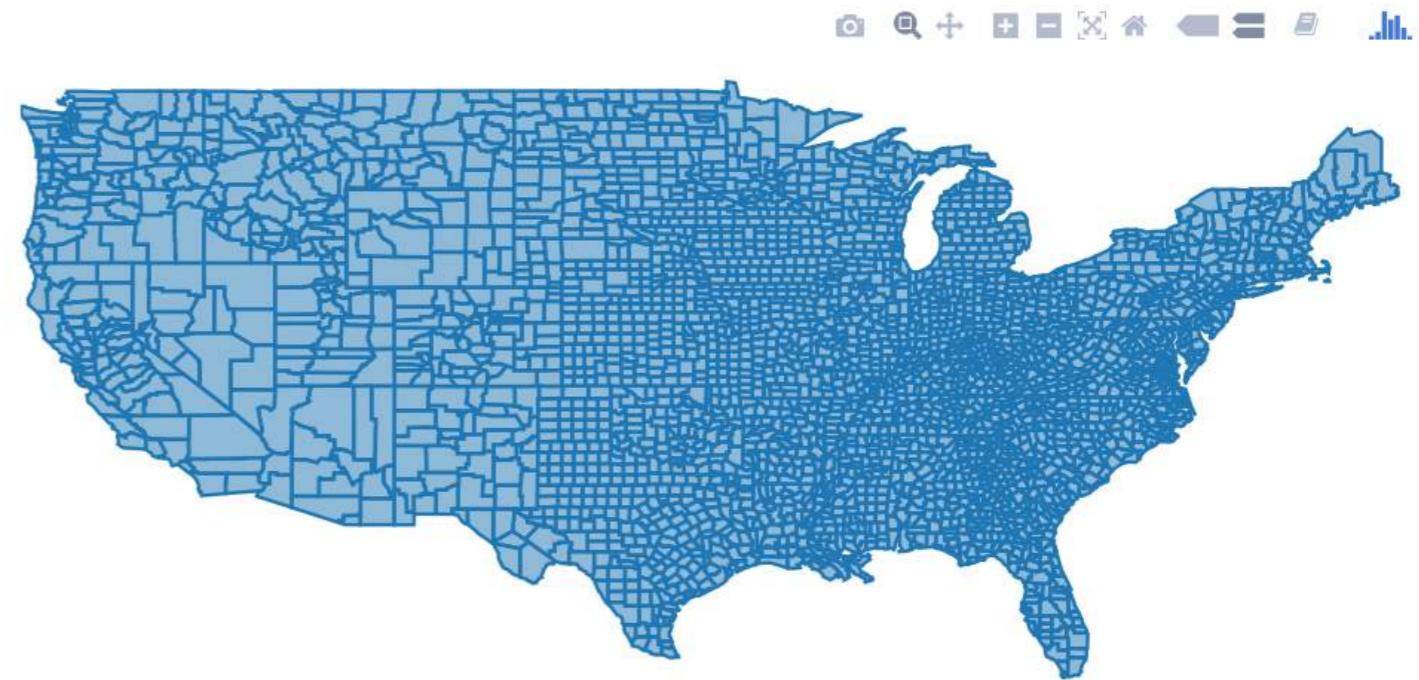
下一个示例是一个“严格本地”方法，利用`layout.geo`属性来设置地图的美学和缩放级别。它还使用来自`maps`的数据库 `world.cities` 来筛选巴西城市，并将它们绘制在“本地”地图之上。

主要变量：`poph`是包含城市及其人口的文本（鼠标悬停时显示）；`q`是基于人口分位数的有序因子。`ge`包含地图布局的信息。更多信息请参见 [package documentation](#)。

```
library(maps)
dfb <- world.cities[world.cities$country.etc=="Brazil",]
library(plotly)
dfb$poph <- paste(dfb$name, "Pop", round(dfb$pop/1e6,2), " millions")
dfb$q <- with(dfb, cut(pop, quantile(pop), include.lowest = T))
levels(dfb$q) <- paste(c("1st", "2nd", "3rd", "4th"), "Quantile")
dfb$q <- as.ordered(dfb$q)

ge <- list(
  scope = '南美洲',
  showland = TRUE,
  landcolor = toRGB("gray85"),
  subunitwidth = 1,
  countrywidth = 1,
  subunitcolor = toRGB("white"),
  countrycolor = toRGB("white")
)

plot_geo(dfb, 经度 = ~long, 纬度 = ~lat, 文本 = ~poph,
  标记 = ~list(大小 = sqrt(pop/10000) + 1, 线条 = list(宽度 = 0)),
  颜色 = ~q, 位置模式 = 'country names') %>%
  layout(geo = ge, title = '人口<br>(点击图例切换)')
```



For a combination of both approaches, swap `plot_ly()` for `plot_geo()` or `plot_mapbox()` in the above example. See the [plotly book](#) for more examples.

The next example is a “strictly native” approach that leverages the `layout.geo` attribute to set the aesthetics and zoom level of the map. It also uses the database `world.cities` from `maps` to filter the Brazilian cities and plot them on top of the “native” map.

The main variables: `poph` is a text with the city and its population (which is shown upon mouse hover); `q` is a ordered factor from the population's quantile. `ge` has information for the layout of the maps. See the [package documentation](#) for more information.

```
library(maps)
dfb <- world.cities[world.cities$country.etc=="Brazil",]
library(plotly)
dfb$poph <- paste(dfb$name, "Pop", round(dfb$pop/1e6,2), " millions")
dfb$q <- with(dfb, cut(pop, quantile(pop), include.lowest = T))
levels(dfb$q) <- paste(c("1st", "2nd", "3rd", "4th"), "Quantile")
dfb$q <- as.ordered(dfb$q)

ge <- list(
  scope = 'south america',
  showland = TRUE,
  landcolor = toRGB("gray85"),
  subunitwidth = 1,
  countrywidth = 1,
  subunitcolor = toRGB("white"),
  countrycolor = toRGB("white")
)

plot_geo(dfb, lon = ~long, lat = ~lat, text = ~poph,
  marker = ~list(size = sqrt(pop/10000) + 1, line = list(width = 0)),
  color = ~q, locationmode = 'country names') %>%
  layout(geo = ge, title = 'Populations<br>(Click legend to toggle)')
```



第33.4节：使用Leaflet制作动态HTML地图

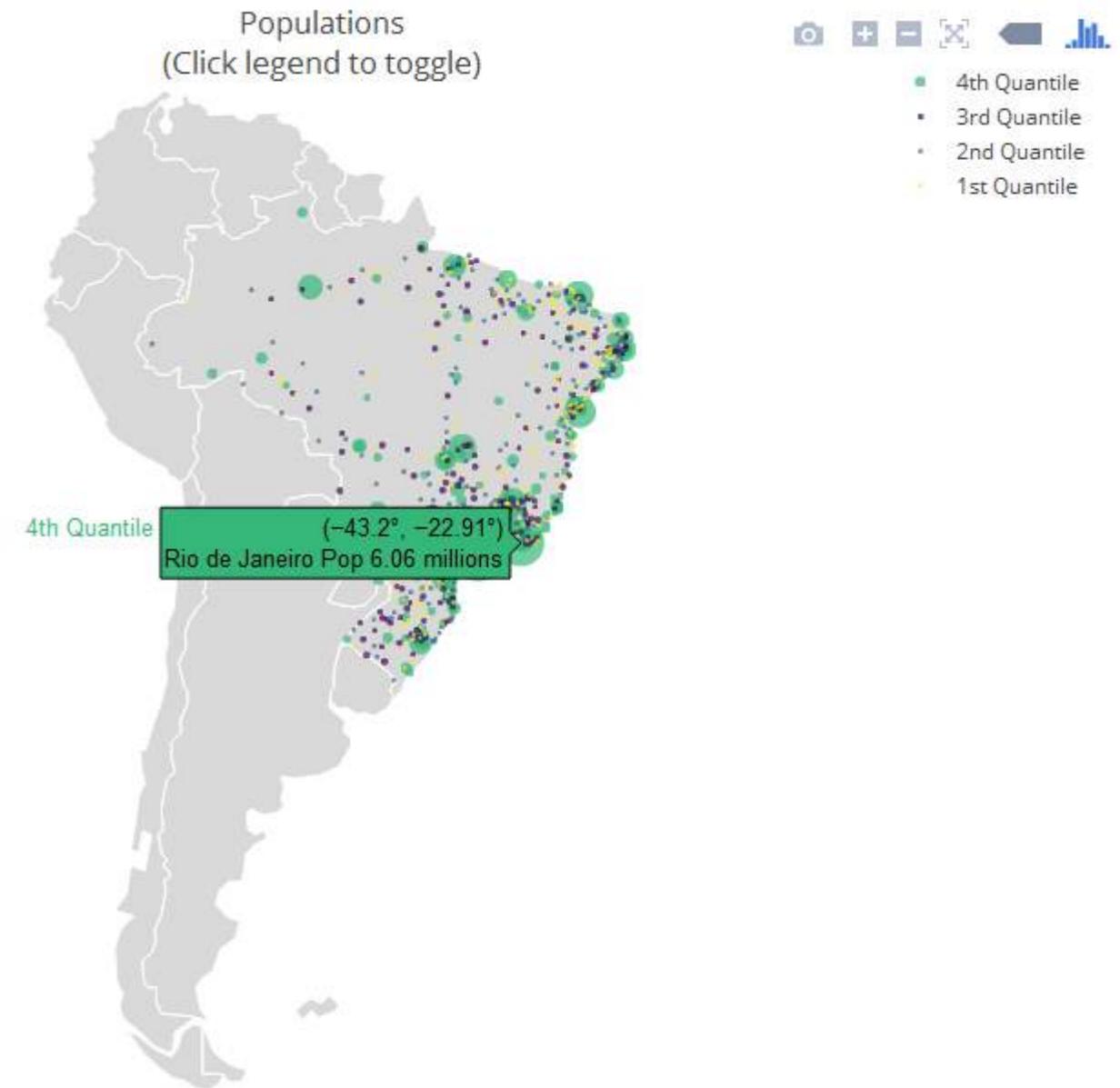
[Leaflet](#) 是一个用于制作网页动态地图的开源JavaScript库。RStudio为Leaflet编写了R绑定，可通过其[leaflet包](#)获得，基于[htmlwidgets](#)构建。Leaflet地图与[RMarkdown](#)和[Shiny](#)生态系统兼容良好。

该接口采用管道操作，使用`leaflet()`函数初始化地图，后续函数用于添加（或移除）地图图层。提供多种图层类型，从带弹出窗口的标记到用于创建分级色彩图的多边形。传递给`leaflet()`的数据框中的变量通过函数式~引用访问。

映射`state.name`和`state.center`数据集：

```
library(leaflet)

data.frame(state.name, state.center) %>%
  leaflet() %>%
addProviderTiles('Stamen.Watercolor') %>%
  addMarkers(lng = ~x, lat = ~y,
popup = ~state.name,
clusterOptions = markerClusterOptions())
```



Section 33.4: Making Dynamic HTML Maps with Leaflet

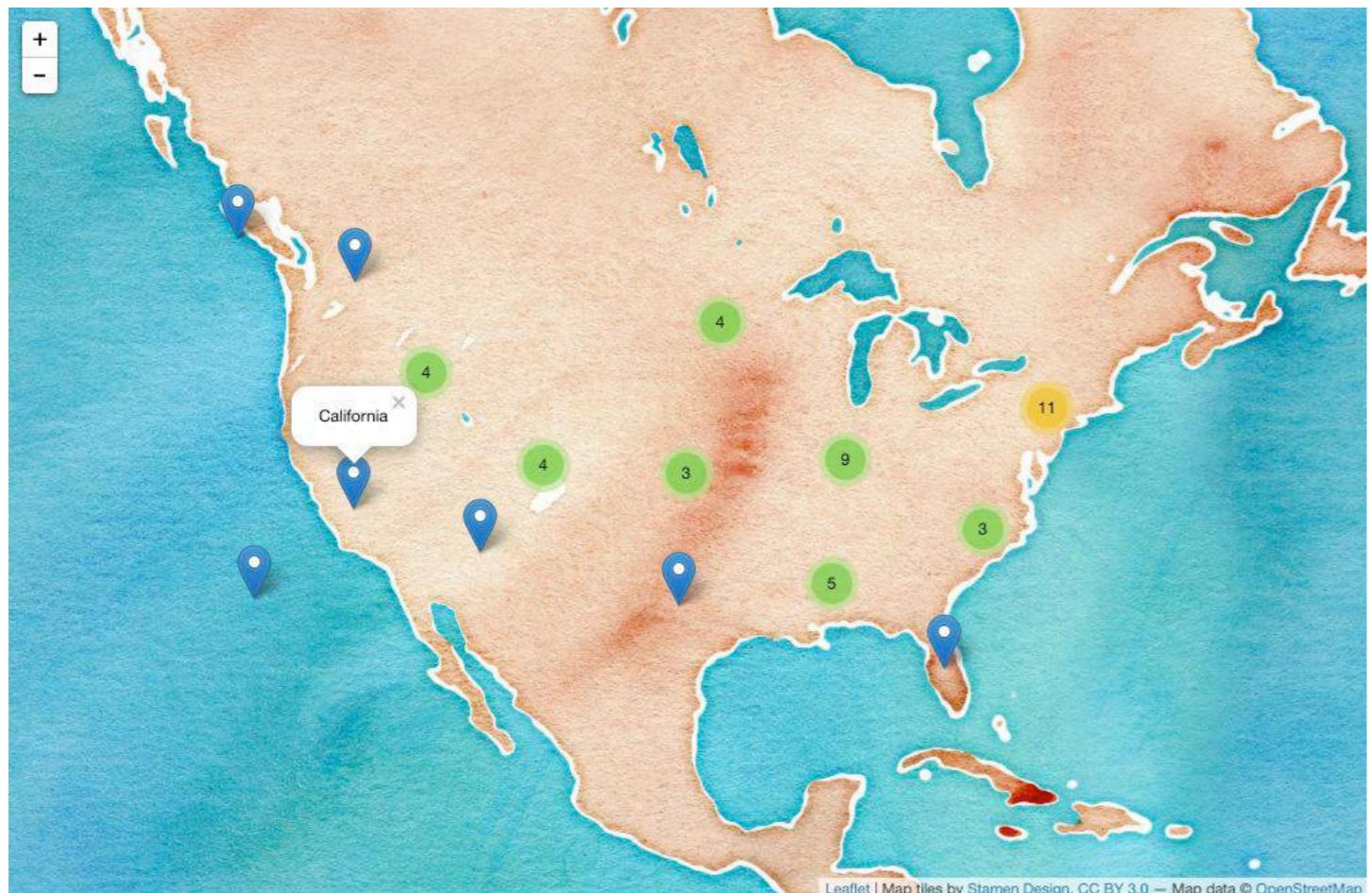
[Leaflet](#) is an open-source JavaScript library for making dynamic maps for the web. RStudio wrote R bindings for Leaflet, available through its [leaflet package](#), built with [htmlwidgets](#). Leaflet maps integrate well with the [RMarkdown](#) and [Shiny](#) ecosystems.

The interface is piped, using a `leaflet()` function to initialize a map and subsequent functions adding (or removing) map layers. Many kinds of layers are available, from markers with popups to polygons for creating choropleth maps. Variables in the `data.frame` passed to `leaflet()` are accessed via function-style ~ quotation.

To map the `state.name` and `state.center` datasets:

```
library(leaflet)

data.frame(state.name, state.center) %>%
  leaflet() %>%
addProviderTiles('Stamen.Watercolor') %>%
  addMarkers(lng = ~x, lat = ~y,
popup = ~state.name,
clusterOptions = markerClusterOptions())
```



(截图；点击查看动态版本。)

第33.5节：Shiny应用中的动态Leaflet地图

Leaflet包设计用于与Shiny集成

在ui中调用leafletOutput(), 在服务器端调用renderLeaflet()

```
library(shiny)
library(leaflet)

ui <- fluidPage(
  leafletOutput("my_leaf")
)

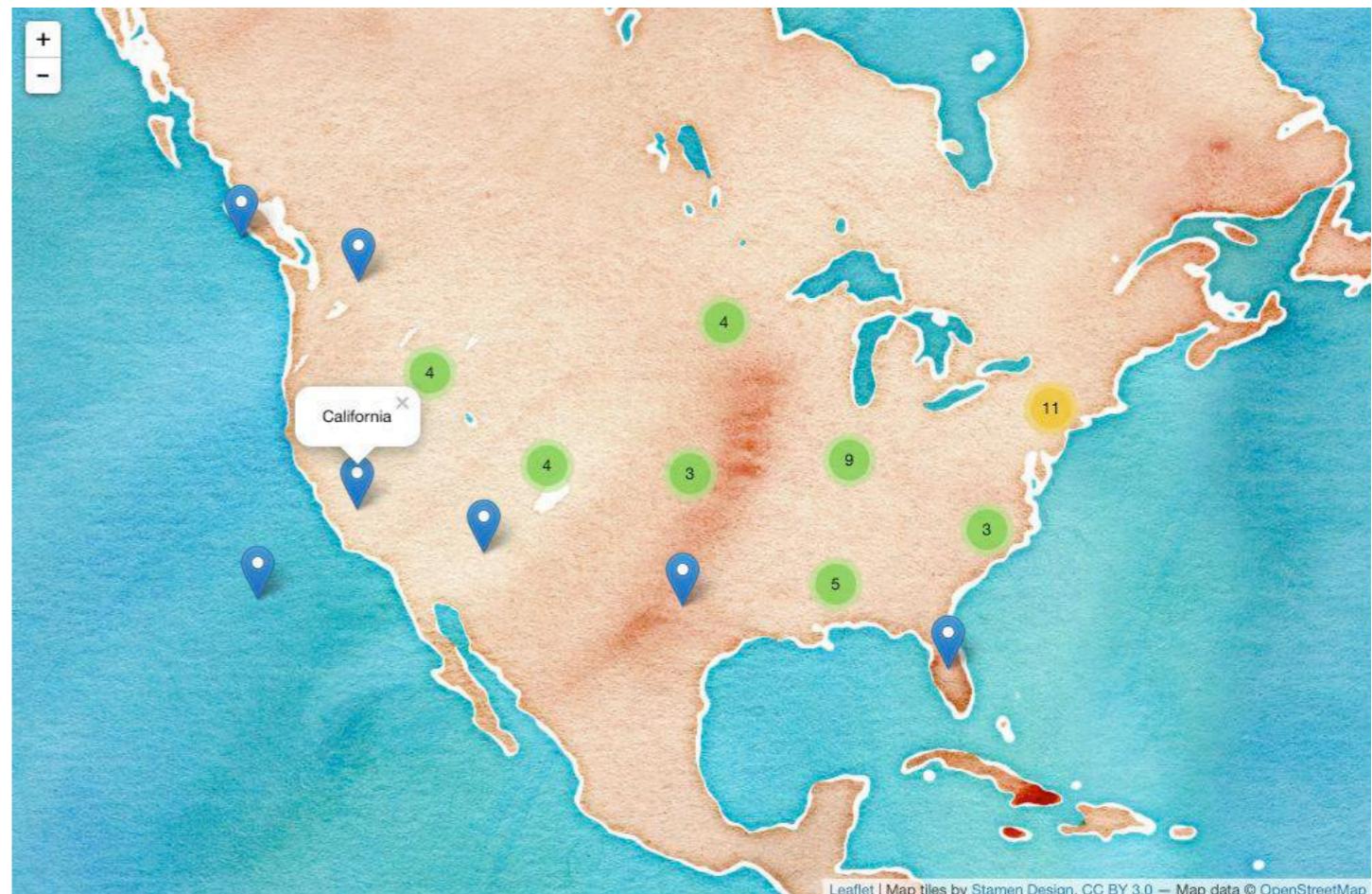
server <- function(input, output, session){

  output$my_leaf <- renderLeaflet{

    leaflet() %>%
      addProviderTiles('Hydda.Full') %>%
      setView(lat = -37.8, lng = 144.8, zoom = 10)
  }
}

shinyApp(ui, server)
```

然而，影响renderLeaflet表达式的响应式输入会导致每次响应式元素更新时整个地图被重新绘制。



(Screenshot; click for dynamic version.)

Section 33.5: Dynamic Leaflet maps in Shiny applications

The [Leaflet](#) package is designed to [integrate with Shiny](#)

In the **ui** you call `leafletOutput()` and in the server you call `renderLeaflet()`

```
library(shiny)
library(leaflet)

ui <- fluidPage(
  leafletOutput("my_leaf")
)

server <- function(input, output, session){

  output$my_leaf <- renderLeaflet{

    leaflet() %>%
      addProviderTiles('Hydda.Full') %>%
      setView(lat = -37.8, lng = 144.8, zoom = 10)
  }
}

shinyApp(ui, server)
```

However, reactive inputs that affect the `renderLeaflet` expression will cause the entire map to be redrawn each time the reactive element is updated.

因此，要修改已经运行的地图，应使用leafletProxy()函数。

通常你使用leaflet来创建地图的静态部分，使用leafletProxy来管理动态元素，例如：

```
library(shiny)
library(leaflet)

ui <- fluidPage(
  sliderInput(inputId = "slider",
              label = "values",
              min = 0,
              max = 100,
              value = 0,
              step = 1),
  leafletOutput("my_leaf")
)

server <- function(input, output, session){
  set.seed(123456)
  df <- data.frame(latitude = sample(seq(-38.5, -37.5, by = 0.01), 100),
                    longitude = sample(seq(144.0, 145.0, by = 0.01), 100),
                    value = seq(1, 100))

  ## 创建静态元素
  output$my_leaf <- renderLeaflet{

    leaflet() %>%
      addProviderTiles('Hydda.Full') %>%
      setView(lat = -37.8, lng = 144.8, zoom = 8)
  }

  ## 过滤数据
  df_filtered <- reactive{
    df[df$value >= input$slider, ]
  }

  ## 响应过滤后的数据
  observe{

    leafletProxy(mapId = "my_leaf", data = df_filtered()) %>%
      clearMarkers() %>% ## 清除之前的标记
      addMarkers()
  }
}

shinyApp(ui, server)
```

Therefore, to modify a map that's already running you should use the `leafletProxy()` function.

Normally you use `leaflet` to create the static aspects of the map, and `leafletProxy` to manage the dynamic elements, for example:

```
library(shiny)
library(leaflet)

ui <- fluidPage(
  sliderInput(inputId = "slider",
              label = "values",
              min = 0,
              max = 100,
              value = 0,
              step = 1),
  leafletOutput("my_leaf")
)

server <- function(input, output, session){
  set.seed(123456)
  df <- data.frame(latitude = sample(seq(-38.5, -37.5, by = 0.01), 100),
                    longitude = sample(seq(144.0, 145.0, by = 0.01), 100),
                    value = seq(1, 100))

  ## create static element
  output$my_leaf <- renderLeaflet{

    leaflet() %>%
      addProviderTiles('Hydda.Full') %>%
      setView(lat = -37.8, lng = 144.8, zoom = 8)
  }

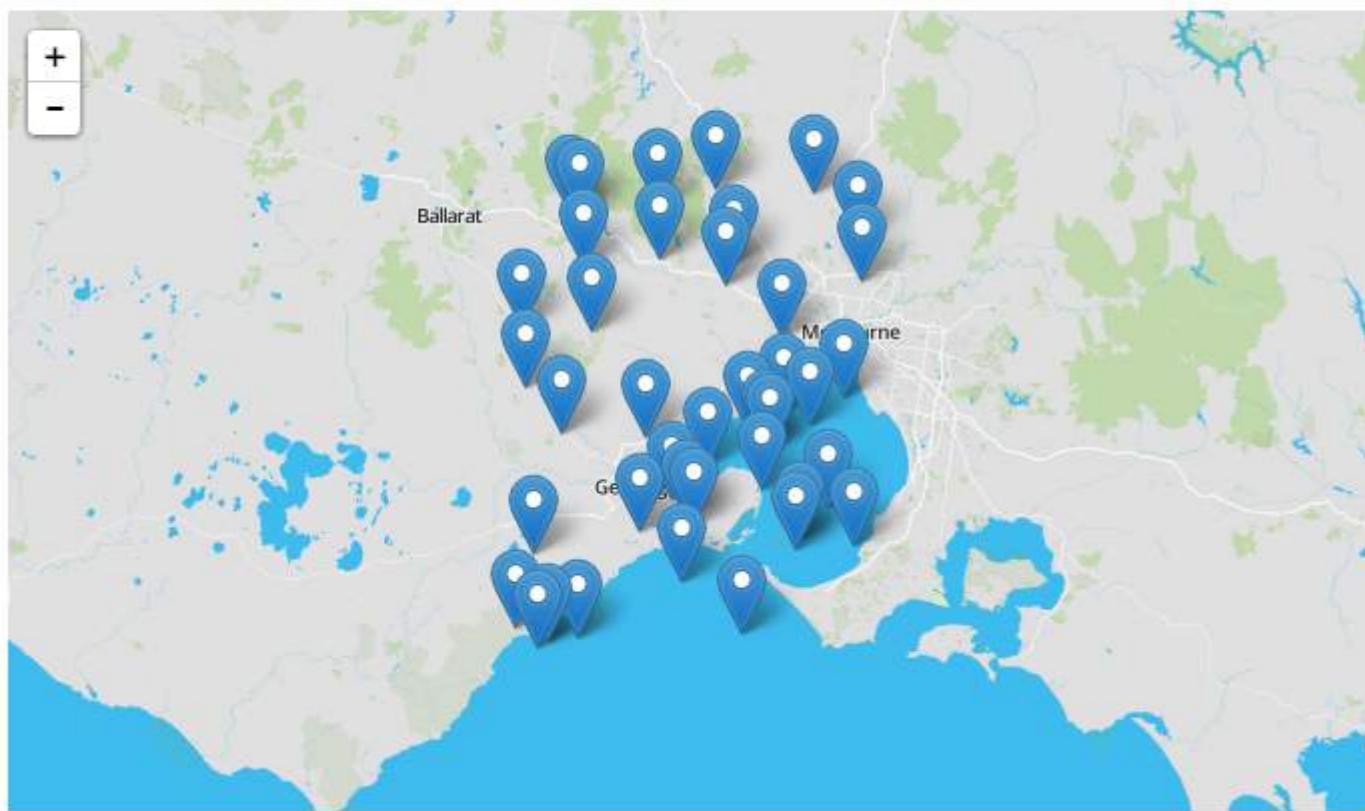
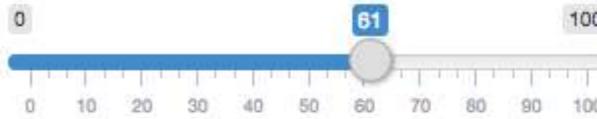
  ## filter data
  df_filtered <- reactive{
    df[df$value >= input$slider, ]
  }

  ## respond to the filtered data
  observe{

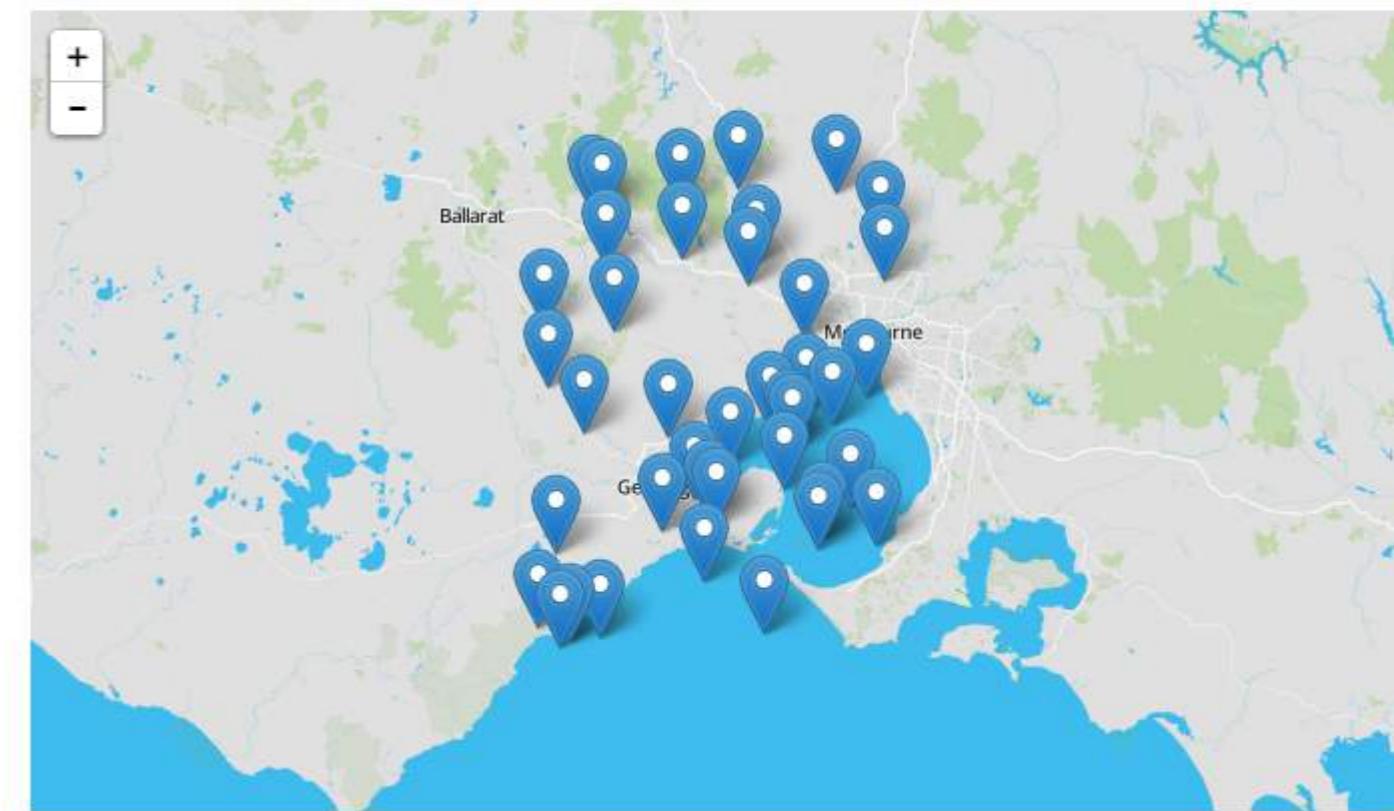
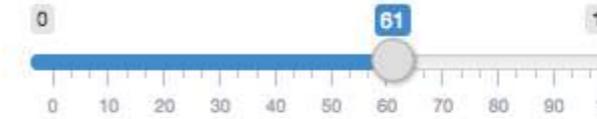
    leafletProxy(mapId = "my_leaf", data = df_filtered()) %>%
      clearMarkers() %>% ## clear previous markers
      addMarkers()
  }
}

shinyApp(ui, server)
```

values



values



第34章：集合运算

第34.1节：向量对的集合运算符

比较集合

在R中，向量可能包含重复元素：

```
v = "A"  
w = c("A", "A")
```

然而，集合中每个元素只有一个副本。R通过只取向量中不同的元素，将向量视为集合，因此上述两个向量被视为相同：

```
setequal(v, w)  
# TRUE
```

集合的合并

关键函数名称直观：

```
x = c(1, 2, 3)  
y = c(2, 4)  
  
union(x, y)  
# 1 2 3 4  
  
intersect(x, y)  
# 2  
  
setdiff(x, y)  
# 1 3
```

这些都记录在同一页面，?union。

第34.2节：向量的笛卡尔积或“交叉”积

为了找到所有形式为(x, y)的向量，其中x取自向量X，y取自Y，我们使用expand.grid：

```
X = c(1, 1, 2)  
Y = c(4, 5)  
  
expand.grid(X, Y)
```

```
#   Var1 Var2  
# 1    1    4  
# 2    1    4  
# 3    2    4  
# 4    1    5  
# 5    1    5  
# 6    2    5
```

结果是一个data.frame，每个传入的向量对应一列。通常，我们想要的是集合的笛卡尔积，而不是展开向量的“网格”。我们可以使用unique、lapply和do.call：

```
m = do.call(expand.grid, lapply(list(X, Y), unique))
```

Chapter 34: Set operations

Section 34.1: Set operators for pairs of vectors

Comparing sets

In R, a vector may contain duplicated elements:

```
v = "A"  
w = c("A", "A")
```

However, a set contains only one copy of each element. R treats a vector like a set by taking only its distinct elements, so the two vectors above are regarded as the same:

```
setequal(v, w)  
# TRUE
```

Combining sets

The key functions have natural names:

```
x = c(1, 2, 3)  
y = c(2, 4)  
  
union(x, y)  
# 1 2 3 4  
  
intersect(x, y)  
# 2  
  
setdiff(x, y)  
# 1 3
```

These are all documented on the same page, ?union.

Section 34.2: Cartesian or "cross" products of vectors

To find every vector of the form (x, y) where x is drawn from vector X and y from Y, we use `expand.grid`:

```
X = c(1, 1, 2)  
Y = c(4, 5)  
  
expand.grid(X, Y)
```

```
#   Var1 Var2  
# 1    1    4  
# 2    1    4  
# 3    2    4  
# 4    1    5  
# 5    1    5  
# 6    2    5
```

The result is a data.frame with one column for each vector passed to it. Often, we want to take the Cartesian product of sets rather than to expand a "grid" of vectors. We can use `unique`, `lapply` and `do.call`:

```
m = do.call(expand.grid, lapply(list(X, Y), unique))
```

```
# Var1 Var2  
# 1 1 4  
# 2 2 4  
# 3 1 5  
# 4 2 5
```

对组合应用函数

如果你想对每个结果组合 $f(x,y)$ 应用一个函数，可以将其作为另一列添加：

```
m$p = with(m, Var1*Var2)  
# Var1 Var2 p  
# 1 1 4 4  
# 2 2 4 8  
# 3 1 5 5  
# 4 2 5 10
```

这种方法适用于任意数量的向量，但在两个向量的特殊情况下，有时将结果放在矩阵中更合适，这可以通过outer实现：

```
uX = unique(X)  
uY = unique(Y)  
  
outer(setNames(uX, uX), setNames(uY, uY), `*`)  
  
# 4 5  
# 1 4 5  
# 2 8 10
```

有关相关概念和工具，请参见组合学主题。

第34.3节：向量的集合成员资格

%in% 操作符用于比较向量与集合。

```
v = "A"  
w = c("A", "A")  
  
w %in% v  
# TRUE TRUE  
  
v %in% w  
# TRUE
```

左侧的每个元素都会被单独处理，并测试其是否属于右侧向量关联的集合（该集合由其所有不同元素组成）。

与相等测试不同，%in% 总是返回 TRUE 或 FALSE：

```
c(1, NA) %in% c(1, 2, 3, 4)  
# TRUE FALSE
```

文档位于 ?`%in%`。

第34.4节：去重 / 删除重复项 / 选择不同元素

```
# Var1 Var2  
# 1 1 4  
# 2 2 4  
# 3 1 5  
# 4 2 5
```

Applying functions to combinations

If you then want to apply a function to each resulting combination $f(x, y)$, it can be added as another column:

```
m$p = with(m, Var1*Var2)  
# Var1 Var2 p  
# 1 1 4 4  
# 2 2 4 8  
# 3 1 5 5  
# 4 2 5 10
```

This approach works for as many vectors as we need, but in the special case of two, it is sometimes a better fit to have the result in a matrix, which can be achieved with outer:

```
uX = unique(X)  
uY = unique(Y)  
  
outer(setNames(uX, uX), setNames(uY, uY), `*`)  
  
# 4 5  
# 1 4 5  
# 2 8 10
```

For related concepts and tools, see the combinatorics topic.

Section 34.3: Set membership for vectors

The %in% operator compares a vector with a set.

```
v = "A"  
w = c("A", "A")  
  
w %in% v  
# TRUE TRUE  
  
v %in% w  
# TRUE
```

Each element on the left is treated individually and tested for membership in the set associated with the vector on the right (consisting of all its distinct elements).

Unlike equality tests, %in% always returns TRUE or FALSE:

```
c(1, NA) %in% c(1, 2, 3, 4)  
# TRUE FALSE
```

The documentation is at ?`%in%`.

Section 34.4: Make unique / drop duplicates / select distinct

向量中的元素

`unique` 会删除重复项，使结果中的每个元素唯一（只出现一次）：

```
x = c(2, 1, 1, 2, 1)
```

```
unique(x)  
# 2 1
```

返回的值按它们首次出现的顺序排列。

重复的 标签每个重复的元素：

```
duplicated(x)  
# FALSE FALSE TRUE TRUE TRUE
```

`anyDuplicated(x) > 0L` 是快速检查向量中是否包含重复项的方法。

第34.5节：测量集合重叠 / 向量的韦恩图

要计算两个集合中有多少元素重叠，可以编写自定义函数：

```
xtab_set <- function(A, B){  
  both <- union(A, B)  
  inA <- both %in% A  
  inB <- both %in% B  
  return(table(inA, inB))  
}  
  
A = 1:20  
B = 10:30  
  
xtab_set(A, B)  
  
#      inB  
# inA  FALSE TRUE  
# FALSE    0   10  
# TRUE     9   11
```

维恩图，由各种软件包提供，可用于可视化多个集合之间的重叠计数。

elements from a vector

`unique` drops duplicates so that each element in the result is unique (only appears once):

```
x = c(2, 1, 1, 2, 1)
```

```
unique(x)  
# 2 1
```

Values are returned in the order they first appeared.

`duplicated` tags each duplicated element:

```
duplicated(x)  
# FALSE FALSE TRUE TRUE TRUE
```

`anyDuplicated(x) > 0L` is a quick way of checking whether a vector contains any duplicates.

Section 34.5: Measuring set overlaps / Venn diagrams for vectors

To count how many elements of two sets overlap, one could write a custom function:

```
xtab_set <- function(A, B){  
  both <- union(A, B)  
  inA <- both %in% A  
  inB <- both %in% B  
  return(table(inA, inB))  
}  
  
A = 1:20  
B = 10:30  
  
xtab_set(A, B)  
  
#      inB  
# inA  FALSE TRUE  
# FALSE    0   10  
# TRUE     9   11
```

A Venn diagram, offered by various packages, can be used to visualize overlap counts across multiple sets.

第35章：tidyverse

第35.1节：tidyverse概述

什么是 tidyverse ?

tidyverse 是一种快速且优雅的方式，将基础的 R 转变为一个经过Hadley/Rstudio重新设计的增强工具。所有包含在 tidyverse 中的软件包的开发都遵循 整洁工具宣言 (The tidy tools manifesto) 的原则规则。但首先，让作者们来描述他们的杰作：

tidyverse是一组协同工作的软件包，因为它们共享通用的数据表示和API设计。tidyverse软件包旨在通过一条命令轻松安装和加载tidyverse的核心软件包。

了解tidyverse中所有软件包及其如何协同工作的最佳途径是《R数据科学》 (R for Data Science) 。在接下来的几个月里，随着我致力于改进软件包网站、简化引用以及为tidyverse数据分析讨论提供统一平台，预计会听到更多关于tidyverse的消息。

([source](#))

如何使用它？

就像普通的 R 软件包一样，你需要安装并加载该软件包。

```
install.package("tidyverse")
library("tidyverse")
```

区别在于，使用一个命令即可安装/加载数十个包。额外好处是，可以放心所有安装/加载的包版本都是兼容的。

这些包都有哪些？

常见且广泛使用的包：

- [ggplot2](#) : 高级数据可视化 SO_doc
- [dplyr](#) : 快速（基于Rcpp）且一致的数据操作方法 SO_doc
- [tidyr](#) : 数据整理工具 SO_doc
- [readr](#) : 用于数据导入。
- [purrr](#) : 通过补充R的函数式编程工具，借鉴JS包underscore.js、lodash和lazy.js的风格，赋予纯函数更多功能，使其更加灵活。
- [tibble](#) : 数据框的现代化重新设计。
- [magrittr](#) : 管道操作，使代码更易读 SO_doc

用于处理特定数据格式的包：

- [hms](#): 易于读取的时间
- [stringr](#): 提供一套连贯的函数，旨在使字符串处理尽可能简单
- [lubridate](#): 高级日期/时间操作 SO_doc
- [forcats](#): 因子（分类变量）的高级处理。

数据导入：

Chapter 35: tidyverse

Section 35.1: tidyverse: an overview

What is tidyverse?

[tidyverse](#) is the fast and elegant way to turn basic R into an enhanced tool, redesigned by Hadley/Rstudio. The development of all packages included in tidyverse follow the principle rules of [The tidy tools manifesto](#). But first, let the authors describe their masterpiece:

The tidyverse is a set of packages that work in harmony because they share common data representations and API design. The tidyverse package is designed to make it easy to install and load core packages from the tidyverse in a single command.

The best place to learn about all the packages in the tidyverse and how they fit together is R for Data Science. Expect to hear more about the tidyverse in the coming months as I work on improved package websites, making citation easier, and providing a common home for discussions about data analysis with the tidyverse.

([source](#))

How to use it?

Just with the ordinary R packages, you need to install and load the package.

```
install.package("tidyverse")
library("tidyverse")
```

The difference is, on a single command a couple of dozens of packages are installed/loaded. As a bonus, one may rest assured that all the installed/loaded packages are of compatible versions.

What are those packages?

The commonly known and widely used packages:

- [ggplot2](#): advanced data visualisation SO_doc
- [dplyr](#): fast (Rcpp) and coherent approach to data manipulation SO_doc
- [tidyr](#): tools for data tidying SO_doc
- [readr](#): for data import.
- [purrr](#): makes your pure functions purr by completing R's functional programming tools with important features from other languages, in the style of the JS packages underscore.js, lodash and lazy.js.
- [tibble](#): a modern re-imagining of data frames.
- [magrittr](#): piping to make code more readable SO_doc

Packages for manipulating specific data formats:

- [hms](#): easily read times
- [stringr](#): provide a cohesive set of functions designed to make working with strings as easy as possible
- [lubridate](#): advanced date/times manipulations SO_doc
- [forcats](#): advanced work with factors.

Data import:

- [DBI](#): 定义了R与数据库管理系统（DBMS）之间的通用接口
- [haven](#): 轻松导入SPSS、SAS和Stata文件 SO_doc
- [httr](#): httr的目标是为curl包提供一个包装器，定制以满足现代网络

API的需求

- [jsonlite](#): 一个快速的JSON解析器和生成器，针对统计数据和网络进行了优化
- [readxl](#): 读取.xls和.xlsx文件，无需依赖其他包 SO_doc
- [rvest](#): rvest帮助你从网页抓取信息 SO_doc
- [xml2](#): 用于XML

以及建模：

- [modelr](#): 提供帮助你创建优雅建模流程的函数
- [broom](#): 轻松将模型提取为整洁数据

最后，tidyverse 建议使用：

- [knitr](#)：令人惊叹的通用型文档编程引擎，具有轻量级API，设计旨在让用户无需大量编码即可完全控制输出。 SO_docs : one, two
- [rmarkdown](#)：Rstudio用于可重复编程的包。 SO_docs : one, two, three, four

第35.2节：创建tbl_df

tbl_df（发音为ibble diff）是数据框的一种变体，常用于tidyverse包中。它由ibble包实现。

使用as_data_frame函数将数据框转换为tbl_df：

```
library(tibble)
mtcars_tbl <- as_data_frame(mtcars)
```

数据框和tbl_df最显著的区别之一是它们的打印方式：

```
# 一个tibble : 32行x11列
mpg cyl disp hp drat wt qsec vs am gear carb
* <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 21.0 6 160.0 110 3.90 2.620 16.46 0 1 4 4
2 21.0 6 160.0 110 3.90 2.875 17.02 0 1 4 4
3 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1
4 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1
5 18.7 8 360.0 175 3.15 3.440 17.02 0 0 3 2
6 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1
7 14.3 8 360.0 245 3.21 3.570 15.84 0 0 3 4
8 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2
9 22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2
10 19.2 6 167.6 123 3.92 3.440 18.30 1 0 4 4
# ... 还有22行
```

- 打印输出包括表格尺寸的摘要（32 x 11）
- 其中包含每列的类型（dbl）
- 它打印有限数量的行。（要更改此设置，请使用options(tibble.print_max = [数字])）。

dplyr包中的许多函数可以自然地与tbl_df一起使用，例如group_by()。

- [DBI](#): defines a common interface between the R and database management systems (DBMS)
- [haven](#): easily import SPSS, SAS and Stata files SO_doc
- [httr](#): the aim of httr is to provide a wrapper for the curl package, customised to the demands of modern web APIs
- [jsonlite](#): a fast JSON parser and generator optimized for statistical data and the web
- [readxl](#): read.xls and .xlsx files without need for dependency packages SO_doc
- [rvest](#): rvest helps you scrape information from web pages SO_doc
- [xml2](#): for XML

And modelling:

- [modelr](#): provides functions that help you create elegant pipelines when modelling
- [broom](#): easily extract the models into tidy data

Finally, tidyverse suggest the use of:

- [knitr](#): the amazing general-purpose literate programming engine, with lightweight API's designed to give users full control of the output without heavy coding work. SO_docs: one, two
- [rmarkdown](#): Rstudio's package for reproducible programming. SO_docs: one, two, three, four

Section 35.2: Creating tbl_df's

A `tbl_df` (pronounced *tibble diff*) is a variation of a data frame that is often used in tidyverse packages. It is implemented in the [tibble](#) package.

Use the `as_data_frame` function to turn a data frame into a `tbl_df`:

```
library(tibble)
mtcars_tbl <- as_data_frame(mtcars)
```

One of the most notable differences between data.frames and `tbl_dfs` is how they print:

```
# A tibble: 32 x 11
  mpg cyl disp hp drat wt qsec vs am gear carb
* <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
1 21.0 6 160.0 110 3.90 2.620 16.46 0 1 4 4
2 21.0 6 160.0 110 3.90 2.875 17.02 0 1 4 4
3 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1
4 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1
5 18.7 8 360.0 175 3.15 3.440 17.02 0 0 3 2
6 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1
7 14.3 8 360.0 245 3.21 3.570 15.84 0 0 3 4
8 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2
9 22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2
10 19.2 6 167.6 123 3.92 3.440 18.30 1 0 4 4
# ... with 22 more rows
```

- The printed output includes a summary of the dimensions of the table (32 x 11)
- It includes the type of each column (dbl)
- It prints a limited number of rows. (To change this use `options(tibble.print_max = [number])`).

Many functions in the dplyr package work naturally with `tbl_dfs`, such as `group_by()`.

第36章：Rcpp

第36.1节：使用插件扩展Rcpp

在C++中，可以使用以下方式设置不同的编译标志：

```
// [[Rcpp::plugins(name)]]
```

内置插件列表：

```
// 内置的C++11插件  
// [[Rcpp::plugins(cpp11)]]
```

```
// 适用于较旧g++编译器的C++11插件  
// [[Rcpp::plugins(cpp0x)]]
```

```
// 适用于C++14标准的内置C++14插件  
// [[Rcpp::plugins(cpp14)]]
```

```
// 适用于C++14和C++17标准的开发中C++1y插件  
// [[Rcpp::plugins(cpp1y)]]
```

```
// built-in OpenMP++11 plugin  
// [[Rcpp::plugins(openmp)]]
```

第36.2节：内联代码编译

Rcpp 提供了两个函数，允许内联编译代码并直接导出到 R 中：`cppFunction()` 和 `evalCpp()`。第三个函数 `sourceCpp()` 用于读取单独文件中的 C++ 代码，但也可以类似于 `cppFunction()` 使用。

下面是一个在 R 中编译 C++ 函数的示例。注意使用 "" 来包围源代码。

```
# 注意 - 这是 R 代码。  
# Rcpp 中的 cppFunction 允许快速测试。  
require(Rcpp)
```

```
# 创建一个函数，将向量中的每个元素乘以一个数  
# 返回修改后的向量。  
cppFunction(  
  NumericVector exfun(NumericVector x, int i){  
    x = x*i;  
    return x;  
  }
```

```
# 在R中调用函数  
exfun(1:5, 3)
```

快速理解C++表达式可使用：

```
# 使用evalCpp评估C++表达式  
evalCpp("std::numeric_limits<double>::max()")  
## [1] 1.797693e+308
```

Chapter 36: Rcpp

Section 36.1: Extending Rcpp with Plugins

Within C++, one can set different compilation flags using:

```
// [[Rcpp::plugins(name)]]
```

List of the built-in plugins:

```
// built-in C++11 plugin  
// [[Rcpp::plugins(cpp11)]]  
  
// built-in C++11 plugin for older g++ compiler  
// [[Rcpp::plugins(cpp0x)]]  
  
// built-in C++14 plugin for C++14 standard  
// [[Rcpp::plugins(cpp14)]]  
  
// built-in C++1y plugin for C++14 and C++17 standard under development  
// [[Rcpp::plugins(cpp1y)]]  
  
// built-in OpenMP++11 plugin  
// [[Rcpp::plugins(openmp)]]
```

Section 36.2: Inline Code Compile

Rcpp features two functions that enable code compilation inline and exportation directly into R: `cppFunction()` and `evalCpp()`. A third function called `sourceCpp()` exists to read in C++ code in a separate file though can be used akin to `cppFunction()`.

Below is an example of compiling a C++ function within R. Note the use of "" to surround the source.

```
# Note - This is R code.  
# cppFunction in Rcpp allows for rapid testing.  
require(Rcpp)  
  
# Creates a function that multiplies each element in a vector  
# Returns the modified vector.  
cppFunction(  
  NumericVector exfun(NumericVector x, int i){  
    x = x*i;  
    return x;  
  })  
  
# Calling function in R  
exfun(1:5, 3)
```

To quickly understand a C++ expression use:

```
# Use evalCpp to evaluate C++ expressions  
evalCpp("std::numeric_limits<double>::max()")  
## [1] 1.797693e+308
```

第36.3节：Rcpp属性

Rcpp属性使得R与C++的协作过程变得简单。属性的形式为：

```
// [[Rcpp::attribute]]
```

属性的使用通常与以下内容相关联：

```
// [[Rcpp::export]]
```

该属性直接放置在通过sourceCpp()读取的C++文件中声明的函数头部上方。

下面是一个使用属性的外部C++文件示例。

```
// 将以下代码添加到C++文件Rcpp_example.cpp中
```

```
#include <Rcpp.h>
using namespace Rcpp;
```

// 将导出标签放在函数声明的正上方。

```
// [[Rcpp::export]]
```

```
double muRcpp(NumericVector x){
```

```
    int n = x.size(); // 向量大小
    double sum = 0; // 求和变量
```

```
    // for循环，注意C++索引从0开始
    for(int i = 0; i < n; i++){
        // 简写形式 sum = sum + x[i]
        sum += x[i];
    }
```

```
    return sum/n; // 计算并返回均值
}
```

// 将依赖函数放在调用之前，或者

// 使用以下方式声明函数定义：

```
double muRcpp(NumericVector x);
```

```
// [[Rcpp::export]]
```

```
double varRcpp(NumericVector x, bool bias = true){
```

```
    // 使用C++函数计算均值
    double mean = muRcpp(x);
    double sum = 0;
```

```
    int n = x.size();
```

```
    for(int i = 0; i < n; i++){
        sum += pow(x[i] - mean, 2.0); // 平方
    }
```

```
    return sum/(n-bias); // 返回方差
}
```

要在R中使用这个外部C++文件，我们执行以下操作：

```
require(Rcpp)
```

Section 36.3: Rcpp Attributes

Rcpp Attributes makes the process of working with R and C++ straightforward. The form of attributes take:

```
// [[Rcpp::attribute]]
```

The use of attributes is typically associated with:

```
// [[Rcpp::export]]
```

that is placed directly above a declared function header when reading in a C++ file via sourceCpp().

Below is an example of an external C++ file that uses attributes.

```
// Add code below into C++ file Rcpp_example.cpp
```

```
#include <Rcpp.h>
using namespace Rcpp;
```

```
// Place the export tag right above function declaration.
// [[Rcpp::export]]
double muRcpp(NumericVector x){
```

```
    int n = x.size(); // Size of vector
    double sum = 0; // Sum value
```

```
    // For loop, note cpp index shift to 0
    for(int i = 0; i < n; i++){
        // Shorthand for sum = sum + x[i]
        sum += x[i];
    }
```

```
    return sum/n; // Obtain and return the Mean
}
```

```
// Place dependent functions above call or
// declare the function definition with:
double muRcpp(NumericVector x);
```

```
// [[Rcpp::export]]
double varRcpp(NumericVector x, bool bias = true){
```

```
    // Calculate the mean using C++ function
    double mean = muRcpp(x);
    double sum = 0;
```

```
    int n = x.size();
```

```
    for(int i = 0; i < n; i++){
        sum += pow(x[i] - mean, 2.0); // Square
    }
```

```
    return sum/(n-bias); // Return variance
}
```

To use this external C++ file within R, we do the following:

```
require(Rcpp)
```

```
# 编译文件
sourceCpp("path/to/file/Rcpp_example.cpp")

# 创建一些示例数据
x = 1:5

all.equal(muRcpp(x), mean(x))
## TRUE

all.equal(varRcpp(x), var(x))
## TRUE
```

```
# Compile File
sourceCpp("path/to/file/Rcpp_example.cpp")

# Make some sample data
x = 1:5

all.equal(muRcpp(x), mean(x))
## TRUE

all.equal(varRcpp(x), var(x))
## TRUE
```

第36.4节：指定额外的构建依赖

要在Rcpp生态系统中使用额外的软件包，正确的头文件可能不是Rcpp.h，而是Rcpp<PACKAGE>.h（例如RcppArmadillo）。通常需要先导入该头文件，然后在

中声明依赖关系

```
// [[Rcpp::depends(Rcpp<PACKAGE>)]]
```

示例：

```
// 使用RcppArmadillo包
// 需要不同于Rcpp.h的头文件
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]
```



```
// 使用RcppEigen包
// 需要不同于Rcpp.h的头文件
#include <RcppEigen.h>
// [[Rcpp::depends(RcppEigen)]]
```

Section 36.4: Specifying Additional Build Dependencies

To use additional packages within the Rcpp ecosystem, the correct header file may not be Rcpp.h but Rcpp<PACKAGE>.h (as e.g. for [RcppArmadillo](#)). It typically needs to be imported and then the dependency is stated within

```
// [[Rcpp::depends(Rcpp<PACKAGE>)]]
```

Examples:

```
// Use the RcppArmadillo package
// Requires different header file from Rcpp.h
#include <RcppArmadillo.h>
// [[Rcpp::depends(RcppArmadillo)]]
```



```
// Use the RcppEigen package
// Requires different header file from Rcpp.h
#include <RcppEigen.h>
// [[Rcpp::depends(RcppEigen)]]
```

第37章：随机数生成器

第37.1节：随机排列

生成5个数字的随机排列：

```
sample(5)
# [1] 4 5 3 1 2
```

生成任意向量的随机排列：

```
sample(10:15)
# [1] 11 15 12 10 14 13
```

也可以使用包pracma

```
randperm(a, k)
# 生成a中k个元素的一个随机排列，如果a是向量，  
# 或者生成1:a的随机排列，如果a是单个整数。  
# a : 整数或长度为n的数值向量。  
# k : 整数，小于a或length(a)。
```

```
# 示例
library(pracma)
randperm(1:10, 3)
[1] 3 7 9
```

```
randperm(10, 10)
[1] 4 5 10 8 2 7 6 9 3 1
```

```
randperm(seq(2, 10, by=2))
[1] 6 4 10 2 8
```

第37.2节：使用各种密度函数生成随机数

下面是使用各种概率分布生成5个随机数的示例。

0到10之间的均匀分布

```
runif(5, min=0, max=10)
[1] 2.1724399 8.9209930 6.1969249 9.3303321 2.4054102
```

均值为0，标准差为1的正态分布

```
rnorm(5, mean=0, sd=1)
[1] -0.97414402 -0.85722281 -0.08555494 -0.37444299 1.20032409
```

二项分布，试验次数为10，成功概率为0.5

```
rbinom(5, size=10, prob=0.5)
[1] 4 3 5 2 3
```

几何分布，成功概率为0.2

```
rgeom(5, prob=0.2)
[1] 14 8 11 1 3
```

Chapter 37: Random Numbers Generator

Section 37.1: Random permutations

To generate random permutation of 5 numbers:

```
sample(5)
# [1] 4 5 3 1 2
```

To generate random permutation of any vector:

```
sample(10:15)
# [1] 11 15 12 10 14 13
```

One could also use the package pracma

```
randperm(a, k)
# Generates one random permutation of k of the elements a, if a is a vector,
# or of 1:a if a is a single integer.
# a: integer or numeric vector of some length n.
# k: integer, smaller as a or length(a).
```

```
# Examples
library(pracma)
randperm(1:10, 3)
[1] 3 7 9
```

```
randperm(10, 10)
[1] 4 5 10 8 2 7 6 9 3 1
```

```
randperm(seq(2, 10, by=2))
[1] 6 4 10 2 8
```

Section 37.2: Generating random numbers using various density functions

Below are examples of generating 5 random numbers using various probability distributions.

Uniform distribution between 0 and 10

```
runif(5, min=0, max=10)
[1] 2.1724399 8.9209930 6.1969249 9.3303321 2.4054102
```

Normal distribution with 0 mean and standard deviation of 1

```
rnorm(5, mean=0, sd=1)
[1] -0.97414402 -0.85722281 -0.08555494 -0.37444299 1.20032409
```

Binomial distribution with 10 trials and success probability of 0.5

```
rbinom(5, size=10, prob=0.5)
[1] 4 3 5 2 3
```

Geometric distribution with 0.2 success probability

```
rgeom(5, prob=0.2)
[1] 14 8 11 1 3
```

超几何分布，白球3个，黑球10个，抽取5个

```
rhyper(5, m=3, n=10, k=5)
[1] 2 0 1 1 1
```

负二项分布，试验次数为10，成功概率为0.8

```
rnbnom(5, size=10, prob=0.8)
[1] 3 1 3 4 2
```

均值和方差 (lambda) 为2的泊松分布

```
rpois(5, lambda=2)
[1] 2 1 2 3 4
```

参数为1.5的指数分布

```
rexp(5, rate=1.5)
[1] 1.8993303 0.4799358 0.5578280 1.5630711 0.6228000
```

位置参数为0，尺度参数为1的逻辑斯蒂分布

```
rlogis(5, location=0, scale=1)
[1] 0.9498992 -1.0287433 -0.4192311 0.7028510 -1.2095458
```

自由度为15的卡方分布

```
rchisq(5, df=15)
[1] 14.89209 19.36947 10.27745 19.48376 23.32898
```

Beta 分布，形状参数 a=1, b=0.5

```
rbeta(5, shape1=1, shape2=0.5)
[1] 0.1670306 0.5321586 0.9869520 0.9548993 0.9999737
```

Gamma 分布，形状参数为 3，尺度参数为 0.5

```
rgamma(5, shape=3, scale=0.5)
[1] 2.2445984 0.7934152 3.2366673 2.2897537 0.8573059
```

Cauchy 分布，位置参数为 0，尺度参数为 1

```
rcauchy(5, location=0, scale=1)
[1] -0.01285116 -0.38918446 8.71016696 10.60293284 -0.68017185
```

对数正态分布，均值为 0，标准差为 1 (对数尺度)

```
rlnorm(5, meanlog=0, sdlog=1)
[1] 0.8725009 2.9433779 0.3329107 2.5976206 2.8171894
```

Weibull 分布，形状参数为 0.5，尺度参数为 1

```
rweibull(5, shape=0.5, scale=1)
[1] 0.337599112 1.307774557 7.233985075 5.840429942 0.005751181
```

第一组样本有10个观测值，第二组有20个观测值的Wilcoxon分布。

```
rwilcox(5, 10, 20)
[1] 111 88 93 100 124
```

使用指定概率的5个对象和3个箱子的多项分布

Hypergeometric distribution with 3 white balls, 10 black balls and 5 draws

```
rhyper(5, m=3, n=10, k=5)
[1] 2 0 1 1 1
```

Negative Binomial distribution with 10 trials and success probability of 0.8

```
rnbnom(5, size=10, prob=0.8)
[1] 3 1 3 4 2
```

Poisson distribution with mean and variance (lambda) of 2

```
rpois(5, lambda=2)
[1] 2 1 2 3 4
```

Exponential distribution with the rate of 1.5

```
rexp(5, rate=1.5)
[1] 1.8993303 0.4799358 0.5578280 1.5630711 0.6228000
```

Logistic distribution with 0 location and scale of 1

```
rlogis(5, location=0, scale=1)
[1] 0.9498992 -1.0287433 -0.4192311 0.7028510 -1.2095458
```

Chi-squared distribution with 15 degrees of freedom

```
rchisq(5, df=15)
[1] 14.89209 19.36947 10.27745 19.48376 23.32898
```

Beta distribution with shape parameters a=1 and b=0.5

```
rbeta(5, shape1=1, shape2=0.5)
[1] 0.1670306 0.5321586 0.9869520 0.9548993 0.9999737
```

Gamma distribution with shape parameter of 3 and scale=0.5

```
rgamma(5, shape=3, scale=0.5)
[1] 2.2445984 0.7934152 3.2366673 2.2897537 0.8573059
```

Cauchy distribution with 0 location and scale of 1

```
rcauchy(5, location=0, scale=1)
[1] -0.01285116 -0.38918446 8.71016696 10.60293284 -0.68017185
```

Log-normal distribution with 0 mean and standard deviation of 1 (on log scale)

```
rlnorm(5, meanlog=0, sdlog=1)
[1] 0.8725009 2.9433779 0.3329107 2.5976206 2.8171894
```

Weibull distribution with shape parameter of 0.5 and scale of 1

```
rweibull(5, shape=0.5, scale=1)
[1] 0.337599112 1.307774557 7.233985075 5.840429942 0.005751181
```

Wilcoxon distribution with 10 observations in the first sample and 20 in second.

```
rwilcox(5, 10, 20)
[1] 111 88 93 100 124
```

Multinomial distribution with 5 object and 3 boxes using the specified probabilities

```
rmultinom(5, 大小=5, 概率=c(0.1,0.1,0.8))
 [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    1    1    0
[2,]    2    0    1    1    0
[3,]    3    5    3    3    5
```

第37.3节：随机数生成器的可复现性

当期望别人复现包含随机元素的R代码时，`set.seed()`函数非常有用。例如，以下两行代码总是会产生不同的输出（因为这正是随机数生成器的意义所在）：

```
> sample(1:10, 5)
[1] 6 9 2 7 10
> sample(1:10, 5)
[1] 7 6 1 2 10
```

这两者也会产生不同的输出：

```
> rnorm(5)
[1] 0.4874291 0.7383247 0.5757814 -0.3053884 1.5117812
> rnorm(5)
[1] 0.38984324 -0.62124058 -2.21469989 1.12493092 -0.04493361
```

但是，如果我们在两种情况下都将种子设置为相同的值（大多数人为了简单起见使用1），我们会得到两个相同的样本：

```
> set.seed(1)
> sample(letters, 2)
[1] "g" "j"
> set.seed(1)
> sample(letters, 2)
[1] "g" "j"
```

同样，对于例如 `rexp()` 抽样也是如此：

```
> set.seed(1)
> rexp(5)
[1] 0.7551818 1.1816428 0.1457067 0.1397953 0.4360686
> set.seed(1)
> rexp(5)
[1] 0.7551818 1.1816428 0.1457067 0.1397953 0.4360686
```

```
rmultinom(5, size=5, prob=c(0.1,0.1,0.8))
 [,1] [,2] [,3] [,4] [,5]
[1,]    0    0    1    1    0
[2,]    2    0    1    1    0
[3,]    3    5    3    3    5
```

Section 37.3: Random number generator's reproducibility

When expecting someone to reproduce an R code that has random elements in it, the `set.seed()` function becomes very handy. For example, these two lines will always produce different output (because that is the whole point of random number generators):

```
> sample(1:10, 5)
[1] 6 9 2 7 10
> sample(1:10, 5)
[1] 7 6 1 2 10
```

These two will also produce different outputs:

```
> rnorm(5)
[1] 0.4874291 0.7383247 0.5757814 -0.3053884 1.5117812
> rnorm(5)
[1] 0.38984324 -0.62124058 -2.21469989 1.12493092 -0.04493361
```

However, if we set the seed to something identical in both cases (most people use 1 for simplicity), we get two identical samples:

```
> set.seed(1)
> sample(letters, 2)
[1] "g" "j"
> set.seed(1)
> sample(letters, 2)
[1] "g" "j"
```

and same with, say, `rexp()` draws:

```
> set.seed(1)
> rexp(5)
[1] 0.7551818 1.1816428 0.1457067 0.1397953 0.4360686
> set.seed(1)
> rexp(5)
[1] 0.7551818 1.1816428 0.1457067 0.1397953 0.4360686
```

第38章：并行处理

第38.1节：使用parallel包进行并行处理

基础包parallel允许通过分叉、套接字和随机数生成进行并行计算。

检测本地主机上的核心数量：

```
parallel::detectCores(all.tests = FALSE, logical = TRUE)
```

创建本地主机上的核心集群：

```
parallelCluster <- parallel::makeCluster(parallel::detectCores())
```

首先，必须创建一个适合并行化的函数。考虑mtcars数据集。通过为每个cyl级别创建单独的回归模型，可以改进对mpg的回归。

```
data <- mtcars
yfactor <- 'cyl'
zlevels <- sort(unique(data[[yfactor]]))
datay <- data[,1]
dataz <- data[,2]
datax <- data[,3:11]

fitmodel <- function(zlevel, datax, datay, dataz) {
  glm.fit(x = datax[dataz == zlevel], y = datay[dataz == zlevel])
}
```

创建一个函数，可以循环遍历所有可能的 zlevels 迭代。这仍然是串行的，但这是一个重要步骤，因为它决定了将被并行化的过程。

```
fitmodel <- function(zlevel, datax, datay, dataz) {
  glm.fit(x = datax[dataz == zlevel], y = datay[dataz == zlevel])
}

for (zlevel in zlevels) {
  print("*****")
  print(zlevel)
  print(fitmodel(zlevel, datax, datay, dataz))
}
```

柯里化此函数：

```
worker <- function(zlevel) {
  fitmodel(zlevel, datax, datay, dataz)
}
```

使用 parallel 进行并行计算时无法访问全局环境。幸运的是，每个函数都会创建一个 parallel 可以访问的局部环境。创建一个包装函数可以实现并行化。要应用的函数也需要放置在该环境中。

```
wrapper <- function(datax, datay, dataz) {
  # 强制求值所有未由并行应用提供的参数
  force(datax)
```

Chapter 38: Parallel processing

Section 38.1: Parallel processing with parallel package

The base package parallel allows parallel computation through forking, sockets, and random-number generation.

Detect the number of cores present on the localhost:

```
parallel::detectCores(all.tests = FALSE, logical = TRUE)
```

Create a cluster of the cores on the localhost:

```
parallelCluster <- parallel::makeCluster(parallel::detectCores())
```

First, a function appropriate for parallelization must be created. Consider the `mtcars` dataset. A regression on mpg could be improved by creating a separate regression model for each level of cyl.

```
data <- mtcars
yfactor <- 'cyl'
zlevels <- sort(unique(data[[yfactor]]))
datay <- data[,1]
dataz <- data[,2]
datax <- data[,3:11]
```

```
fitmodel <- function(zlevel, datax, datay, dataz) {
  glm.fit(x = datax[dataz == zlevel], y = datay[dataz == zlevel])
}
```

Create a function that can loop through all the possible iterations of zlevels. This is still in serial, but is an important step as it determines the exact process that will be parallelized.

```
fitmodel <- function(zlevel, datax, datay, dataz) {
  glm.fit(x = datax[dataz == zlevel], y = datay[dataz == zlevel])
}
```

```
for (zlevel in zlevels) {
  print("*****")
  print(zlevel)
  print(fitmodel(zlevel, datax, datay, dataz))
}
```

Curry this function:

```
worker <- function(zlevel) {
  fitmodel(zlevel, datax, datay, dataz)
}
```

Parallel computing using parallel cannot access the global environment. Luckily, each function creates a local environment parallel can access. Creation of a wrapper function allows for parallelization. The function to be applied also needs to be placed within the environment.

```
wrapper <- function(datax, datay, dataz) {
  # force evaluation of all parameters not supplied by parallelization apply
  force(datax)
```

```

force(datay)
force(dataz)
# 这些变量现在在一个并行函数可访问的环境中

# 函数也将在该环境中应用
fitmodel <- function(zlevel, datax, datay, dataz) {
  glm.fit(x = datax[dataz == zlevel,], y = datay[dataz == zlevel])
}

# 在此环境中调用，迭代单个参数 zlevel
worker <- function(zlevel) {
  fitmodel(zlevel, datax, datay, dataz)
}
return(worker)
}

```

现在创建一个集群并运行包装函数。

```

parallelcluster <- parallel::makeCluster(parallel::detectCores())
models <- parallel::parLapply(parallelcluster, zlevels,
  wrapper(datax, datay, dataz))

```

完成后务必停止集群。

```
parallel::stopCluster(parallelcluster)
```

parallel包包含整个apply()家族，前缀为par。

第38.2节：使用foreach包进行并行处理

foreach包为R带来了并行处理的能力。但在使用多核CPU之前，必须先分配一个多核集群。doSNOW包是其中一种选择。

foreach循环的一个简单用法是计算从1到所有数字的平方根与平方的和100000。

```

library(foreach)
library(doSNOW)

cl <- makeCluster(5, type = "SOCK")
registerDoSNOW(cl)

f <- foreach(i = 1:100000, .combine = c, .inorder = F) %dopar% {
  k <- i ** 2 + sqrt(i)
k
}

```

foreach的输出结构由.combine参数控制。默认的输出结构是一个列表。在上面的代码中，使用了c来返回一个向量。注意，计算函数（或运算符）如“+”也可以用来执行计算并返回进一步处理的对象。

需要指出的是，每个foreach循环的结果是最后一次调用的结果。因此，在此示例中，k将被添加到结果中。

参数

.combine combine函数。决定循环结果如何合并。可能的取值有c、cbind、rbind、"+"、"*"等

详细信息

```

force(datay)
force(dataz)
# these variables are now in an environment accessible by parallel function

# function to be applied also in the environment
fitmodel <- function(zlevel, datax, datay, dataz) {
  glm.fit(x = datax[dataz == zlevel,], y = datay[dataz == zlevel])
}

# calling in this environment iterating over single parameter zlevel
worker <- function(zlevel) {
  fitmodel(zlevel, datax, datay, dataz)
}
return(worker)
}

```

Now create a cluster and run the wrapper function.

```

parallelcluster <- parallel::makeCluster(parallel::detectCores())
models <- parallel::parLapply(parallelcluster, zlevels,
  wrapper(datax, datay, dataz))

```

Always stop the cluster when finished.

```
parallel::stopCluster(parallelcluster)
```

The parallel package includes the entire `apply()` family, prefixed with `par`.

Section 38.2: Parallel processing with foreach package

The foreach package brings the power of parallel processing to R. But before you want to use multi core CPUs you have to assign a multi core cluster. The doSNOW package is one possibility.

A simple use of the foreach loop is to calculate the sum of the square root and the square of all numbers from 1 to 100000.

```

library(foreach)
library(doSNOW)

cl <- makeCluster(5, type = "SOCK")
registerDoSNOW(cl)

f <- foreach(i = 1:100000, .combine = c, .inorder = F) %dopar% {
  k <- i ** 2 + sqrt(i)
k
}

```

The structure of the output of foreach is controlled by the .combine argument. The default output structure is a list. In the code above, c is used to return a vector instead. Note that a calculation function (or operator) such as "+" may also be used to perform a calculation and return a further processed object.

It is important to mention that the result of each foreach-loop is the last call. Thus, in this example k will be added to the result.

Parameter

.combine combine Function. Determines how the results of the loop are combined. Possible values are c, cbind, rbind, "+", "*..."

Details

.inorder

如果为TRUE，结果将按照迭代变量（此处为i）的顺序排列。如果为FALSE，结果不排序。这可能对计算时间有积极影响。

.packages 用于指定除base之外的任何包中提供的函数，例如mass、randomForest等，您必须为这些包提供c("mass", "randomForest")

第38.3节：随机数生成

并行化的一个主要问题是随机数生成器（RNG）作为种子的使用。随机数是通过从会话开始或最近一次set.seed()调用后的操作次数进行迭代生成的。由于并行进程来自同一函数，它们可能使用相同的种子，导致结果相同！调用将在不同核心上串行运行，无法带来优势。

必须生成一组种子并发送给每个并行进程。一些包（如parallel、snow等）会自动完成此操作，但在其他包中必须显式处理。

```
s <- seed
for (i in 1:numofcores) {
  s <- nextRNGStream(s)
  # 将 s 作为 .Random.seed 发送给工作进程 i
}
```

种子也可以设置以保证可重复性。

```
clusterSetRNGStream(cl = parallelcluster, iseed)
```

第38.4节：mcparallelDo

mcparallelDo包允许在类Unix（如Linux和MacOSX）操作系统上异步执行R代码。该包的基本理念更符合探索性数据分析的需求，而非编码。若需编码异步操作，可考虑使用future包。

示例

创建数据

```
data(ToothGrowth)
```

触发 mcparallelDo 以在分叉上执行分析

```
mcparallelDo({glm(len ~ supp * dose, data=ToothGrowth)}, "interactionPredictorModel")
```

执行其他操作，例如

```
binaryPredictorModel <- glm(len ~ supp, data=ToothGrowth)
gaussianPredictorModel <- glm(len ~ dose, data=ToothGrowth)
```

mcparallelDo 的结果在完成时会返回到你的目标环境，例如 .GlobalEnv，并带有一条消息（默认情况下）

```
summary(interactionPredictorModel)
```

其他示例

```
# 示例：直到返回到顶层才返回值
for (i in 1:10) {
  if (i == 1) {
```

.inorder

if TRUE the result is ordered according to the order of the iteration variable (here i). If FALSE the result is not ordered. This can have positive effects on computation time.

.packages

for functions which are provided by any package except base, like e.g. mass, randomForest or else, you have to provide these packages with c("mass", "randomForest")

Section 38.3: Random Number Generation

A major problem with parallelization is the use of RNG as seeds. Random numbers by the number are iterated by the number of operations from either the start of the session or the most recent set.seed(). Since parallel processes arise from the same function, it can use the same seed, possibly causing identical results! Calls will run in serial on the different cores, provide no advantage.

A set of seeds must be generated and sent to each parallel process. This is automatically done in some packages (parallel, snow, etc.), but must be explicitly addressed in others.

```
s <- seed
for (i in 1:numofcores) {
  s <- nextRNGStream(s)
  # send s to worker i as .Random.seed
}
```

Seeds can also be set for reproducibility.

```
clusterSetRNGStream(cl = parallelcluster, iseed)
```

Section 38.4: mcparallelDo

The mcparallelDo package allows for the evaluation of R code asynchronously on Unix-alike (e.g. Linux and MacOSX) operating systems. The underlying philosophy of the package is aligned with the needs of exploratory data analysis rather than coding. For coding asynchrony, consider the future package.

Example

Create data

```
data(ToothGrowth)
```

Trigger mcparallelDo to perform analysis on a fork

```
mcparallelDo({glm(len ~ supp * dose, data=ToothGrowth)}, "interactionPredictorModel")
```

Do other things, e.g.

```
binaryPredictorModel <- glm(len ~ supp, data=ToothGrowth)
gaussianPredictorModel <- glm(len ~ dose, data=ToothGrowth)
```

The result from mcparallelDo returns in your targetEnvironment, e.g. .GlobalEnv, when it is complete with a message (by default)

```
summary(interactionPredictorModel)
```

Other Examples

```
# Example of not returning a value until we return to the top level
for (i in 1:10) {
  if (i == 1) {
```

```
mcparallelDo({2+2}, targetValue = "output")
}
if (exists("output")) print(i)
}

# 获取值而不返回顶层的示例
for (i in 1:10) {
  if (i == 1) {
    mcparallelDo({2+2}, targetValue = "output")
  }
  mcparallelDoCheck()
  if (exists("output")) print(i)
}
```

```
mcparallelDo({2+2}, targetValue = "output")
}
if (exists("output")) print(i)
}

# Example of getting a value without returning to the top level
for (i in 1:10) {
  if (i == 1) {
    mcparallelDo({2+2}, targetValue = "output")
  }
  mcparallelDoCheck()
  if (exists("output")) print(i)
}
```

第39章：子集选择

给定一个R对象，我们可能需要对其中包含的一个或多个部分数据进行单独分析。从给定对象中获取这些数据部分的过程称为子集选择。

第39.1节：数据框

将数据框子集化为更小的数据框可以像子集化列表一样完成。

```
> df3 <- data.frame(x = 1:3, y = c("a", "b", "c"), stringsAsFactors = FALSE)

> df3
##   x y
## 1 1 a
## 2 2 b
## 3 3 c

> df3[1] # 按数字子集变量
##   x
## 1 1
## 2 2
## 3 3

> df3["x"] # 按名称子集变量
##   x
## 1 1
## 2 2
## 3 3

> is.data.frame(df3[1])
## TRUE

> is.list(df3[1])
## TRUE
```

将数据框子集为列向量可以使用双括号[[]]或美元符号

操作符\$来完成。

```
> df3[[2]] # 使用[[ ]]按数字子集变量
## [1] "a" "b" "c"

> df3[["y"]] # 使用[[ ]]按名称子集变量
## [1] "a" "b" "c"

> df3$x # 使用$按名称子集变量
## [1] 1 2 3

> typeof(df3$x)
## "integer"

> is.vector(df3$x)
## TRUE
```

将数据作为二维矩阵子集可以使用 i 和 j 术语来完成。

```
> df3[1, 2] # 按数字子集行和列
## [1] "a"
```

Chapter 39: Subsetting

Given an R object, we may require separate analysis for one or more parts of the data contained in it. The process of obtaining these parts of the data from a given object is called subsetting.

Section 39.1: Data frames

Subsetting a data frame into a smaller data frame can be accomplished the same as subsetting a list.

```
> df3 <- data.frame(x = 1:3, y = c("a", "b", "c"), stringsAsFactors = FALSE)

> df3
##   x y
## 1 1 a
## 2 2 b
## 3 3 c

> df3[1] # Subset a variable by number
##   x
## 1 1
## 2 2
## 3 3

> df3[ "x" ] # Subset a variable by name
##   x
## 1 1
## 2 2
## 3 3

> is.data.frame(df3[1])
## TRUE

> is.list(df3[1])
## TRUE
```

Subsetting a dataframe into a column vector can be accomplished using double brackets [[]] or the dollar sign operator \$.

```
> df3[[2]] # Subset a variable by number using [[ ]]
## [1] "a" "b" "c"

> df3[["y"]] # Subset a variable by name using [[ ]]
## [1] "a" "b" "c"

> df3$x # Subset a variable by name using $
## [1] 1 2 3

> typeof(df3$x)
## "integer"

> is.vector(df3$x)
## TRUE
```

Subsetting a data as a two dimensional matrix can be accomplished using i and j terms.

```
> df3[1, 2] # Subset row and column by number
## [1] "a"
```

```

> df3[1, "y"] # 按数字子集行, 按名称子集列
## [1] "a"

> df3[2, ]    # 按数字子集整行
##   x y
## 2 2 b

> df3[, 1]    # 子集所有第一个变量
## [1] 1 2 3

> df3[, 1, drop = FALSE]
##   x
## 1 1
## 2 2
## 3 3

```

注意：仅按j（列）子集时会简化为变量自身的类型，但仅按i子集时会返回一个 **data.frame**，因为不同变量可能有不同的类型和类。将 **drop**参数设置为FALSE可以保持数据框。

```

> is.vector(df3[, 2])
## TRUE

> is.data.frame(df3[2, ])
## TRUE

> is.data.frame(df3[, 2, drop = FALSE])
## TRUE

```

第39.2节：原子向量

原子向量（不包括列表和表达式，这些也是向量）使用[操作符进行子集操作：

```

# 创建一个示例向量
v1 <- c("a", "b", "c", "d")

# 选择第三个元素
v1[3]
## [1] "c"

```

[操作符也可以接受向量作为参数。例如，选择第一个和第三个元素：

```

v1 <- c("a", "b", "c", "d")

v1[c(1, 3)]
## [1] "a" "c"

```

有时我们可能需要从向量中省略某个特定值。这可以通过在该值的索引前使用负号 (-) 来实现。例如，要从 v1 中省略第一个值，使用 v1[-1]。这个方法可以很容易地扩展到多个值。例如，v1[-c(1,3)]。

```

> v1[-1]
[1] "b" "c" "d"
> v1[-c(1,3)]
[1] "b" "d"

```

在某些情况下，尤其是当向量长度较大时，我们希望知道特定值的索引（如果存在）：

```

> df3[1, "y"] # Subset row by number and column by name
## [1] "a"

> df3[2, ]    # Subset entire row by number
##   x y
## 2 2 b

> df3[, 1]    # Subset all first variables
## [1] 1 2 3

> df3[, 1, drop = FALSE]
##   x
## 1 1
## 2 2
## 3 3

```

Note: Subsetting by j (column) alone simplifies to the variable's own type, but subsetting by i alone returns a **data.frame**, as the different variables may have different types and classes. Setting the **drop** parameter to FALSE keeps the data frame.

```

> is.vector(df3[, 2])
## TRUE

> is.data.frame(df3[2, ])
## TRUE

> is.data.frame(df3[, 2, drop = FALSE])
## TRUE

```

Section 39.2: Atomic vectors

Atomic vectors (which excludes lists and expressions, which are also vectors) are subset using the [operator:

```

# create an example vector
v1 <- c("a", "b", "c", "d")

# select the third element
v1[3]
## [1] "c"

```

The [operator can also take a vector as the argument. For example, to select the first and third elements:

```

v1 <- c("a", "b", "c", "d")

v1[c(1, 3)]
## [1] "a" "c"

```

Some times we may require to omit a particular value from the vector. This can be achieved using a negative sign(-) before the index of that value. For example, to omit the first value from v1, use v1[-1]. This can be extended to more than one value in a straight forward way. For example, v1[-c(1,3)].

```

> v1[-1]
[1] "b" "c" "d"
> v1[-c(1,3)]
[1] "b" "d"

```

On some occasions, we would like to know, especially, when the length of the vector is large, index of a particular

值：

```
> v1=="c"
[1] FALSE FALSE TRUE FALSE
> which(v1=="c")
[1] 3
```

如果原子向量有名称（一个 `names` 属性），则可以使用名称的字符向量进行子集选择：

```
v <- 1:3
names(v) <- c("one", "two", "three")

v
## one two three
## 1   2   3

v["two"]
## two
## 2
```

`[[` 操作符也可以用于索引原子向量，区别在于它接受长度为一的索引向量，并且会去除任何存在的名称：

```
v[[c(1, 2)]]
## Error in v[[c(1, 2)]] :
## attempt to select more than one element in vectorIndex

v[["two"]]
## [1] 2
```

向量也可以使用逻辑向量进行子集化。与使用数值和字符向量进行子集化不同，用于子集化的逻辑向量必须与被提取元素的向量长度相等，因此如果使用逻辑向量 `y` 对子集化 `x`，即 `x[y]`，若 `length(y) < length(x)`，则 `y` 会被回收以匹配 `length(x)`：

```
v[c(TRUE, FALSE, TRUE)]
## 一三
## 1   3

v[c(FALSE, TRUE)] # 循环使用为 'c(FALSE, TRUE, FALSE)'
## 二
## 2

v[TRUE] # 循环使用为 'c(TRUE, TRUE, TRUE)'
## 一 二三
## 1   2   3

v[FALSE] # 方便丢弃元素但保留向量的类型和基本结构
## 命名的 integer(0)
```

第39.3节：矩阵

对于对象的每个维度，`[` 操作符接受一个参数。向量有一个维度，接受一个参数。矩阵和数据框有两个维度，接受两个参数，表示为 `[i, j]`，其中 `i` 是行，`j` 是列。索引从 1 开始。

```
## 一个示例矩阵
mat <- 矩阵(1:6, 行数 = 2, 维度名称 = 列表(列("row1", "row2"), 列("col1", "col2", "col3")))
```

value, if it exists:

```
> v1=="c"
[1] FALSE FALSE TRUE FALSE
> which(v1=="c")
[1] 3
```

If the atomic vector has names (a `names` attribute), it can be subset using a character vector of names:

```
v <- 1:3
names(v) <- c("one", "two", "three")

v
## one two three
## 1   2   3

v["two"]
## two
## 2
```

The `[[` operator can also be used to index atomic vectors, with differences in that it accepts a indexing vector with a length of one and strips any names present:

```
v[[c(1, 2)]]
## Error in v[[c(1, 2)]] :
## attempt to select more than one element in vectorIndex

v[["two"]]
## [1] 2
```

Vectors can also be subset using a logical vector. In contrast to subsetting with numeric and character vectors, the logical vector used to subset has to be equal to the length of the vector whose elements are extracted, so if a logical vector `y` is used to subset `x`, i.e. `x[y]`, if `length(y) < length(x)` then `y` will be recycled to match `length(x)`:

```
v[c(TRUE, FALSE, TRUE)]
## one three
## 1   3

v[c(FALSE, TRUE)] # recycled to 'c(FALSE, TRUE, FALSE)'
## two
## 2

v[TRUE] # recycled to 'c(TRUE, TRUE, TRUE)'
## one two three
## 1   2   3

v[FALSE] # handy to discard elements but save the vector's type and basic structure
## named integer(0)
```

Section 39.3: Matrices

For each dimension of an object, the `[` operator takes one argument. Vectors have one dimension and take one argument. Matrices and data frames have two dimensions and take two arguments, given as `[i, j]` where `i` is the row and `j` is the column. Indexing starts at 1.

```
## a sample matrix
mat <- matrix(1:6, nrow = 2, dimnames = list(c("row1", "row2"), c("col1", "col2", "col3")))
```

```
mat
#      col1 col2 col3
# row1  1    3    5
# row2  2    4    6
```

mat[i,j] 是矩阵 mat 中第 i 行、第 j 列的元素。例如，i 值为 2, j 值为 1 表示矩阵第二行第一列的数字。省略 i 或 j 会返回该维度的所有值。

```
mat[ , 3]
## row1 row2
## 5   6

mat[1, ]
# col1 col2 col3
# 1    3    5
```

当矩阵有行名或列名（非必需）时，可以使用它们进行子集选择：

```
mat[ , 'col1']
# row1 row2
# 1    2
```

默认情况下，子集的结果将在可能的情况下被简化。如果子集只有一个维度，如上面的示例中，结果将是一个一维向量，而不是二维矩阵。此默认行为可以通过将 drop = FALSE 参数传递给 [来覆盖。

```
## 这将第一行选为一个向量
class(mat[1, ])
# [1] "integer"

## 而这将第一行选为一个1x3矩阵：
class(mat[1, , drop = F])
# [1] "matrix"
```

当然，如果所选部分本身具有两个维度，则不能丢弃维度：

```
mat[1:2, 2:3] ## 一个2x2矩阵
#      col2 col3
# row1  3    5
# row2  4    6
```

按位置选择矩阵中的单个元素

也可以使用一个Nx2矩阵从一个矩阵中选择N个单独的元素（类似于坐标系的工作方式）。如果你想提取一个向量，包含矩阵中(第1行, 第1列)、(第1行, 第3列)、(第2行, 第3列)、(第2行, 第1列)的元素，这可以通过创建一个包含这些坐标的索引矩阵并用它来子集矩阵轻松实现：

```
mat
#      col1 col2 col3
# row1  1    3    5
# row2  2    4    6

ind = rbind(c(1, 1), c(1, 3), c(2, 3), c(2, 1))
ind
#      [,1] [,2]
```

```
mat
#      col1 col2 col3
# row1  1    3    5
# row2  2    4    6
```

mat[i,j] is the element in the i-th row, j-th column of the matrix mat. For example, an i value of 2 and a j value of 1 gives the number in the second row and the first column of the matrix. Omitting i or j returns all values in that dimension.

```
mat[ , 3]
## row1 row2
## 5   6

mat[1, ]
# col1 col2 col3
# 1    3    5
```

When the matrix has row or column names (not required), these can be used for subsetting:

```
mat[ , 'col1']
# row1 row2
# 1    2
```

By default, the result of a subset will be simplified if possible. If the subset only has one dimension, as in the examples above, the result will be a one-dimensional vector rather than a two-dimensional matrix. This default can be overridden with the drop = FALSE argument to [:

```
## This selects the first row as a vector
class(mat[1, ])
# [1] "integer"

## Whereas this selects the first row as a 1x3 matrix:
class(mat[1, , drop = F])
# [1] "matrix"
```

Of course, dimensions cannot be dropped if the selection itself has two dimensions:

```
mat[1:2, 2:3] ## A 2x2 matrix
#      col2 col3
# row1  3    5
# row2  4    6
```

Selecting individual matrix entries by their positions

It is also possible to use a Nx2 matrix to select N individual elements from a matrix (like how a coordinate system works). If you wanted to extract, in a vector, the entries of a matrix in the (1st row, 1st column), (1st row, 3rd column), (2nd row, 3rd column), (2nd row, 1st column) this can be done easily by creating a index matrix with those coordinates and using that to subset the matrix:

```
mat
#      col1 col2 col3
# row1  1    3    5
# row2  2    4    6

ind = rbind(c(1, 1), c(1, 3), c(2, 3), c(2, 1))
ind
#      [,1] [,2]
```

```
# [1] 1 1
# [2] 1 3
# [3] 2 3
# [4] 2 1

mat[ind]
# [1] 1 5 6 2
```

在上述示例中，`ind` 矩阵的第1列指的是 `mat` 中的行，`ind` 的第2列指的是 `mat` 中的列。

第39.4节：列表

列表可以用 [进行子集操作：

```
l1 <- list(c(1, 2, 3), 'two' = c("a", "b", "c"), list(10, 20))
l1
## [[1]]
## [1] 1 2 3
##
## $two
## [1] "a" "b" "c"
##
## [[3]]
## [[3]][[1]]
## [1] 10
##
## [[3]][[2]]
## [1] 20

l1[1]
## [[1]]
## [1] 1 2 3

l1['two']
## $two
## [1] "a" "b" "c"

l1[[2]]
## [1] "a" "b" "c"

l1[['two']]
## [1] "a" "b" "c"
```

注意，`l1[2]` 的结果仍然是一个列表，因为 [操作符选择列表的元素，返回一个更小的列表。 [[操作符提取列表元素，返回该元素类型的对象。

元素可以通过数字或名称的字符串（如果存在）进行索引。可以通过传递数字或名称字符串的向量，使用 [选择多个元素。

使用向量长度 > 1 的 [和 [[索引时，分别返回带有指定元素和递归子集（如果有）的“列表”：

```
l1[c(3, 1)]
## [[1]]
## [[1]][[1]]
## [1] 10
##
## [[1]][[2]]
## [1] 20
##
```

```
# [1] 1 1
# [2] 1 3
# [3] 2 3
# [4] 2 1

mat[ind]
# [1] 1 5 6 2
```

In the above example, the 1st column of the `ind` matrix refers to rows in `mat`, the 2nd column of `ind` refers to columns in `mat`.

Section 39.4: Lists

A list can be subset with [:

```
l1 <- list(c(1, 2, 3), 'two' = c("a", "b", "c"), list(10, 20))
l1
## [[1]]
## [1] 1 2 3
##
## $two
## [1] "a" "b" "c"
##
## [[3]]
## [[3]][[1]]
## [1] 10
##
## [[3]][[2]]
## [1] 20

l1[1]
## [[1]]
## [1] 1 2 3

l1['two']
## $two
## [1] "a" "b" "c"

l1[[2]]
## [1] "a" "b" "c"

l1[['two']]
## [1] "a" "b" "c"
```

Note the result of `l1[2]` is still a list, as the [operator selects elements of a list, returning a smaller list. The [[operator extracts list elements, returning an object of the type of the list element.

Elements can be indexed by number or a character string of the name (if it exists). Multiple elements can be selected with [by passing a vector of numbers or strings of names. Indexing with a vector of `length > 1` in [and [[returns a "list" with the specified elements and a recursive subset (if available), respectively:

```
l1[c(3, 1)]
## [[1]]
## [[1]][[1]]
## [1] 10
##
## [[1]][[2]]
## [1] 20
##
```

```
##  
## [[2]]  
## [1] 1 2 3
```

相比之下：

```
l1[[c(3, 1)]]  
## [1] 10
```

等同于：

```
l1[[3]][[1]]  
## [1] 10
```

\$操作符允许你仅通过名称选择列表元素，但与[和[[不同，它不需要引号。作为一个中缀操作符，\$只能接受单个名称：

```
l1$two  
## [1] "a" "b" "c"
```

此外，\$ 操作符默认允许部分匹配：

```
l1$t  
## [1] "a" "b" "c"
```

与 [[不同，后者需要指定是否允许部分匹配：

```
l1[["t"]]  
## NULL  
l1[["t", exact = FALSE]]  
## [1] "a" "b" "c"
```

设置 options(warnPartialMatchDollar = TRUE) 时，当使用 \$ 发生部分匹配会给出“警告”：

```
l1$t  
## [1] "a" "b" "c"  
## 警告信息：  
## 在 l1$t 中 : 't' 部分匹配到 'two'
```

第39.5节：向量索引

在此示例中，我们将使用向量：

```
> x <- 11:20  
> x  
[1] 11 12 13 14 15 16 17 18 19 20
```

R 向量是从 1 开始索引的，例如 x[1] 会返回 11。我们也可以通过传递一个索引向量给括号操作符来提取 x 的子向量：

```
> x[c(2,4,6)]  
[1] 12 14 16
```

如果传递一个负索引向量，R 会返回一个排除指定索引的子向量：

```
##  
## [[2]]  
## [1] 1 2 3
```

Compared to:

```
l1[[c(3, 1)]]  
## [1] 10
```

which is equivalent to:

```
l1[[3]][[1]]  
## [1] 10
```

The \$ operator allows you to select list elements solely by name, but unlike [and [[, does not require quotes. As an infix operator, \$ can only take a single name:

```
l1$two  
## [1] "a" "b" "c"
```

Also, the \$ operator allows for partial matching by default:

```
l1$t  
## [1] "a" "b" "c"
```

in contrast with [[where it needs to be specified whether partial matching is allowed:

```
l1[["t"]]  
## NULL  
l1[["t", exact = FALSE]]  
## [1] "a" "b" "c"
```

Setting `options(warnPartialMatchDollar = TRUE)`, a "warning" is given when partial matching happens with \$:

```
l1$t  
## [1] "a" "b" "c"  
## Warning message:  
## In l1$t : partial match of 't' to 'two'
```

Section 39.5: Vector indexing

For this example, we will use the vector:

```
> x <- 11:20  
> x  
[1] 11 12 13 14 15 16 17 18 19 20
```

R vectors are 1-indexed, so for example x[1] will return 11. We can also extract a sub-vector of x by passing a vector of indices to the bracket operator:

```
> x[c(2,4,6)]  
[1] 12 14 16
```

If we pass a vector of negative indices, R will return a sub-vector with the specified indices excluded:

```
> x[c(-1,-3)]  
[1] 12 14 15 16 17 18 19 20
```

我们也可以传递一个布尔向量给括号操作符，这种情况下它会返回对应于索引向量中为 TRUE 的坐标的子向量：

```
> x[c(rep(TRUE, 5), rep(FALSE, 5))]  
[1] 11 12 13 14 15 16
```

如果索引向量比数组长度短，则会重复使用该索引向量，如下所示：

```
> x[c(TRUE, FALSE)]  
[1] 11 13 15 17 19  
> x[c(TRUE, FALSE, FALSE)]  
[1] 11 14 17 20
```

第39.6节：其他对象

[和[[操作符是通用的原始函数。这意味着R中的任何对象（具体来说 `isTRUE(is.object(x))`——即具有显式“class”属性）在子集化时都可以有其特定的行为；即拥有自己针对[和/或[[的方法。

例如，“data.frame”对象 (`is.object(iris)`) 就是这种情况，其中定义了`[.data.frame`和`[[.data.frame`方法，使其表现出既像“矩阵”又像“列表”的子集化行为。当强制在子集化“data.frame”时出错，我们会看到实际上调用了函数`[.data.frame`，即使我们只是使用了[。

```
iris[invalidArgument, ]  
## 在`[.data.frame`(iris, invalidArgument, )中出错：  
## 找不到对象'invalidArgument'
```

在没有更多当前主题细节的情况下，举一个[方法的例子：

```
x = structure(1:5, class = "myClass")  
x[c(3, 2, 4)]  
## [1] 3 2 4  
'[.myClass' = function(x, i) cat(sprintf("我们期望返回'%s[%s]'，但这是一个自定义的`[`方法，应有一个`? [.myClass`帮助页面说明其行为", deparse(substitute(x)),  
deparse(substitute(i)))  
  
x[c(3, 2, 4)]  
## 我们期望返回 `x[c(3, 2, 4)]`，但这是一个自定义的 `[` 方法，应该有一个 `? [.myClass` 的帮助页面说明其行为  
## NULL
```

我们可以通过使用等效的非泛型函数 `.subset`（以及 `.subset2` 用于 `[[`）来克服 [的方法分派。

当我们编写自己的“类”并希望避免使用变通方法（例如 `unclass(x)`）时，这尤其有用且高效，可以在对我们的“类”进行计算时避免方法分派和对象复制：

```
.subset(x, c(3, 2, 4))  
## [1] 3 2 4
```

第39.7节：矩阵的元素级操作

设 A 和 B 是两个维度相同的矩阵。运算符 +、-、/、*、^ 在用于维度相同的矩阵时，会对矩阵对应元素执行相应操作，并返回一个新的

```
> x[c(-1, -3)]  
[1] 12 14 15 16 17 18 19 20
```

We can also pass a boolean vector to the bracket operator, in which case it returns a sub-vector corresponding to the coordinates where the indexing vector is TRUE:

```
> x[c(rep(TRUE, 5), rep(FALSE, 5))]  
[1] 11 12 13 14 15 16
```

If the indexing vector is shorter than the length of the array, then it will be repeated, as in:

```
> x[c(TRUE, FALSE)]  
[1] 11 13 15 17 19  
> x[c(TRUE, FALSE, FALSE)]  
[1] 11 14 17 20
```

Section 39.6: Other objects

The [and [[operators are primitive functions that are generic. This means that any *object* in R (specifically `isTRUE(is.object(x))` --i.e. has an explicit "class" attribute) can have its own specified behaviour when subsetted; i.e. has its own *methods* for [and/or [[.

For example, this is the case with "data.frame" (`is.object(iris)`) objects where `[.data.frame` and `[[.data.frame` methods are defined and they are made to exhibit both "matrix"-like and "list"-like subsetting. With forcing an error when subsetting a "data.frame", we see that, actually, a function `[.data.frame` was called when we just used [.

```
iris[invalidArgument, ]  
## Error in `[.data.frame`(iris, invalidArgument, ) :  
##   object 'invalidArgument' not found
```

Without further details on the current topic, an example[method:

```
x = structure(1:5, class = "myClass")  
x[c(3, 2, 4)]  
## [1] 3 2 4  
'[.myClass' = function(x, i) cat(sprintf("We'd expect '%s[%s]' to be returned but this a custom `['` method and should have a `? [.myClass` help page for its behaviour\n", deparse(substitute(x)),  
deparse(substitute(i)))  
  
x[c(3, 2, 4)]  
## We'd expect `x[c(3, 2, 4)]` to be returned but this a custom `['` method and should have a `? [.myClass` help page for its behaviour  
## NULL
```

We can overcome the method dispatching of [by using the equivalent non-generic `.subset` (and `.subset2` for [[). This is especially useful and efficient when programming our own "class"es and want to avoid work-arounds (like `unclass(x)`) when computing on our "class"es efficiently (avoiding method dispatch and copying objects):

```
.subset(x, c(3, 2, 4))  
## [1] 3 2 4
```

Section 39.7: Elementwise Matrix Operations

Let A and B be two matrices of same dimension. The operators +, -, /, *, ^ when used with matrices of same dimension perform the required operations on the corresponding elements of the matrices and return a new

同维度矩阵。这些操作通常称为元素级操作。

操作符 A op B	含义
+	A + B 对应元素相加
-	A - B 将 B 的元素从 A 的对应元素中减去
/	A / B 将 A 的元素除以 B 的对应元素
*	A * B 将 A 的元素与 B 的对应元素相乘
^	A^(−1) 例如，给出一个元素为 A 的倒数的矩阵

对于“真正的”矩阵乘法，如线性代数中所见，使用%*%。例如，A 与 B 的乘法为：A %*% B。维度要求是A的ncol() 与B的nrow() 相同

一些用于矩阵的函数

函数	示例	用途
nrow()	nrow(A)	确定矩阵A的行数
ncol()	ncol(A)	确定矩阵A的列数rownames() rowname
s(A)	打印矩阵A的行名colnames() colnames(A)	打印矩阵A的列名row
Means()	rowMeans(A)	计算矩阵A每一行的均值colMeans() colMeans(A)
upper.tri()	upper.tri(A)	计算矩阵A每一列的均值upper.tri() upper.tri(A) 返回一个向量，其元素是方阵A的上三角矩阵
lower.tri()	lower.tri(A)	返回一个向量，其元素是方阵A的下三角矩阵
det()	det(A)	导致矩阵A的行列式
solve()	solve(A)	返回非奇异矩阵A的逆矩阵
diag()	diag(A)	返回一个对角矩阵，其非对角元素为零，对角元素与方阵A相同
t()	t(A)	返回矩阵A的转置矩阵
eigen()	eigen(A)	返回矩阵A的特征值和特征向量
is.matrix()	is.matrix(A)	根据A是否为矩阵返回TRUE或FALSE。
as.matrix()	as.matrix(x)	将向量x转换为矩阵

matrix of the same dimension. These operations are usually referred to as element-wise operations.

Operator A op B	Meaning
+	A + B Addition of corresponding elements of A and B
-	A - B Subtracts the elements of B from the corresponding elements of A
/	A / B Divides the elements of A by the corresponding elements of B
*	A * B Multiplies the elements of A by the corresponding elements of B
^	A^(−1) For example, gives a matrix whose elements are reciprocals of A

For "true" matrix multiplication, as seen in *Linear Algebra*, use %*%. For example, multiplication of A with B is: A %*% B. The dimensional requirements are that the `ncol()` of A be the same as `nrow()` of B

Some Functions used with Matrices

Function	Example	Purpose
nrow()	nrow(A)	determines the number of rows of A
ncol()	ncol(A)	determines the number of columns of A
rownames()	rownames(A)	prints out the row names of the matrix A
colnames()	colnames(A)	prints out the column names of the matrix A
rowMeans()	rowMeans(A)	computes means of each row of the matrix A
colMeans()	colMeans(A)	computes means of each column of the matrix A
upper.tri()	upper.tri(A)	returns a vector whose elements are the upper triangular matrix of square matrix A
lower.tri()	lower.tri(A)	returns a vector whose elements are the lower triangular matrix of square matrix A
det()	det(A)	results in the determinant of the matrix A
solve()	solve(A)	results in the inverse of the non-singular matrix A
diag()	diag(A)	returns a diagonal matrix whose off-diagonal elements are zeros and diagonals are the same as that of the square matrix A
t()	t(A)	returns the transpose of the matrix A
eigen()	eigen(A)	returns the eigenvalues and eigenvectors of the matrix A
is.matrix()	is.matrix(A)	returns TRUE or FALSE depending on whether A is a matrix or not.
as.matrix()	as.matrix(x)	creates a matrix out of the vector x

第40章：调试

第40.1节：使用debug

你可以使用debug为任何函数设置调试。

```
debug(mean)  
mean(1:3)
```

之后对该函数的所有调用都会进入调试模式。你可以使用undebbug来禁用此行为。

```
undebbug(mean)  
mean(1:3)
```

如果你只想进入函数的调试模式一次，可以考虑使用debugonce。

```
debugonce(mean)  
mean(1:3)  
mean(1:3)
```

第40.2节：使用browser

浏览器功能可以像断点一样使用：代码执行将在调用该功能的点暂停。然后用户可以检查变量值，执行任意R代码，并逐行单步执行代码。

一旦代码中触发browser()，交互式解释器将启动。任何R代码都可以照常运行，此外还提供以下命令，

命令	含义
c	退出浏览器并继续程序
f	完成当前循环或函数
n	跳过（执行下一条语句，跳过函数调用）
s	单步进入（执行下一条语句，进入函数调用）
哪里	打印堆栈跟踪
r	调用“resume”重启
Q	退出浏览器并退出

例如，我们可能有如下脚本，

```
toDebug <- function() {  
  a = 1  
  b = 2  
  
  browser()  
  
  for(i in 1:100) {  
    a = a * b  
  }  
}  
  
toDebug()
```

运行上述脚本时，我们最初会看到类似如下内容，

Chapter 40: Debugging

Section 40.1: Using debug

You can set any function for debugging with **debug**.

```
debug(mean)  
mean(1:3)
```

All subsequent calls to the function will enter debugging mode. You can disable this behavior with **undebbug**.

```
undebbug(mean)  
mean(1:3)
```

If you know you only want to enter the debugging mode of a function once, consider the use of **debugonce**.

```
debugonce(mean)  
mean(1:3)  
mean(1:3)
```

Section 40.2: Using browser

The **browser** function can be used like a breakpoint: code execution will pause at the point it is called. Then user can then inspect variable values, execute arbitrary R code and step through the code line by line.

Once **browser()** is hit in the code the interactive interpreter will start. Any R code can be run as normal, and in addition the following commands are present,

Command	Meaning
c	Exit browser and continue program
f	Finish current loop or function \
n	Step Over (evaluate next statement, stepping over function calls)
s	Step Into (evaluate next statement, stepping into function calls)
where	Print stack trace
r	Invoke "resume" restart
Q	Exit browser and quit

For example we might have a script like,

```
toDebug <- function() {  
  a = 1  
  b = 2  
  
  browser()  
  
  for(i in 1:100) {  
    a = a * b  
  }  
}  
  
toDebug()
```

When running the above script we initially see something like,

调用自: toDebug
浏览器[1]>

然后我们可以这样与提示交互,

```
调用自: toDebug
浏览器[1]> a
[1] 1
浏览器[1]> b
[1] 2
浏览器[1]> n
调试 在 #7: for (i in 1:100) {
  a = a * b
}
浏览器[2]> n
调试 在 #8: a = a * b
浏览器[2]> a
[1] 1
浏览器[2]> n
调试 在 #8: a = a * b
浏览器[2]> a
[1] 2
浏览器[2]> Q
```

browser() 也可以作为函数链的一部分使用, 示例如下:

```
mtcars %>% 按(cyl) %>% {浏览器()}
```

Called from: toDebug
Browser[1]>

We could then interact with the prompt as so,

```
Called from: toDebug
Browser[1]> a
[1] 1
Browser[1]> b
[1] 2
Browse[1]> n
debug at #7: for (i in 1:100) {
  a = a * b
}
Browse[2]> n
debug at #8: a = a * b
Browse[2]> a
[1] 1
Browse[2]> n
debug at #8: a = a * b
Browse[2]> a
[1] 2
Browse[2]> Q
```

browser() can also be used as part of a functional chain, like so:

```
mtcars %>% group_by(cyl) %>% {browser()}
```

第41章：安装软件包

参数详情

pkgs	软件包名称的字符向量。如果 <code>repos = NULL</code> , 则为文件路径的字符向量。
lib	表示安装软件包的库目录的字符向量。
repos	字符向量，使用的仓库基础URL，可以为NULL以从本地文件安装
method	下载方法
destdir	存放已下载包的目录
	到/导入/建议（递归值，指示是否同时安装这些包所依赖/链接的未安装包） 如果 <code>repos=NULL</code> 则不使用。
... <small>Windows二进制安装函数的参数。</small>	传递给'download.file'或用于OS X和 Windows。

第41.1节：从GitHub安装包

要直接从GitHub安装包，请使用devtools包：

```
library(devtools)
install_github("authorName/repositoryName")
```

要从github安装ggplot2：

```
devtools::install_github("tidyverse/ggplot2")
```

上述命令将安装对应于master分支的ggplot2版本。要从仓库的不同分支安装，请使用ref参数提供分支名称。例如，以下命令将安装googleway包的dev_general分支。

```
devtools::install_github("SymbolixAU/googleway", ref = "dev_general")
```

另一种选择是使用ghit包。它提供了一个轻量级的从github安装包的替代方案：

```
install.packages("ghit")
ghit::install_github("google/CausalImpact")
```

要安装位于Github上的private仓库中的包，请在

<http://www.github.com/settings/tokens/>生成一个personal access token（有关详细文档，请参见?install_github）。请按照以下步骤操作：

1. `install.packages(c("curl", "httr"))`
2. `config = httr::config(ssl_verifypeer = FALSE)`
3. `install.packages("RCurl")`
`options(RCurlOptions = c(getOption("RCurlOptions"), ssl.verifypeer = FALSE, ssl.verifyhost = FALSE))`
4. `getOption("RCurlOptions")`

你应该看到以下内容：

```
ssl.verifypeer ssl.verifyhost
FALSE      FALSE
5. library(httr)
```

Chapter 41: Installing packages

Parameter Details

pkgs	character vector of the names of packages. If <code>repos = NULL</code> , a character vector of file paths.
lib	character vector giving the library directories where to install the packages.
repos	character vector, the base URL(s) of the repositories to use, can be NULL to install from local files
method	download method
destdir	directory where downloaded packages are stored
dependencies	logical indicating whether to also install uninstalled packages which these packages depend on/link to/import/suggest (and so on recursively). Not used if <code>repos = NULL</code> .
...	Arguments to be passed to 'download.file' or to the functions for binary installs on OS X and Windows.

Section 41.1: Install packages from GitHub

To install packages directly from GitHub use the devtools package:

```
library(devtools)
install_github("authorName/repositoryName")
```

To install ggplot2 from github:

```
devtools::install_github("tidyverse/ggplot2")
```

The above command will install the version of ggplot2 that corresponds to the *master* branch. To install from a different branch of a repository use the ref argument to provide the name of the branch. For example, the following command will install the dev_general branch of the googleway package.

```
devtools::install_github("SymbolixAU/googleway", ref = "dev_general")
```

Another option is to use the ghit package. It provides a lightweight alternative for installing packages from github:

```
install.packages("ghit")
ghit::install_github("google/CausalImpact")
```

To install a package that is in a **private** repository on Github, generate a **personal access token** at

<http://www.github.com/settings/tokens/> (See ?install_github for documentation on the same). Follow these steps:

1. `install.packages(c("curl", "httr"))`
2. `config = httr::config(ssl_verifypeer = FALSE)`
3. `install.packages("RCurl")`
`options(RCurlOptions = c(getOption("RCurlOptions"), ssl.verifypeer = FALSE, ssl.verifyhost = FALSE))`
4. `getOption("RCurlOptions")`

You should see the following:

```
ssl.verifypeer ssl.verifyhost
FALSE      FALSE
5. library(httr)
```

```
set_config(config(ssl_verifypeer = 0L))
```

这可以防止常见错误：“无法使用给定的CA证书验证对等证书”

6.最后，使用以下命令无缝安装你的包

```
install_github("username/package_name", auth_token="abc")
```

或者，使用以下方式设置环境变量GITHUB_PAT，

```
Sys.setenv(GITHUB_PAT = "access_token")
devtools::install_github("organisation/package_name")
```

在Github中生成的PAT（个人访问令牌）只会显示一次，即在最初创建时，因此建议将该令牌保存在.Rprofile中。这对于组织拥有许多私有仓库时也很有帮助。

第41.2节：从仓库下载和安装包

包是以明确定义的格式包含R函数、数据和编译代码的集合。公共（和私有）仓库用于托管R包集合。最大的R包集合可从CRAN获得。

使用CRAN

可以使用以下代码从CRAN安装包：

```
install.packages("dplyr")
```

其中“dplyr”称为字符向量。

可以使用组合函数c()一次安装多个包，传入一系列包名的字符向量：

```
install.packages(c("dplyr", "tidyverse", "ggplot2"))
```

在某些情况下，`install.packages` 可能会提示选择 CRAN 镜像或失败，这取决于 `getOption("repos")` 的值。为防止这种情况，请指定一个 CRAN 镜像 作为 repos 参数：

```
install.packages("dplyr", repos = "https://cloud.r-project.org/")
```

使用 repos 参数也可以从其他仓库安装。有关所有可用选项的完整信息，请运行 `?install.packages`。

大多数包需要其他包中实现的函数（例如包 `data.table`）。为了安装一个包（或多个包）及其所依赖的所有包，应将参数 `dependencies` 设置为 `TRUE`：

```
install.packages("data.table", dependencies = TRUE)
```

使用 Bioconductor

Bioconductor 托管了大量与生物信息学相关的包。它们提供了以 `biocLite` 函数为中心的包管理：

```
set_config(config(ssl_verifypeer = 0L))
```

This prevents the common error: "Peer certificate cannot be authenticated with given CA certificates"

6. Finally, use the following command to install your package seamlessly

```
install_github("username/package_name", auth_token="abc")
```

Alternatively, set an environment variable GITHUB_PAT, using

```
Sys.setenv(GITHUB_PAT = "access_token")
devtools::install_github("organisation/package_name")
```

The PAT generated in Github is only visible once, i.e., when created initially, so its prudent to save that token in .Rprofile. This is also helpful if the organisation has many private repositories.

Section 41.2: Download and install packages from repositories

Packages are collections of R functions, data, and compiled code in a well-defined format. Public (and private) repositories are used to host collections of R packages. The largest collection of R packages is available from CRAN.

Using CRAN

A package can be installed from CRAN using following code:

```
install.packages("dplyr")
```

Where “`dplyr`” is referred to as a character vector.

More than one packages can be installed in one go by using the combine function `c()` and passing a series of character vector of package names:

```
install.packages(c("dplyr", "tidyverse", "ggplot2"))
```

In some cases, `install.packages` may prompt for a CRAN mirror or fail, depending on the value of `getOption("repos")`. To prevent this, specify a CRAN mirror as repos argument:

```
install.packages("dplyr", repos = "https://cloud.r-project.org/")
```

Using the repos argument it is also possible to install from other repositories. For complete information about all the available options, run `?install.packages`.

Most packages require functions, which were implemented in other packages (e.g. the package `data.table`). In order to install a package (or multiple packages) with all the packages, which are used by this given package, the argument `dependencies` should be set to `TRUE`:

```
install.packages("data.table", dependencies = TRUE)
```

Using Bioconductor

Bioconductor hosts a substantial collection of packages related to Bioinformatics. They provide their own package management centred around the `biocLite` function:

```
## 如果不支持 https:// URL, 请尝试使用 http://  
source("https://bioconductor.org/biocLite.R")  
biocLite()
```

默认情况下，这会安装提供最常用功能的部分包。可以通过传递包名向量来安装特定包。例如，要从 Bioconductor 安装 RImmPort：

```
source("https://bioconductor.org/biocLite.R")  
biocLite("RImmPort")
```

第41.3节：从本地源安装软件包

从本地源文件安装软件包：

```
install.packages(path_to_source, repos = NULL, type="source")  
  
install.packages("~/Downloads/dplyr-master.zip", repos=NULL, type="source")
```

这里，path_to_source是本地源文件的绝对路径。

另一个打开窗口以选择已下载的zip或tar.gz源文件的命令是：

```
install.packages(file.choose(), repos=NULL)
```

另一种可能的方法是使用基于GUI的RStudio：

步骤1：进入工具（Tools）。

步骤2：进入安装软件包（Install Packages）。

步骤3：在“从安装”中设置为“软件包归档文件 (.zip ; .tar.gz) ”

步骤4：然后“浏览”找到你的软件包文件（例如 crayon_1.3.1.zip），并且经过一段时间（在“软件包归档”标签中显示软件包路径和文件名之后）

另一种从本地源安装R包的方法是使用devtools包中的install_local()函数。

```
library(devtools)  
install_local("~/Downloads/dplyr-master.zip")
```

第41.4节：安装本地开发版本的包

在开发R包时，通常需要安装该包的最新版本。

这可以通过首先构建包的源代码分发版（在命令行中）来实现

```
R CMD build my_package
```

然后在R中安装它。任何已加载旧版本包的R会话都需要重新加载该包。

```
unloadNamespace("my_package")  
library(my_package)
```

更方便的方法是使用devtools包来简化此过程。在工作目录设置为包目录的R会话中

```
## Try http:// if https:// URLs are not supported  
source("https://bioconductor.org/biocLite.R")  
biocLite()
```

By default this installs a subset of packages that provide the most commonly used functionality. Specific packages can be installed by passing a vector of package names. For example, to install RImmPort from Bioconductor:

```
source("https://bioconductor.org/biocLite.R")  
biocLite("RImmPort")
```

Section 41.3: Install package from local source

To install package from local source file:

```
install.packages(path_to_source, repos = NULL, type="source")  
  
install.packages("~/Downloads/dplyr-master.zip", repos=NULL, type="source")
```

Here, path_to_source is absolute path of local source file.

Another command that opens a window to choose downloaded zip or tar.gz source files is:

```
install.packages(file.choose(), repos=NULL)
```

Another possible way is using the GUI based RStudio:

Step 1: Go to Tools.

Step 2: Go to Install Packages.

Step 3: In the Install From set it as Package Archive File (.zip; .tar.gz)

Step 4: Then Browse find your package file (say crayon_1.3.1.zip) and after some time (after it shows the **Package path and file name** in the Package Archive tab)

Another way to install R package from local source is using install_local() function from devtools package.

```
library(devtools)  
install_local("~/Downloads/dplyr-master.zip")
```

Section 41.4: Install local development version of a package

While working on the development of an R package it is often necessary to install the latest version of the package. This can be achieved by first building a source distribution of the package (on the command line)

```
R CMD build my_package
```

and then installing it in R. Any running R sessions with previous version of the package loaded will need to reload it.

```
unloadNamespace("my_package")  
library(my_package)
```

A more convenient approach uses the devtools package to simplify the process. In an R session with the working directory set to the package directory

```
devtools::install()
```

将构建、安装并重新加载该软件包。

第41.5节：使用命令行界面（CLI）包管理器——基本的pacman用法

pacman 是一个简单的R语言包管理器。

pacman 允许用户通过单个命令 `p_load` 紧凑地加载所有所需的包，自动安装任何缺失的包（及其依赖项）

。pacman 不要求用户在包名周围输入引号。基本用法如下：

```
p_load(data.table, dplyr, ggplot2)
```

使用这种方法，唯一需要 `library`、`require` 或 `install.packages` 语句的包是 pacman 本身：

```
library(pacman)  
p_load(data.table, dplyr, ggplot2)
```

或者，同样有效的是：

```
pacman::p_load(data.table, dplyr, ggplot2)
```

除了通过减少管理包的代码量节省时间外，pacman 还通过仅在包未安装时安装所需包，促进了可重复代码的构建。

由于您可能不确定pacman是否已安装在将使用您代码的用户的库中（或者您自己将来使用自己的代码时），最佳做法是包含一个条件语句，以便在pacman尚未加载时安装它：

```
if(!(require(pacman)) install.packages("pacman")  
pacman::p_load(data.table, dplyr, ggplot2)
```

```
devtools::install()
```

will build, install and reload the package.

Section 41.5: Using a CLI package manager -- basic pacman usage

pacman is a simple package manager for R.

pacman allows a user to compactly load all desired packages, installing any which are missing (and their dependencies), with a single command, `p_load`. pacman does not require the user to type quotation marks around a package name. Basic usage is as follows:

```
p_load(data.table, dplyr, ggplot2)
```

The only package requiring a `library`, `require`, or `install.packages` statement with this approach is pacman itself:

```
library(pacman)  
p_load(data.table, dplyr, ggplot2)
```

or, equally valid:

```
pacman::p_load(data.table, dplyr, ggplot2)
```

In addition to saving time by requiring less code to manage packages, pacman also facilitates the construction of reproducible code by installing any needed packages if and only if they are not already installed.

Since you may not be sure if pacman is installed in the library of a user who will use your code (or by yourself in future uses of your own code) a best practice is to include a conditional statement to install pacman if it is not already loaded:

```
if(!(require(pacman)) install.packages("pacman")  
pacman::p_load(data.table, dplyr, ggplot2)
```

第42章：检查包

包是基于基础R构建的。本文档解释了如何检查已安装的包及其功能。相关文档：安装包

第42.1节：查看包版本

条件：包应至少已安装。如果当前会话中未加载，也没问题。

```
## 检查过去安装的包版本或  
## 当前已安装但未在当前会话中加载的包版本  
  
packageVersion("seqinr")  
# [1] '3.3.3'  
packageVersion("RWeka")  
# [1] '0.4.29'
```

第42.2节：查看当前会话中加载的包

查看已加载包的列表

```
search()
```

或

```
(.packages())
```

第42.3节：查看包信息

要检索dplyr包及其函数描述的信息：

```
help(package = "dplyr")
```

无需先加载该包。

第42.4节：查看包内置数据集

查看dplyr包中的内置数据集

```
data(package = "dplyr")
```

无需先加载该包。

第42.5节：列出软件包导出的函数

要获取dplyr包中的函数列表，首先必须加载该包：

```
library(dplyr)  
ls("package:dplyr")
```

Chapter 42: Inspecting packages

Packages build on base R. This document explains how to inspect installed packages and their functionality. Related Docs: Installing packages

Section 42.1: View Package Version

Conditions: package should be at least installed. If not loaded in the current session, not a problem.

```
## Checking package version which was installed at past or  
## installed currently but not loaded in the current session  
  
packageVersion("seqinr")  
# [1] '3.3.3'  
packageVersion("RWeka")  
# [1] '0.4.29'
```

Section 42.2: View Loaded packages in Current Session

To check the list of loaded packages

```
search()
```

OR

```
(.packages())
```

Section 42.3: View package information

To retrieve information about dplyr package and its functions' descriptions:

```
help(package = "dplyr")
```

No need to load the package first.

Section 42.4: View package's built-in data sets

To see built-in data sets from package dplyr

```
data(package = "dplyr")
```

No need to load the package first.

Section 42.5: List a package's exported functions

To get the list of functions within package dplyr, we first must load the package:

```
library(dplyr)  
ls("package:dplyr")
```

第43章：使用devtools创建包

本主题将介绍如何使用devtools包从零开始创建R包。

第43.1节：创建和分发包

这是一个简明指南，介绍如何快速从代码创建R包。详尽的文档将在可用时链接，若想深入了解，请务必阅读。更多资源见备注部分。

代码所在的目录将称为`./`，所有命令均假定在该文件夹的R提示符下执行。

文档的创建

代码的文档必须采用与LaTeX非常相似的格式。

不过，我们将使用一个名为roxygen的工具来简化该过程：

```
install.packages("devtools")
library("devtools")
install.packages("roxygen2")
library("roxygen2")
```

roxygen的完整手册页可在此处获得。它与doxygen非常相似。

这里有一个关于如何使用roxygen记录函数的实用示例：

```
'# 增加一个变量。
#
#' 注意，如果`x`不是`numeric`类，则此函数的行为未定义。
#
#' @export
#' @author another guy
#' @name 增加函数
#' @title 增加
#
#' @param x 要增加的变量
#' @return `x`增加1
#
#' @seealso `other_function`
#
#' @examples
#' increment(3)
#' > 4
increment <- function(x) {
  return (x+1)
}
```

然后 [将会得到结果](#)。

还建议创建一个示例文档（参见主题 [创建示例文档](#)），它是关于你的包的完整指南。

Chapter 43: Creating packages with devtools

This topic will cover the creation of R packages from scratch with the devtools package.

Section 43.1: Creating and distributing packages

This is a *compact guide* about how to quickly create an R package from your code. Exhaustive documentations will be linked when available and should be read if you want a deeper knowledge of the situation. See *Remarks* for more resources.

The directory where your code stands will be referred as `./`, and all the commands are meant to be executed from a R prompt in this folder.

Creation of the documentation

The documentation for your code has to be in a format which is very similar to LaTeX.

However, we will use a tool named roxygen in order to simplify the process:

```
install.packages("devtools")
library("devtools")
install.packages("roxygen2")
library("roxygen2")
```

The full man page for roxygen is available [here](#). It is very similar to doxygen.

Here is a practical sample about how to document a function with roxygen:

```
'# Increment a variable.
#
#' Note that the behavior of this function
#' is undefined if `x` is not of class `numeric`.
#
#' @export
#' @author another guy
#' @name Increment Function
#' @title increment
#
#' @param x Variable to increment
#' @return `x` incremented of 1
#
#' @seealso `other_function`
#
#' @examples
#' increment(3)
#' > 4
increment <- function(x) {
  return (x+1)
}
```

And [here will be the result](#).

It is also recommended to create a vignette (see the topic *Creating vignettes*), which is a full guide about your package.

构建包的骨架

假设你的代码写在文件 `./script1.R` 和 `./script2.R` 中，运行以下命令以创建你的包的文件结构：

```
package.skeleton(name="MyPackage", code_files=c("script1.R","script2.R"))
```

然后删除 `./MyPackage/man/` 中的所有文件。现在你需要编译文档：

```
roxygenize("MyPackage")
```

你还应该使用命令提示符中从./启动的R CMD Rd2pdf MyPackage命令，从你的文档生成参考手册。

包属性编辑

1. 包描述

根据需要修改`./MyPackage/DESCRIPTION`。字段Package、Version、License、Description、Title、Author和Maintainer是必填项，其他字段为可选项。

如果你的包依赖其他包，请在名为Depends (R版本 < 3.2.0) 或Imports (R版本 > 3.2.0) 的字段中指定它们。

2. 可选文件夹

一旦你启动了骨架构建，`./MyPackage/`只包含R/和man/子文件夹。但它还可以包含其他文件夹：

- `data/`：这里可以放置你的库需要且不是代码的数据。必须以.RData扩展名保存为数据集，可以在运行时通过`data()`和`load()`加载。
- `tests/`：此文件夹中的所有代码文件将在安装时运行。如果有任何错误，安装将失败。
- `src/`：用于你需要的C/C++/Fortran源文件（使用Rcpp等）。
- `exec/`：用于其他可执行文件。
- `misc/`：几乎用于所有其他情况。

最终定稿与构建

您可以删除`./MyPackage/Read-and-delete-me`。

目前，您的包已准备好安装。

您可以使用`devtools::install("MyPackage")`进行安装。

要将您的包构建为源代码压缩包，您需要在命令提示符中执行以下命令，位置为
`./ : R CMD build MyPackage`

通过Github分发您的包

只需创建一个名为`MyPackage`的新仓库，并将`MyPackage/`中的所有内容上传到主分支。这里是一个示例。

Construction of the package skeleton

Assuming that your code is written for instance in files `./script1.R` and `./script2.R`, launch the following command in order to create the file tree of your package:

```
package.skeleton(name="MyPackage", code_files=c("script1.R","script2.R"))
```

Then delete all the files in `./MyPackage/man/`. You have now to compile the documentation:

```
roxygenize("MyPackage")
```

You should also generate a reference manual from your documentation using R CMD Rd2pdf MyPackage from a command prompt started in `./`.

Edition of the package properties

1. Package description

Modify `./MyPackage/DESCRIPTION` according to your needs. The fields Package, Version, License, Description, Title, Author and Maintainer are mandatory, the other are optional.

If your package depends on others packages, specify them in a field named Depends (R version < 3.2.0) or Imports (R version > 3.2.0).

2. Optional folders

Once you launched the skeleton build, `./MyPackage/` only had R/ and man/ subfolders. However, it can have some others:

- `data/`: here you can place the data that your library needs and that isn't code. It must be saved as dataset with the .RData extension, and you can load it at runtime with `data()` and `load()`
- `tests/`: all the code files in this folder will be ran at install time. If there is any error, the installation will fail.
- `src/`: for C/C++/Fortran source files you need (using Rcpp...).
- `exec/`: for other executables.
- `misc/`: for barely everything else.

Finalization and build

You can delete `./MyPackage/Read-and-delete-me`.

As it is now, your package is ready to be installed.

You can install it with `devtools::install("MyPackage")`.

To build your package as a source tarball, you need to execute the following command, from a command prompt in `./ : R CMD build MyPackage`

Distribution of your package

Through Github

Simply create a new repository called `MyPackage` and upload everything in `MyPackage/` to the master branch. Here is an example.

然后任何人都可以使用devtools从github安装您的包：

```
install_package("MyPackage", "your_github_username")
```

通过CRAN

您的软件包需要遵守CRAN仓库政策。包括但不限于：您的软件包必须是跨平台的（除了一些非常特殊的情况），并且应通过R CMD check测试。

这是提交表单。您必须上传源代码压缩包。

第43.2节：创建vignette

vignette是您软件包的长篇指南。如果您知道所需函数的名称，函数文档非常有用，但否则'毫无用处。vignette就像一本书的章节或一篇学术论文：它可以描述您的软件包旨在解决的问题，然后向读者展示如何解决该问题。

vignette将完全用markdown创建。

要求

- Rmarkdown : `install.packages("rmarkdown")`
- [Pandoc](#)

vignette创建

```
devtools::use_vignette("MyVignette", "MyPackage")
```

您现在可以编辑您的示例文档，路径为./vignettes/MyVignette.Rmd。

您的示例文档中的文本格式为Markdown。

对原始Markdown的唯一补充是一个标签，它接受R代码，运行它，捕获输出，并将其转换为格式化的Markdown：

```
```{r}
将两个数字相加
add <- function(a, b) a + b
add(10, 20)
...``
```

将显示为：

```
将两个数字相加
add <- function(a, b) a + b
add(10, 20)
[1] 30
```

因此，您在示例文档中使用的所有包必须在./DESCRIPTION中列为依赖项。

Then anyone can install your package from github with devtools:

```
install_package("MyPackage", "your_github_username")
```

#### Through CRAN

Your package needs to comply to the [CRAN Repository Policy](#). Including but not limited to: your package must be cross-platforms (except some very special cases), it should pass the R CMD check test.

Here is the [submission form](#). You must upload the source tarball.

## Section 43.2: Creating vignettes

A vignette is a long-form guide to your package. Function documentation is great if you know the name of the function you need, but it's useless otherwise. A vignette is like a book chapter or an academic paper: it can describe the problem that your package is designed to solve, and then show the reader how to solve it.

Vignettes will be created entirely in markdown.

#### Requirements

- Rmarkdown: `install.packages("rmarkdown")`
- [Pandoc](#)

#### Vignette creation

```
devtools::use_vignette("MyVignette", "MyPackage")
```

You can now edit your vignette at ./vignettes/MyVignette.Rmd.

The text in your vignette is formatted as [Markdown](#).

The only addition to the original Markdown, is a tag that takes R code, runs it, captures the output, and translates it into formatted Markdown:

```
```{r}
# Add two numbers together
add <- function(a, b) a + b
add(10, 20)
...``
```

Will display as:

```
# Add two numbers together
add <- function(a, b) a + b
add(10, 20)
## [1] 30
```

Thus, all the packages you will use in your vignettes must be listed as dependencies in ./DESCRIPTION.

第44章：在您自己的包中使用管道赋值 %<>%：如何操作？

为了在用户创建的包中使用管道操作符，必须像导入的其他函数一样将其列在NAMESPACE中。

第44.1节：将管道放入实用函数文件中

一种方法是在包内部导出管道操作符。这可以在许多包用来存放未作为包一部分导出的有用小函数的“传统”zzz.R或utils.R文件中完成。例如，放入：

```
#' 管道操作符
#
#' @name %>%
#' @rdname pipe
#' @keywords internal
#' @export
#' @importFrom magrittr %>%
#' @usage lhs \%>\% rhs
NULL
```

Chapter 44: Using pipe assignment in your own package %<>%: How to ?

In order to use the pipe in a user-created package, it must be listed in the NAMESPACE like any other function you choose to import.

Section 44.1: Putting the pipe in a utility-functions file

One option for doing this is to export the pipe from within the package itself. This may be done in the 'traditional' zzz.R or utils.R files that many packages utilise for useful little functions that are not exported as part of the package. For example, putting:

```
#' Pipe operator
#
#' @name %>%
#' @rdname pipe
#' @keywords internal
#' @export
#' @importFrom magrittr %>%
#' @usage lhs \%>\% rhs
NULL
```

第45章：Arima模型

第45.1节：使用Arima建模AR1过程

我们将对该过程进行建模

$$x_t = .7x_{t-1} + \epsilon \quad \epsilon \sim N(0, 1)$$

```
#加载预测包  
library(forecast)  
  
# 生成长度为 n 的 AR1 过程 (来自 Cowpertwait & Meltcalfe)  
# 设置变量  
set.seed(1234)  
n <- 1000  
x <- matrix(0, 1000, 1)  
w <- rnorm(n)  
  
# 循环创建 x  
for (t in 2:n) x[t] <- 0.7 * x[t-1] + w[t]  
plot(x, type='l')
```

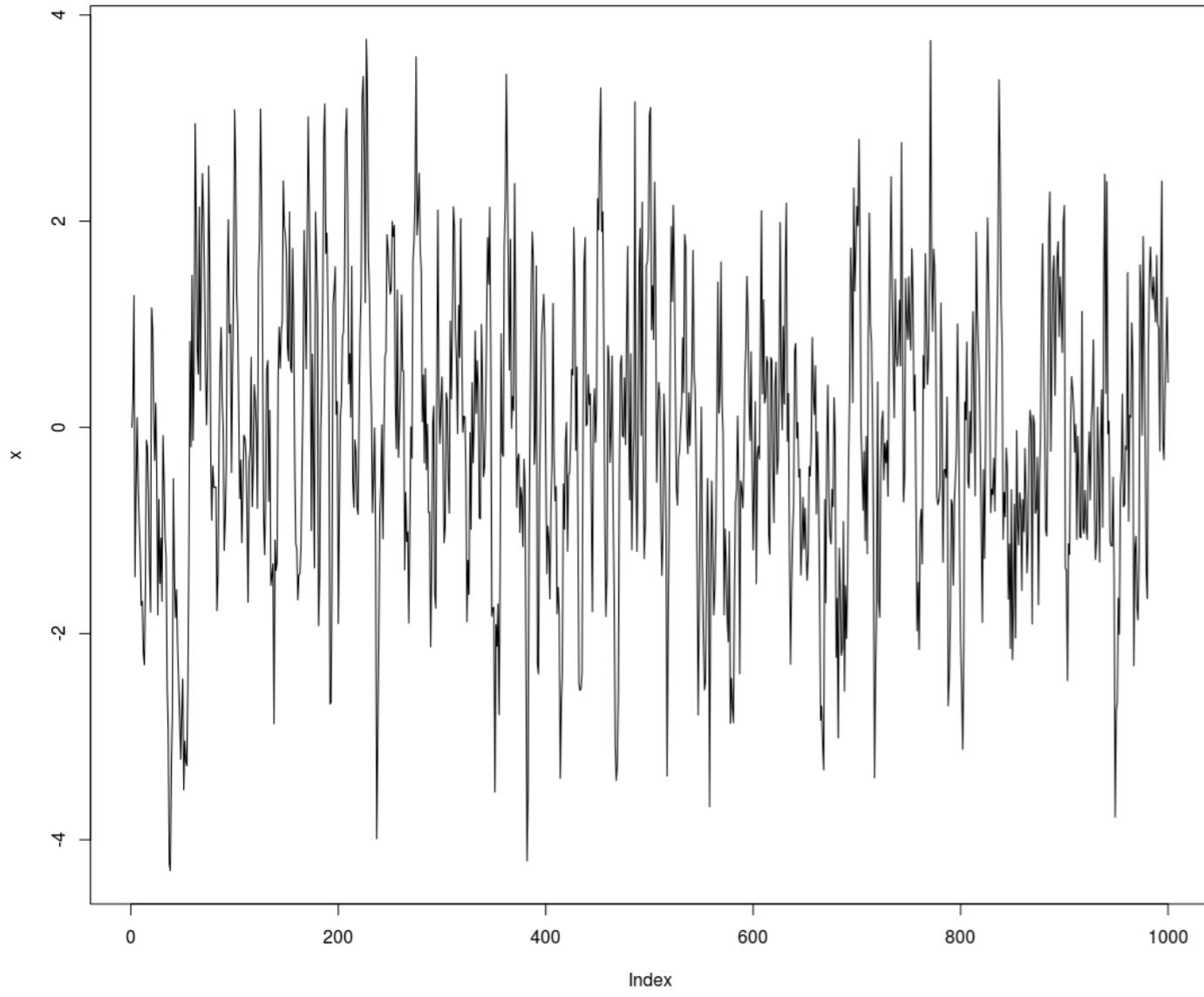
Chapter 45: Arima Models

Section 45.1: Modeling an AR1 Process with Arima

We will model the process

$$x_t = .7x_{t-1} + \epsilon \quad \epsilon \sim N(0, 1)$$

```
#Load the forecast package  
library(forecast)  
  
#Generate an AR1 process of length n (from Cowpertwait & Meltcalfe)  
# Set up variables  
set.seed(1234)  
n <- 1000  
x <- matrix(0, 1000, 1)  
w <- rnorm(n)  
  
# loop to create x  
for (t in 2:n) x[t] <- 0.7 * x[t-1] + w[t]  
plot(x, type='l')
```



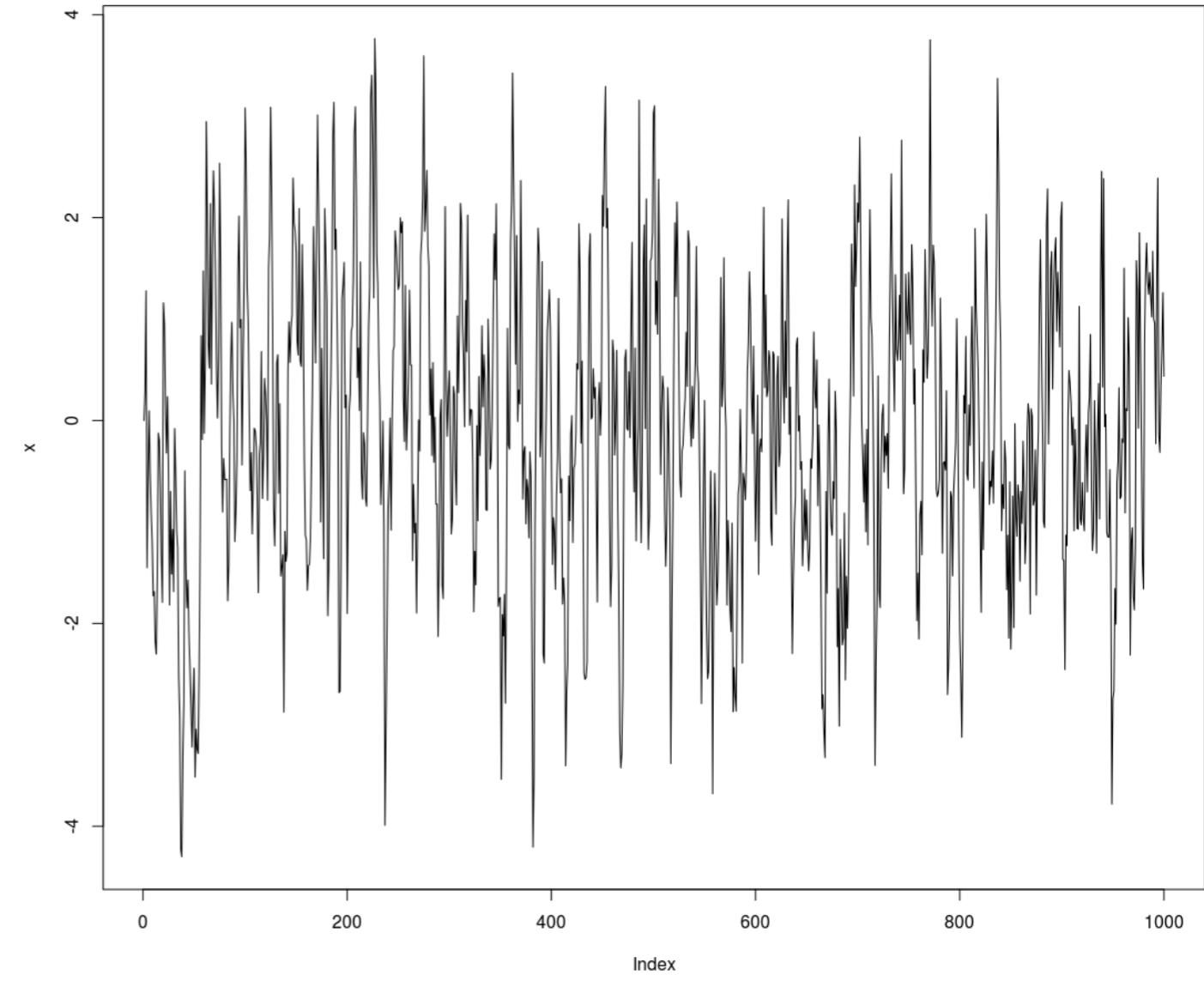
我们将拟合一个自回归阶数为1，差分阶数为0，移动平均阶数为0的 Arima 模型。

```
# 使用 Arima 拟合 AR1 模型
fit <- Arima(x, order = c(1, 0, 0))
summary(fit)
# 序列：x
# ARIMA(1,0,0) 带非零均值

# 系数：
#       ar1 截距
#       0.7040 -0.0842
# 标准误差 0.0224  0.1062

# sigma^2 估计为 0.9923：对数似然=-1415.39
# AIC=2836.79  AICc=2836.81  BIC=2851.51
#
# 训练集误差指标：
#           ME      RMSE      MAE      MPE      MAPE      MASE      ACF1
# 训练集 -8.369365e-05 0.9961194 0.7835914 Inf Inf 0.91488 0.02263595
# 验证模型是否捕捉到了真实的AR参数
```

注意我们的系数接近生成数据中的真实值



We will fit an Arima model with autoregressive order 1, 0 degrees of differencing, and an MA order of 0.

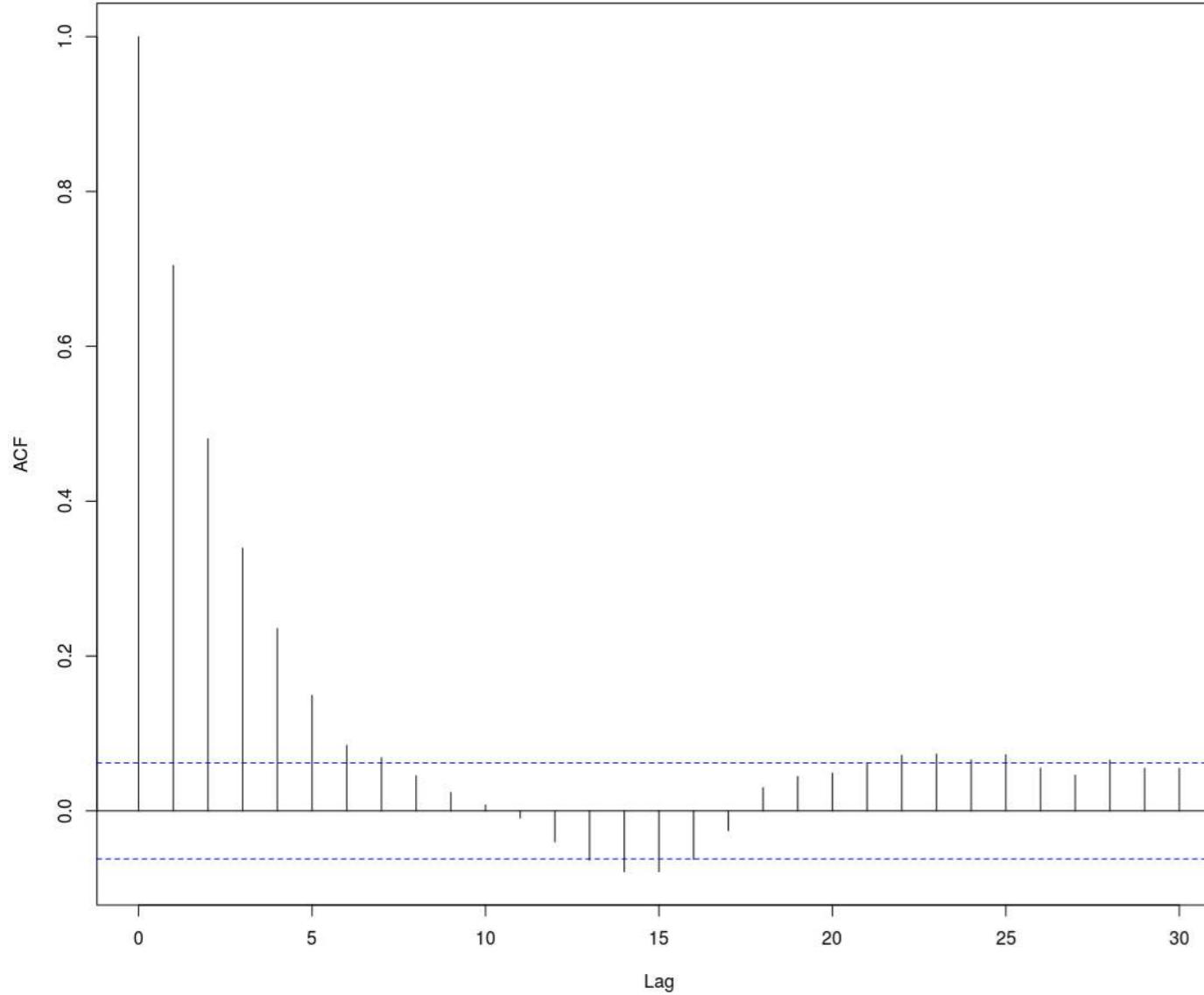
```
#Fit an AR1 model using Arima
fit <- Arima(x, order = c(1, 0, 0))
summary(fit)
# Series: x
# ARIMA(1,0,0) with non-zero mean
#
# Coefficients:
#       ar1 intercept
#       0.7040 -0.0842
# s.e.  0.0224   0.1062
#
# sigma^2 estimated as 0.9923: log likelihood=-1415.39
# AIC=2836.79  AICc=2836.81  BIC=2851.51
#
# Training set error measures:
#           ME      RMSE      MAE      MPE      MAPE      MASE      ACF1
# Training set -8.369365e-05 0.9961194 0.7835914 Inf Inf 0.91488 0.02263595
# Verify that the model captured the true AR parameter
```

Notice that our coefficient is close to the true value from the generated data

```
fit$coef[1]  
#      ar1  
# 0.7040085
```

```
#验证模型是否消除了自相关  
acf(x)
```

Series 1

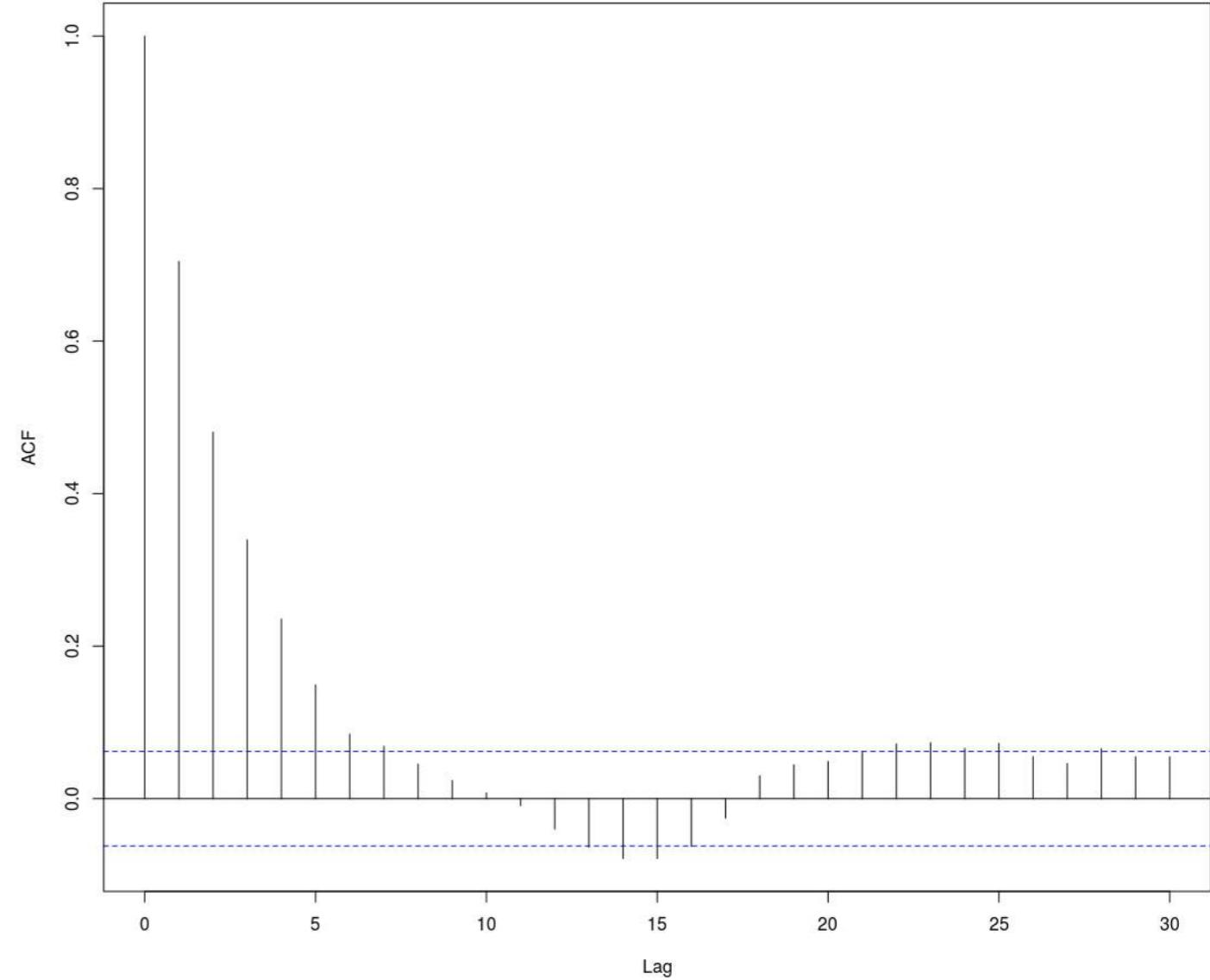


```
acf(fit$resid)
```

```
fit$coef[1]  
#      ar1  
# 0.7040085
```

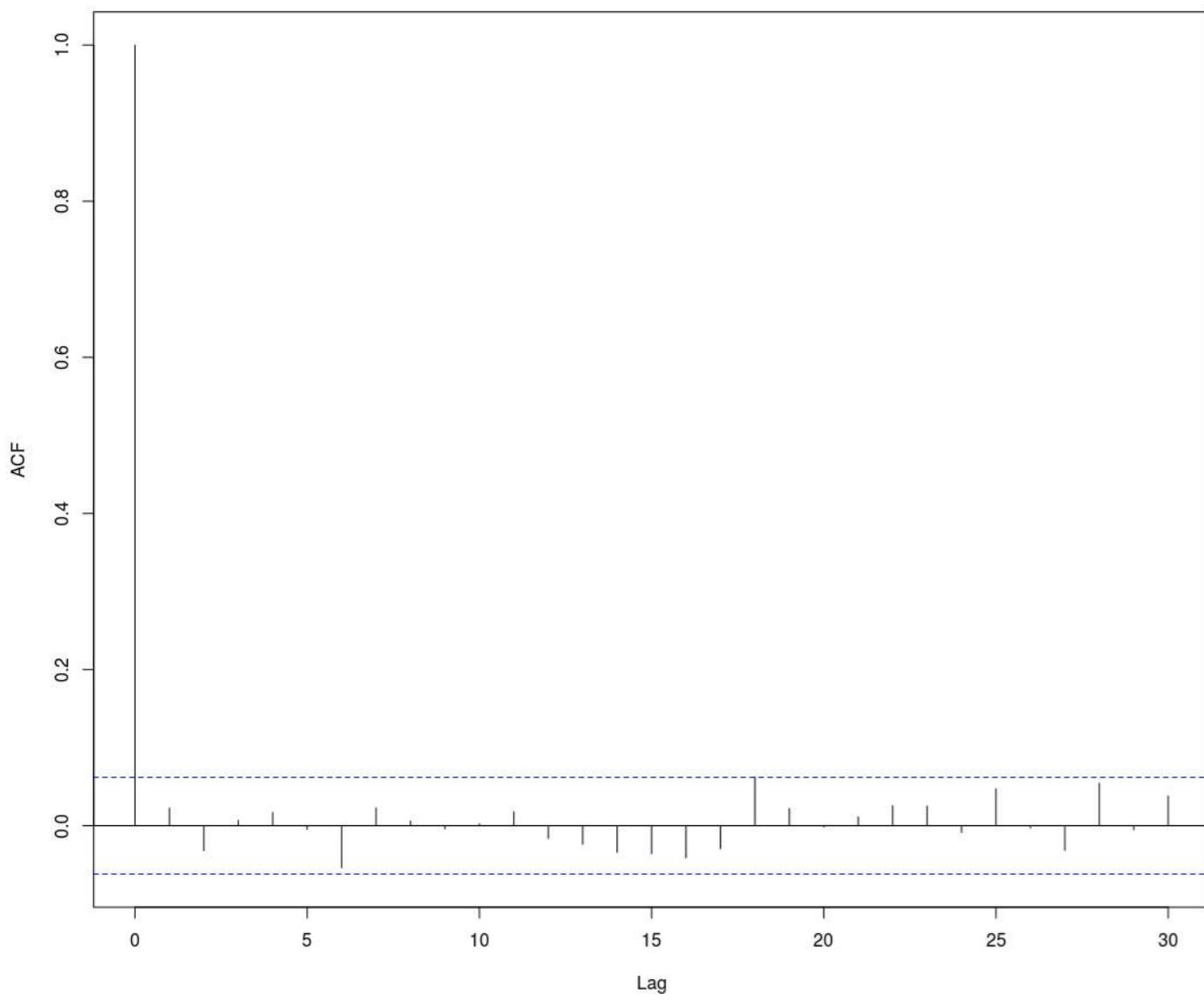
```
#Verify that the model eliminates the autocorrelation  
acf(x)
```

Series 1



```
acf(fit$resid)
```

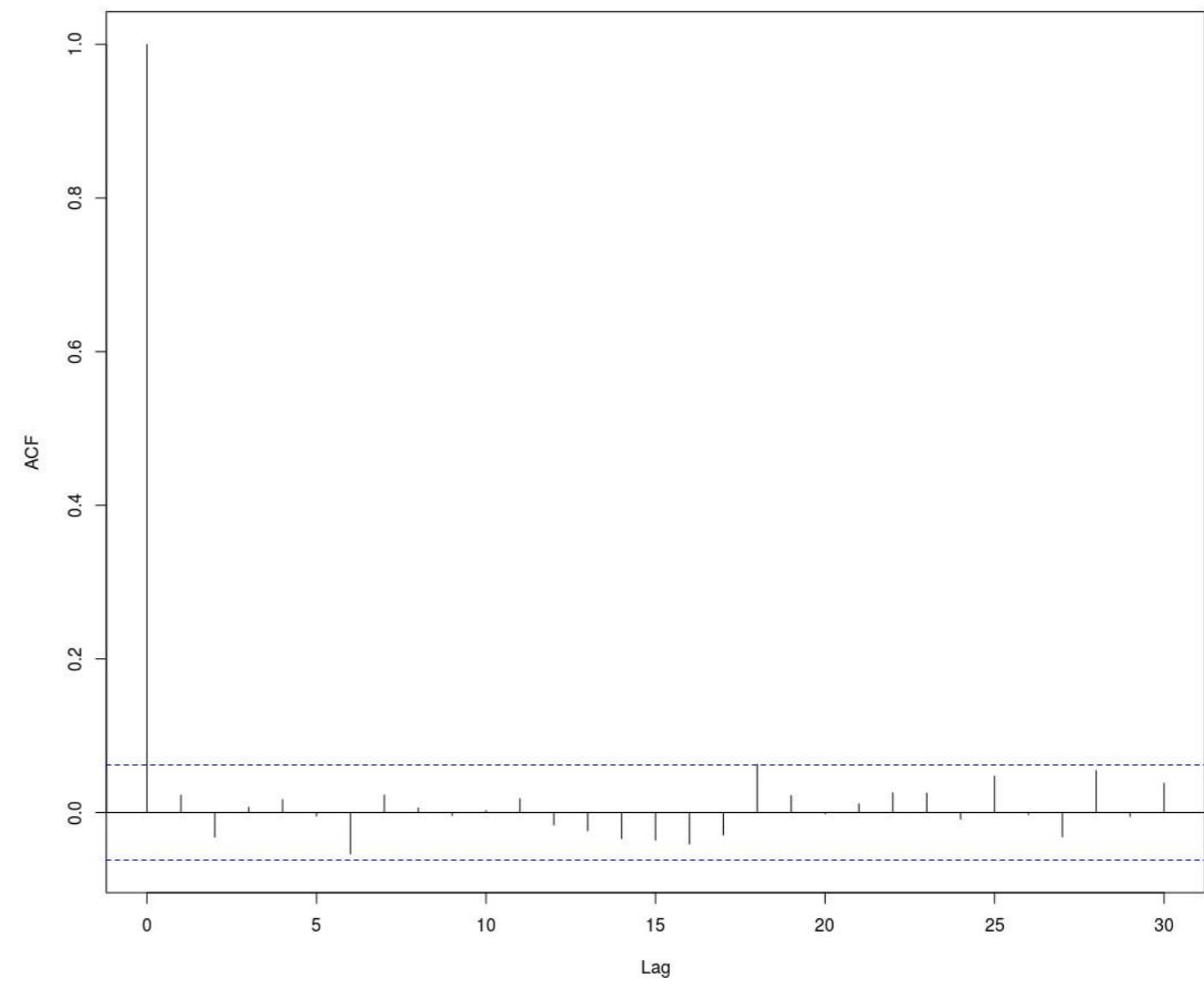
Series fit\$resid



```
#预测10期
fcst <- forecast(fit, h = 100)
fcst
点预测     80%下限   80%上限   95%下限   95%上限
1001  0.282529070 -0.9940493 1.559107 -1.669829 2.234887
1002  0.173976408 -1.3872262 1.735179 -2.213677 2.561630
1003  0.097554408 -1.5869850 1.782094 -2.478726 2.673835
1004  0.043752667 -1.6986831 1.786188 -2.621073 2.708578
1005  0.005875783 -1.7645535 1.776305 -2.701762 2.713514
...
```

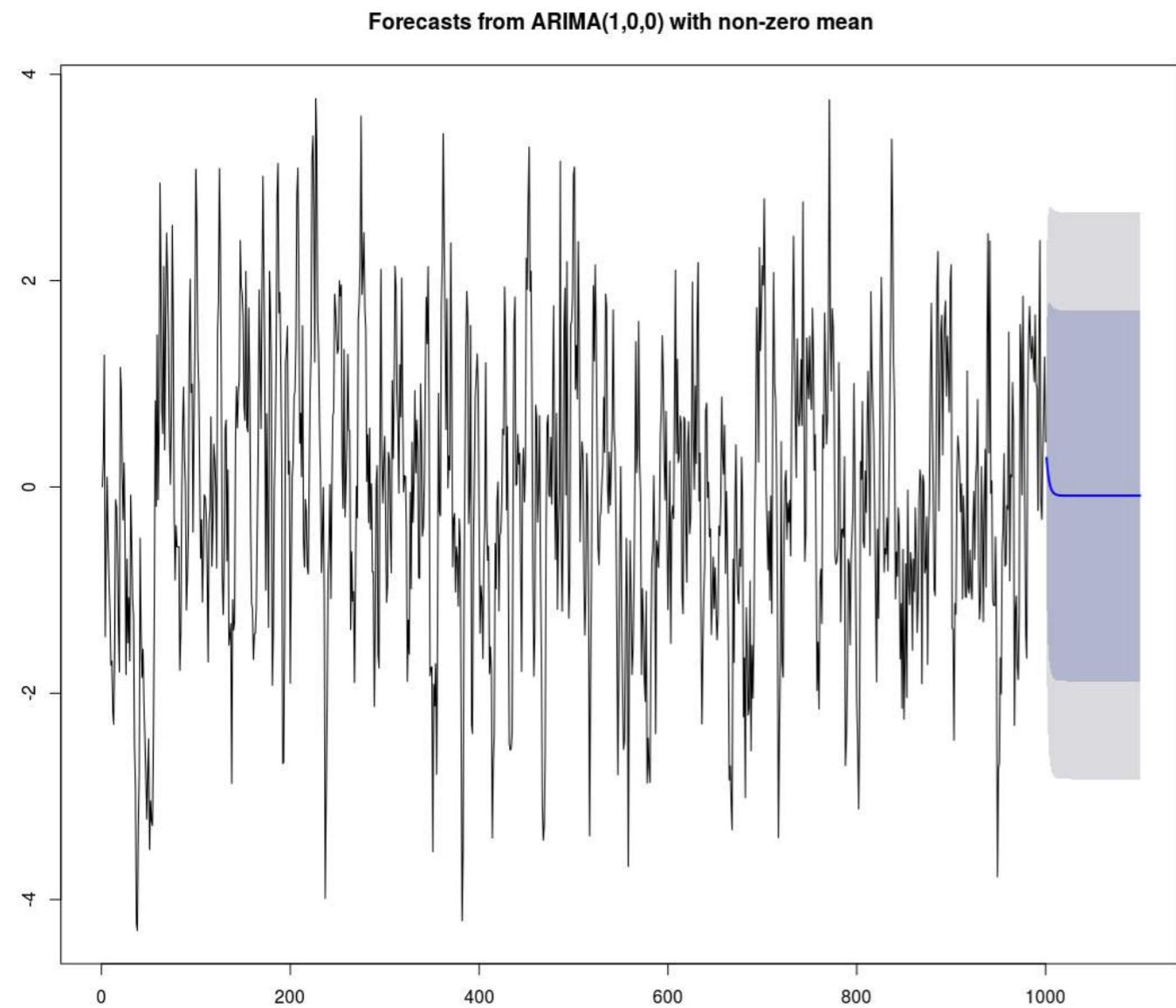
```
#调用点预测
fcst$mean
# 时间序列：
# 起始 = 1001
# 结束 = 1100
# 频率 = 1
[1] 0.282529070 0.173976408 0.097554408 0.043752667 0.005875783 -0.020789866 -0.039562711
-0.052778954
[9] -0.062083302
...
```

Series fit\$resid

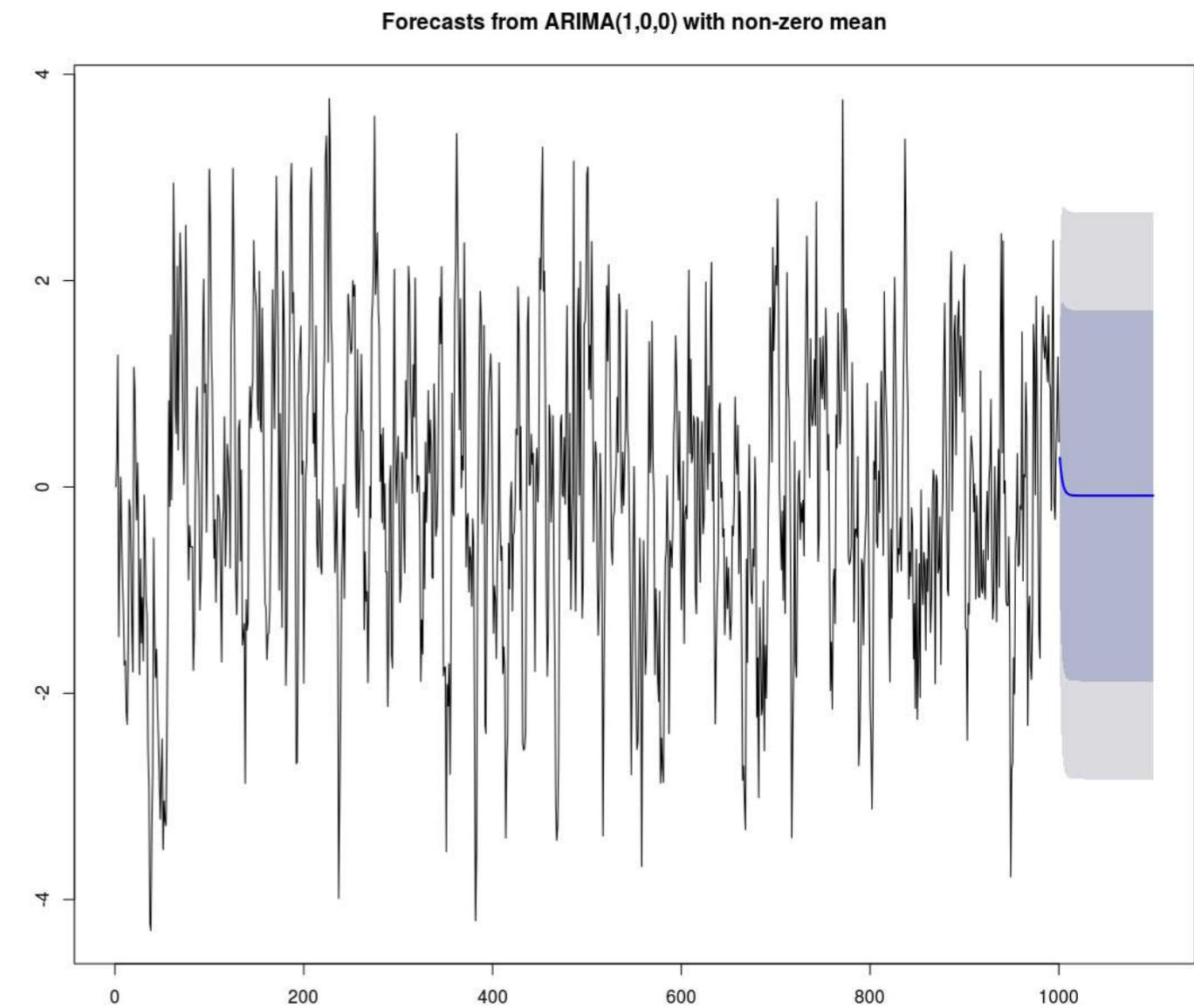


```
#Forecast 10 periods
fcst <- forecast(fit, h = 100)
fcst
Point Forecast     Lo 80      Hi 80      Lo 95      Hi 95
1001    0.282529070 -0.9940493 1.559107 -1.669829 2.234887
1002    0.173976408 -1.3872262 1.735179 -2.213677 2.561630
1003    0.097554408 -1.5869850 1.782094 -2.478726 2.673835
1004    0.043752667 -1.6986831 1.786188 -2.621073 2.708578
1005    0.005875783 -1.7645535 1.776305 -2.701762 2.713514
...
#Call the point predictions
fcst$mean
# Time Series:
# Start = 1001
# End = 1100
# Frequency = 1
[1] 0.282529070 0.173976408 0.097554408 0.043752667 0.005875783 -0.020789866 -0.039562711
-0.052778954
[9] -0.062083302
...
```

```
#绘制预测图  
plot(fcst)
```



```
#Plot the forecast  
plot(fcst)
```



第46章：分布函数

R 有许多内置函数用于处理概率分布，官方文档起始于?Distributions。

第46.1节：正态分布

我们以*norm为例。根据文档：

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

所以如果我想知道标准正态分布在0处的值，我会这样做

```
dnorm(0)
```

这给出了0.3989423，这是一个合理的答案。

同样，pnorm(0)给出.5。再次说明这是合理的，因为分布的一半在0的左侧。

qnorm基本上执行的是pnorm的反操作。qnorm(.5)给出0。

最后，还有 rnorm函数：

```
rnorm(10)
```

将从标准正态分布生成10个样本。

如果您想更改给定分布的参数，只需像这样更改它们

```
rnorm(10, mean=4, sd= 3)
```

第46.2节：二项分布

我们现在演示为二项分布定义的函数dbinom、pbinom、qbinom和rbinom。

函数 dbinom() 给出二项变量各种取值的概率。最少需要三个参数。该函数的第一个参数必须是分位数向量（随机变量X的可能取值）。第二和第三个参数是分布的定义参数，即 n（独立试验次数）和 p（每次试验成功的概率）。例如，对于二项分布，n = 5, p = 0.5, X的可能取值为0、1、2、3、4、5。也就是说，函数 dbinom(x,n,p)给出概率值P(X = x), x = 0、1、2、3、4、5。

```
#Binom(n = 5, p = 0.5) 概率
> n <- 5; p<- 0.5; x <- 0:n
> dbinom(x,n,p)
[1] 0.03125 0.15625 0.31250 0.31250 0.15625 0.03125
#验证总概率为1
> sum(dbinom(x,n,p))
[1] 1
>
```

二项概率分布图可以显示如下图所示：

Chapter 46: Distribution Functions

R has many built-in functions to work with probability distributions, with official docs starting at ?Distributions.

Section 46.1: Normal distribution

Let's use *norm as an example. From the documentation:

```
dnorm(x, mean = 0, sd = 1, log = FALSE)
pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
rnorm(n, mean = 0, sd = 1)
```

So if I wanted to know the value of a standard normal distribution at 0, I would do

```
dnorm(0)
```

Which gives us 0.3989423, a reasonable answer.

In the same way pnorm(0) gives .5. Again, this makes sense, because half of the distribution is to the left of 0.

qnorm will essentially do the opposite of pnorm. qnorm(.5) gives 0.

Finally, there's the rnorm function:

```
rnorm(10)
```

Will generate 10 samples from standard normal.

If you want to change the parameters of a given distribution, simply change them like so

```
rnorm(10, mean=4, sd= 3)
```

Section 46.2: Binomial Distribution

We now illustrate the functions `dbinom`, `pbinom`, `qbinom` and `rbinom` defined for *Binomial distribution*.

The `dbinom()` function gives the probabilities for various values of the binomial variable. Minimally it requires three arguments. The first argument for this function must be a vector of quantiles(the possible values of the random variable X). The second and third arguments are the defining parameters of the distribution, namely, n(the number of independent trials) and p(the probability of success in each trial). For example, for a binomial distribution with n = 5, p = 0.5, the possible values for X are 0, 1, 2, 3, 4, 5. That is, the `dbinom(x,n,p)` function gives the probability values P(X = x) for x = 0, 1, 2, 3, 4, 5.

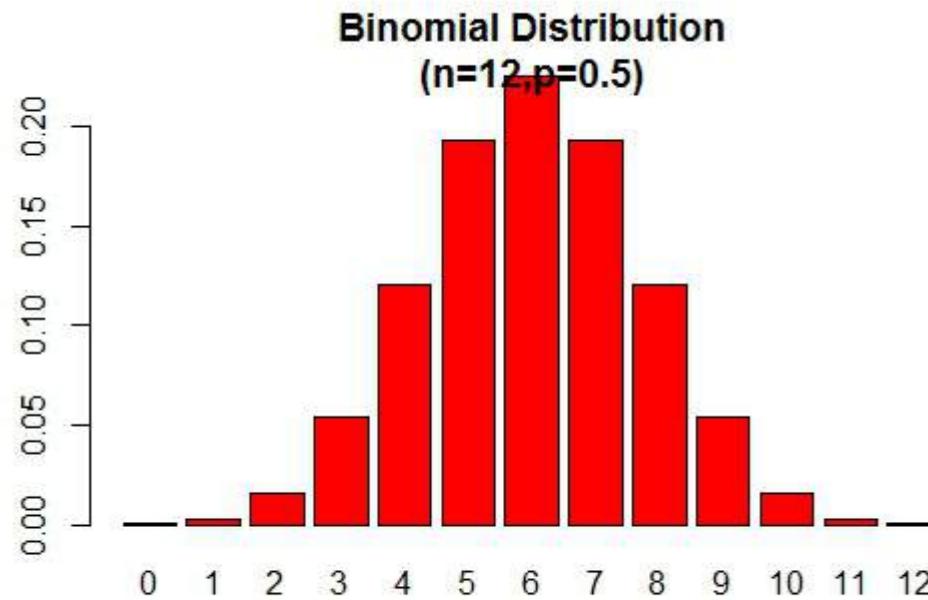
```
#Binom(n = 5, p = 0.5) probabilities
> n <- 5; p<- 0.5; x <- 0:n
> dbinom(x,n,p)
[1] 0.03125 0.15625 0.31250 0.31250 0.15625 0.03125
#To verify the total probability is 1
> sum(dbinom(x,n,p))
[1] 1
>
```

The binomial probability distribution plot can be displayed as in the following figure:

```

> x <- 0:12
> prob <- dbinom(x,12,.5)
> barplot(prob,col = "red",ylim = c(0,.2),names.arg=x,
  main="二项分布(n=12,p=0.5)")

```

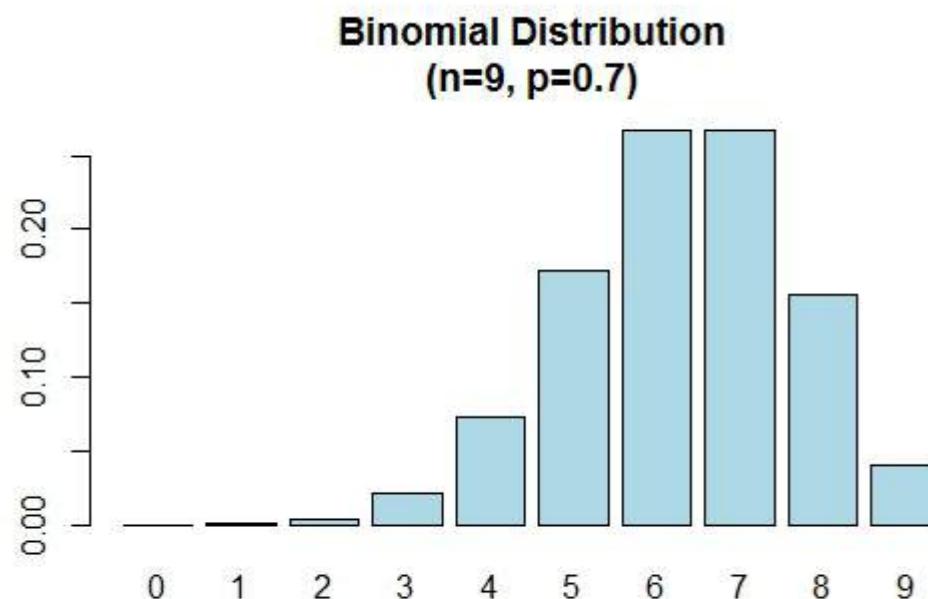


注意，当 $p = 0.5$ 时，二项分布是对称的。为了说明当 p 大于 0.5 时二项分布是负偏的，考虑以下示例：

```

> n=9; p=.7; x=0:n; prob=dbinom(x,n,p);
> barplot(prob,names.arg = x,main="二项分布(n=9, p=0.7)",col="lightblue")

```

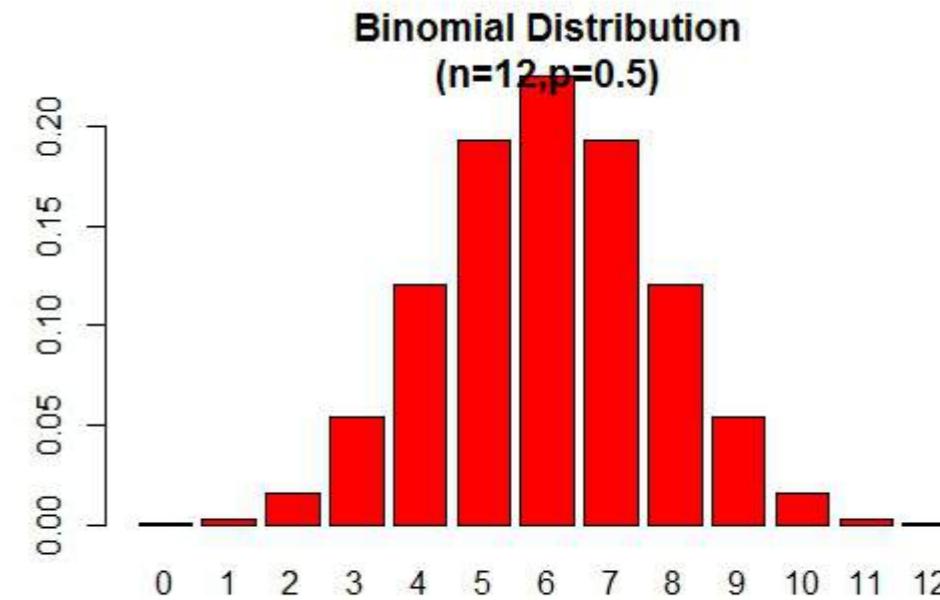


当 p 小于 0.5 时，二项分布是正偏的，如下所示。

```

> x <- 0:12
> prob <- dbinom(x,12,.5)
> barplot(prob,col = "red",ylim = c(0,.2),names.arg=x,
  main="Binomial Distribution\\n(n=12, p=0.5)")

```

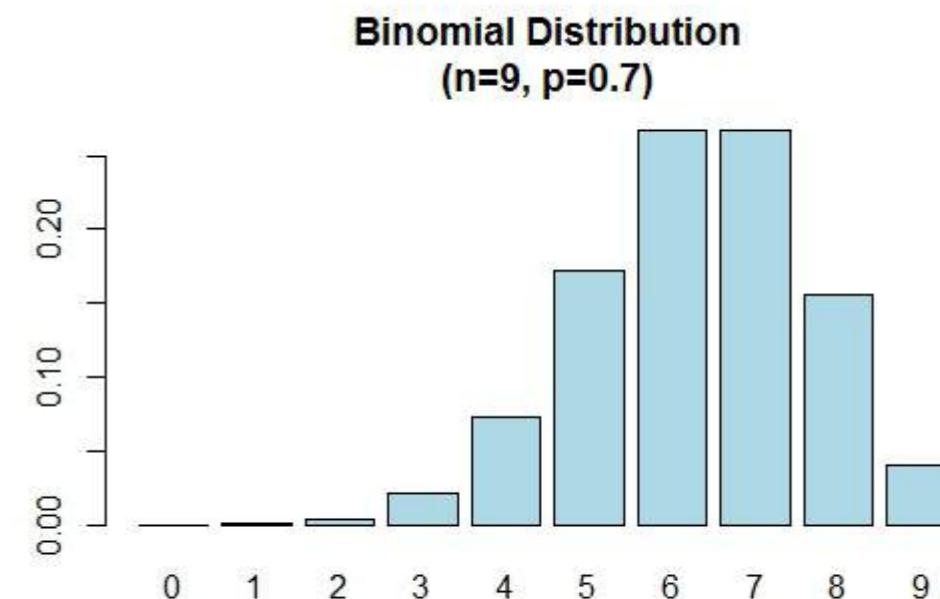


Note that the binomial distribution is symmetric when $p = 0.5$. To demonstrate that the binomial distribution is negatively skewed when p is larger than 0.5, consider the following example:

```

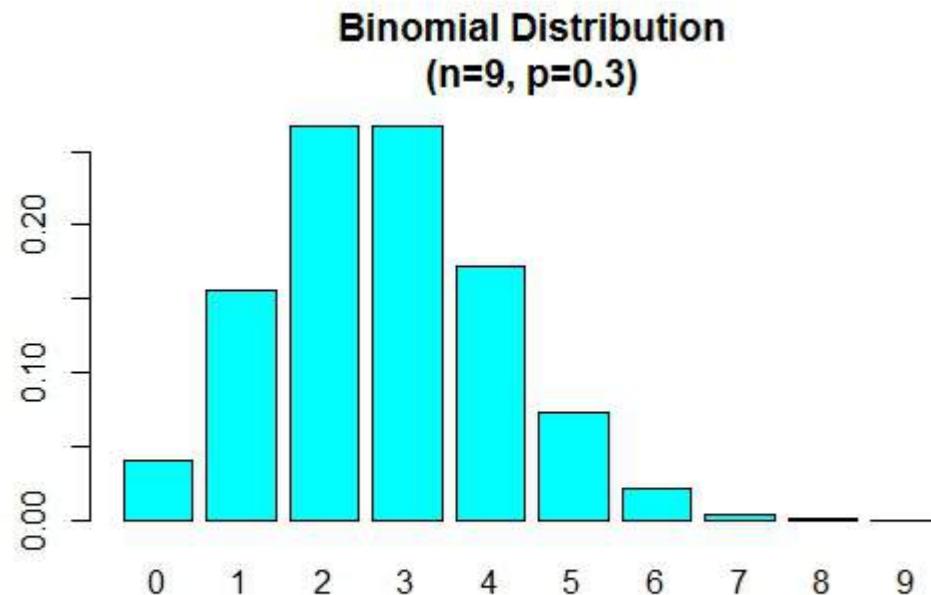
> n=9; p=.7; x=0:n; prob=dbinom(x,n,p);
> barplot(prob,names.arg = x,main="Binomial Distribution\\n(n=9, p=0.7)",col="lightblue")

```

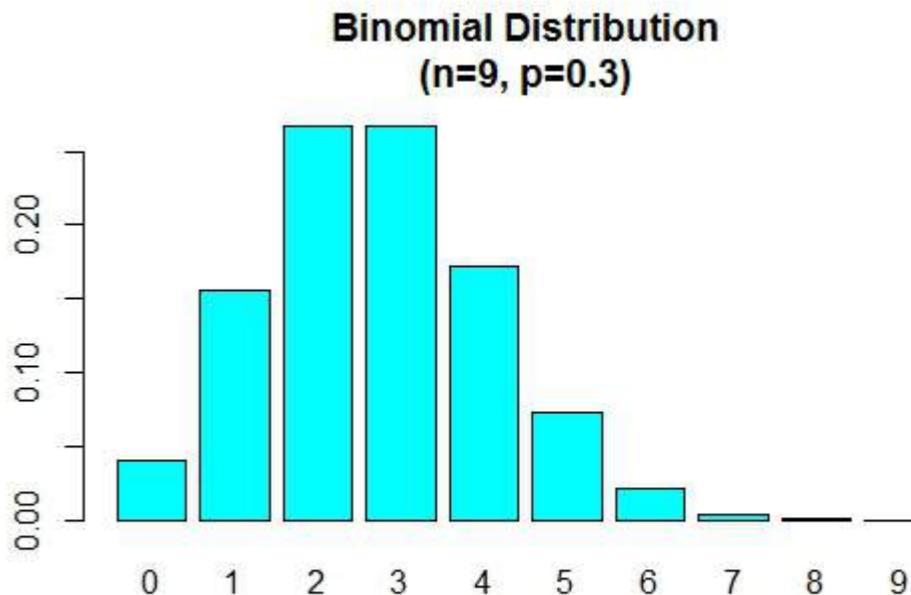


When p is smaller than 0.5 the binomial distribution is positively skewed as shown below.

```
> n=9; p=.3; x=0:n; prob=dbinom(x,n,p);
> barplot(prob,names.arg = x,main="二项分布(n=9, p=0.3)",col="cyan")
```



```
> n=9; p=.3; x=0:n; prob=dbinom(x,n,p);
> barplot(prob,names.arg = x,main="Binomial Distribution\n(n=9, p=0.3)",col="cyan")
```



我们现在将演示累积分布函数 `pbinom()` 的用法。该函数可用于计算诸如 $P(X \leq x)$ 的概率。该函数的第一个参数是分位数向量 (x 的取值)。

```
# 计算概率
# 在 Bin(n=5,p=0.5) 分布中计算 P(X <= 2)
> pbinom(2,5,0.5)
[1] 0.5
```

上述概率也可以通过以下方式获得：

```
# P(X <= 2) = P(X=0) + P(X=1) + P(X=2)
> sum(dbinom(0:2,5,0.5))
[1] 0.5
```

计算类型概率： $P(a \leq X \leq b)$

```
# 在 Bin(n=9,p=0.6) 分布中, P(3<= X <= 5) = P(X=3) + P(X=4) + P(X=5)
> sum(dbinom(c(3,4,5),9,0.6))
[1] 0.4923556
>
```

以表格形式展示二项分布：

```
> n = 10; p = 0.4; x = 0:n;
> prob = dbinom(x,n,p)
> cdf = pbinom(x,n,p)
> distTable = cbind(x,prob,cdf)
> distTable
x      prob      cdf
[1,] 0 0.0060466176 0.006046618
[2,] 1 0.0403107840 0.046357402
[3,] 2 0.1209323520 0.167289754
```

We will now illustrate the usage of the cumulative distribution function `pbinom()`. This function can be used to calculate probabilities such as $P(X \leq x)$. The first argument to this function is a vector of quantiles(values of x).

```
# Calculating Probabilities
# P(X <= 2) in a Bin(n=5,p=0.5) distribution
> pbinom(2,5,0.5)
[1] 0.5
```

The above probability can also be obtained as follows:

```
# P(X <= 2) = P(X=0) + P(X=1) + P(X=2)
> sum(dbinom(0:2,5,0.5))
[1] 0.5
```

To compute, probabilities of the type: $P(a \leq X \leq b)$

```
# P(3<= X <= 5) = P(X=3) + P(X=4) + P(X=5) in a Bin(n=9,p=0.6) dist
> sum(dbinom(c(3,4,5),9,0.6))
[1] 0.4923556
>
```

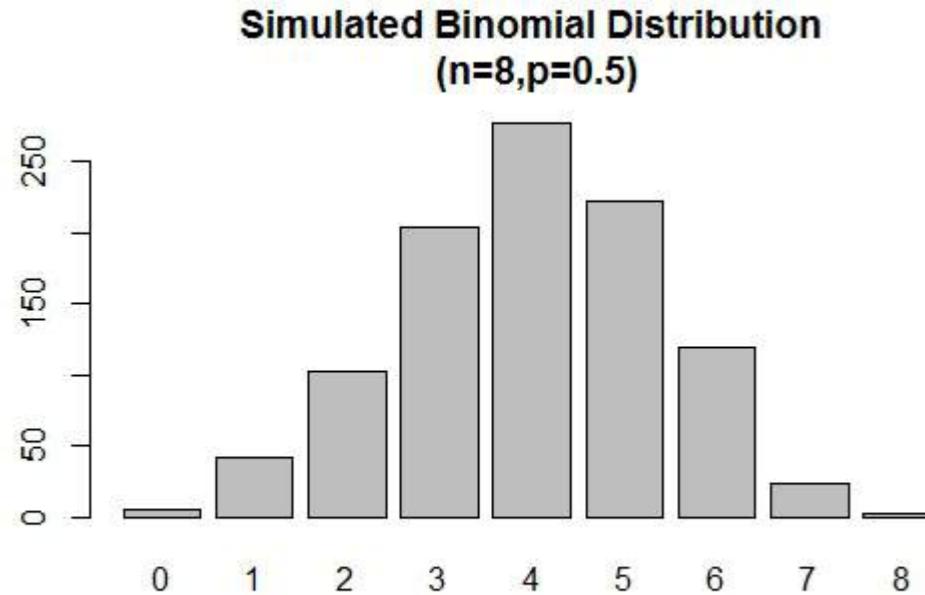
Presenting the binomial distribution in the form of a table:

```
> n = 10; p = 0.4; x = 0:n;
> prob = dbinom(x,n,p)
> cdf = pbinom(x,n,p)
> distTable = cbind(x,prob,cdf)
> distTable
x      prob      cdf
[1,] 0 0.0060466176 0.006046618
[2,] 1 0.0403107840 0.046357402
[3,] 2 0.1209323520 0.167289754
```

```
[4,] 3 0.2149908480 0.382280602
[5,] 4 0.2508226560 0.633103258
[6,] 5 0.2006581248 0.833761382
[7,] 6 0.1114767360 0.945238118
[8,] 7 0.0424673280 0.987705446
[9,] 8 0.0106168320 0.998322278
[10,] 9 0.0015728640 0.999895142
[11,] 10 0.0001048576 1.000000000
>
```

rbinom() 用于生成具有指定参数值的指定大小的随机样本。

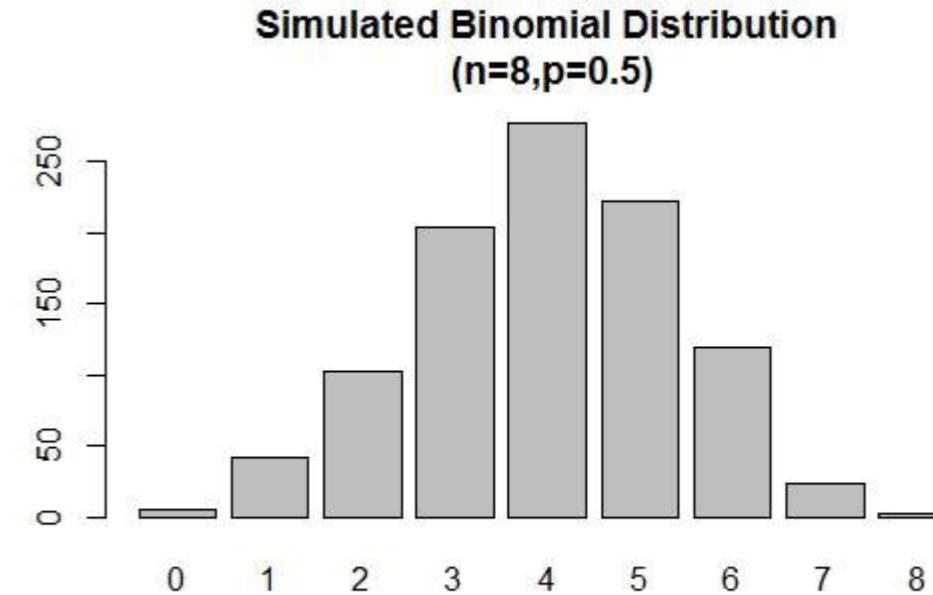
```
# 模拟
> xVal<-names(table(rbinom(1000,8,.5)))> barplot
t(as.vector(table(rbinom(1000,8,.5))),names.arg =xVal,
n="模拟的二项分布 (n=8,p=0.5)" mai
```



```
[4,] 3 0.2149908480 0.382280602
[5,] 4 0.2508226560 0.633103258
[6,] 5 0.2006581248 0.833761382
[7,] 6 0.1114767360 0.945238118
[8,] 7 0.0424673280 0.987705446
[9,] 8 0.0106168320 0.998322278
[10,] 9 0.0015728640 0.999895142
[11,] 10 0.0001048576 1.000000000
>
```

The `rbinom()` is used to generate random samples of specified sizes with a given parameter values.

```
# Simulation
> xVal<-names(table(rbinom(1000,8,.5)))
> barplot(as.vector(table(rbinom(1000,8,.5))),names.arg =xVal,
main="Simulated Binomial Distribution\n (n=8,p=0.5)")
```



第47章：Shiny

第47.1节：创建应用程序

Shiny 是由 [RStudio](#) 开发的一个 R 包，允许创建网页以交互式地显示 R 中分析的结果。

创建 Shiny 应用有两种简单方法：

- 在一个 .R 文件中，或
- 在两个文件中：ui.R 和 server.R。

Shiny 应用分为两个部分：

- ui：用户界面脚本，控制应用的布局和外观。
- server：服务器脚本，包含使应用响应的代码。

单文件

```
library(shiny)

# 创建用户界面
ui <- shinyUI(fluidPage(
  # 应用标题
  titlePanel("Hello World!")
))

# 创建服务器函数
server <- shinyServer(function(input, output){})

# 运行应用
shinyApp(ui = ui, server = server)
```

两个文件

创建ui.R文件

```
library(shiny)

# 定义应用程序的用户界面
shinyUI(fluidPage(
  # 应用程序标题
  titlePanel("Hello World!")
))
```

创建 server.R 文件

```
library(shiny)

# 定义服务器逻辑
shinyServer(function(input, output){})
```

第47.2节：复选框组

创建一组复选框，可用于独立切换多个选项。服务器将接收输入作为所选值的字符向量。

```
library(shiny)

ui <- fluidPage(
```

Chapter 47: Shiny

Section 47.1: Create an app

Shiny is an [R package](#) developed by [RStudio](#) that allows the creation of web pages to interactively display the results of an analysis in R.

There are two simple ways to create a Shiny app:

- in one .R file, or
- in two files: ui.R and server.R.

A Shiny app is divided into two parts:

- **ui**: A user interface script, controlling the layout and appearance of the application.
- **server**: A server script which contains code to allow the application to react.

One file

```
library(shiny)

# Create the UI
ui <- shinyUI(fluidPage(
  # Application title
  titlePanel("Hello World!")
))

# Create the server function
server <- shinyServer(function(input, output){})

# Run the app
shinyApp(ui = ui, server = server)
```

Two files

Create ui.R file

```
library(shiny)

# Define UI for application
shinyUI(fluidPage(
  # Application title
  titlePanel("Hello World!")
))
```

Create server.R file

```
library(shiny)

# Define server logic
shinyServer(function(input, output){})
```

Section 47.2: Checkbox Group

Create a group of checkboxes that can be used to toggle multiple choices independently. The server will receive the input as a character vector of the selected values.

```
library(shiny)

ui <- fluidPage(
```

```

checkboxGroupInput("checkGroup1", 标签 = h3("这是一个复选框组"),
  选项 = list("1" = 1, "2" = 2, "3" = 3),
  默认选中 = 1),
  fluidRow(column(3, verbatimTextOutput("text_choice")))
)

server <- function(input, output){
  output$text_choice <- renderPrint({
    return(paste0("你选择了选项 ", input$checkGroup1)))
  })

shinyApp(ui = ui, server = server)

```

This is a Checkbox group

1
 2
 3

[1] "You have chosen the choice 1"

可以更改设置：

- 标签：标题
- 选项：选中的值
- 选中项：初始选中的值（无选中时为NULL）
- inline：水平或垂直
- 宽度

也可以添加HTML。

第47.3节：单选按钮

你可以创建一组单选按钮，用于从列表中选择一个项目。

可以更改设置：

- 选中项：初始选中的值（无选中时为character(0)）
- inline：水平或垂直
- 宽度

也可以添加HTML。

library(shiny)

```

ui <- fluidPage(
  radioButtons("radio",
  label = HTML('<FONT color="red"><FONT size="5pt">欢迎</FONT></FONT><br> <b>你最喜欢的颜色是红色吗？</b>'),
  choices = list("TRUE" = 1, "FALSE" = 2),
  selected = 1,
  inline = T,
  width = "100%"),

```

```

checkboxGroupInput("checkGroup1", label = h3("This is a Checkbox group"),
  choices = list("1" = 1, "2" = 2, "3" = 3),
  selected = 1),
  fluidRow(column(3, verbatimTextOutput("text_choice")))
)

server <- function(input, output){
  output$text_choice <- renderPrint({
    return(paste0("You have chosen the choice ", input$checkGroup1)))
  })

shinyApp(ui = ui, server = server)

```

This is a Checkbox group

1
 2
 3

[1] "You have chosen the choice 1"

It's possible to change the settings :

- label : title
- choices : selected values
- selected : The initially selected value (NULL for no selection)
- inline : horizontal or vertical
- width

It is also possible to add HTML.

Section 47.3: Radio Button

You can create a set of radio buttons used to select an item from a list.

It's possible to change the settings :

- selected : The initially selected value (character(0) for no selection)
- inline : horizontal or vertical
- width

It is also possible to add HTML.

library(shiny)

```

ui <- fluidPage(
  radioButtons("radio",
  label = HTML('<FONT color="red"><FONT size="5pt">Welcome</FONT></FONT><br> Your
favorite color is red ?</b>'),
  choices = list("TRUE" = 1, "FALSE" = 2),
  selected = 1,
  inline = T,
  width = "100%"),

```

```
fluidRow(column(3, textOutput("value"))))
```

```
server <- function(input, output){  
  output$value <- renderPrint({  
    if(input$radio == 1){return('太棒了 !')}  
    else{return("抱歉 !")}}})  
shinyApp(ui = ui, server = server)
```

Welcome

Your favorite color is red ?

TRUE FALSE

[1] "Great !"

第47.4节：调试

debug() 和 debugonce() 在大多数Shiny调试环境中效果不佳。然而，browser() 语句插入关键位置可以让你深入了解Shiny代码的运行（或未运行）情况。另见：使用 browser() 调试

展示模式

[展示模式](#) 显示你的应用及其生成代码，并在运行时高亮显示 server.R 中的代码行。

有两种方式可以启用展示模式：

- 使用参数 display.mode = "showcase" 启动 Shiny 应用，例如：runApp("MyApp", display.mode = "showcase")。
- 在你的 Shiny 应用文件夹中创建名为 DESCRIPTION 的文件，并在其中添加这一行：DisplayMode: Showcase。

反应日志可视化工具

反应日志可视化工具 提供了一个基于浏览器的交互式工具，用于可视化应用中的反应依赖关系和执行。要启用反应日志可视化工具，请在 R 控制台执行 options(shiny.reactlog=TRUE)，或者在你的 server.R 文件中添加该代码行。应用运行时，按下 Windows 上的 Ctrl+F3 或 Mac 上的 Command+F3 即可启动反应日志可视化工具。使用左右箭头键在反应日志可视化工具中导航。

第47.5节：选择框

创建一个选择列表，用于从一组值中选择单个或多个项目。

```
library(shiny)
```

```
ui <- fluidPage(  
  selectInput("id_selectInput",  
    label = HTML('<B><FONT size="3">你最喜欢的颜色是什么 ?</FONT></B>'),  
    multiple = TRUE,  
    choices = list("red" = "red", "green" = "green", "blue" = "blue", "yellow" = "yellow"),  
    selected = NULL),  
  br(), br(),  
  fluidRow(column(3, textOutput("text_choice"))))
```

```
fluidRow(column(3, textOutput("value"))))
```

```
server <- function(input, output){  
  output$value <- renderPrint({  
    if(input$radio == 1){return('Great !')}  
    else{return("Sorry !")}}})  
shinyApp(ui = ui, server = server)
```

Welcome

Your favorite color is red ?

TRUE FALSE

[1] "Great !"

Section 47.4: Debugging

debug() and debugonce() won't work well in the context of most Shiny debugging. However, browser() statements inserted in critical places can give you a lot of insight into how your Shiny code is (not) working. See also: Debugging using browser()

Showcase mode

[Showcase mode](#) displays your app alongside the code that generates it and highlights lines of code in server.R as it runs them.

There are two ways to enable Showcase mode:

- Launch Shiny app with the argument display.mode = "showcase", e.g., runApp("MyApp", display.mode = "showcase").
- Create file called DESCRIPTION in your Shiny app folder and add this line in it: DisplayMode: Showcase.

Reactive Log Visualizer

[Reactive Log Visualizer](#) provides an interactive browser-based tool for visualizing reactive dependencies and execution in your application. To enable Reactive Log Visualizer, execute `options(shiny.reactlog=TRUE)` in R console and or add that line of code in your server.R file. To start Reactive Log Visualizer, hit Ctrl+F3 on Windows or Command+F3 on Mac when your app is running. Use left and right arrow keys to navigate in Reactive Log Visualizer.

Section 47.5: Select box

Create a select list that can be used to choose a single or multiple items from a list of values.

```
library(shiny)
```

```
ui <- fluidPage(  
  selectInput("id_selectInput",  
    label = HTML('<B><FONT size="3">What is your favorite color ?</FONT></B>'),  
    multiple = TRUE,  
    choices = list("red" = "red", "green" = "green", "blue" = "blue", "yellow" = "yellow"),  
    selected = NULL),  
  br(), br(),  
  fluidRow(column(3, textOutput("text_choice"))))
```

```
server <- function(input, output){
  output$text_choice <- renderPrint({
    return(input$id_selectInput)})
}

shinyApp(ui = ui, server = server)
```

What is your favorite color ?

red green blue

yellow

[1] "red" "green" "blue"

可以更改设置：

- 标签：标题
- 选项：选中的值
- selected：初始选中的值（无选择时为NULL）
- multiple：TRUE或FALSE
- 宽度
- size
- selectize：TRUE或FALSE（是否使用selectize.js，改变显示效果）

也可以添加HTML。

第47.6节：启动Shiny应用

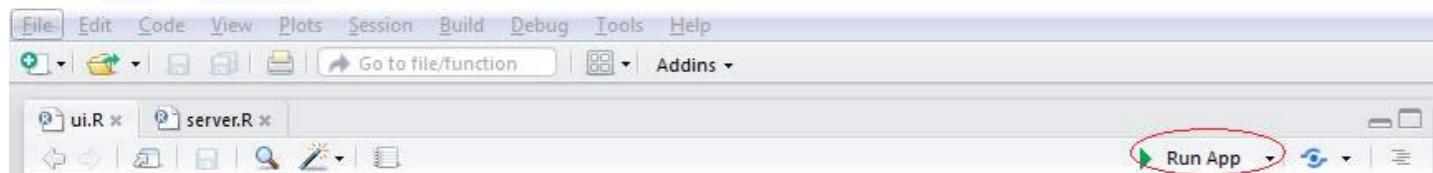
根据你创建应用的方式，可以通过多种方式启动应用。如果你的应用分为两个文件 ui.R 和 server.R，或者你的应用全部写在一个文件中。

1. 两文件应用

你的两个文件ui.R和server.R必须放在同一文件夹中。然后你可以通过在控制台运行shinyApp()函数，并传入包含Shiny应用的目录路径来启动应用。

```
shinyApp("path_to_the_folderContaining_the_files")
```

你也可以通过按下 Rstudio 中出现的运行应用程序按钮直接从 Rstudio 启动应用程序，当你打开ui.R或server.R文件时。



或者，如果你的工作目录是 Shiny 应用程序目录，你也可以直接在控制台输入runApp()来启动。

2. 单文件应用程序

如果你在一个R文件中创建应用程序，也可以使用shinyApp()函数启动它。

- 在你的代码中：

```
server <- function(input, output){
  output$text_choice <- renderPrint({
    return(input$id_selectInput)})
}

shinyApp(ui = ui, server = server)
```

What is your favorite color ?

red green blue

yellow

[1] "red" "green" "blue"

It's possible to change the settings :

- label : title
- choices : selected values
- selected : The initially selected value (NULL for no selection)
- multiple : TRUE or FALSE
- width
- size
- selectize: TRUE or FALSE (for use or not selectize.js, change the display)

It is also possible to add HTML.

Section 47.6: Launch a Shiny app

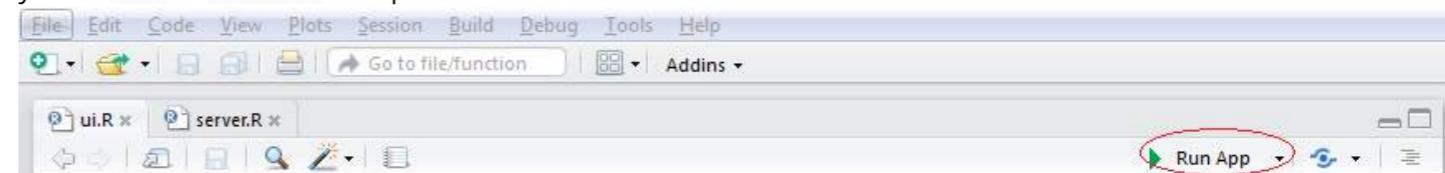
You can launch an application in several ways, depending on how you create your app. If your app is divided in two files ui.R and server.R or if all of your app is in one file.

1. Two files app

Your two files ui.R and server.R have to be in the same folder. You could then launch your app by running in the console the shinyApp() function and by passing the path of the directory that contains the Shiny app.

```
shinyApp("path_to_the_folderContaining_the_files")
```

You can also launch the app directly from Rstudio by pressing the **Run App** button that appears on Rstudio when you an ui.R or server.R file open.



Or you can simply write runApp() on the console if your working directory is Shiny App directory.

2. One file app

If you create your in one R file you can also launch it with the shinyApp() function.

- inside of your code :

```
library(shiny)
```

```
ui <- fluidPage() #创建用户界面  
server <- function(input, output){} #创建服务器  
  
shinyApp(ui = ui, server = server) #运行应用程序
```

- 在控制台中，通过添加包含 Shiny 应用程序的.R文件路径，并使用参数appFile：

```
shinyApp(appFile="path_to_my_R_file_containig_the_app")
```

第47.7节：控件部件

函数	组件
操作按钮	操作按钮
checkboxGroupInput	一组复选框
checkboxInput	单个复选框
dateInput	用于辅助日期选择的日历
dateRangeInput	用于选择日期范围的一对日历
fileInput	文件上传控件向导
helpText	可以添加到输入表单的帮助文本
numericInput	用于输入数字的字段
radioButtons	一组单选按钮
selectInput	一个可供选择的下拉框
sliderInput	滑动条
submitButton	提交按钮
文本输入	用于输入文本的字段

```
library(shiny)
```

```
# 创建用户界面  
ui <- shinyUI(fluidPage(  
  标题面板("基本控件"),  
  
  流式布局行(  
  
    列(3,  
      h3("按钮"),  
      actionButton("action", 标签 = "操作"),  
      br(),  
      br(),  
      submitButton("提交")),  
  
    列(3,  
      h3("单选框"),  
      checkboxInput("checkbox", 标签 = "选项A", 默认值 = TRUE)),  
  
    列(3,  
      checkboxGroupInput("checkGroup",  
        标签 = h3("复选框组"),  
        选项 = list("选项 1" = 1,  
                   "选项 2" = 2, "选项 3" = 3),  
        默认选中 = 1)),  
  
    column(3,  
      dateInput("date",  
        标签 = h3("日期输入"))  
    ))  
  ))  
))
```

```
library(shiny)  
  
ui <- fluidPage() #Create the ui  
server <- function(input, output){} #create the server  
  
shinyApp(ui = ui, server = server) #run the App
```

```
library(shiny)
```

```
ui <- fluidPage() #Create the ui  
server <- function(input, output){} #create the server  
  
shinyApp(ui = ui, server = server) #run the App
```

- in the console by adding path to a .R file containing the Shiny application with the parameter appFile:

```
shinyApp(appFile="path_to_my_R_file_containig_the_app")
```

Section 47.7: Control widgets

Function	Widget
actionButton	Action Button
checkboxGroupInput	A group of check boxes
checkboxInput	A single check box
dateInput	A calendar to aid date selection
dateRangeInput	A pair of calendars for selecting a date range
fileInput	A file upload control wizard
helpText	Help text that can be added to an input form
numericInput	A field to enter numbers
radioButtons	A set of radio buttons
selectInput	A box with choices to select from
sliderInput	A slider bar
submitButton	A submit button
textInput	A field to enter text

```
library(shiny)
```

```
# Create the UI  
ui <- shinyUI(fluidPage(  
  titlePanel("Basic widgets"),  
  
  fluidRow(  
  
    column(3,  
      h3("Buttons"),  
      actionButton("action", label = "Action"),  
      br(),  
      br(),  
      submitButton("Submit")),  
  
    column(3,  
      h3("Single checkbox"),  
      checkboxInput("checkbox", label = "Choice A", value = TRUE)),  
  
    column(3,  
      checkboxGroupInput("checkGroup",  
        label = h3("Checkbox group"),  
        choices = list("Choice 1" = 1,  
                      "Choice 2" = 2, "Choice 3" = 3),  
        selected = 1)),  
  
    column(3,  
      dateInput("date",  
        label = h3("Date input"))  
    ))  
  ))  
))
```

```
默认值 = "2014-01-01"))
),
fluidRow(
列(3,
dateRangeInput("dates", 标签 = h3("日期范围"))),
column(3,
fileInput("file", 标签 = h3("文件输入"))),
column(3,
h3("帮助文本"),
helpText("注意：帮助文本不是一个真正的控件，",
"但它提供了一种简单的方式来添加文本，",
"以配合其他控件使用。")),
列(3,
numericInput("num",
            标签 = h3("数字输入"),
            默认值 = 1)),
),
fluidRow(
列(3,
radioButtons("radio", 标签 = h3("单选按钮"),
            选项 = list("选项 1" = 1, "选项 2" = 2,
                        "选项 3" = 3), 默认选中 = 1)),
column(3,
selectInput("select", 标签 = h3("选择框"),
            选项 = list("选项 1" = 1, "选项 2" = 2,
                        "选项 3" = 3), 默认选中 = 1)),
column(3,
sliderInput("slider1", 标签 = h3("滑块"),
            最小值 = 0, 最大值 = 100, 默认值 = 50),
sliderInput("slider2", "", 
            最小值 = 0, 最大值 = 100, 默认值 = c(25, 75)),
),
列(3,
textInput("text", 标签 = h3("文本输入"),
            默认值 = "请输入文本..."))
)
))

# 创建服务器函数
server <- shinyServer(function(input, output){})

# 运行应用
shinyApp(ui = ui, server = server)
```

```

        value = "2014-01-01"))
),
fluidRow(
  column(3,
    dateRangeInput("dates", label = h3("Date range"))),
  column(3,
    fileInput("file", label = h3("File input"))),
  column(3,
    h3("Help text"),
    helpText("Note: help text isn't a true widget,",
             "but it provides an easy way to add text to",
             "accompany other widgets.")),
  column(3,
    numericInput("num",
      label = h3("Numeric input"),
      value = 1))
),
fluidRow(
  column(3,
    radioButtons("radio", label = h3("Radio buttons"),
      choices = list("Choice 1" = 1, "Choice 2" = 2,
                     "Choice 3" = 3), selected = 1)),
  column(3,
    selectInput("select", label = h3("Select box"),
      choices = list("Choice 1" = 1, "Choice 2" = 2,
                     "Choice 3" = 3), selected = 1)),
  column(3,
    sliderInput("slider1", label = h3("Sliders"),
      min = 0, max = 100, value = 50),
    sliderInput("slider2", "",
      min = 0, max = 100, value = c(25, 75)))
),
column(3,
 textInput("text", label = h3("Text input"),
            value = "Enter text..."))
)
))

# Create the server function
server <- shinyServer(function(input, output){})

# Run the app
shinyApp(ui = ui, server = server)

```

第48章：空间分析

第48.1节：从XY数据集创建空间点

谈到地理数据，R被证明是一个强大的数据处理、分析和可视化工具。

通常，空间数据以表格形式的XY坐标数据集存在。此示例将展示如何从XY数据集创建空间数据集。

软件包 rgdal 和 sp 提供了强大的功能。R中的空间数据可以存储为 Spatial* DataFrame (其中 * 可以是 Points、 Lines 或 Polygons)。

本示例使用的数据可以从 [OpenGeocode](#) 下载。

首先，工作目录必须设置为下载的CSV数据集所在的文件夹。此外，还需要加载软件包 rgdal。

```
setwd("D:/GeocodeExample/")
library(rgdal)
```

随后，将存储城市及其地理坐标的CSV文件作为 data.frame 加载到R中

```
xy <- read.csv("worldcities.csv", stringsAsFactors = FALSE)
```

通常，查看数据及其结构（例如列名、数据类型等）是很有用的。

```
head(xy)
str(xy)
```

这表明纬度和经度列被解释为字符值，因为它们包含类似"-33.532"的条目。然而，后面使用的函数SpatialPointsDataFrame()用于创建空间数据集，要求坐标值的数据类型为numeric。因此，这两列必须进行转换。

```
xy$latitude <- as.numeric(xy$latitude)
xy$longitude <- as.numeric(xy$longitude)
```

部分值无法转换为数值数据，因此生成了NA值。必须将它们移除。

```
xy <- xy[!is.na(xy$longitude),]
```

最后，XY数据集可以转换为空间数据集。这需要坐标以及坐标所存储的坐标参考系统（CRS）的指定。

```
xySPoints <- SpatialPointsDataFrame(coords = c(xy[,c("longitude", "latitude")]),
proj4string = CRS("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"),
data = xy
)
```

基本的绘图函数可以轻松用于快速查看生成的空间点。

```
plot(xySPoints, pch = ".")
```

Chapter 48: spatial analysis

Section 48.1: Create spatial points from XY data set

When it comes to geographic data, R shows to be a powerful tool for data handling, analysis and visualisation.

Often, spatial data is available as an XY coordinate data set in tabular form. This example will show how to create a spatial data set from an XY data set.

The packages rgdal and sp provide powerful functions. Spatial data in R can be stored as Spatial*DataFrame (where * can be Points, Lines or Polygons).

This example uses data which can be downloaded at [OpenGeocode](#).

At first, the working directory has to be set to the folder of the downloaded CSV data set. Furthermore, the package rgdal has to be loaded.

```
setwd("D:/GeocodeExample/")
library(rgdal)
```

Afterwards, the CSV file storing cities and their geographical coordinates is loaded into R as a [data.frame](#)

```
xy <- read.csv("worldcities.csv", stringsAsFactors = FALSE)
```

Often, it is useful to get a glimpse of the data and its structure (e.g. column names, data types etc.).

```
head(xy)
str(xy)
```

This shows that the latitude and longitude columns are interpreted as character values, since they hold entries like "-33.532". Yet, the later used function SpatialPointsDataFrame() which creates the spatial data set requires the coordinate values to be of the data type [numeric](#). Thus the two columns have to be converted.

```
xy$latitude <- as.numeric(xy$latitude)
xy$longitude <- as.numeric(xy$longitude)
```

Few of the values cannot be converted into numeric data and thus, NA values are created. They have to be removed.

```
xy <- xy[!is.na(xy$longitude),]
```

Finally, the XY data set can be converted into a spatial data set. This requires the coordinates and the specification of the Coordinate Reference System (CRS) in which the coordinates are stored.

```
xySPoints <- SpatialPointsDataFrame(coords = c(xy[,c("longitude", "latitude")]),
proj4string = CRS("+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs"),
data = xy
)
```

The basic plot function can easily be used to sneak peak the produced spatial points.

```
plot(xySPoints, pch = ".")
```



第48.2节：导入形状文件 (.shp)

rgdal

可以使用readOGR()函数从 rgdal包中轻松导入ESRI shape文件。

```
library(rgdal)  
shp <- readOGR(dsn = "/path/to/your/file", layer = "filename")
```

需要注意的是，dsn不能以/结尾，且layer不允许包含文件后缀（例如.shp）

raster

另一种导入shapefile的方法是通过 raster库和 shapefile函数：

```
library(raster)  
shp <- shapefile("path/to/your/file.shp")
```

注意路径定义与rgdal导入语句的不同。

tmap

tmap包为 rgdal::readORG函数提供了一个很好的封装。

```
library(tmap)  
shp <- read_shape("path/to/your/file.shp")
```

Section 48.2: Importing a shape file (.shp)

rgdal

ESRI shape files can easily be imported into R by using the function readOGR() from the rgdal package.

```
library(rgdal)  
shp <- readOGR(dsn = "/path/to/your/file", layer = "filename")
```

It is important to know, that the dsn must not end with / and the layer does not allow the file ending (e.g. .shp)

raster

Another possible way of importing shapefiles is via the raster library and the shapefile function:

```
library(raster)  
shp <- shapefile("path/to/your/file.shp")
```

Note how the path definition is different from the rgdal import statement.

tmap

tmap package provides a nice wrapper for the rgdal::readORG function.

```
library(tmap)  
shp <- read_shape("path/to/your/file.shp")
```

第49章 : sqldf

第49.1节：基本用法示例

`sqldf()` 来自包 `sqldf`, 允许在R中使用SQLite查询来选择和操作数据。SQL查询以字符字符串形式输入。

例如，要选择包 `ggplot2` 中“diamonds”数据集的前10行：

```
data("diamonds")
head(diamonds)
```

# 一个6行10列的tibble											
		克拉	切工	颜色	净度	深度	台宽	价格	x	y	z
		<dbl>	<ord>	<ord>	<ord>	<dbl>	<dbl>	<int>	<dbl>	<dbl>	<dbl>
	1	0.23	理想	E	SI2	61.5	55	326	3.95	3.98	2.43
2	0.21	优质	E	SI1	59.8	61	326	3.89	3.84	2.31	
3	0.23	好	E	VS1	56.9	65	327	4.05	4.07	2.31	
4	0.29	优质	I	VS2	62.4	58	334	4.20	4.23	2.63	
5	0.31	好	J	SI2	63.3	58	335	4.34	4.35	2.75	
6	0.24	非常好	J	VVS2	62.8	57	336	3.94	3.96	2.48	

```
require(sqldf)  
sqldf("select * from diamonds limit 10")
```

克拉		切工	颜色	净度	深度	台面	价格	x	y	z
1	0.23	理想	E	SI2	61.5	55	326	3.95	3.98	2.43
2	0.21	优质	E	SI1	59.8	61	326	3.89	3.84	2.31
3	0.23	好	E	VS1	56.9	65	327	4.05	4.07	2.31
4	0.29	优质	I	VS2	62.4	58	334	4.20	4.23	2.63
5	0.31	好	J	SI2	63.3	58	335	4.34	4.35	2.75
6	0.24	非常好	J	VVS2	62.8	57	336	3.94	3.96	2.48
7	0.24	非常好	I	VVS1	62.3	57	336	3.95	3.98	2.47
8	0.26	非常好	H	SI1	61.9	55	337	4.07	4.11	2.53
9	0.22	一般	E	VS2	65.1	61	337	3.87	3.78	2.49
10	0.23	非常好	H	VS1	59.4	61	338	4.00	4.05	2.39

选择颜色为“E”的前10行：

```
sqldf("select * from diamonds where color = 'E' limit 10")
```

克拉		切工	颜色	净度	深度	台面	价格	x	y	z
1	0.23	理想	E	SI2	61.5	55	326	3.95	3.98	2.43
2	0.21	优质	E	SI1	59.8	61	326	3.89	3.84	2.31
3	0.23	好	E	VS1	56.9	65	327	4.05	4.07	2.31
4	0.22	一般	E	VS2	65.1	61	337	3.87	3.78	2.49
5	0.20	优质	E	SI2	60.2	62	345	3.79	3.75	2.27
6	0.32	优质	E	I1	60.9	58	345	4.38	4.42	2.68
7	0.23	非常好	E	VS2	63.8	55	352	3.85	3.92	2.48
8	0.23	非常好	E	VS1	60.7	59	402	3.97	4.01	2.42
9	0.23	非常好	E	VS1	59.5	58	402	4.01	4.06	2.40
10	0.23	好	E	VS1	64.1	59	402	3.83	3.85	2.46

注意上面示例中，SQL查询中的引号字符串如果整体查询用“”引起起来，则内部用“”引起起来（反之亦然）。

Chapter 49: sqldf

Section 49.1: Basic Usage Examples

`sqldf()` from the package `sqldf` allows the use of SQLite queries to select and manipulate data in R. SQL queries are entered as character strings.

To select the first 10 rows of the "diamonds" dataset from the package `ggplot2`, for example:

```
data("diamonds")
head(diamonds)
```

```
# A tibble: 6 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
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
6 0.24 Very Good J     VVS2  62.8   57   336  3.94  3.96  2.48
```

```
require(sqldf)  
sqldf("select * from diamonds limit 10")
```

	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
6	0.24	Very Good	J	VVS2	62.8	57	336	3.94	3.96	2.48
7	0.24	Very Good	I	VVS1	62.3	57	336	3.95	3.98	2.47
8	0.26	Very Good	H	SI1	61.9	55	337	4.07	4.11	2.53
9	0.22	Fair	E	VS2	65.1	61	337	3.87	3.78	2.49
10	0.23	Very Good	H	VS1	59.4	61	338	4.00	4.05	2.39

To select the first 10 rows where for the color "E":

```
sqldf("select * from diamonds where color = 'E' limit 10")
```

	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	0.22	Fair	E	VS2	65.1	61	337	3.87	3.78	2.49
5	0.20	Premium	E	SI2	60.2	62	345	3.79	3.75	2.27
6	0.32	Premium	E	I1	60.9	58	345	4.38	4.42	2.68
7	0.23	Very Good	E	VS2	63.8	55	352	3.85	3.92	2.48
8	0.23	Very Good	E	VS1	60.7	59	402	3.97	4.01	2.42
9	0.23	Very Good	E	VS1	59.5	58	402	4.01	4.06	2.40
10	0.23	Good	E	VS1	64.1	59	402	3.83	3.85	2.46

Notice in the example above that quoted strings within the SQL query are quoted using " if the overall query is quoted with "" (this also works in reverse).

假设我们想添加一列，用于统计克拉数超过1且为优质切工的钻石数量：

```
sqldf("select count(*) from diamonds where carat > 1 and color = 'E'")
```

```
count(*)  
1 1892
```

创建的值结果也可以作为新列返回：

```
sqldf("select *, count(*) as cnt_big_E_colored_stones from diamonds where carat > 1 and color = 'E'  
group by clarity")
```

克拉	切工	颜色	净度	深度	台宽	价格	长	宽	高	cnt_big_E_colored_stones
1 1.30	一般	E	I1	66.5	58	2571	6.79	6.75	4.50	65
2 1.28	理想	E	IF	60.7	57	18700	7.09	6.99	4.27	28
3 2.02	很好	E	SI1	59.8	59	18731	8.11	8.20	4.88	499
4 2.03	优质	E	SI2	61.5	59	18477	8.24	8.16	5.04	666
5 1.51	理想	E	VS1	61.5	57	18729	7.34	7.40	4.53	158
6 1.72	很好	E	VS2	63.4	56	18557	7.65	7.55	4.82	318
7 1.20	理想	E	VVS1	61.8	56	16256	6.78	6.87	4.22	52
8 1.55	理想	E	VVS2	62.5	55	18188	7.38	7.40	4.62	106

如果有人想知道根据切工的最大价格是多少：

```
sqldf("select cut, max(price) from diamonds group by cut")
```

切工	最大(价格)
1一般	18574
2好	18788
3理想	18806
4优质	18823
5非常好	18818

Suppose that we wish to add a new column to count the number of Premium cut diamonds over 1 carat:

```
sqldf("select count(*) from diamonds where carat > 1 and color = 'E'")
```

```
count(*)  
1 1892
```

Results of created values can also be returned as new columns:

```
sqldf("select *, count(*) as cnt_big_E_colored_stones from diamonds where carat > 1 and color = 'E'  
group by clarity")
```

carat	cut	color	clarity	depth	table	price	x	y	z	cnt_big_E_colored_stones
1 1.30	Fair	E	I1	66.5	58	2571	6.79	6.75	4.50	65
2 1.28	Ideal	E	IF	60.7	57	18700	7.09	6.99	4.27	28
3 2.02	Very Good	E	SI1	59.8	59	18731	8.11	8.20	4.88	499
4 2.03	Premium	E	SI2	61.5	59	18477	8.24	8.16	5.04	666
5 1.51	Ideal	E	VS1	61.5	57	18729	7.34	7.40	4.53	158
6 1.72	Very Good	E	VS2	63.4	56	18557	7.65	7.55	4.82	318
7 1.20	Ideal	E	VVS1	61.8	56	16256	6.78	6.87	4.22	52
8 1.55	Ideal	E	VVS2	62.5	55	18188	7.38	7.40	4.62	106

If one would be interested what is the **max** price of the diamond **according** to the **cut**:

```
sqldf("select cut, max(price) from diamonds group by cut")
```

cut	max(price)
1 Fair	18574
2 Good	18788
3 Ideal	18806
4 Premium	18823
5 Very Good	18818

第50章：代码性能分析

第50.1节：使用微基准测试进行基准比较

你可以使用microbenchmark包来进行“亚毫秒级准确的表达式评估计时”。

在这个例子中，我们比较了六个等效的data.table表达式在基于某个条件更新分组元素时的速度。

更具体地说：

一个包含3列的数据表：id、time和status。对于每个id，我想找到时间最大的记录——然后如果该记录的状态为true，且时间大于7，我想将其设置为false。

```
library(microbenchmark)
library(data.table)

set.seed(20160723)
dt <- data.table(id = c(rep(seq(1:10000), 每个 = 10)),
                  time = c(rep(seq(1:10000), 10)),
                  status = c(sample(c(TRUE, FALSE), 10000*10, 替换 = TRUE)))setkey(dt, id, time) ## 创建副本，以便“按引用更新”不会影响其他表达式

dt1 <- copy(dt)
dt2 <- copy(dt)
dt3 <- copy(dt)
dt4 <- copy(dt)
dt5 <- copy(dt)
dt6 <- copy(dt)

microbenchmark()

expression_1 = {
  dt1[ dt1[order(time), .I[.N], by = id]$V1, status := status * time < 7 ]
}

expression_2 = {
  dt2[, status := c(.SD[-.N, status], .SD[.N, status * time > 7]), by = id]
}

expression_3 = {
  dt3[dt3[,.N, by = id][,cumsum(N)], status := status * time > 7]
}

expression_4 = {
  y <- dt4[, .SD[.N], by=id]
  dt4[y, status := status & time > 7]
}

expression_5 = {
  y <- dt5[, .SD[.N, .(time, status)], by = id][time > 7 & status]
  dt5[y, status := FALSE]
}

expression_6 = {
  dt6[ dt6[, .I == .I[which.max(time)], by = id]$V1 & time > 7, status := FALSE]
}
```

Chapter 50: Code profiling

Section 50.1: Benchmarking using microbenchmark

You can use the [microbenchmark](#) package to conduct "sub-millisecond accurate timing of expression evaluation".

In [this example](#) we are comparing the speeds of six equivalent data.table expressions for updating elements in a group, based on a certain condition.

More specifically:

A data.table with 3 columns: id, time and status. For each id, I want to find the record with the maximum time - then if for that record if the status is true, I want to set it to false if the time is > 7

```
library(microbenchmark)
library(data.table)

set.seed(20160723)
dt <- data.table(id = c(rep(seq(1:10000), each = 10)),
                  time = c(rep(seq(1:10000), 10)),
                  status = c(sample(c(TRUE, FALSE), 10000*10, replace = TRUE)))
setkey(dt, id, time) ## create copies of the data so the 'updates-by-reference' don't affect other expressions
dt1 <- copy(dt)
dt2 <- copy(dt)
dt3 <- copy(dt)
dt4 <- copy(dt)
dt5 <- copy(dt)
dt6 <- copy(dt)

microbenchmark()

expression_1 = {
  dt1[ dt1[order(time), .I[.N], by = id]$V1, status := status * time < 7 ]
}

expression_2 = {
  dt2[, status := c(.SD[-.N, status], .SD[.N, status * time > 7]), by = id]
}

expression_3 = {
  dt3[dt3[,.N, by = id][,cumsum(N)], status := status * time > 7]
}

expression_4 = {
  y <- dt4[, .SD[.N], by=id]
  dt4[y, status := status & time > 7]
}

expression_5 = {
  y <- dt5[, .SD[.N, .(time, status)], by = id][time > 7 & status]
  dt5[y, status := FALSE]
}

expression_6 = {
  dt6[ dt6[, .I == .I[which.max(time)], by = id]$V1 & time > 7, status := FALSE]
}
```

```

times = 10L ## 指定每个表达式评估的次数
)

# 单位：毫秒
#      expr      min       lq     mean   median      uq     max neval
# expression_1 11.646149 13.201670 16.808399 15.643384 18.78640 26.321346    10
# expression_2 8051.898126 8777.016935 9238.323459 8979.553856 9281.93377 12610.869058    10
# expression_3  3.208773  3.385841  4.207903  4.089515  4.70146  5.654702    10
# expression_4 15.758441 16.247833 20.677038 19.028982 21.04170 36.373153    10
# expression_5 7552.970295 8051.080753 8702.064620 8861.608629 9308.62842 9722.234921    10
# expression_6 18.403105 18.812785 22.427984 21.966764 24.66930 28.607064    10

```

输出显示在此测试中，expression_3 是最快的。

参考文献

[data.table - 添加和修改列](#)

[data.table - data.table中的特殊分组符号](#)

第50.2节 : proc.time()

最简单的情况下，proc.time() 返回当前进程的总CPU运行时间（秒）。在控制台执行时会得到如下类型的输出：

proc.time()

```

#      user      system      elapsed
# 284.507 120.397 515029.305

```

这对于基准测试特定代码行特别有用。例如：

```

t1 <- proc.time()
fibb <- function (n) {
  if (n < 3) {
    return(c(0,1)[n])
  } else {
    return(fibb(n - 2) + fibb(n - 1))
  }
}
print("时间一")
print(proc.time() - t1)

t2 <- proc.time()
fibb(30)

print("时间二")
print(proc.time() - t2)

```

这将产生以下输出：

```

source('~/active-rstudio-document')

# [1] "时间一"
#   用户 系统 经过时间
#   0     0     0
#
# [1] "时间二"

```

```

times = 10L ## specify the number of times each expression is evaluated
)
```

Unit: milliseconds

```

#      expr      min       lq     mean   median      uq     max neval
# expression_1 11.646149 13.201670 16.808399 15.643384 18.78640 26.321346    10
# expression_2 8051.898126 8777.016935 9238.323459 8979.553856 9281.93377 12610.869058    10
# expression_3  3.208773  3.385841  4.207903  4.089515  4.70146  5.654702    10
# expression_4 15.758441 16.247833 20.677038 19.028982 21.04170 36.373153    10
# expression_5 7552.970295 8051.080753 8702.064620 8861.608629 9308.62842 9722.234921    10
# expression_6 18.403105 18.812785 22.427984 21.966764 24.66930 28.607064    10

```

The output shows that in this test expression_3 is the fastest.

References

[data.table - Adding and modifying columns](#)

[data.table - special grouping symbols in data.table](#)

Section 50.2: proc.time()

At its simplest, `proc.time()` gives the total elapsed CPU time in seconds for the current process. Executing it in the console gives the following type of output:

proc.time()

```

#      user      system      elapsed
# 284.507 120.397 515029.305

```

This is particularly useful for benchmarking specific lines of code. For example:

```

t1 <- proc.time()
fibb <- function (n) {
  if (n < 3) {
    return(c(0,1)[n])
  } else {
    return(fibb(n - 2) + fibb(n - 1))
  }
}
print("Time one")
print(proc.time() - t1)

t2 <- proc.time()
fibb(30)

print("Time two")
print(proc.time() - t2)

```

This gives the following output:

```

source('~/active-rstudio-document')

# [1] "Time one"
#   user      system      elapsed
#   0        0        0
#
# [1] "Time two"

```

```
# 用户 系统 经过时间  
# 1.534 0.012 1.572
```

system.time() 是 proc.time() 的一个封装，返回特定命令/表达式的经过时间。

```
print(t1 <- system.time(replicate(1000,12^2)))  
## 用户 系统 经过时间  
## 0.000 0.000 0.002
```

请注意，返回的对象，类为proc.time，比表面看起来稍微复杂一些：

```
str(t1)  
## 类 'proc_time' 命名数值 [1:5] 0 0 0.002 0 0  
## ..- 属性(*, "names")= 字符 [1:5] "user.self" "sys.self" "elapsed" "user.child" ...
```

第50.3节：微基准测试

微基准测试对于估计本来非常快的过程所需时间非常有用。例如，考虑估计打印hello world所需的时间。

```
system.time(print("hello world"))  
  
# [1] "hello world"  
#   用户 系统 经过时间  
#       0       0       0
```

这是因为 system.time本质上是proc.time的一个包装函数，后者以秒为单位测量时间。由于打印"hello world"所需时间不到一秒，因此看起来所用时间少于一秒，然而这并不真实。为了验证这一点，我们可以使用microbenchmark包：

```
library(microbenchmark)  
microbenchmark(print("hello world"))  
  
# 单位：微妙  
#          表达式 最小    下四分位   平均 中位数 上四分位 最大 评估次数  
# print("hello world") 26.336 29.984 44.11637 44.6835 45.415 158.824 100
```

这里我们可以看到，在运行print("hello world") 100次后，平均耗时实际上是44微妙。（注意，运行此代码将在控制台上打印100次"hello world"。）

我们可以将其与等效的过程 cat("hello world") 进行比较，以查看它是否比print("hello world") 更快：

```
microbenchmark(cat("hello world"))# 单位：  
  
微妙  
#          expr 最小    下四分位   平均 中位数 上四分位 最大 评估次数  
# cat("hello world") 14.093 17.6975 23.73829 19.319 20.996 119.382 100
```

在这种情况下，cat() 的速度几乎是 print() 的两倍。

或者，可以在同一个 microbenchmark 调用中比较两个过程：

```
microbenchmark(print("hello world"), cat("hello world"))# 单位：微妙  
  
# 表达式 最小    下四分位   平均 中位数 上四分位 最大 评估次数
```

```
#   user  system elapsed  
# 1.534 0.012 1.572
```

system.time() is a wrapper for proc.time() that returns the elapsed time for a particular command/expression.

```
print(t1 <- system.time(replicate(1000,12^2)))  
## user  system elapsed  
## 0.000 0.000 0.002
```

Note that the returned object, of class proc.time, is slightly more complicated than it appears on the surface:

```
str(t1)  
## Class 'proc_time' Named num [1:5] 0 0 0.002 0 0  
## ..- attr(*, "names")= chr [1:5] "user.self" "sys.self" "elapsed" "user.child" ...
```

Section 50.3: Microbenchmark

Microbenchmark is useful for estimating the time taking for otherwise fast procedures. For example, consider estimating the time taken to print hello world.

```
system.time(print("hello world"))  
  
# [1] "hello world"  
#   user  system elapsed  
#       0       0       0
```

This is because system.time is essentially a wrapper function for proc.time, which measures in seconds. As printing "hello world" takes less than a second it appears that the time taken is less than a second, however this is not true. To see this we can use the package microbenchmark:

```
library(microbenchmark)  
microbenchmark(print("hello world"))  
  
# Unit: microseconds  
#          expr      min       lq     mean   median      uq      max neval  
# print("hello world") 26.336 29.984 44.11637 44.6835 45.415 158.824 100
```

Here we can see after running print("hello world") 100 times, the average time taken was in fact 44 microseconds. (Note that running this code will print "hello world" 100 times onto the console.)

We can compare this against an equivalent procedure, cat("hello world\n"), to see if it is faster than print("hello world"):

```
microbenchmark(cat("hello world\n"))  
  
# Unit: microseconds  
#          expr      min       lq     mean   median      uq      max neval  
# cat("hello world\\n") 14.093 17.6975 23.73829 19.319 20.996 119.382 100
```

In this case cat() is almost twice as fast as print().

Alternatively one can compare two procedures within the same microbenchmark call:

```
microbenchmark(print("hello world"), cat("hello world\\n"))  
# Unit: microseconds  
#          expr      min       lq     mean   median      uq      max neval
```

```
# print("hello world") 29.122 31.654 39.64255 34.5275 38.852 192.779 100# cat("hello  
world\\") 9.381 12.356 13.83820 12.9930 13.715 52.564 100
```

第50.4节 : System.time

System.time 给出执行 R 表达式所需的 CPU 时间，例如：

```
system.time(print("hello world"))

# [1] "hello world"
#   user  system elapsed
#       0       0       0
```

你可以通过使用大括号添加更大块的代码：

```
system.time({
  library(numbers)
  Primes(1,10^5)
})
```

或者用它来测试函数：

```
fibb <- function (n) {
  if (n < 3) {
    return(c(0,1)[n])
  } else {
    return(fibb(n - 2) + fibb(n -1))
  }
}

system.time(fibb(30))
```

第50.5节 : 行级性能分析

一个用于行级性能分析的包是lineprof，由Hadley Wickham编写和维护。下面是它如何与forecast包中的auto.arima一起使用的快速演示：

```
library(lineprof)
library(forecast)

l <- lineprof(auto.arima(AirPassengers))
shine(l)
```

这将为你提供一个shiny应用，允许你深入查看每个函数调用。这样你就可以轻松地看到是什么导致你的R代码变慢。下面是shiny应用的截图：

```
# print("hello world") 29.122 31.654 39.64255 34.5275 38.852 192.779 100
# cat("hello world\\n") 9.381 12.356 13.83820 12.9930 13.715 52.564 100
```

Section 50.4: System.time

System time gives you the CPU time required to execute a R expression, for example:

```
system.time(print("hello world"))

# [1] "hello world"
#   user  system elapsed
#       0       0       0
```

You can add larger pieces of code through use of braces:

```
system.time({
  library(numbers)
  Primes(1,10^5)
})
```

Or use it to test functions:

```
fibb <- function (n) {
  if (n < 3) {
    return(c(0,1)[n])
  } else {
    return(fibb(n - 2) + fibb(n -1))
  }
}

system.time(fibb(30))
```

Section 50.5: Line Profiling

One package for line profiling is [lineprof](#) which is written and maintained by Hadley Wickham. Here is a quick demonstration of how it works with auto.arima in the forecast package:

```
library(lineprof)
library(forecast)

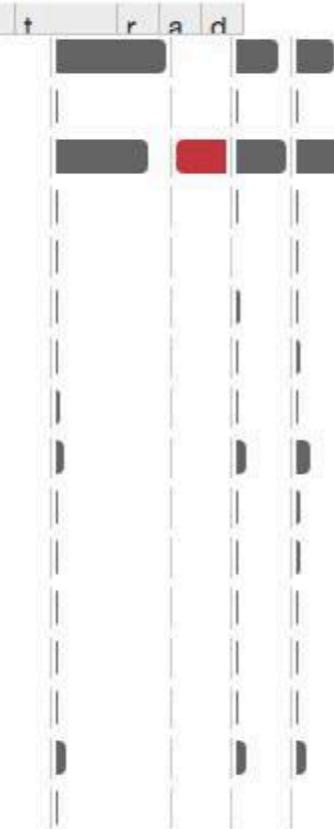
l <- lineprof(auto.arima(AirPassengers))
shine(l)
```

This will provide you with a shiny app, which allows you to delve deeper into every function call. This enables you to see with ease what is causing your R code to slow down. There is a screenshot of the shiny app below:

Line profiling

Back

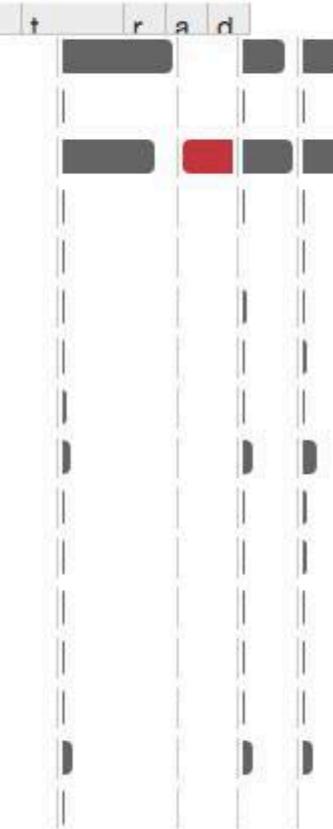
#	Source code
1	nsdiffs/OCSBtest
2	nsdiffs/diff
3	nsdiffs/OCSBtest
4	diff/diff.ts
5	ndiffs/suppressWarnings
6	ndiffs/diff
7	diff/diff.ts
8	try/tryCatch
9	myarima/suppressWarnings
10	
11	myarima/suppressWarnings
12	myarima
13	myarima/suppressWarnings
14	
15	myarima/suppressWarnings
16	data.frame



Line profiling

Back

#	Source code
1	nsdiffs/OCSBtest
2	nsdiffs/diff
3	nsdiffs/OCSBtest
4	diff/diff.ts
5	ndiffs/suppressWarnings
6	ndiffs/diff
7	diff/diff.ts
8	try/tryCatch
9	myarima/suppressWarnings
10	
11	myarima/suppressWarnings
12	myarima
13	myarima/suppressWarnings
14	
15	myarima/suppressWarnings
16	data.frame



第51章：控制流结构

第51.1节：for循环的最佳构造

为了说明良好for循环构造的效果，我们将用四种不同的方法计算每列的平均值：

1. 使用优化较差的for循环
2. 使用优化良好的for循环
3. 使用*`apply`函数族
4. 使用`colMeans`函数

将展示每种选项的代码；比较执行每个选项的计算时间；最后给出差异的讨论。

优化较差的for循环

```
column_mean_poor <- NULL
for (i in 1:length(mtcars)){
  column_mean_poor[i] <- mean(mtcars[[i]])
}
```

优化良好的for循环

```
column_mean_optimal <- vector("numeric", length(mtcars))
for (i in seq_along(mtcars)){
  column_mean_optimal <- mean(mtcars[[i]])
}
```

vapply 函数

```
column_mean_vapply <- vapply(mtcars, mean, numeric(1))
```

colMeans 函数

```
column_mean_colMeans <- colMeans(mtcars)
```

效率比较

以下显示了这四种方法的基准测试结果（代码未显示）

单位: 微秒

expr	min	lq	mean	median	uq	max	neval	cld
poor	240.986	262.0820	287.1125	275.8160	307.2485	442.609	100	d
optimal	220.313	237.4455	258.8426	247.0735	280.9130	362.469	100	c
vapply	107.042	109.7320	124.4715	113.4130	132.6695	202.473	100	a
colMeans	155.183	161.6955	180.2067	175.0045	194.2605	259.958	100	b

注意，优化后的for循环略胜于构造不佳的for循环。构造不佳的for循环不断增加输出对象的长度，而每次长度变化时，R都会重新评估该对象的类。

通过在开始循环之前声明输出对象的类型及其长度，优化的for循环去除了一部分开销负担。

然而，在此示例中，使用vapply函数使计算效率提高了一倍，这主要是因为我们

Chapter 51: Control flow structures

Section 51.1: Optimal Construction of a For Loop

To illustrate the effect of good for loop construction, we will calculate the mean of each column in four different ways:

1. Using a poorly optimized for loop
2. Using a well optimized for loop
3. Using an *`apply` family of functions
4. Using the `colMeans` function

Each of these options will be shown in code; a comparison of the computational time to execute each option will be shown; and lastly a discussion of the differences will be given.

Poorly optimized for loop

```
column_mean_poor <- NULL
for (i in 1:length(mtcars)){
  column_mean_poor[i] <- mean(mtcars[[i]])
}
```

Well optimized for loop

```
column_mean_optimal <- vector("numeric", length(mtcars))
for (i in seq_along(mtcars)){
  column_mean_optimal <- mean(mtcars[[i]])
}
```

vapply Function

```
column_mean_vapply <- vapply(mtcars, mean, numeric(1))
```

colMeans Function

```
column_mean_colMeans <- colMeans(mtcars)
```

Efficiency comparison

The results of benchmarking these four approaches is shown below (code not displayed)

Unit: microseconds

expr	min	lq	mean	median	uq	max	neval	cld
poor	240.986	262.0820	287.1125	275.8160	307.2485	442.609	100	d
optimal	220.313	237.4455	258.8426	247.0735	280.9130	362.469	100	c
vapply	107.042	109.7320	124.4715	113.4130	132.6695	202.473	100	a
colMeans	155.183	161.6955	180.2067	175.0045	194.2605	259.958	100	b

Notice that the optimized `for` loop edged out the poorly constructed for loop. The poorly constructed for loop is constantly increasing the length of the output object, and at each change of the length, R is reevaluating the class of the object.

Some of this overhead burden is removed by the optimized for loop by declaring the type of output object and its length before starting the loop.

In this example, however, the use of an `vapply` function doubles the computational efficiency, largely because we

告诉R结果必须是数值型（如果有任何一个结果不是数值型，将返回错误）。

使用`colMeans`函数比`vapply`函数稍慢一些。这种差异归因于`colMeans`中执行的一些错误检查，主要是因为`mtcars`是一个`data.frame`。`colMeans`中进行了`as.matrix`转换，而`vapply`函数中没有执行这些操作。

第51.2节：基本的For循环构造

在此示例中，我们将计算数据框中每一列的平方偏差，这里以`mtcars`为例。

选项A：整数索引

```
squared_deviance <- vector("list", length(mtcars))
for (i in seq_along(mtcars)){
  squared_deviance[[i]] <- (mtcars[[i]] - mean(mtcars[[i]]))^2
}
```

`squared_deviance`是一个包含11个元素的列表，符合预期。

```
class(squared_deviance)
length(squared_deviance)
```

选项 B：字符索引

```
squared_deviance <- vector("list", length(mtcars))
Squared_deviance <- setNames(squared_deviance, names(mtcars))
for (k in names(mtcars)){
  squared_deviance[[k]] <- (mtcars[[k]] - mean(mtcars[[k]]))^2
}
```

如果我们想要一个`data.frame`作为结果呢？其实有很多方法可以将列表转换成其他对象。不过，在这种情况下，也许最简单的方法是将for循环的结果存储在一个`data.frame`中。

```
squared_deviance <- mtcars #复制原始数据
squared_deviance[TRUE] <- NA #替换为NA，或者使用squared_deviance[,] <- NA
for (i in seq_along(mtcars)){
  squared_deviance[[i]] <- (mtcars[[i]] - mean(mtcars[[i]]))^2
}
dim(squared_deviance)
[1] 32 11
```

即使我们使用字符选项（B），结果也将是相同的。

第51.3节：其他循环结构：`while`和`repeat`

R 提供了另外两种循环结构，`while` 和 `repeat`，通常用于需要不确定次数迭代的情况。

while 循环

`while` 循环的一般形式如下，

```
while (condition) {
  ## 执行某些操作
  ## 在循环体内
```

told R that the result had to be numeric (if any one result were not numeric, an error would be returned).

Use of the `colMeans` function is a touch slower than the `vapply` function. This difference is attributable to some error checks performed in `colMeans` and mainly to the `as.matrix` conversion (because `mtcars` is a `data.frame`) that weren't performed in the `vapply` function.

Section 51.2: Basic For Loop Construction

In this example we will calculate the squared deviance for each column in a data frame, in this case the `mtcars`.

Option A: integer index

```
squared_deviance <- vector("list", length(mtcars))
for (i in seq_along(mtcars)){
  squared_deviance[[i]] <- (mtcars[[i]] - mean(mtcars[[i]]))^2
}
```

`squared_deviance` is an 11 elements list, as expected.

```
class(squared_deviance)
length(squared_deviance)
```

Option B: character index

```
squared_deviance <- vector("list", length(mtcars))
Squared_deviance <- setNames(squared_deviance, names(mtcars))
for (k in names(mtcars)){
  squared_deviance[[k]] <- (mtcars[[k]] - mean(mtcars[[k]]))^2
}
```

What if we want a `data.frame` as a result? Well, there are many options for transforming a list into other objects. However, and maybe the simplest in this case, will be to store the `for` results in a `data.frame`.

```
squared_deviance <- mtcars #copy the original
squared_deviance[TRUE] <- NA #replace with NA or do squared_deviance[,] <- NA
for (i in seq_along(mtcars)){
  squared_deviance[[i]] <- (mtcars[[i]] - mean(mtcars[[i]]))^2
}
dim(squared_deviance)
[1] 32 11
```

The result will be the same even though we use the character option (B).

Section 51.3: The Other Looping Constructs: `while` and `repeat`

R provides two additional looping constructs, `while` and `repeat`, which are typically used in situations where the number of iterations required is indeterminate.

The `while` loop

The general form of a `while` loop is as follows,

```
while (condition) {
  ## do something
  ## in loop body
```

```
}
```

其中 condition 在进入循环体之前进行评估。如果 condition 计算结果为 TRUE，循环体内的代码将被执行，该过程会重复，直到 condition 计算结果为 FALSE（或遇到 break 语句；见下文）。

与 for 循环不同，如果 while 循环使用变量进行递增迭代，该变量必须提前声明和初始化，并且必须在循环体内更新。例如，以下循环完成相同的任务：

```
for (i in 0:4) {cat(i,
  "")}
```

```
# 0
# 1
# 2
# 3
# 4
```

```
i <- 0
while (i < 5) {cat(i
  , "") i <- i
+ 1}
```

```
# 0
# 1
# 2
# 3
# 4
```

在上面的 while 循环中，语句 `i <- i + 1` 是防止无限循环所必需的。

此外，可以通过在循环体内调用 break 来终止 while 循环：

```
iter <- 0
while (TRUE) {
  if (runif(1) < 0.25) {
    break
  } else {
    iter <- iter + 1
  }
}
迭代
#[1] 4
```

在此示例中，condition 始终为 TRUE，因此终止循环的唯一方法是在循环体内调用 break。请注意，iter 的最终值将取决于运行此示例时您的伪随机数生成器（PRNG）的状态，并且每次执行代码时应产生不同的结果（本质上）。

repeat 循环

repeat 结构本质上与 `while (TRUE) { ## something }` 相同，形式如下：

```
repeat ({
  ## 执行某些操作
  ## 在循环体内
})
```

额外的 {} 不是必需的，但 () 是必需的。使用 repeat 重写前面的示例，

```
}
```

where condition is evaluated prior to entering the loop body. If condition evaluates to TRUE, the code inside of the loop body is executed, and this process repeats until condition evaluates to FALSE (or a `break` statement is reached; see below). Unlike the `for` loop, if a `while` loop uses a variable to perform incremental iterations, the variable must be declared and initialized ahead of time, and must be updated within the loop body. For example, the following loops accomplish the same task:

```
for (i in 0:4) {
  cat(i, "\n")
}
# 0
# 1
# 2
# 3
# 4
```

```
i <- 0
while (i < 5) {
  cat(i, "\n")
  i <- i + 1
}
# 0
# 1
# 2
# 3
# 4
```

In the `while` loop above, the line `i <- i + 1` is necessary to prevent an infinite loop.

Additionally, it is possible to terminate a `while` loop with a call to `break` from inside the loop body:

```
iter <- 0
while (TRUE) {
  if (runif(1) < 0.25) {
    break
  } else {
    iter <- iter + 1
  }
}
iter
#[1] 4
```

In this example, condition is always TRUE, so the only way to terminate the loop is with a call to `break` inside the body. Note that the final value of iter will depend on the state of your PRNG when this example is run, and should produce different results (essentially) each time the code is executed.

The repeat loop

The `repeat` construct is essentially the same as `while (TRUE) { ## something }`, and has the following form:

```
repeat ({
  ## do something
  ## in loop body
})
```

The extra {} are not required, but the () are. Rewriting the previous example using `repeat`,

```

iter <- 0
repeat ({
  if (runif(1) < 0.25) {
    break
  } else {
  iter <- iter + 1
  }
})
iter
#[1] 2

```

关于 break 的更多内容

需要注意的是，`break` 只会终止最内层的循环。也就是说，下面的代码是一个无限循环：

```

while (TRUE) {wh
  ile (TRUE) {cat("内层循环")break
}
cat("外层循环")

```

不过，通过一些创造性地方法，可以实现从嵌套循环中完全跳出。例如，考虑下面的表达式，其当前状态下将无限循环：

```

while (TRUE) {cat
  ("外层循环体")while (TRUE) {
    cat("内层循环体")      x <-
runif(1)if (x < .3) {
  break
} else {
  cat(sprintf("x 的值是 %.5f", x))
}
}

```

一种可能的方法是认识到，与`break`不同，`return`表达式确实有能力跨过多层循环返回控制权。然而，由于`return`仅在函数内部有效，我们不能简单地将上面的`break`替换为`return()`，还需要将整个表达式包装成一个匿名函数：

```

(function() {
  while (TRUE) {
    cat("外层循环体")while (TRUE)
    {
      cat("内层循环体")      x
<- runif(1)if (x < .3) {
        return()
} else {
  cat(sprintf("x 的值是 %.5f", x))
}
}
}

```

```

iter <- 0
repeat ({
  if (runif(1) < 0.25) {
    break
  } else {
    iter <- iter + 1
  }
})
iter
#[1] 2

```

More on `break`

It's important to note that `break` will *only terminate the immediately enclosing loop*. That is, the following is an infinite loop:

```

while (TRUE) {
  while (TRUE) {
    cat("inner loop\n")
    break
  }
  cat("outer loop\n")
}

```

With a little creativity, however, it is possible to break entirely from within a nested loop. As an example, consider the following expression, which, in its current state, will loop infinitely:

```

while (TRUE) {
  cat("outer loop body\n")
  while (TRUE) {
    cat("inner loop body\n")
    x <- runif(1)
    if (x < .3) {
      break
    } else {
      cat(sprintf("x is %.5f\n", x))
    }
  }
}

```

One possibility is to recognize that, unlike `break`, the `return` expression **does** have the ability to return control across multiple levels of enclosing loops. However, since `return` is only valid when used within a function, we cannot simply replace `break` with `return()` above, but also need to wrap the entire expression as an anonymous function:

```

(function() {
  while (TRUE) {
    cat("outer loop body\n")
    while (TRUE) {
      cat("inner loop body\n")
      x <- runif(1)
      if (x < .3) {
        return()
      } else {
        cat(sprintf("x is %.5f\n", x))
      }
    }
}

```

)()

或者，我们可以在表达式之前创建一个哑变量（exit），并在准备终止时通过内层循环中的<<-激活它：

```
exit <- FALSE
while (TRUE) {cat
  ("外层循环体")while (TRUE) {

    cat("内层循环体")      x <-
  runif(1)if (x < .3) {

  exit <- TRUE
    break
  } else {
    cat(sprintf("x 的值是 %.5f", x))}

}
  if (exit) break
}
```

)()

Alternatively, we can create a dummy variable (exit) prior to the expression, and activate it via <<- from the inner loop when we are ready to terminate:

```
exit <- FALSE
while (TRUE) {
  cat("outer loop body\n")
  while (TRUE) {
    cat("inner loop body\n")
    x <- runif(1)
    if (x < .3) {
      exit <- TRUE
      break
    } else {
      cat(sprintf("x is %.5f\n", x))
    }
  if (exit) break
}
```

第52章：按列操作

第52.1节：每列求和

假设我们需要对数据集中的每一列进行求和

```
set.seed(20)
df1 <- data.frame(ID = rep(c("A", "B", "C"), 每个 = 3), V1 = rnorm(9), V2 = rnorm(9))
m1 <- as.matrix(df1[-1])
```

有很多方法可以做到这一点。使用base R，最好的选择是colSums

```
colSums(df1[-1], na.rm = TRUE)
```

这里，我们去掉了第一列，因为它是非数值型的，然后对每列进行了求和，指定了na.rm = TRUE（以防数据集中有任何NA）

这对matrix也适用

```
colSums(m1, na.rm = TRUE)
```

这也可以用lapply/sapply/vapply循环完成

```
lapply(df1[-1], sum, na.rm = TRUE)
```

需要注意的是，输出是一个列表。如果我们需要一个向量输出

```
sapply(df1[-1], sum, na.rm = TRUE)
```

或者

```
vapply(df1[-1], sum, na.rm = TRUE, numeric(1))
```

对于矩阵，如果我们想遍历列，则使用apply，参数MARGIN = 1

```
apply(m1, 2, FUN = sum, na.rm = TRUE)
```

也可以使用像 dplyr或data.table这样的包来实现

```
library(dplyr)
df1 %>%
summarise_at(vars(matches("^V\\d+")), sum, na.rm = TRUE)
```

这里，我们传入了一个正则表达式来匹配需要在 summarise_at中求和的列名。该正则表达式将匹配所有以V开头，后跟一个或多个数字 (\d+) 的列。

A data.table 选项是

```
library(data.table)
setDT(df1)[, lapply(.SD, sum, na.rm = TRUE), .SDcols = 2:ncol(df1)]
```

我们将 'data.frame' 转换为 'data.table' (setDT(df1))，在 .SDcols中指定要应用函数的列，循环遍历数据表的子集 (.SD) 并计算sum。

Chapter 52: Column wise operation

Section 52.1: sum of each column

Suppose we need to do the **sum** of each column in a dataset

```
set.seed(20)
df1 <- data.frame(ID = rep(c("A", "B", "C"), each = 3), V1 = rnorm(9), V2 = rnorm(9))
m1 <- as.matrix(df1[-1])
```

There are many ways to do this. Using base R, the best option would be **colSums**

```
colSums(df1[-1], na.rm = TRUE)
```

Here, we removed the first column as it is non-numeric and did the **sum** of each column, specifying the **na.rm = TRUE** (in case there are any NAs in the dataset)

This also works with **matrix**

```
colSums(m1, na.rm = TRUE)
```

This can be done in a loop with **lapply/sapply/vapply**

```
lapply(df1[-1], sum, na.rm = TRUE)
```

It should be noted that the output is a **list**. If we need a **vector** output

```
sapply(df1[-1], sum, na.rm = TRUE)
```

Or

```
vapply(df1[-1], sum, na.rm = TRUE, numeric(1))
```

For matrices, if we want to loop through columns, then use **apply** with **MARGIN = 1**

```
apply(m1, 2, FUN = sum, na.rm = TRUE)
```

There are ways to do this with packages like dplyr or data.table

```
library(dplyr)
df1 %>%
summarise_at(vars(matches("^V\\d+")), sum, na.rm = TRUE)
```

Here, we are passing a regular expression to match the column names that we need to get the **sum** in summarise_at. The regex will match all columns that start with V followed by one or more numbers (\d+).

A data.table option is

```
library(data.table)
setDT(df1)[, lapply(.SD, sum, na.rm = TRUE), .SDcols = 2:ncol(df1)]
```

We convert the 'data.frame' to 'data.table' (setDT(df1)), specified the columns to be applied the function in .SDcols and loop through the Subset of Data.table (.SD) and get the **sum**.

如果需要使用分组操作，可以通过指定分组列轻松实现

```
df1 %>%
  group_by(ID) %>%
  summarise_at(vars(matches("^V\\d+")), sum, na.rm = TRUE)
```

在需要所有列的总和时，可以使用summarise_each代替summarise_at

```
df1 %>%
  group_by(ID) %>%
  summarise_each(funs(sum(., na.rm = TRUE)))
```

data.table 选项是

```
setDT(df1)[, lapply(.SD, sum, na.rm = TRUE), by = ID]
```

If we need to use a group by operation, we can do this easily by specifying the group by column/columns

```
df1 %>%
  group_by(ID) %>%
  summarise_at(vars(matches("^V\\d+")), sum, na.rm = TRUE)
```

In cases where we need the **sum** of all the columns, summarise_each can be used instead of summarise_at

```
df1 %>%
  group_by(ID) %>%
  summarise_each(funs(sum(., na.rm = TRUE)))
```

The data.table option is

```
setDT(df1)[, lapply(.SD, sum, na.rm = TRUE), by = ID]
```

第53章：JSON

第53.1节：JSON与R对象之间的转换

jsonlite包是一个快速的JSON解析器和生成器，针对统计数据和网络进行了优化。读取和写入JSON的两个主要函数分别是fromJSON()和toJSON()，它们设计用于处理向量、矩阵和数据框，以及来自网络的JSON流。

从向量创建JSON数组，反之亦然

```
library(jsonlite)

## 向量转JSON
toJSON(c(1,2,3))
# [1] 1 2 3

fromJSON('[1,2,3]')
# [1] 1 2 3
```

从列表创建命名的JSON数组，反之亦然

```
toJSON(list(myVec = c(1,2,3)))
# {"myVec": [1,2,3]}

fromJSON('{"myVec": [1,2,3]}')
# $myVec
# [1] 1 2 3
```

更复杂的列表结构

```
## 列表结构
lst <- list(a = c(1,2,3),
            b = list(letters[1:6]))

toJSON(lst)
# {"a": [1,2,3], "b": [{"a": "a", "b": "b", "c": "c", "d": "d", "e": "e", "f": "f"}]}

fromJSON('{"a": [1,2,3], "b": [{"a": "a", "b": "b", "c": "c", "d": "d", "e": "e", "f": "f"}]} ')
# $a
# [1] 1 2 3
#
# $b
# [1] [2] [3] [4] [5] [6]
# [1,] "a" "b" "c" "d" "e" "f"
```

从data.frame创建JSON，反之亦然

```
## 将data.frame转换为JSON
df <- data.frame(id = seq_along(1:10),
                  val = letters[1:10])

toJSON(df)
#
#[{"id":1,"val":"a"}, {"id":2,"val":"b"}, {"id":3,"val":"c"}, {"id":4,"val":"d"}, {"id":5,"val":"e"}, {"id":6,"val":"f"}, {"id":7,"val":"g"}, {"id":8,"val":"h"}, {"id":9,"val":"i"}, {"id":10,"val":"j"}]
```

Chapter 53: JSON

Section 53.1: JSON to / from R objects

The [jsonlite package](#) is a fast JSON parser and generator optimized for statistical data and the web. The two main functions used to read and write JSON are fromJSON() and toJSON() respectively, and are designed to work with vectors, matrices and data.frames, and streams of JSON from the web.

Create a JSON array from a vector, and vice versa

```
library(jsonlite)

## vector to JSON
toJSON(c(1,2,3))
# [1] 1 2 3

fromJSON('[1,2,3]')
# [1] 1 2 3
```

Create a named JSON array from a list, and vice versa

```
toJSON(list(myVec = c(1,2,3)))
# {"myVec": [1,2,3]}

fromJSON('{"myVec": [1,2,3]}')
# $myVec
# [1] 1 2 3
```

More complex list structures

```
## list structures
lst <- list(a = c(1,2,3),
            b = list(letters[1:6]))

toJSON(lst)
# {"a": [1,2,3], "b": [{"a": "a", "b": "b", "c": "c", "d": "d", "e": "e", "f": "f"}]}

fromJSON('{"a": [1,2,3], "b": [{"a": "a", "b": "b", "c": "c", "d": "d", "e": "e", "f": "f"}]} ')
# $a
# [1] 1 2 3
#
# $b
# [1] [2] [3] [4] [5] [6]
# [1,] "a" "b" "c" "d" "e" "f"
```

Create JSON from a [data.frame](#), and vice versa

```
## converting a data.frame to JSON
df <- data.frame(id = seq_along(1:10),
                  val = letters[1:10])

toJSON(df)
#
#[{"id":1,"val":"a"}, {"id":2,"val":"b"}, {"id":3,"val":"c"}, {"id":4,"val":"d"}, {"id":5,"val":"e"}, {"id":6,"val":"f"}, {"id":7,"val":"g"}, {"id":8,"val":"h"}, {"id":9,"val":"i"}, {"id":10,"val":"j"}]
```

```

## 读取JSON字符串
fromJSON('[{"id":1,"val":"a"}, {"id":2,"val":"b"}, {"id":3,"val":"c"}, {"id":4,"val":"d"}, {"id":5,"val":"e"}, {"id":6,"val":"f"}, {"id":7,"val":"g"}, {"id":8,"val":"h"}, {"id":9,"val":"i"}, {"id":10,"val":"j"}]')
#   id val
# 1 1 a
# 2 2 b
# 3 3 c
# 4 4 d
# 5 5 e
# 6 6 f
# 7 7 g
# 8 8 h
# 9 9 i
# 10 10 j

```

直接从互联网读取JSON

```

## 从URL读取JSON
googleway_issues <- fromJSON("https://api.github.com/repos/SymbolixAU/googleway/issues")

googleway_issues$url
# [1] "https://api.github.com/repos/SymbolixAU/googleway/issues/20"
"https://api.github.com/repos/SymbolixAU/googleway/issues/19"
# [3] "https://api.github.com/repos/SymbolixAU/googleway/issues/14"
"https://api.github.com/repos/SymbolixAU/googleway/issues/11"
# [5] "https://api.github.com/repos/SymbolixAU/googleway/issues/9"
"https://api.github.com/repos/SymbolixAU/googleway/issues/5"
# [7] "https://api.github.com/repos/SymbolixAU/googleway/issues/2"

```

```

## reading a JSON string
fromJSON('[{"id":1,"val":"a"}, {"id":2,"val":"b"}, {"id":3,"val":"c"}, {"id":4,"val":"d"}, {"id":5,"val":"e"}, {"id":6,"val":"f"}, {"id":7,"val":"g"}, {"id":8,"val":"h"}, {"id":9,"val":"i"}, {"id":10,"val":"j"}]')
#   id val
# 1 1 a
# 2 2 b
# 3 3 c
# 4 4 d
# 5 5 e
# 6 6 f
# 7 7 g
# 8 8 h
# 9 9 i
# 10 10 j

```

Read JSON direct from the internet

```

## Reading JSON from URL
googleway_issues <- fromJSON("https://api.github.com/repos/SymbolixAU/googleway/issues")

googleway_issues$url
# [1] "https://api.github.com/repos/SymbolixAU/googleway/issues/20"
"https://api.github.com/repos/SymbolixAU/googleway/issues/19"
# [3] "https://api.github.com/repos/SymbolixAU/googleway/issues/14"
"https://api.github.com/repos/SymbolixAU/googleway/issues/11"
# [5] "https://api.github.com/repos/SymbolixAU/googleway/issues/9"
"https://api.github.com/repos/SymbolixAU/googleway/issues/5"
# [7] "https://api.github.com/repos/SymbolixAU/googleway/issues/2"

```

第54章：RODBC

第54.1节：通过RODBC连接Excel文件

虽然RODBC仅限于Windows电脑，且R与任何目标关系数据库管理系统（RDMS）之间的架构需兼容，它的一个关键灵活性是能够像操作SQL数据库一样操作Excel文件。

```
require(RODBC)
con = odbcConnectExcel("myfile.xlsx") # 打开与Excel文件的连接
sqlTables(con)$TABLE_NAME # 显示所有工作表
df = sqlFetch(con, "Sheet1") # 读取一个工作表
df = sqlQuery(con, "select * from [Sheet1 $]") # 读取一个工作表 (替代的SQL语法)
close(con) # 关闭与文件的连接
```

第54.2节：SQL Server管理数据库连接以获取单个表

RODBC的另一个用途是连接SQL Server管理数据库。我们需要指定“驱动程序”，即这里是SQL Server，数据库名为“Atilla”，然后使用sqlQuery提取整个表或其中一部分。

```
library(RODBC)
cn <- odbcDriverConnect(connection="Driver={SQL
Server};server=localhost;database=Atilla;trusted_connection=yes;")
tbl <- sqlQuery(cn, 'select top 10 * from table_1')
```

第54.3节：连接关系型数据库

```
library(RODBC)
con <- odbcDriverConnect("driver={Sql Server};server=servername;trusted connection=true")
dat <- sqlQuery(con, "select * from table");
close(con)
```

这将连接到一个 SQL Server 实例。有关连接字符串应如何编写的更多信息，请访问 connectionstrings.com

此外，由于未指定数据库，您应确保完全限定您想查询的对象，例如 database.schema.objectname

Chapter 54: RODBC

Section 54.1: Connecting to Excel Files via RODBC

While RODBC is restricted to Windows computers with compatible architecture between R and any target RDMS, one of its key flexibilities is to work with Excel files as if they were SQL databases.

```
require(RODBC)
con = odbcConnectExcel("myfile.xlsx") # open a connection to the Excel file
sqlTables(con)$TABLE_NAME # show all sheets
df = sqlFetch(con, "Sheet1") # read a sheet
df = sqlQuery(con, "select * from [Sheet1 $]") # read a sheet (alternative SQL syntax)
close(con) # close the connection to the file
```

Section 54.2: SQL Server Management Database connection to get individual table

Another use of RODBC is in connecting with SQL Server Management Database. We need to specify the 'Driver' i.e. SQL Server here, the database name "Atilla" and then use the sqlQuery to extract either the full table or a fraction of it.

```
library(RODBC)
cn <- odbcDriverConnect(connection="Driver={SQL
Server};server=localhost;database=Atilla;trusted_connection=yes;")
tbl <- sqlQuery(cn, 'select top 10 * from table_1')
```

Section 54.3: Connecting to relational databases

```
library(RODBC)
con <- odbcDriverConnect("driver={Sql Server};server=servername;trusted connection=true")
dat <- sqlQuery(con, "select * from table");
close(con)
```

This will connect to a SQL Server instance. For more information on what your connection string should look like, visit connectionstrings.com

Also, since there's no database specified, you should make sure you fully qualify the object you're wanting to query like this database.schema.objectname

第55章：lubridate

第55.1节：使用lubridate从字符串解析日期和日期时间

lubridate 包提供了方便的函数，用于从字符字符串格式化日期和日期时间对象。这些函数是以下的排列组合

字母	要解析的元素	Base R 等价物
y	年份	%y, %Y
m (与 y 和 d 一起)	月份	%m, %b, %h, %B
d	日	%d, %e
h	小时	%H, %I%p
m (带有 h 和 s)	分钟	%M
s	秒	%S

例如，`ymd()`用于解析按年、月、日顺序排列的日期，例如“2016-07-22”，或者
`ymd_hms()`用于解析按年、月、日、小时、分钟、秒顺序排列的日期时间，例如“2016-07-22 13:04:47”。

这些函数能够识别大多数分隔符（如/、-和空白符），无需额外参数。
它们也能处理不一致的分隔符。

日期

日期函数返回一个Date类的对象。

```
library(lubridate)

mdy(c('07/02/2016', '7 / 03 / 2016', '7 / 4 / 16'))
## [1] "2016-07-02" "2016-07-03" "2016-07-04"

ymd(c("20160724","2016/07/23","2016-07-25")) # 分隔符不一致
## [1] "2016-07-24" "2016-07-23" "2016-07-25"
```

日期时间

实用函数

日期时间可以使用包含`ymd_hms`变体的函数解析，包括`ymd_hm`和`ymd_h`。所有日期时间函数都可以接受一个`tz`时区参数，类似于`as.POSIXct`或`strptime`的参数，但默认时区为“UTC”，而非本地时区。

日期时间函数返回一个POSIXct类的对象。

```
x <- c("20160724 130102", "2016/07/23 14:02:01", "2016-07-25 15:03:00")
ymd_hms(x, tz="EST")
## [1] "2016-07-24 13:01:02 EST" "2016-07-23 14:02:01 EST"
## [3] "2016-07-25 15:03:00 EST"

ymd_hms(x)
## [1] "2016-07-24 13:01:02 UTC" "2016-07-23 14:02:01 UTC"
## [3] "2016-07-25 15:03:00 UTC"
```

Chapter 55: lubridate

Section 55.1: Parsing dates and datetimes from strings with lubridate

The `lubridate` package provides convenient functions to format date and datetime objects from character strings.
The functions are permutations of

Letter	Element to parse	Base R equivalent
y	year	%y, %Y
m (with y and d)	month	%m, %b, %h, %B
d	day	%d, %e
h	hour	%H, %I%p
m (with h and s)	minute	%M
s	seconds	%S

e.g. `ymd()` for parsing a date with the year followed by the month followed by the day, e.g. “2016-07-22”，or
`ymd_hms()` for parsing a datetime in the order year, month, day, hours, minutes, seconds, e.g. “2016-07-22 13:04:47”.

The functions are able to recognize most separators (such as /, -, and whitespace) without additional arguments.
They also work with inconsistent separators.

Dates

The date functions return an object of class Date.

```
library(lubridate)

mdy(c('07/02/2016', '7 / 03 / 2016', '7 / 4 / 16'))
## [1] "2016-07-02" "2016-07-03" "2016-07-04"

ymd(c("20160724","2016/07/23","2016-07-25")) # inconsistent separators
## [1] "2016-07-24" "2016-07-23" "2016-07-25"
```

Datetimes

Utility functions

Datetimes can be parsed using `ymd_hms` variants including `ymd_hm` and `ymd_h`. All datetime functions can accept a `tz` timezone argument akin to that of `as.POSIXct` or `strptime`, but which defaults to “UTC” instead of the local timezone.

The datetime functions return an object of class POSIXct.

```
x <- c("20160724 130102", "2016/07/23 14:02:01", "2016-07-25 15:03:00")
ymd_hms(x, tz="EST")
## [1] "2016-07-24 13:01:02 EST" "2016-07-23 14:02:01 EST"
## [3] "2016-07-25 15:03:00 EST"

ymd_hms(x)
## [1] "2016-07-24 13:01:02 UTC" "2016-07-23 14:02:01 UTC"
## [3] "2016-07-25 15:03:00 UTC"
```

解析函数

lubridate 还包括三个用于解析日期时间的函数，使用格式化字符串，如 `as.POSIXct` 或 `strptime`：

函数	输出类	接受的格式化字符串
<code>parse_date_timePOSIXct</code>		灵活。将接受带有 % 的 <code>strptime</code> 风格或 lubridate 日期时间函数名风格，例如 "ymd hms"。将接受一个顺序向量用于异构数据，并猜测哪个合适。
<code>parse_date_time2</code>	默认 <code>POSIXct</code> ；如果 <code>lt = TRUE</code> , 则为 <code>POSIXlt</code>	严格。仅接受有限集合中的 <code>strptime</code> 标记（带或不带 %）。
<code>fast_strptime</code>	默认 <code>POSIXlt</code> ；如果 <code>lt = FALSE</code> , 则为 <code>POSIXct</code>	严格。仅接受带有分隔符 (-、/、: 等) 且由有限集合中的 % 分隔的 <code>strptime</code> 标记。

```
x <- c('2016-07-22 13:04:47', '07/22/2016 1:04:47 pm')

parse_date_time(x, orders = c('mdy Imsp', 'ymd hms'))
## [1] "2016-07-22 13:04:47 UTC" "2016-07-22 13:04:47 UTC"

x <- c('2016-07-22 13:04:47', '2016-07-22 14:47:58')

parse_date_time2(x, orders = 'Ymd HMS')
## [1] "2016-07-22 13:04:47 UTC" "2016-07-22 14:47:58 UTC"

fast_strptime(x, format = '%Y-%m-%d %H:%M:%S')
## [1] "2016-07-22 13:04:47 UTC" "2016-07-22 14:47:58 UTC"
```

`parse_date_time2` 和 `fast_strptime` 使用快速的 C 解析器以提高效率。

有关格式化标记，请参见 `?parse_date_time`。

第 55.2 节：周期与持续时间的区别

与持续时间不同，周期可以用来准确模拟时钟时间，而无需知道闰秒、闰日和夏令时变更等事件何时发生。

```
start_2012 <- ymd_hms("2012-01-01 12:00:00")
## [1] "2012-01-01 12:00:00 UTC"

# period() 考虑闰年计算。
start_2012 + period(1, "years")
## [1] "2013-01-01 12:00:00 UTC"

# 这里 duration() 不考虑闰年计算。
start_2012 + duration(1)
## [1] "2012-12-31 12:00:00 UTC"
```

第55.3节：瞬间

瞬间是指特定的时间点。任何指向某一时间点的日期时间对象都被视为一个瞬间。要测试一个对象是否为瞬间，使用 `is.instant`。

```
library(lubridate)

today_start <- dmy_hms("22.07.2016 12:00:00", tz = "IST") # 默认时区="UTC"
today_start
## [1] "2016-07-22 12:00:00 IST"
is.instant(today_start)
## [1] TRUE
```

Parser functions

lubridate 也包括三个函数用于解析日期时间，使用格式化字符串，如 `as.POSIXct` 或 `strptime`：

Function	Output Class	Formatting strings accepted
<code>parse_date_time</code>	<code>POSIXct</code>	Flexible. Will accept <code>strptime</code> -style with % or lubridate datetime function name style, e.g "ymd hms". Will accept a vector of orders for heterogeneous data and guess which is appropriate.
<code>parse_date_time2</code>	Default <code>POSIXct</code> ; if <code>lt = TRUE</code> , <code>POSIXlt</code>	Strict. Accepts only <code>strptime</code> tokens (with or without %) from a limited set.
<code>fast_strptime</code>	Default <code>POSIXlt</code> ; if <code>lt = FALSE</code> , <code>POSIXct</code>	Strict. Accepts only %-delimited <code>strptime</code> tokens with delimiters (-, /, :, etc.) from a limited set.

```
x <- c('2016-07-22 13:04:47', '07/22/2016 1:04:47 pm')

parse_date_time(x, orders = c('mdy Imsp', 'ymd hms'))
## [1] "2016-07-22 13:04:47 UTC" "2016-07-22 13:04:47 UTC"

x <- c('2016-07-22 13:04:47', '2016-07-22 14:47:58')

parse_date_time2(x, orders = 'Ymd HMS')
## [1] "2016-07-22 13:04:47 UTC" "2016-07-22 14:47:58 UTC"

fast_strptime(x, format = '%Y-%m-%d %H:%M:%S')
## [1] "2016-07-22 13:04:47 UTC" "2016-07-22 14:47:58 UTC"
```

`parse_date_time2` 和 `fast_strptime` 使用快速 C 解析器以提高效率。

有关格式化标记，请参见 `?parse_date_time`。

Section 55.2: Difference between period and duration

与持续时间不同，周期可以用来准确模拟时钟时间，而无需知道闰秒、闰日和夏令时变更等事件何时发生。

```
start_2012 <- ymd_hms("2012-01-01 12:00:00")
## [1] "2012-01-01 12:00:00 UTC"

# period() 考虑闰年计算。
start_2012 + period(1, "years")
## [1] "2013-01-01 12:00:00 UTC"

# Here duration() doesn't consider leap year calculations.
start_2012 + duration(1)
## [1] "2012-12-31 12:00:00 UTC"
```

Section 55.3: Instants

瞬间是指特定的时间点。任何指向某一时间点的日期时间对象都被视为一个瞬间。要测试一个对象是否为瞬间，使用 `is.instant`。

```
library(lubridate)

today_start <- dmy_hms("22.07.2016 12:00:00", tz = "IST") # default tz="UTC"
today_start
## [1] "2016-07-22 12:00:00 IST"
is.instant(today_start)
## [1] TRUE
```

```

now_dt <- ymd_hms(now(), tz="IST")
now_dt
## [1] "2016-07-22 13:53:09 IST"
is.instant(now_dt)
## [1] TRUE

is.instant("helloworld")
## [1] FALSE
is.instant(60)
## [1] FALSE

```

第55.4节：区间、持续时间和周期

区间是lubridate中记录时间跨度的最简单方式。区间是发生在两个特定时刻之间的一段时间。

```

# 通过减去两个时刻创建区间
today_start <- ymd_hms("2016-07-22 12:00:00", tz="IST")
today_start
## [1] "2016-07-22 12:00:00 IST"
today_end <- ymd_hms("2016-07-22 23:59:59", tz="IST")
today_end
## [1] "2016-07-22 23:59:59 IST"
span <- today_end - today_start
span
## 时间差为11.99972小时
as.interval(span, today_start)
## [1] 2016-07-22 12:00:00 IST--2016-07-22 23:59:59 IST

# 使用interval()函数创建区间
span <- interval(today_start, today_end)
[1] 2016-07-22 12:00:00 IST--2016-07-22 23:59:59 IST

```

持续时间测量两个时刻之间发生的确切时间量。

```

duration(60, "seconds")
## [1] "60秒"

duration(2, "minutes")
## [1] "120秒 (约2分钟) "

```

注意：由于单位变化较大，不使用大于周的单位。

持续时间可以使用 dseconds、dminutes 以及其他持续时间辅助函数创建。
运行 ?quick_durations 查看完整列表。

```

dseconds(60)
## [1] "60秒"

dhours(2)
## [1] "7200秒 (约2小时) "

dyears(1)
## [1] "31536000秒 (约365天) "

```

持续时间可以从时间点中减去或加上，以获得新的时间点。

```
today_start + dhours(5)
```

```

now_dt <- ymd_hms(now(), tz="IST")
now_dt
## [1] "2016-07-22 13:53:09 IST"
is.instant(now_dt)
## [1] TRUE

is.instant("helloworld")
## [1] FALSE
is.instant(60)
## [1] FALSE

```

Section 55.4: Intervals, Durations and Periods

Intervals are simplest way of recording timespans in lubridate. An interval is a span of time that occurs between two specific **instants**.

```

# create interval by subtracting two instants
today_start <- ymd_hms("2016-07-22 12:00:00", tz="IST")
today_start
## [1] "2016-07-22 12:00:00 IST"
today_end <- ymd_hms("2016-07-22 23:59:59", tz="IST")
today_end
## [1] "2016-07-22 23:59:59 IST"
span <- today_end - today_start
span
## Time difference of 11.99972 hours
as.interval(span, today_start)
## [1] 2016-07-22 12:00:00 IST--2016-07-22 23:59:59 IST

# create interval using interval() function
span <- interval(today_start, today_end)
[1] 2016-07-22 12:00:00 IST--2016-07-22 23:59:59 IST

```

Durations measure the exact amount of time that occurs between two instants.

```

duration(60, "seconds")
## [1] "60s"

duration(2, "minutes")
## [1] "120s (~2 minutes)"

```

Note: Units larger than weeks are not used due to their variability.

Durations can be created using dseconds, dminutes and other duration helper functions.
Run ?quick_durations for complete list.

```

dseconds(60)
## [1] "60s"

dhours(2)
## [1] "7200s (~2 hours)"

dyears(1)
## [1] "31536000s (~365 days)"

```

Durations can be subtracted and added to instants to get new instants.

```
today_start + dhours(5)
```

```
## [1] "2016-07-22 17:00:00 IST"
today_start + dhours(5) + dminutes(30) + dseconds(15)
## [1] "2016-07-22 17:30:15 IST"
```

持续时间可以由时间区间创建。

```
as.duration(span)
[1] "43199秒 (约12小时)"
```

期间 (Periods) 测量两个时间点之间时钟时间的变化。

期间可以使用period函数以及其他辅助函数如seconds、hours等创建。要获取完整的期间辅助函数列表，请运行?quick_periods。

```
period(1, "hour")
## [1] "1小时 0分 0秒"
```

```
hours(1)
## [1] "1小时 0分 0秒"
```

```
period(6, "months")
## [1] "6个月 0天 0小时 0分 0秒"
```

```
months(6)
## [1] "6个月0天0小时0分0秒"
```

```
年(1)
## [1] "1年0月0日0时0分0秒"
```

is.period 函数可用于检查对象是否为周期。

```
is.period(years(1))
## [1] TRUE
```

```
is.period(dydays(1))
## [1] FALSE
```

第55.5节：在lubridate中操作日期和时间

```
date <- now()
date
## "2016-07-22 03:42:35 IST"

year(date)
## 2016

minute(date)
## 42

wday(date, label = T, abbr = T)
# [1] 周五
# 等级：周日 < 周一 < 周二 < 周三 < 周四 < 周五 < 周六

day(date) <- 31
## "2016-07-31 03:42:35 IST"

# 如果某个元素被设置为超过其支持的最大值，差值
# 将会进位到下一个更高的元素
```

```
## [1] "2016-07-22 17:00:00 IST"
```

```
today_start + dhours(5) + dminutes(30) + dseconds(15)
## [1] "2016-07-22 17:30:15 IST"
```

Durations can be created from intervals.

```
as.duration(span)
[1] "43199s (~12 hours)"
```

Periods 测量两个时间点之间时钟时间的变化。

Periods can be created using period function as well other helper functions like seconds, hours, etc. To get a complete list of period helper functions, Run ?quick_periods.

```
period(1, "hour")
## [1] "1H 0M 0S"
```

```
hours(1)
## [1] "1H 0M 0S"
```

```
period(6, "months")
## [1] "6m 0d 0H 0M 0S"
```

```
months(6)
## [1] "6m 0d 0H 0M 0S"
```

```
years(1)
## [1] "1y 0m 0d 0H 0M 0S"
```

is.period function can be used to check if an object is a period.

```
is.period(years(1))
## [1] TRUE
```

```
is.period(dydays(1))
## [1] FALSE
```

Section 55.5: Manipulating date and time in lubridate

```
date <- now()
date
## "2016-07-22 03:42:35 IST"

year(date)
## 2016

minute(date)
## 42

wday(date, label = T, abbr = T)
# [1] Fri
# Levels: Sun < Mon < Tues < Wed < Thurs < Fri < Sat

day(date) <- 31
## "2016-07-31 03:42:35 IST"
```

```
# If an element is set to a larger value than it supports, the difference
# will roll over into the next higher element
```

```
日(日期) <- 32
## "2016-08-01 03:42:35 IST"
```

第55.6节：时区

with_tz 返回在不同时间区域中显示的日期时间。

```
nyc_time <- 现在("America/New_York")
nyc_time
## [1] "2016-07-22 05:49:08 EDT"
```

```
# 对应的欧洲/莫斯科时间
with_tz(nyc_time, tzzone = "Europe/Moscow")
## [1] "2016-07-22 12:49:08 MSK"
```

force_tz 返回在新时区中与x具有相同时钟时间的日期时间。

```
nyc_time <- 现在("America/New_York")
nyc_time
## [1] "2016-07-22 05:49:08 EDT"
```

```
force_tz(nyc_time, tzzone = "Europe/Moscow") # 仅更改时区
## [1] "2016-07-22 05:49:08 MSK"
```

第55.7节：在lubridate中解析日期和时间

Lubridate 提供了 ymd() 系列函数，用于将字符字符串解析为日期。字母 y、m 和 d 分别对应日期时间的年、月和日元素。

```
mdy("07-21-2016")          # 返回日期
## [1] "2016-07-21"

mdy("07-21-2016", tz = "UTC")    # 返回 POSIXt 类的向量
## "2016-07-21 UTC"

dmy("21-07-2016")          # 返回日期
## [1] "2016-07-21"

dmy(c("21.07.2016", "22.07.2016")) # 返回 Date 类的向量
## [1] "2016-07-21" "2016-07-22"
```

第 55.8 节：日期四舍五入

```
now_dt <- ymd_hms(now(), tz="IST")
now_dt
## [1] "2016-07-22 13:53:09 IST"
```

round_date() 接受一个日期时间对象，并将其四舍五入到指定时间单位的最接近整数值。

```
round_date(now_dt, "minute")
## [1] "2016-07-22 13:53:00 IST"

round_date(now_dt, "hour")
```

```
day(date) <- 32
## "2016-08-01 03:42:35 IST"
```

Section 55.6: Time Zones

with_tz returns a date-time as it would appear in a different time zone.

```
nyc_time <- now("America/New_York")
nyc_time
## [1] "2016-07-22 05:49:08 EDT"

# corresponding Europe/Moscow time
with_tz(nyc_time, tzzone = "Europe/Moscow")
## [1] "2016-07-22 12:49:08 MSK"
```

force_tz returns a the date-time that has the same clock time as x in the new time zone.

```
nyc_time <- now("America/New_York")
nyc_time
## [1] "2016-07-22 05:49:08 EDT"

force_tz(nyc_time, tzzone = "Europe/Moscow") # only timezone changes
## [1] "2016-07-22 05:49:08 MSK"
```

Section 55.7: Parsing date and time in lubridate

Lubridate provides ymd() series of functions for parsing character strings into dates. The letters y, m, and d correspond to the year, month, and day elements of a date-time.

```
mdy("07-21-2016")          # Returns Date
## [1] "2016-07-21"

mdy("07-21-2016", tz = "UTC")    # Returns a vector of class POSIXt
## "2016-07-21 UTC"

dmy("21-07-2016")          # Returns Date
## [1] "2016-07-21"

dmy(c("21.07.2016", "22.07.2016")) # Returns vector of class Date
## [1] "2016-07-21" "2016-07-22"
```

Section 55.8: Rounding dates

```
now_dt <- ymd_hms(now(), tz="IST")
now_dt
## [1] "2016-07-22 13:53:09 IST"
```

round_date() takes a date-time object and rounds it to the nearest integer value of the specified time unit.

```
round_date(now_dt, "minute")
## [1] "2016-07-22 13:53:00 IST"

round_date(now_dt, "hour")
```

```
## [1] "2016-07-22 14:00:00 IST"
```

```
round_date(now_dt, "year")
## [1] "2017-01-01 IST"
```

`floor_date()` 接受一个日期时间对象，并将其向下舍入到指定时间单位的最接近整数值。

```
floor_date(now_dt, "minute")
## [1] "2016-07-22 13:53:00 IST"
```

```
floor_date(now_dt, "hour")
## [1] "2016-07-22 13:00:00 IST"
```

```
floor_date(now_dt, "year")
## [1] "2016-01-01 IST"
```

`ceiling_date()` 接受一个日期时间对象，并将其向上舍入到指定时间单位的最接近整数值。

```
ceiling_date(now_dt, "minute")
## [1] "2016-07-22 13:54:00 IST"
```

```
ceiling_date(now_dt, "hour")
## [1] "2016-07-22 14:00:00 IST"
```

```
ceiling_date(now_dt, "year")
## [1] "2017-01-01 IST"
```

```
## [1] "2016-07-22 14:00:00 IST"
```

```
round_date(now_dt, "year")
## [1] "2017-01-01 IST"
```

`floor_date()` takes a date-time object and rounds it down to the nearest integer value of the specified time unit.

```
floor_date(now_dt, "minute")
## [1] "2016-07-22 13:53:00 IST"
```

```
floor_date(now_dt, "hour")
## [1] "2016-07-22 13:00:00 IST"
```

```
floor_date(now_dt, "year")
## [1] "2016-01-01 IST"
```

`ceiling_date()` takes a date-time object and rounds it up to the nearest integer value of the specified time unit.

```
ceiling_date(now_dt, "minute")
## [1] "2016-07-22 13:54:00 IST"
```

```
ceiling_date(now_dt, "hour")
## [1] "2016-07-22 14:00:00 IST"
```

```
ceiling_date(now_dt, "year")
## [1] "2017-01-01 IST"
```

第56章：时间序列与预测

第56.1节：创建ts对象

时间序列数据可以存储为一个sts对象。sts对象包含有关季节频率的信息，ARIMA函数会使用这些信息。它还允许使用window命令按日期调用序列中的元素。

```
#创建一个包含100个观测值的虚拟数据集  
x <- rnorm(100)  
  
#将该向量转换为具有100个年度观测值的时间序列对象  
x <- ts(x, start = c(1900), freq = 1)  
  
#将该向量转换为从7月开始的100个月度观测值的时间序列对象  
x <- ts(x, start = c(1900, 7), freq = 12)  
  
#或者，起始观测值也可以是一个数字：  
x <- ts(x, start = 1900.5, freq = 12)  
  
#将该向量转换为从1900年第1周开始的100个每日观测值且频率为每周的时间序列对象  
x <- ts(x, start = c(1900, 1), freq = 7)  
  
#ts对象的默认绘图为折线图  
plot(x)
```

```
#window函数可以按日期调用单个元素或元素集合  
  
#调用1900年前4周的数据  
window(x, start = c(1900, 1), end = (1900, 4))  
  
#仅调用1900年第10周的数据  
window(x, start = c(1900, 10), end = (1900, 10))  
  
#调用包括并且在1900年第10周之后的所有周  
window(x, start = c(1900, 10))
```

可以创建包含多条序列的 ts 对象：

```
#创建一个包含3条序列、每条序列100个观测值的虚拟矩阵  
x <- cbind(rnorm(100), rnorm(100), rnorm(100))  
  
#创建一个从1900年开始、频率为1的多序列年观测 ts 对象  
x <- ts(x, start = 1900, freq = 1)  
  
#R会为对象中的每条序列绘制一张图  
plot(x)
```

第56.2节：时间序列数据的探索性数据分析

```
data(AirPassengers)  
class(AirPassengers)
```

```
| 1 "ts"
```

本着探索性数据分析（EDA）的精神，第一步的好方法是查看时间序列数据的图形：

Chapter 56: Time Series and Forecasting

Section 56.1: Creating a ts object

Time series data can be stored as a ts object. ts objects contain information about seasonal frequency that is used by ARIMA functions. It also allows for calling of elements in the series by date using the `window` command.

```
#Create a dummy dataset of 100 observations  
x <- rnorm(100)  
  
#Convert this vector to a ts object with 100 annual observations  
x <- ts(x, start = c(1900), freq = 1)  
  
#Convert this vector to a ts object with 100 monthly observations starting in July  
x <- ts(x, start = c(1900, 7), freq = 12)  
  
#Alternatively, the starting observation can be a number:  
x <- ts(x, start = 1900.5, freq = 12)  
  
#Convert this vector to a ts object with 100 daily observations and weekly frequency starting in  
the first week of 1900  
x <- ts(x, start = c(1900, 1), freq = 7)  
  
#The default plot for a ts object is a line plot  
plot(x)  
  
#The window function can call elements or sets of elements by date  
  
#Call the first 4 weeks of 1900  
window(x, start = c(1900, 1), end = (1900, 4))  
  
#Call only the 10th week in 1900  
window(x, start = c(1900, 10), end = (1900, 10))  
  
#Call all weeks including and after the 10th week of 1900  
window(x, start = c(1900, 10))
```

It is possible to create ts objects with multiple series:

```
#Create a dummy matrix of 3 series with 100 observations each  
x <- cbind(rnorm(100), rnorm(100), rnorm(100))  
  
#Create a multi-series ts with annual observation starting in 1900  
x <- ts(x, start = 1900, freq = 1)  
  
#R will draw a plot for each series in the object  
plot(x)
```

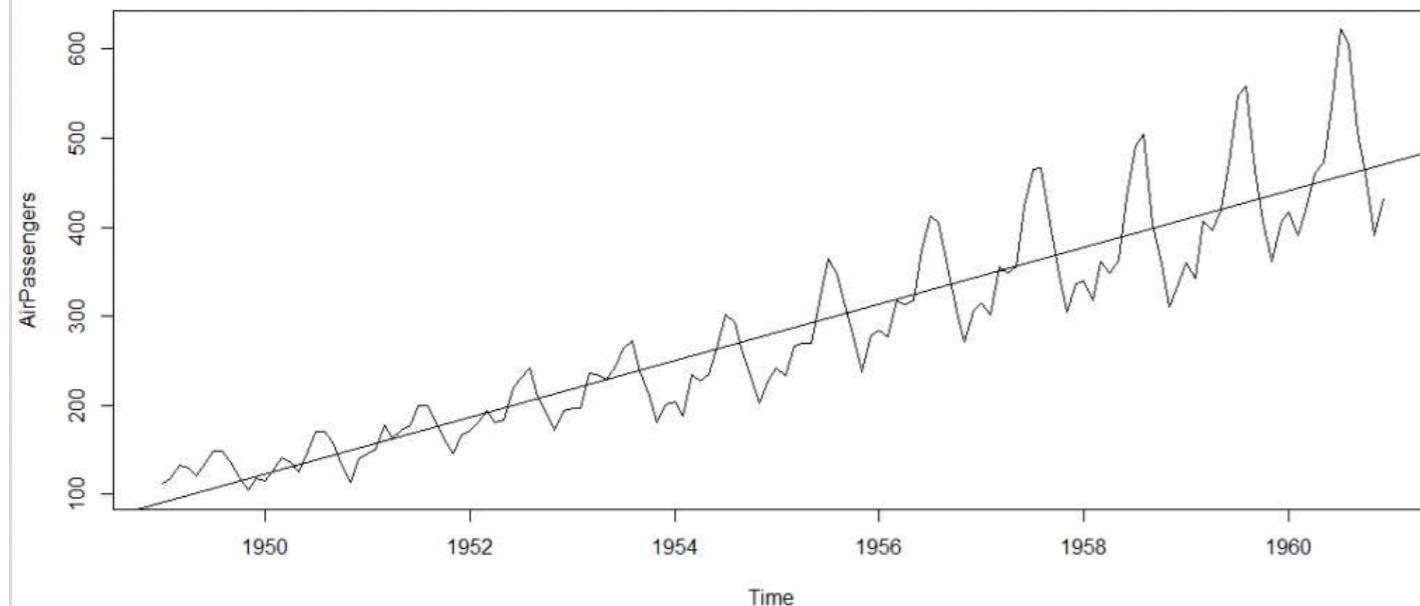
Section 56.2: Exploratory Data Analysis with time-series data

```
data(AirPassengers)  
class(AirPassengers)
```

```
| 1 "ts"
```

In the spirit of Exploratory Data Analysis (EDA) a good first step is to look at a plot of your time-series data:

```
plot(AirPassengers) # 绘制原始数据
abline(reg=lm(AirPassengers~time(AirPassengers))) # 拟合趋势线
```

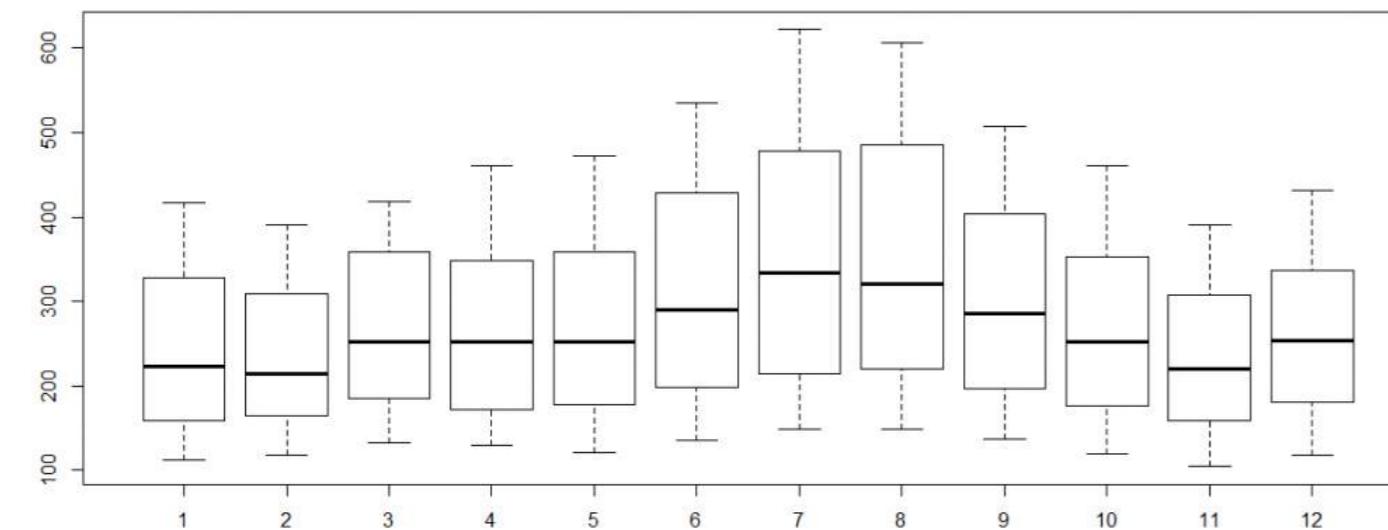


为了进一步的EDA，我们检查跨年份的周期：

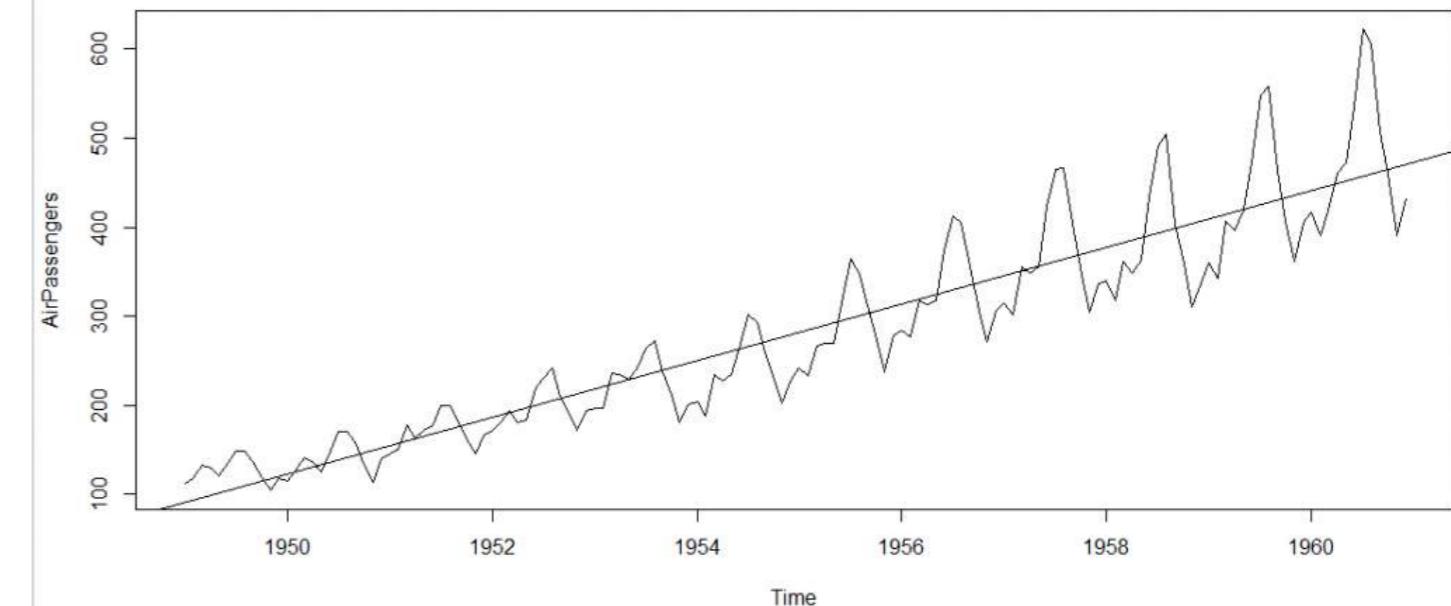
周期(AirPassengers)

	一月	二月	三月	四月	五月	六月	七月	八月	九月	十月	十一月	十二月
1949	1	2	3	4	5	6	7	8	9	10	11	12
1950	1	2	3	4	5	6	7	8	9	10	11	12
1951	1	2	3	4	5	6	7	8	9	10	11	12
1952	1	2	3	4	5	6	7	8	9	10	11	12
1953	1	2	3	4	5	6	7	8	9	10	11	12
1954	1	2	3	4	5	6	7	8	9	10	11	12
1955	1	2	3	4	5	6	7	8	9	10	11	12
1956	1	2	3	4	5	6	7	8	9	10	11	12
1957	1	2	3	4	5	6	7	8	9	10	11	12
1958	1	2	3	4	5	6	7	8	9	10	11	12
1959	1	2	3	4	5	6	7	8	9	10	11	12
1960	1	2	3	4	5	6	7	8	9	10	11	12

```
boxplot(AirPassengers~cycle(AirPassengers)) #按月份绘制箱线图以探索季节性影响
```



```
plot(AirPassengers) # plot the raw data
abline(reg=lm(AirPassengers~time(AirPassengers))) # fit a trend line
```

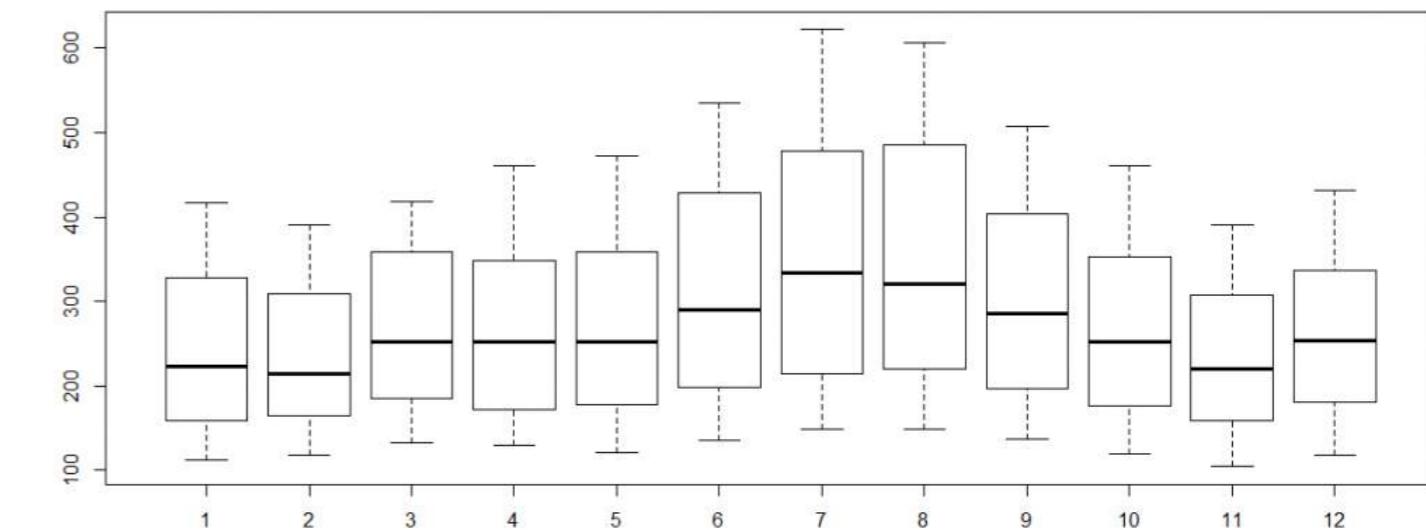


For further EDA we examine cycles across years:

cycle(AirPassengers)

	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
1949	1	2	3	4	5	6	7	8	9	10	11	12
1950	1	2	3	4	5	6	7	8	9	10	11	12
1951	1	2	3	4	5	6	7	8	9	10	11	12
1952	1	2	3	4	5	6	7	8	9	10	11	12
1953	1	2	3	4	5	6	7	8	9	10	11	12
1954	1	2	3	4	5	6	7	8	9	10	11	12
1955	1	2	3	4	5	6	7	8	9	10	11	12
1956	1	2	3	4	5	6	7	8	9	10	11	12
1957	1	2	3	4	5	6	7	8	9	10	11	12
1958	1	2	3	4	5	6	7	8	9	10	11	12
1959	1	2	3	4	5	6	7	8	9	10	11	12
1960	1	2	3	4	5	6	7	8	9	10	11	12

```
boxplot(AirPassengers~cycle(AirPassengers)) #Box plot across months to explore seasonal effects
```



第57章：strsplit函数

第57.1节：简介

strsplit 是一个用于根据某些字符模式将向量拆分成列表的有用函数。使用典型的R工具，整个列表可以重新合并为data.frame，或者列表的一部分可以用于绘图操作。

以下是strsplit的常见用法：沿逗号分隔符拆分字符串：

```
temp <- c("this,that,other", "hat,scarf,food", "woman,man,child")
# 获取一个以逗号分隔的列表
myList <- strsplit(temp, split=",")
# 打印 myList
myList
[[1]]
[1] "this"  "that"  "other"

[[2]]
[1] "hat"   "scarf" "food"

[[3]]
[1] "woman" "man"   "child"
```

如上所示，split 参数不仅限于字符，还可以遵循正则表达式所规定的模式。例如，temp2 与上面的 temp 相同，只是每个元素的分隔符有所不同。我们可以利用 split 参数接受正则表达式的特点来缓解向量中的不规则性。

```
temp2 <- c("this, that, other", "hat,scarf ,food", "woman; man ; child")
myList2 <- strsplit(temp2, split=" ?[,;] ?")
myList2
[[1]]
[1] "this"  "that"  "other"

[[2]]
[1] "hat"   "scarf" "food"

[[3]]
[1] "woman" "man"   "child"
```

注意事项:

1. 分解正则表达式语法超出本示例范围。
2. 有时匹配正则表达式会减慢进程。与许多允许使用正则表达式的R函数一样，fixed参数可用于告诉R对分割字符进行字面匹配。

Chapter 57: strsplit function

Section 57.1: Introduction

strsplit is a useful function for breaking up a vector into a list on some character pattern. With typical R tools, the whole list can be reincorporated to a data.frame or part of the list might be used in a graphing exercise.

Here is a common usage of **strsplit**: break a character vector along a comma separator:

```
temp <- c("this,that,other", "hat,scarf,food", "woman,man,child")
# get a list split by commas
myList <- strsplit(temp, split=",")
# print myList
myList
[[1]]
[1] "this"  "that"  "other"

[[2]]
[1] "hat"   "scarf" "food"

[[3]]
[1] "woman" "man"   "child"
```

As hinted above, the split argument is not limited to characters, but may follow a pattern dictated by a regular expression. For example, temp2 is identical to temp above except that the separators have been altered for each item. We can take advantage of the fact that the split argument accepts regular expressions to alleviate the irregularity in the vector.

```
temp2 <- c("this, that, other", "hat,scarf ,food", "woman; man ; child")
myList2 <- strsplit(temp2, split=" ?[,;] ?")
myList2
[[1]]
[1] "this"  "that"  "other"

[[2]]
[1] "hat"   "scarf" "food"

[[3]]
[1] "woman" "man"   "child"
```

Notes:

1. breaking down the regular expression syntax is out of scope for this example.
2. Sometimes matching regular expressions can slow down a process. As with many R functions that allow the use of regular expressions, the fixed argument is available to tell R to match on the split characters literally.

第58章：网页抓取与解析

第58.1节：使用rvest进行基础抓取

rvest是Hadley Wickham开发的一个网页抓取与解析包，灵感来自Python的Beautiful Soup。它利用了Hadley的xml2包中libxml2绑定进行HTML解析。

作为tidyverse的一部分，rvest支持管道操作。它使用

- xml2::read_html来抓取网页的HTML，
- 然后可以使用CSS或XPath选择器通过其html_node和html_nodes函数进行子集选择，并用html_text和html_table等函数解析为R对象。

要抓取维基百科关于R的里程碑表格，代码如下

```
library(rvest)  
url <- 'https://en.wikipedia.org/wiki/R_(programming_language)'
```

```
# 从网站抓取HTML  
url %>% read_html() %>%  
  # 选择class="wikitable"的HTML标签  
  html_node(css = '.wikitable') %>%  
    # 将表格解析为data.frame  
  html_table() %>%  
    # 修剪以便打印  
  dplyr::mutate(Description = substr(Description, 1, 70))
```

Description

```
##   Release      Date  
## 1   0.16        这是由伊哈卡 (Ihaka) 主要开发的最后一个alpha版本  
## 2   0.49 1997-04-23 这是目前可用的最早的源代码版本  
## 3   0.60 1997-12-05 R成为GNU项目的官方组成部分。代码被  
## 4   0.65.1 1999-10-07 update.packages和install.packages函数的第一个版本  
## 5   1.0 2000-02-29 被开发者认为足够稳定，可用于生产环境  
## 6   1.4 2001-12-19 引入了S4方法，并发布了Mac OS X的第一个版本  
## 7   2.0 2004-10-04 引入了惰性加载，支持快速加载数据  
## 8   2.1 2005-04-18 支持UTF-8编码，并开始国际化  
## 9   2.11 2010-04-22 支持Windows 64位系统。  
## 10  2.13 2011-04-14 添加了一个新的编译器函数，可以加速函数执行  
## 11  2.14 2011-10-31 为包添加了强制命名空间。新增了并行功能  
## 12  2.15 2012-03-30 新的负载均衡函数。提高了序列化速度  
## 13  3.0 2013-04-03 支持64位系统上数值索引值231及以上
```

虽然这会返回一个data.frame，但请注意，正如抓取数据的典型情况一样，仍需进一步清理数据：例如格式化日期、插入NA等。

请注意，数据如果格式不够规整，可能需要循环或其他进一步处理才能成功解析。如果网站使用了jQuery或其他方式插入内容，read_html可能不足以抓取，可能需要更强大的抓取工具，如RSelenium。

第58.2节：登录时使用 rvest

抓取网页时常见的问题是如何输入用户ID和密码登录网站。

在这个示例中，我创建了一个用于跟踪我在 Stack Overflow 上发布答案的流程。整体流程是登录，访问网页收集信息，将其添加到数据框中，然后跳转到下一页。

Chapter 58: Web scraping and parsing

Section 58.1: Basic scraping with rvest

rvest is a package for web scraping and parsing by Hadley Wickham inspired by Python's BeautifulSoup. It leverages Hadley's xml2 package's libxml2 bindings for HTML parsing.

As part of the tidyverse, rvest is piped. It uses

- xml2::read_html to scrape the HTML of a webpage,
- which can then be subset with its html_node and html_nodes functions using CSS or XPath selectors, and
- parsed to R objects with functions like html_text and html_table.

To scrape the table of milestones from the Wikipedia page on R, the code would look like

```
library(rvest)  
  
url <- 'https://en.wikipedia.org/wiki/R_(programming_language)'  
  
# scrape HTML from website  
url %>% read_html() %>%  
  # select HTML tag with class="wikitable"  
  html_node(css = '.wikitable') %>%  
    # parse table into data.frame  
  html_table() %>%  
    # trim for printing  
  dplyr::mutate(Description = substr(Description, 1, 70))
```

```
##   Release      Date  
## 1   0.16        This is the last alpha version developed primarily by Ihaka  
## 2   0.49 1997-04-23 This is the oldest source release which is currently available.  
## 3   0.60 1997-12-05 R becomes an official part of the GNU Project. The code is now  
## 4   0.65.1 1999-10-07 First versions of update.packages and install.packages function  
## 5   1.0 2000-02-29 Considered by its developers stable enough for production use.  
## 6   1.4 2001-12-19 S4 methods are introduced and the first version for Mac OS X  
## 7   2.0 2004-10-04 Introduced lazy loading, which enables fast loading of data  
## 8   2.1 2005-04-18 Support for UTF-8 encoding, and the beginnings of internationalization  
## 9   2.11 2010-04-22 Support for Windows 64 bit systems.  
## 10  2.13 2011-04-14 Adding a new compiler function that allows speeding up function calls  
## 11  2.14 2011-10-31 Added mandatory namespaces for packages. Added a new parallel  
## 12  2.15 2012-03-30 New load balancing functions. Improved serialization speed for  
## 13  3.0 2013-04-03 Support for numeric index values 231 and larger on 64 bit systems.
```

While this returns a data.frame, note that as is typical for scraped data, there is still further data cleaning to be done: here, formatting dates, inserting NAs, and so on.

Note that data in a less consistently rectangular format may take looping or other further munging to successfully parse. If the website makes use of jQuery or other means to insert content, read_html may be insufficient to scrape, and a more robust scraper like RSelenium may be necessary.

Section 58.2: Using rvest when login is required

I common problem encounter when scrapping a web is how to enter a userid and password to log into a web site.

In this example which I created to track my answers posted here to stack overflow. The overall flow is to login, go to a web page collect information, add it a dataframe and then move to the next page.

```
library(rvest)
```

```
# 登录网页地址
login<-  
"https://stackoverflow.com/users/login?ssrc=head&returnurl=http%3a%2f%2fstackoverflow.com%2f"

# 创建一个包含目标登录地址的网页会话
pgsession<-html_session(login)
pgform<-html_form(pgsession)[[2]] # 在此案例中，提交表单是第二个表单
filled_form<-set_values(pgform, email="*****", password="*****")
submit_form(pgsession, filled_form)

# 预分配最终结果的数据框。
results<-data.frame()

#遍历所有包含所需信息的页面
for (i in 1:5)
{
  #提取信息的页面基础地址
  url<-"http://stackoverflow.com/users/*****?tab=answers&sort=activity&page="
  url<-paste0(url, i)
  page<-jump_to(pgsession, url)

  #收集问题投票和问题标题的信息
  summary<-html_nodes(page, "div .answer-summary")
  question<-matrix(html_text(html_nodes(summary, "div"), trim=TRUE), ncol=2, byrow = TRUE)

  #查找回答日期、超链接及是否被采纳
  dateans<-html_node(summary, "span") %>% html_attr("title")
  hyperlink<-html_node(summary, "div a") %>% html_attr("href")
  accepted<-html_node(summary, "div") %>% html_attr("class")

  #创建临时结果然后合并到最终结果中
  rtemp<-cbind(question, dateans, accepted, hyperlink)
  results<-rbind(results, rtemp)
}

#数据框清理
names(results)<-c("Votes", "Answer", "Date", "Accepted", "HyperLink")
results$Votes<-as.integer(as.character(results$Votes))
results$Accepted<-ifelse(results$Accepted=="answer-votes default", 0, 1)
```

此处循环限制为仅5页，需根据您的应用进行调整。我将用户特定的值替换为*****，希望这能为您的问题提供一些指导。

```
library(rvest)
```

```
#Address of the login webpage
login<-
"https://stackoverflow.com/users/login?ssrc=head&returnurl=http%3a%2f%2fstackoverflow.com%2f"

#create a web session with the desired login address
pgsession<-html_session(login)
pgform<-html_form(pgsession)[[2]] #in this case the submit is the 2nd form
filled_form<-set_values(pgform, email="****", password="****")
submit_form(pgsession, filled_form)

#pre allocate the final results dataframe.
results<-data.frame()

#loop through all of the pages with the desired info
for (i in 1:5)
{
  #base address of the pages to extract information from
  url<-"http://stackoverflow.com/users/*****?tab=answers&sort=activity&page="
  url<-paste0(url, i)
  page<-jump_to(pgsession, url)

  #collect info on the question votes and question title
  summary<-html_nodes(page, "div .answer-summary")
  question<-matrix(html_text(html_nodes(summary, "div"), trim=TRUE), ncol=2, byrow = TRUE)

  #find date answered, hyperlink and whether it was accepted
  dateans<-html_node(summary, "span") %>% html_attr("title")
  hyperlink<-html_node(summary, "div a") %>% html_attr("href")
  accepted<-html_node(summary, "div") %>% html_attr("class")

  #create temp results then bind to final results
  rtemp<-cbind(question, dateans, accepted, hyperlink)
  results<-rbind(results, rtemp)
}

#Dataframe Clean-up
names(results)<-c("Votes", "Answer", "Date", "Accepted", "HyperLink")
results$Votes<-as.integer(as.character(results$Votes))
results$Accepted<-ifelse(results$Accepted=="answer-votes default", 0, 1)
```

The loop in this case is limited to only 5 pages, this needs to change to fit your application. I replaced the user specific values with *****, hopefully this will provide some guidance for you problem.

第59章：广义线性模型

第59.1节：泰坦尼克号数据集上的逻辑回归

逻辑回归是广义线性模型的一种特殊情况，用于建模二分结果（概率单位和互补对数-对数模型与之密切相关）。

名称来源于所用的连接函数，即logit或对数几率函数。logit的逆函数称为逻辑函数，定义为：

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

该函数接受一个介于 $]-\infty; +\infty[$ 之间的值，返回一个介于0和1之间的值；即逻辑函数将线性预测值转换为概率。

逻辑回归可使用glm函数执行，选项为family = binomial（即family = binomial(link="logit")的简写；logit是二项分布族的默认连接函数）。

在本例中，我们尝试预测RMS泰坦尼克号上乘客的命运。

读取数据：

```
url <- "http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic.txt"
titanic <- read.csv(file = url, stringsAsFactors = FALSE)
```

清理缺失值：

在这种情况下，我们用一个近似值——平均值来替代缺失值。

```
titanic$age[is.na(titanic$age)] <- mean(titanic$age, na.rm = TRUE)
```

训练模型：

```
titanic.train <- glm(survived ~ pclass + sex + age,
                      family = binomial, data = titanic)
```

模型摘要：

```
summary(titanic.train)
```

输出结果：

调用：

```
glm(formula = survived ~ pclass + sex + age, family = binomial, data = titanic)
```

Deviance Residuals:

最小值	第一四分位数	中位数	第三四分位数	最大值
-2.6452	-0.6641	-0.3679	0.6123	2.5615

系数：

估计值	标准误差	z值	P(> z)
(截距) 3.552261	0.342188	10.381	< 2e-16 ***
舱位2等舱 -1.170777	0.211559	-5.534	3.13e-08 ***
舱位3等舱 -2.430672	0.195157	-12.455	< 2e-16 ***

Chapter 59: Generalized linear models

Section 59.1: Logistic regression on Titanic dataset

Logistic regression is a particular case of the *generalized linear model*, used to model dichotomous outcomes (*probit* and *complementary log-log* models are closely related).

The name comes from the *link function* used, the *logit* or *log-odds* function. The inverse function of the *logit* is called the *logistic function* and is given by:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

This function takes a value between $]-\infty; +\infty[$ and returns a value between 0 and 1; i.e the *logistic function* takes a linear predictor and returns a probability.

Logistic regression can be performed using the `glm` function with the option `family = binomial` (shortcut for `family = binomial(link="logit")`; the *logit* being the default link function for the binomial family).

In this example, we try to predict the fate of the passengers aboard the RMS Titanic.

Read the data:

```
url <- "http://biostat.mc.vanderbilt.edu/wiki/pub/Main/DataSets/titanic.txt"
titanic <- read.csv(file = url, stringsAsFactors = FALSE)
```

Clean the missing values:

In that case, we replace the missing values by an approximation, the average.

```
titanic$age[is.na(titanic$age)] <- mean(titanic$age, na.rm = TRUE)
```

Train the model:

```
titanic.train <- glm(survived ~ pclass + sex + age,
                      family = binomial, data = titanic)
```

Summary of the model:

```
summary(titanic.train)
```

The output:

Call:

```
glm(formula = survived ~ pclass + sex + age, family = binomial, data = titanic)
```

Deviance Residuals:

Min	1Q	Median	3Q	Max
-2.6452	-0.6641	-0.3679	0.6123	2.5615

Coefficients:

	Estimate	Std. Error	z value	Pr(> z)
(Intercept)	3.552261	0.342188	10.381	< 2e-16 ***
pclass2nd	-1.170777	0.211559	-5.534	3.13e-08 ***
pclass3rd	-2.430672	0.195157	-12.455	< 2e-16 ***

```

性别男 -2.463377 0.154587 -15.935 < 2e-16 ***
年龄 -0.042235 0.007415 -5.696 1.23e-08 ***
---
显著性代码: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '' 1

```

(二项分布族的离散参数取为 1)

```

零模型 偏差: 1686.8 自由度 1312
残差 偏差: 1165.7 自由度 1308
赤池信息量准则(AIC): 1175.7

```

Fisher评分迭代次数: 5

- 首先显示的是调用信息。它提醒用户模型及指定的选项。
- 接下来看到的是偏差残差，它是模型拟合优度的度量。输出的这一部分显示了模型中各个个案的偏差残差分布。
- 输出的下一部分显示了系数、它们的标准误差、z统计量（有时称为Wald z统计量）及其对应的p值。
 - 定性变量被“虚拟化”。某一类别被视为参考类别。参考类别可以通过公式中的I进行更改。
 - 所有四个预测变量在0.1%的显著性水平上均具有统计学意义。
 - 逻辑回归系数表示预测变量增加一个单位时结果的对数几率的变化。
 - 要查看几率比（预测变量每增加一个单位，生存几率的乘法变化），对参数取指数。
 - 要查看参数的置信区间（CI），使用confint。
- 系数表下方是拟合指标，包括零残差和偏差残差以及赤池信息量准则（AIC），可用于比较模型性能。
 - 在比较用最大似然法拟合同一数据的模型时，AIC越小，拟合越好。
 - 模型拟合的一个衡量标准是整体模型的显著性。该检验用于判断带有预测变量的模型是否显著优于仅含截距项的模型（即零模型）。

几率比示例：

```

exp(coef(titanic.train)[3])
pclass3rd
0.08797765

```

使用该模型，与头等舱相比，三等舱乘客的生存几率约为十分之一。

参数置信区间示例：

```

confint(titanic.train)
正在等待分析完成...
2.5 % 97.5 %
(截距) 2.89486872 4.23734280
pclass2nd -1.58986065 -0.75987230
pclass3rd -2.81987935 -2.05419500

```

```

sexmale -2.463377 0.154587 -15.935 < 2e-16 ***
age -0.042235 0.007415 -5.696 1.23e-08 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '' 1

```

(Dispersion parameter for binomial family taken to be 1)

```

Null deviance: 1686.8 on 1312 degrees of freedom
Residual deviance: 1165.7 on 1308 degrees of freedom
AIC: 1175.7

```

Number of Fisher Scoring iterations: 5

- The first thing displayed is the call. It is a reminder of the model and the options specified.
- Next we see the deviance residuals, which are a measure of model fit. This part of output shows the distribution of the deviance residuals for individual cases used in the model.
- The next part of the output shows the coefficients, their standard errors, the z-statistic (sometimes called a Wald z-statistic), and the associated p-values.
 - The qualitative variables are "dummified". A modality is considered as the reference. The reference modality can be change with I in the formula.
 - All four predictors are statistically significant at a 0.1 % level.
 - The logistic regression coefficients give the change in the log odds of the outcome for a one unit increase in the predictor variable.
 - To see the odds ratio (multiplicative change in the odds of survival per unit increase in a predictor variable), exponentiate the parameter.
 - To see the confidence interval (CI) of the parameter, use confint.
- Below the table of coefficients are fit indices, including the null and deviance residuals and the Akaike Information Criterion (AIC), which can be used for comparing model performance.
 - When comparing models fitted by maximum likelihood to the same data, the smaller the AIC, the better the fit.
 - One measure of model fit is the significance of the overall model. This test asks whether the model with predictors fits significantly better than a model with just an intercept (i.e., a null model).

Example of odds ratios:

```

exp(coef(titanic.train)[3])
pclass3rd
0.08797765

```

With this model, compared to the first class, the 3rd class passengers have about a tenth of the odds of survival.

Example of confidence interval for the parameters:

```

confint(titanic.train)
Waiting for profiling to be done...
2.5 % 97.5 %
(Intercept) 2.89486872 4.23734280
pclass2nd -1.58986065 -0.75987230
pclass3rd -2.81987935 -2.05419500

```

```
sexmale      -2.77180962 -2.16528316  
age         -0.05695894 -0.02786211
```

整体模型显著性计算示例：

检验统计量服从卡方分布，自由度等于当前模型与零模型之间自由度的差值（即模型中预测变量的数量）。

```
with(titanic.train, pchisq(null.deviance - deviance, df.null - df.residual  
, lower.tail = FALSE))  
[1] 1.892539e-111
```

p值接近0，表明模型具有高度显著性。

```
sexmale      -2.77180962 -2.16528316  
age         -0.05695894 -0.02786211
```

Exemple of calculating the significance of the overall model:

The test statistic is distributed chi-squared with degrees of freedom equal to the differences in degrees of freedom between the current and the null model (i.e., the number of predictor variables in the model).

```
with(titanic.train, pchisq(null.deviance - deviance, df.null - df.residual  
, lower.tail = FALSE))  
[1] 1.892539e-111
```

The p-value is near 0, showing a strongly significant model.

第60章：在长格式和宽格式之间重塑数据

在R语言中，表格数据存储在数据框中。本章节涵盖了转换单个表格的各种方法。

第60.1节：重塑数据

数据通常以表格形式出现。一般来说，可以将这种表格数据分为宽格式和长格式。在宽格式中，每个变量都有自己的列。

人员 身高[厘米] 年龄[岁]

艾莉森	178	20
鲍勃	174	45
卡尔	182	31

然而，有时使用长格式更为方便，在长格式中，所有变量都在一列中，数值则在第二列中。

人员 变量 值

Alison	身高[厘米]	178
Bob	身高[厘米]	174
Carl	身高[厘米]	182
Alison	年龄[岁]	20
Bob	年龄[岁]	45
Carl	年龄[岁]	31

可以使用基础R以及第三方包来简化此过程。对于每个选项，将使用mtcars数据集。默认情况下，该数据集是长格式。为了使这些包能够工作，我们将把行名插入为第一列。

```
mtcars # 显示数据集  
data <- data.frame(observation=row.names(mtcars),mtcars)
```

基础R

在基础R中有两个函数可以用来在宽格式和长格式之间转换：`stack()` 和 `unstack()`。

```
long <- stack(data)  
long # 这显示了长格式  
wide <- unstack(long)  
wide # 这显示了宽格式
```

然而，对于更高级的用例，这些函数可能变得非常复杂。幸运的是，还有其他选项使用第三方包。

tidyR 包

该包使用 `gather()` 将宽格式转换为长格式，使用 `spread()` 将长格式转换为宽格式。

```
library(tidyR)  
long <- gather(data, variable, value, 2:12) # 其中 variable 是变量列的名称, value 表示值列的名称  
, 2:12 指的是
```

Chapter 60: Reshaping data between long and wide forms

In R, tabular data is stored in data frames. This topic covers the various ways of transforming a single table.

Section 60.1: Reshaping data

Often data comes in tables. Generally one can divide this tabular data in wide and long formats. In a wide format, each variable has its own column.

Person Height [cm] Age [yr]

Alison	178	20
Bob	174	45
Carl	182	31

However, sometimes it is more convenient to have a long format, in which all variables are in one column and the values are in a second column.

Person Variable Value

Alison	Height [cm]	178
Bob	Height [cm]	174
Carl	Height [cm]	182
Alison	Age [yr]	20
Bob	Age [yr]	45
Carl	Age [yr]	31

Base R, as well as third party packages can be used to simplify this process. For each of the options, the `mtcars` dataset will be used. By default, this dataset is in a long format. In order for the packages to work, we will insert the row names as the first column.

```
mtcars # shows the dataset  
data <- data.frame(observation=row.names(mtcars),mtcars)
```

Base R

There are two functions in base R that can be used to convert between wide and long format: `stack()` and `unstack()`.

```
long <- stack(data)  
long # this shows the long format  
wide <- unstack(long)  
wide # this shows the wide format
```

However, these functions can become very complex for more advanced use cases. Luckily, there are other options using third party packages.

The tidyR package

This package uses `gather()` to convert from wide to long and `spread()` to convert from long to wide.

```
library(tidyR)  
long <- gather(data, variable, value, 2:12) # where variable is the name of the  
# variable column, value indicates the name of the value column and 2:12 refers to
```

```
# 要转换的列。
long # 显示长格式结果
wide <- spread(long, variable, value)
wide # 显示宽格式结果 (~data)
```

data.table 包

data.table 包扩展了 reshape2 函数，使用 melt() 函数将宽格式转换为长格式，使用 dcast() 函数将长格式转换为宽格式。

```
library(data.table)
long <- melt(data, 'observation', 2:12, 'variable', 'value')
long # 显示长格式结果
wide <- dcast(long, observation ~ variable)
wide # 显示宽格式结果 (~data)
```

第60.2节：reshape函数

最灵活的基础R数据重塑函数是reshape。语法请参见?reshape。

```
# 创建不平衡的纵向（面板）数据集
set.seed(1234)
df <- data.frame(identifier=rep(1:5, each=3),
                  location=rep(c("up", "down", "left", "up", "center"), each=3),
                  period=rep(1:3, 5), counts=sample(35, 15, replace=TRUE),
                  values=rnorm(15, 5, 10))[-c(4, 8, 11),]
df
```

	identifier	location	period	counts	values
1	1	up	1	4	9.186478
2	1	up	2	22	6.431116
3	1	向上	3	22	6.334104
5	2	向下	2	31	6.161130
6	2	向下	3	23	6.583062
7	3	向左	1	1	6.513467
9	3	向左	3	24	5.199980
10	4	向上	1	18	6.093998
12	4	向上	3	20	7.628488
13	5	中心	1	10	9.573291
14	5	中心	2	33	9.156725
15	5	中心	3	11	5.228851

请注意，该数据框是不平衡的，即单位2在第一个时期缺少一个观测值，而单位3和4在第二个时期缺少观测值。同时，请注意有两个变量随时期变化：计数和值，以及两个不变的变量：标识符和位置。

长格式转宽格式

将数据框重塑为宽格式，

```
# 按时间变量重塑为宽格式
df.wide <- reshape(df, idvar="identifier", timevar="period",
v.names=c("values", "counts"), direction="wide")
df.wide
```

	标识符	位置	values.1	counts.1	values.2	counts.2	values.3	counts.3
1	1	上	9.186478	4	6.431116	22	6.334104	22
5	2	下	NA	NA	6.161130	31	6.583062	23
7	3	左	6.513467	1	NA	NA	5.199980	24
10	4	上	6.093998	18	NA	NA	7.628488	20

```
# the columns to be converted.
long # shows the long result
wide <- spread(long, variable, value)
wide # shows the wide result (~data)
```

The data.table package

The data.table package extends the reshape2 functions and uses the function melt() to go from wide to long and dcast() to go from long to wide.

```
library(data.table)
long <- melt(data, 'observation', 2:12, 'variable', 'value')
long # shows the long result
wide <- dcast(long, observation ~ variable)
wide # shows the wide result (~data)
```

Section 60.2: The reshape function

The most flexible base R function for reshaping data is `reshape`. See `?reshape` for its syntax.

```
# create unbalanced longitudinal (panel) data set
set.seed(1234)
df <- data.frame(identifier=rep(1:5, each=3),
                  location=rep(c("up", "down", "left", "up", "center"), each=3),
                  period=rep(1:3, 5), counts=sample(35, 15, replace=TRUE),
                  values=rnorm(15, 5, 10))[-c(4, 8, 11),]
df
```

	identifier	location	period	counts	values
1	1	up	1	4	9.186478
2	1	up	2	22	6.431116
3	1	up	3	22	6.334104
5	2	down	2	31	6.161130
6	2	down	3	23	6.583062
7	3	left	1	1	6.513467
9	3	left	3	24	5.199980
10	4	up	1	18	6.093998
12	4	up	3	20	7.628488
13	5	center	1	10	9.573291
14	5	center	2	33	9.156725
15	5	center	3	11	5.228851

Note that the data.frame is unbalanced, that is, unit 2 is missing an observation in the first period, while units 3 and 4 are missing observations in the second period. Also, note that there are two variables that vary over the periods: counts and values, and two that do not vary: identifier and location.

Long to Wide

To reshape the data.frame to wide format,

```
# reshape wide on time variable
df.wide <- reshape(df, idvar="identifier", timevar="period",
v.names=c("values", "counts"), direction="wide")
df.wide
```

	identifier	location	values.1	counts.1	values.2	counts.2	values.3	counts.3
1	1	up	9.186478	4	6.431116	22	6.334104	22
5	2	down	NA	NA	6.161130	31	6.583062	23
7	3	left	6.513467	1	NA	NA	5.199980	24
10	4	up	6.093998	18	NA	NA	7.628488	20

注意缺失的时间段用NA填充。

在重塑为宽格式时，“v.names”参数指定随时间变化的列。如果位置变量不是必需的，可以在重塑之前通过“drop”参数将其删除。在从数据框中删除唯一的不变/非ID列后，v.names参数变得不再必要。

```
reshape(df, idvar="identifier", timevar="period", direction="wide",
       drop="location")
```

宽格式转长格式

要用当前的df.wide重塑成长格式，最简语法是

```
reshape(df.wide, direction="long")
```

然而，这通常更复杂：

```
# 删除df.wide中计数和值名称中的"."分隔符
names(df.wide)[grep("\\.", names(df.wide))] <-
  gsub("\\.", "", names(df.wide)[grep("\\.", names(df.wide))])
```

现在简单的语法会产生关于未定义列的错误。

对于列名较难被reshape函数自动解析的情况，有时需要添加“varying”参数，该参数告诉reshape将宽格式中特定变量分组，以便转换成长格式。该参数接受变量名或索引的向量列表。

```
reshape(df.wide, idvar="identifier",
       varying=list(c(3,5,7), c(4,6,8)), direction="long")
```

在转换成长格式时，可以提供“v.names”参数来重命名生成的变化变量。

有时可以通过使用“sep”参数来避免指定“varying”，该参数告诉reshape变量名的哪一部分指定值参数，哪一部分指定时间参数。

Notice that the missing time periods are filled in with NAs.

In reshaping wide, the “v.names” argument specifies the columns that vary over time. If the location variable is not necessary, it can be dropped prior to reshaping with the “drop” argument. In dropping the only non-varying / non-id column from the data.frame, the v.names argument becomes unnecessary.

```
reshape(df, idvar="identifier", timevar="period", direction="wide",
       drop="location")
```

Wide to Long

To reshape long with the current df.wide, a minimal syntax is

```
reshape(df.wide, direction="long")
```

However, this is typically trickier:

```
# remove "." separator in df.wide names for counts and values
names(df.wide)[grep("\\.", names(df.wide))] <-
  gsub("\\.", "", names(df.wide)[grep("\\.", names(df.wide))])
```

Now the simple syntax will produce an error about undefined columns.

With column names that are more difficult for the **reshape** function to automatically parse, it is sometimes necessary to add the “varying” argument which tells **reshape** to group particular variables in wide format for the transformation into long format. This argument takes a list of vectors of variable names or indices.

```
reshape(df.wide, idvar="identifier",
       varying=list(c(3,5,7), c(4,6,8)), direction="long")
```

In reshaping long, the “v.names” argument can be provided to rename the resulting varying variables.

Sometimes the specification of “varying” can be avoided by use of the “sep” argument which tells **reshape** what part of the variable name specifies the value argument and which specifies the time argument.

第61章：RMarkdown和knitr 演示

参数	定义
标题	文档的标题
作者	文件的作者
日期	文档的日期：可以是 "r format(Sys.time(), '%d %B, %Y')"
作者	文件的作者
输出	文档的输出格式：至少有10种格式可用。对于html文档，使用 <code>html_output</code> 。对于PDF文档，使用 <code>pdf_document</code> ，等等。

第61.1节：为ioslides演示文稿添加页脚

原生不支持添加页脚。幸运的是，我们可以利用jQuery和CSS为使用knitr渲染的ioslides演示文稿的幻灯片添加页脚。首先，我们必须引入jQuery插件。这通过以下代码实现：

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.2/jquery.min.js"></script>
```

现在我们可以使用jQuery来修改演示文稿的DOM（文档对象模型）。换句话说：我们修改文档的HTML结构。当演示文稿加载完成时 `$(document).ready(function() { ... })`，我们选择所有不带有类属性 `.title-slide`、`.backdrop` 或 `.segue` 的幻灯片，并在每个幻灯片“关闭”之前（即在 `</slide>` 之前）添加标签`<footer></footer>`。属性 `label` 包含稍后显示的内容。

现在我们只需用CSS来布局我们的页脚：

在每个`<footer>` (`footer::after`) 之后：

- 显示属性`label`
- 内容，使用字体大小12
- 将页脚定位（距离幻灯片底部20像素，距离左侧60像素）

（其他属性可以忽略，但如果演示文稿使用不同的样式

模板，可能需要修改）。

```
---  
标题：“向演示幻灯片添加页脚”  
作者：“马丁·施梅尔策”  
日期：“2016年7月26日”  
输出：ioslides_presentation  
---
```

```
```{r setup, include=FALSE}  
knitr::opts_chunk$set(echo = FALSE)
```
```

Chapter 61: RMarkdown and knitr presentation

| Parameter | definition |
|-----------|---|
| title | the title of the document |
| author | The author of the document |
| date | The date of the document: Can be "r <code>format(Sys.time(), '%d %B, %Y')</code> " |
| author | The author of the document |
| output | The output format of the document: at least 10 format available. For html document, <code>html_output</code> . For PDF document, <code>pdf_document</code> , .. |

Section 61.1: Adding a footer to an ioslides presentation

Adding a footer is not natively possible. Luckily, we can make use of jQuery and CSS to add a footer to the slides of an ioslides presentation rendered with knitr. First of all we have to include the jQuery plugin. This is done by the line

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.2/jquery.min.js"></script>
```

Now we can use jQuery to alter the DOM (*document object model*) of our presentation. In other words: we alter the HTML structure of the document. As soon as the presentation is loaded `$(document).ready(function() { ... })`，we select all slides, that do not have the class attributes `.title-slide`, `.backdrop`, or `.segue` and add the tag `<footer></footer>` right before each slide is 'closed' (so before `</slide>`). The attribute `label` carries the content that will be displayed later on.

All we have to do now is to layout our footer with CSS:

After each `<footer>` (`footer::after`):

- display the content of the attribute `label`
- use font size 12
- position the footer (20 pixels from the bottom of the slide and 60 pxs from the left)

(the other properties can be ignored but might have to be modified if the presentation uses a different style template).

```
---  
title: "Adding a footer to presentation slides"  
author: "Martin Schmelzer"  
date: "26 Juli 2016"  
output: ioslides_presentation  
---
```

```
```{r setup, include=FALSE}  
knitr::opts_chunk$set(echo = FALSE)
```
```

```
## 幻灯片1
```

这是幻灯片1。

```
## 幻灯片2
```

这是幻灯片2

```
# 测试
```

```
## 幻灯片 3
```

以及幻灯片 3。

结果将如下所示：

Slide 1

This is slide 1.

My amazing footer

2/5

第61.2节：Rstudio示例

这是一个保存为.Rmd的脚本，与保存为.R的r脚本相反。

要编织该脚本，可以使用render函数或使用Rstudio中的快捷按钮。

```
---
title: "Rstudio的rmd文件示例"
author: 'stack user'
date: "2016年7月22日"
```

```
## Slide 1
```

This is slide 1.

```
## Slide 2
```

This is slide 2

```
# Test
```

```
## Slide 3
```

And slide 3.

The result will look like this:

Slide 1

This is slide 1.

My amazing footer

2/5

Section 61.2: Rstudio example

This is a script saved as .Rmd, on the contrary of r scripts saved as .R.

To knit the script, either use the render function or use the shortcut button in Rstudio.

```
---
title: "Rstudio exemple of a rmd file"
author: 'stack user'
date: "22 July 2016"
```

```
output: html_document
```

```
---
```

头部用于定义通用参数和元数据。

```
## R Markdown
```

这是一个 R Markdown 文档。

它是用 markdown 编写的脚本，可以插入 R 代码块。

要插入 R 代码，需要将其封装在反引号中。

长代码块示例如下：

```
```{r cars}
summary(cars)
```
```

小代码块示例如 ``r cat("that")``。

```
## 插入图表
```

你也可以嵌入图表，例如：

```
```{r echo=FALSE}
plot(pressure)
```
```

```
output: html_document
```

```
---
```

The header is used to define the general parameters and the metadata.

```
## R Markdown
```

This is an R Markdown document.

It is a script written in markdown with the possibility to insert chunk of R code in it.

To insert R code, it needs to be encapsulated into inverted quote.

Like that for a long piece of code:

```
```{r cars}
summary(cars)
```
```

And like ``r cat("that")`` for small piece of code.

```
## Including Plots
```

You can also embed plots, for example:

```
```{r echo=FALSE}
plot(pressure)
```
```

第62章：变量的作用域

第62.1节：环境与功能

函数内部声明的变量仅存在于该函数内部（除非作为参数传入）。

```
x <- 1

foo <- function(x) {
  y <- 3
  z <- x + y
  return(z)
}

y
```

错误：未找到对象 'y'

传入函数的变量然后被重新赋值，会被覆盖，但仅在函数内部。

```
foo <- function(x) {
  x <- 2
  y <- 3
  z <- x + y
  return(z)
}

foo(1)
x
```

5
1

在函数之外的更高环境中赋值的变量存在于该函数内，无需传递。

```
foo <- function() {
  y <- 3
  z <- x + y
  return(z)
}

foo()
```

4

第62.2节：函数退出

on.exit()函数对于变量清理非常有用，尤其是在必须分配全局变量时。

某些参数，特别是图形参数，只能全局设置。当

Chapter 62: Scope of variables

Section 62.1: Environments and Functions

Variables declared inside a function only exist (unless passed) inside that function.

```
x <- 1

foo <- function(x) {
  y <- 3
  z <- x + y
  return(z)
}

y
```

Error: object 'y' not found

Variables passed into a function and then reassigned are overwritten, *but only inside the function*.

```
foo <- function(x) {
  x <- 2
  y <- 3
  z <- x + y
  return(z)
}

foo(1)
x
```

5
1

Variables assigned in a higher environment than a function exist within that function, without being passed.

```
foo <- function() {
  y <- 3
  z <- x + y
  return(z)
}

foo()
```

4

Section 62.2: Function Exit

The `on.exit()` function is handy for variable clean up if global variables must be assigned.

Some parameters, especially those for graphics, can only be set globally. This small function is common when

创建更专业的图形时，这个小函数很常见。

```
new_plot <- function(...) {  
  old_pars <- par(mar = c(5,4,4,2) + .1, mfrow = c(1,1))  
  on.exit(par(old_pars))  
  plot(...)  
}
```

第62.3节：子功能

函数内部调用的函数（即子函数）必须在该函数内部定义，才能访问局部环境中定义的变量，而无需传递这些变量。

这会失败：

```
bar <- function() {  
  z <- x + y  
  return(z)  
}  
  
foo <- function() {  
  y <- 3  
  z <- bar()  
  return(z)  
}  
  
foo()
```

bar() 出错：找不到对象 'y'

这可以正常运行：

```
foo <- function() {  
  bar <- function() {  
    z <- x + y  
    return(z)  
  }  
  
  y <- 3  
  z <- bar()  
  return(z)  
}  
  
foo()
```

4

第62.4节：全局赋值

变量可以使用<<-从任何环境进行全局赋值。bar()现在可以访问 y。

```
bar <- function() {
```

creating more specialized plots.

```
new_plot <- function(...) {  
  old_pars <- par(mar = c(5,4,4,2) + .1, mfrow = c(1,1))  
  on.exit(par(old_pars))  
  plot(...)  
}
```

Section 62.3: Sub functions

Functions called within a function (ie subfunctions) must be defined within that function to access any variables defined in the local environment without being passed.

This fails:

```
bar <- function() {  
  z <- x + y  
  return(z)  
}  
  
foo <- function() {  
  y <- 3  
  z <- bar()  
  return(z)  
}  
  
foo()
```

Error in bar(): object 'y' not found

This works:

```
foo <- function() {  
  bar <- function() {  
    z <- x + y  
    return(z)  
  }  
  
  y <- 3  
  z <- bar()  
  return(z)  
}  
  
foo()
```

4

Section 62.4: Global Assignment

Variables can be assigned globally from any environment using <<-. bar() can now access y.

```
bar <- function() {
```

```
z <- x + y
  return(z)
}

foo <- function() {
  y <<- 3
  z <- bar()
  return(z)
}

foo()
```

4

强烈不建议使用全局赋值。更推荐使用包装函数或显式调用另一个局部环境中的变量。

第62.5节：环境和变量的显式赋值

R中的环境可以被显式调用和命名。变量可以被显式赋值并从这些环境中调用或传入。

一个常见创建的环境是包含package:base或package:base内的子环境的环境。

```
e1 <- new.env(parent = baseenv())
e2 <- new.env(parent = e1)
```

变量可以被显式赋值并从这些环境中调用或传入。

```
assign("a", 3, envir = e1)
get("a", envir = e1)
get("a", envir = e2)
```

3

3

由于 e2 继承自 e1，a 在 e1 和 e2 中都是 3。然而，在 e2 中赋值 a 并不会改变 e1 中 a 的值。

```
assign("a", 2, envir = e2)
get("a", envir = e2)
get("a", envir = e1)
```

3

2

```
z <- x + y
  return(z)
}

foo <- function() {
  y <<- 3
  z <- bar()
  return(z)
}

foo()
```

4

Global assignment is highly discouraged. Use of a wrapper function or explicitly calling variables from another local environment is greatly preferred.

Section 62.5: Explicit Assignment of Environments and Variables

Environments in R can be explicitly call and named. Variables can be explicitly assigned and call to or from those environments.

A commonly created environment is one which encloses package :base or a subenvironment within package :base.

```
e1 <- new.env(parent = baseenv())
e2 <- new.env(parent = e1)
```

Variables can be explicitly assigned and call to or from those environments.

```
assign("a", 3, envir = e1)
get("a", envir = e1)
get("a", envir = e2)
```

3

3

Since e2 inherits from e1, a is 3 in both e1 and e2. However, assigning a within e2 does not change the value of a in e1.

```
assign("a", 2, envir = e2)
get("a", envir = e2)
get("a", envir = e1)
```

3

2

第63章：执行置换检验

第63.1节：一个相当通用的函数

我们将使用内置的tooth_growth dataset。我们关注的是，当豚鼠服用维生素C与橙汁时，牙齿生长是否存在统计学上的显著差异。

以下是完整示例：

```
teethVC = ToothGrowth[ToothGrowth$supp == 'VC',]  
teethOJ = ToothGrowth[ToothGrowth$supp == 'OJ',]  
  
permutationTest = function(vectorA, vectorB, testStat){  
  N = 10^5  
  fullSet = c(vectorA, vectorB)  
  lengthA = length(vectorA)  
  lengthB = length(vectorB)  
  trials <- replicate(N,  
    {index <- sample(lengthB + lengthA, size = lengthA, replace = FALSE)  
     testStat((fullSet[index]), fullSet[-index]) } )  
  trials  
}  
vec1 = teethVC$len;  
vec2 = teethOJ$len;  
subtractMeans = function(a, b){ return (mean(a) - mean(b))}  
result = permutationTest(vec1, vec2, subtractMeans)  
observedMeanDifference = subtractMeans(vec1, vec2)  
result = c(result, observedMeanDifference)  
hist(result)  
abline(v=observedMeanDifference, col = "blue")  
pValue = 2*mean(result <= (observedMeanDifference))  
pValue
```

读取CSV后，我们定义函数

```
permutationTest = function(vectorA, vectorB, testStat){  
  N = 10^5  
  fullSet = c(vectorA, vectorB)  
  lengthA = length(vectorA)  
  lengthB = length(vectorB)  
  trials <- replicate(N,  
    {index <- sample(lengthB + lengthA, size = lengthA, replace = FALSE)  
     testStat((fullSet[index]), fullSet[-index]) } )  
  trials  
}
```

此函数接受两个向量，并将它们的内容混合在一起，然后对混合后的向量执行函数 testStat 。
testStat 的结果被加到 trials 中， trials 是返回值。

它执行此操作 $N = 10^5$ 次。请注意，值 N 也可以作为函数的一个参数。

这给我们留下了一组新的数据， trials，即如果两个变量之间确实没有关系，可能得到的均值集合。

现在定义我们的检验统计量：

Chapter 63: Performing a Permutation Test

Section 63.1: A fairly general function

We will use the built in [tooth growth dataset](#). We are interested in whether there is a statistically significant difference in tooth growth when the guinea pigs are given vitamin C vs orange juice.

Here's the full example:

```
teethVC = ToothGrowth[ToothGrowth$supp == 'VC',]  
teethOJ = ToothGrowth[ToothGrowth$supp == 'OJ',]  
  
permutationTest = function(vectorA, vectorB, testStat){  
  N = 10^5  
  fullSet = c(vectorA, vectorB)  
  lengthA = length(vectorA)  
  lengthB = length(vectorB)  
  trials <- replicate(N,  
    {index <- sample(lengthB + lengthA, size = lengthA, replace = FALSE)  
     testStat((fullSet[index]), fullSet[-index]) } )  
  trials  
}  
vec1 = teethVC$len;  
vec2 = teethOJ$len;  
subtractMeans = function(a, b){ return (mean(a) - mean(b))}  
result = permutationTest(vec1, vec2, subtractMeans)  
observedMeanDifference = subtractMeans(vec1, vec2)  
result = c(result, observedMeanDifference)  
hist(result)  
abline(v=observedMeanDifference, col = "blue")  
pValue = 2*mean(result <= (observedMeanDifference))  
pValue
```

After we read in the CSV, we define the function

```
permutationTest = function(vectorA, vectorB, testStat){  
  N = 10^5  
  fullSet = c(vectorA, vectorB)  
  lengthA = length(vectorA)  
  lengthB = length(vectorB)  
  trials <- replicate(N,  
    {index <- sample(lengthB + lengthA, size = lengthA, replace = FALSE)  
     testStat((fullSet[index]), fullSet[-index]) } )  
  trials  
}
```

This function takes two vectors, and shuffles their contents together, then performs the function testStat on the shuffled vectors. The result of teststat is added to trials, which is the return value.

It does this $N = 10^5$ times. Note that the value N could very well have been a parameter to the function.

This leaves us with a new set of data, trials, the set of means that might result if there truly is no relationship between the two variables.

Now to define our test statistic:

```
subtractMeans = function(a, b){ return (mean(a) - mean(b))}
```

执行检验：

```
result = permutationTest(vec1, vec2, subtractMeans)
```

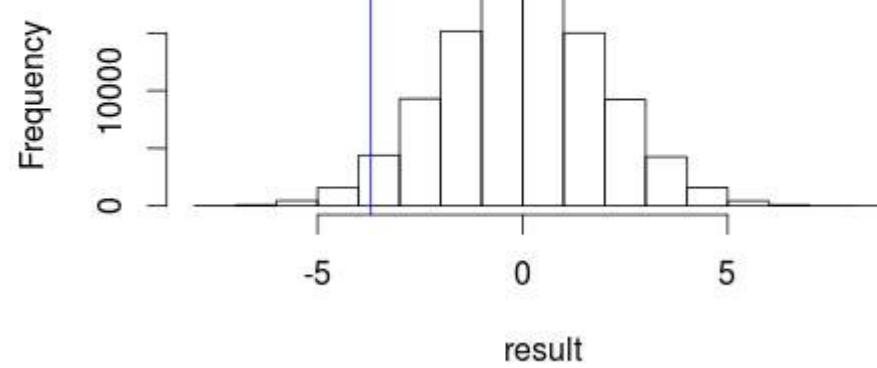
计算我们实际观察到的均值差：

```
observedMeanDifference = subtractMeans(vec1, vec2)
```

让我们看看观察值在检验统计量的直方图上的表现。

```
hist(result)  
abline(v=observedMeanDifference, col = "blue")
```

Histogram of result



看起来我们的观察结果很可能不是随机发生的...

我们想计算p值，即如果两个变量之间没有关系，原始观察结果出现的可能性。

```
pValue = 2*mean(result >= (observedMeanDifference))
```

让我们稍微分解一下：

```
result >= (observedMeanDifference)
```

将创建一个布尔向量，如：

```
FALSE TRUE FALSE FALSE TRUE FALSE ...
```

每当result的值大于或等于observedMean时，该向量对应位置为TRUE。

函数mean会将该向量中的TRUE视为1，FALSE视为0，并给出1的比例，即我们打乱后的向量均值差超过或等于观察值的次数比例。

最后，我们乘以2，因为我们的检验统计量的分布高度对称，而我们真正想知道的是哪些结果比我们观察到的结果“更极端”。

```
subtractMeans = function(a, b){ return (mean(a) - mean(b))}
```

Perform the test:

```
result = permutationTest(vec1, vec2, subtractMeans)
```

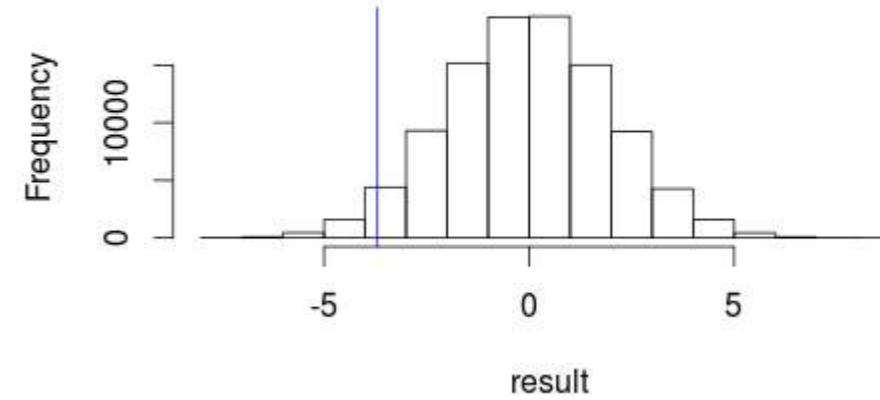
Calculate our actual observed mean difference:

```
observedMeanDifference = subtractMeans(vec1, vec2)
```

Let's see what our observation looks like on a histogram of our test statistic.

```
hist(result)  
abline(v=observedMeanDifference, col = "blue")
```

Histogram of result



It doesn't look like our observed result is very likely to occur by random chance...

We want to calculate the p-value, the likelihood of the original observed result if there is no relationship between the two variables.

```
pValue = 2*mean(result >= (observedMeanDifference))
```

Let's break that down a bit:

```
result >= (observedMeanDifference)
```

Will create a boolean vector, like:

```
FALSE TRUE FALSE FALSE TRUE FALSE ...
```

With TRUE every time the value of result is greater than or equal to the observedMean.

The function `mean` will interpret this vector as 1 for TRUE and 0 for FALSE, and give us the percentage of 1's in the mix, ie the number of times our shuffled vector mean difference surpassed or equalled what we observed.

Finally, we multiply by 2 because the distribution of our test statistic is highly symmetric, and we really want to know which results are "more extreme" than our observed result.

剩下的就是输出p值，结果是0.06093939。对这个值的解释是主观的，
但我会说看起来维生素C比橙汁更能促进牙齿生长。

All that's left is to output the p-value, which turns out to be 0.06093939. Interpretation of this value is subjective, but I would say that it looks like Vitamin C promotes tooth growth quite a lot more than Orange Juice does.

第64章：xgboost

第64.1节：使用xgboost进行交叉验证和调参

```
library(caret) # 用于dummyVars
library(RCurl) # 下载https数据
library(Metrics) # 计算误差
library(xgboost) # 模型

#####
# 从UCI机器学习库加载数据 (http://archive.ics.uci.edu/ml/datasets.html)
urlfile <- 'https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data'
x <- getURL(urlfile, ssl.verifypeer = FALSE)
adults <- read.csv(textConnection(x), header=F)

# adults <-read.csv('https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data',
# header=F)
names(adults)=c('age', 'workclass', 'fnlwgt', 'education', 'educationNum',
'maritalStatus', 'occupation', 'relationship', 'race',
'sex', 'capitalGain', 'capitalLoss', 'hoursWeek',
'nativeCountry', 'income')

# 清理数据
adults$income <- ifelse(adults$income==' <=50K',0,1)
# 对所有因子进行二值化
library(caret)
dmy <- dummyVars(~ ., data = adults)
adultsTrsf <- data.frame(predict(dmy, newdata = adults))
#####

# 我们试图预测收入超过50k的成年人
outcomeName <- c('income')
# 特征列表
predictors <- names(adultsTrsf)[!names(adultsTrsf) %in% outcomeName]

# 调整xgboost (极端梯度提升树) 库的参数
# https://github.com/tqchen/xgboost/wiki/Parameters
# max.depth - 树的最大深度
# nrounds - 最大迭代次数

# 只取前10%的数据！
trainPortion <- floor(nrow(adultsTrsf)*0.1)

trainSet <- adultsTrsf[ 1:floor(trainPortion/2),]
testSet <- adultsTrsf[(floor(trainPortion/2)+1):trainPortion,]

smallestError <- 100
for (depth in seq(1,10,1)) {
    for (rounds in seq(1,20,1)) {

        # 训练
        bst <- xgboost(data = as.matrix(trainSet[,predictors]),
                        label = trainSet[,outcomeName],
                        max.depth=depth, nround=rounds,
                        objective = "reg:linear", verbose=0)
        gc()

        # 预测
        predictions <- predict(bst, as.matrix(testSet[,predictors]), outputmargin=TRUE)
        err <- rmse(as.numeric(testSet[,outcomeName]), as.numeric(predictions))
```

Chapter 64: xgboost

Section 64.1: Cross Validation and Tuning with xgboost

```
library(caret) # for dummyVars
library(RCurl) # download https data
library(Metrics) # calculate errors
library(xgboost) # model

#####
# Load data from UCI Machine Learning Repository (http://archive.ics.uci.edu/ml/datasets.html)
urlfile <- 'https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data'
x <- getURL(urlfile, ssl.verifypeer = FALSE)
adults <- read.csv(textConnection(x), header=F)

# adults <-read.csv('https://archive.ics.uci.edu/ml/machine-learning-databases/adult/adult.data',
# header=F)
names(adults)=c('age', 'workclass', 'fnlwgt', 'education', 'educationNum',
'maritalStatus', 'occupation', 'relationship', 'race',
'sex', 'capitalGain', 'capitalLoss', 'hoursWeek',
'nativeCountry', 'income')

# clean up data
adults$income <- ifelse(adults$income==' <=50K', 0, 1)
# binarize all factors
library(caret)
dmy <- dummyVars(~ ., data = adults)
adultsTrsf <- data.frame(predict(dmy, newdata = adults))
#####

# what we're trying to predict adults that make more than 50k
outcomeName <- c('income')
# list of features
predictors <- names(adultsTrsf)[!names(adultsTrsf) %in% outcomeName]

# play around with settings of xgboost - eXtreme Gradient Boosting (Tree) library
# https://github.com/tqchen/xgboost/wiki/Parameters
# max.depth - maximum depth of the tree
# nrounds - the max number of iterations

# take first 10% of the data only!
trainPortion <- floor(nrow(adultsTrsf)*0.1)

trainSet <- adultsTrsf[ 1:floor(trainPortion/2),]
testSet <- adultsTrsf[(floor(trainPortion/2)+1):trainPortion,]

smallestError <- 100
for (depth in seq(1,10,1)) {
    for (rounds in seq(1,20,1)) {

        # train
        bst <- xgboost(data = as.matrix(trainSet[,predictors]),
                        label = trainSet[,outcomeName],
                        max.depth=depth, nround=rounds,
                        objective = "reg:linear", verbose=0)
        gc()

        # predict
        predictions <- predict(bst, as.matrix(testSet[,predictors]), outputmargin=TRUE)
        err <- rmse(as.numeric(testSet[,outcomeName]), as.numeric(predictions))
```

```

    if (err < smallestError) {
        smallestError = err
        print(paste(depth, rounds, err))
    }
}

cv <- 30
trainSet <- adultsTrsf[1:trainPortion,]
cvDivider <- floor(nrow(trainSet) / (cv+1))

smallestError <- 100
for (depth in seq(1,10,1)) {
    for (rounds in seq(1,20,1)) {
        totalError <- c()
        indexCount <- 1
        for (cv in seq(1:cv)) {
            # 分配数据块到测试集
            dataTestIndex <- c((cv * cvDivider):(cv * cvDivider + cvDivider))
            dataTest <- trainSet[dataTestIndex,]
            # 其余全部作为训练集
            dataTrain <- trainSet[-dataTestIndex,]

            bst <- xgboost(data = as.matrix(dataTrain[,predictors]),
                           label = dataTrain[,outcomeName],
                           max.depth=depth, nround=rounds,
                           objective = "reg:linear", verbose=0)
            gc()
            predictions <- predict(bst, as.matrix(dataTest[,predictors]),
                                   outputmargin=TRUE)

            err <- rmse(as.numeric(dataTest[,outcomeName]), as.numeric(predictions))
            totalError <- c(totalError, err)
        }
        if (mean(totalError) < smallestError) {
            smallestError = mean(totalError)
            print(paste(depth, rounds, smallestError))
        }
    }
}

#####
# 在完整数据集上测试两个模型

trainSet <- adultsTrsf[ 1:trainPortion,]

# 将其余部分分配为测试集
testSet <- adultsTrsf[(trainPortion+1):nrow(adultsTrsf),]

bst <- xgboost(data = as.matrix(trainSet[,predictors]),
               label = trainSet[,outcomeName],
               max.depth=4, nround=19, objective = "reg:linear", verbose=0)
pred <- predict(bst, as.matrix(testSet[,predictors]), outputmargin=TRUE)
rmse(as.numeric(testSet[,outcomeName]), as.numeric(pred))

bst <- xgboost(data = as.matrix(trainSet[,predictors]),
               label = trainSet[,outcomeName],
               max.depth=3, nround=20, objective = "reg:linear", verbose=0)
pred <- predict(bst, as.matrix(testSet[,predictors]), outputmargin=TRUE)
rmse(as.numeric(testSet[,outcomeName]), as.numeric(pred))

```

```

    if (err < smallestError) {
        smallestError = err
        print(paste(depth, rounds, err))
    }
}

cv <- 30
trainSet <- adultsTrsf[1:trainPortion,]
cvDivider <- floor(nrow(trainSet) / (cv+1))

smallestError <- 100
for (depth in seq(1,10,1)) {
    for (rounds in seq(1,20,1)) {
        totalError <- c()
        indexCount <- 1
        for (cv in seq(1:cv)) {
            # assign chunk to data test
            dataTestIndex <- c((cv * cvDivider):(cv * cvDivider + cvDivider))
            dataTest <- trainSet[dataTestIndex,]
            # everything else to train
            dataTrain <- trainSet[-dataTestIndex,]

            bst <- xgboost(data = as.matrix(dataTrain[,predictors]),
                           label = dataTrain[,outcomeName],
                           max.depth=depth, nround=rounds,
                           objective = "reg:linear", verbose=0)
            gc()
            predictions <- predict(bst, as.matrix(dataTest[,predictors]),
                                   outputmargin=TRUE)

            err <- rmse(as.numeric(dataTest[,outcomeName]), as.numeric(predictions))
            totalError <- c(totalError, err)
        }
        if (mean(totalError) < smallestError) {
            smallestError = mean(totalError)
            print(paste(depth, rounds, smallestError))
        }
    }
}

#####
# Test both models out on full data set

trainSet <- adultsTrsf[ 1:trainPortion,]

# assign everything else to test
testSet <- adultsTrsf[(trainPortion+1):nrow(adultsTrsf),]

bst <- xgboost(data = as.matrix(trainSet[,predictors]),
               label = trainSet[,outcomeName],
               max.depth=4, nround=19, objective = "reg:linear", verbose=0)
pred <- predict(bst, as.matrix(testSet[,predictors]), outputmargin=TRUE)
rmse(as.numeric(testSet[,outcomeName]), as.numeric(pred))

bst <- xgboost(data = as.matrix(trainSet[,predictors]),
               label = trainSet[,outcomeName],
               max.depth=3, nround=20, objective = "reg:linear", verbose=0)
pred <- predict(bst, as.matrix(testSet[,predictors]), outputmargin=TRUE)
rmse(as.numeric(testSet[,outcomeName]), as.numeric(pred))

```

第65.1节：按行操作

向量化R代码的关键是减少或消除“按行操作”或R函数的方法调度。

这意味着当遇到一个乍看之下需要“按行操作”的问题时，比如计算每行的均值，需要问自己：

- 我处理的数据集的类别是什么？
- 是否存在已编译的代码可以实现这一点，而无需重复评估R函数？
- 如果没有，我能否改为按列而不是按行进行这些操作？
- 最后，花大量时间开发复杂的向量化代码是否值得，还是直接运行一个简单的apply循环更好？换句话说，数据是否足够大/复杂，以至于R无法通过简单循环高效处理？

撇开内存预分配问题和循环中对象增长的问题，这个例子将重点介绍如何可能避免apply循环、方法调度或在循环中重新评估R函数。

计算每行均值的标准/简单方法是：

| apply(mtcars, 1, mean) | | | | | |
|------------------------|----------------|----------------|--------------|----------------|--|
| 马自达 RX4 | 马自达 RX4 Wagon | 达特桑 710 | 霍内特 4 驱动 | 霍内特 Sportabout | |
| Valiant | Duster 360 | | | | |
| 29.90727 | 29.98136 | 23.59818 | 38.73955 | 53.66455 | |
| 35.04909 | 59.72000 | | | | |
| 梅赛德斯 240D | 梅赛德斯 230 | 梅赛德斯 280 | 梅赛德斯 280C | 梅赛德斯 450SE | |
| 梅赛德斯 450SL | 梅赛德斯 450SLC | | | | |
| 24.63455 | 27.23364 | 31.86000 | 31.78727 | 46.43091 | |
| 46.50000 | 46.35000 | | | | |
| 凯迪拉克 Fleetwood | 林肯 Continental | 克莱斯勒 Imperial | 菲亚特 128 | 本田 Civic | |
| 丰田 Corolla | 丰田 Corona | | | | |
| 66.23273 | 66.05855 | 65.97227 | 19.44091 | 17.74227 | |
| 18.81409 | 24.88864 | | | | |
| 道奇 Challenger | AMC Javelin | 雪佛兰 Camaro Z28 | 庞蒂克 Firebird | 菲亚特 X1-9 | |
| 保时捷 914-2 | 莲花 Europa | | | | |
| 47.24091 | 46.00773 | 58.75273 | 57.37955 | 18.92864 | |
| 24.77909 | 24.88027 | | | | |
| 福特潘特拉L | 法拉利迪诺 | 玛莎拉蒂博拉 | 沃尔沃142E | | |
| 60.97182 | 34.50818 | 63.15545 | 26.26273 | | |

但是我们能做得更好吗？让我们看看这里发生了什么：

- 首先，我们将一个data.frame转换成了一个matrix。（注意这发生在apply函数内部。）这既效率低下且危险。一个matrix不能同时容纳多种列类型。因此，这种转换很可能导致信息丢失，有时还会产生误导性结果（比较apply(iris, 2, class)与str(iris)或与sapply(iris, class))）。

其次，我们对每一行重复执行了操作。也就是说，我们必须评估某个R函数 nrow(mtcars)次。在这个特定情况下，mean不是一个计算量大的函数，因此即使是大数据集，R也可能轻松处理，但如果我们需要按行计算标准差（这涉及一个计算量大的平方根操作）会怎样？这就引出了下一个问题：

Chapter 65: R code vectorization best practices

Section 65.1: By row operations

The key in vectorizing R code, is to reduce or eliminate "by row operations" or method dispatching of R functions.

That means that when approaching a problem that at first glance requires "by row operations", such as calculating the means of each row, one needs to ask themselves:

- What are the classes of the data sets I'm dealing with?
- Is there an existing compiled code that can achieve this without the need of repetitive evaluation of R functions?
- If not, can I do these operation by columns instead by row?
- Finally, is it worth spending a lot of time on developing complicated vectorized code instead of just running a simple apply loop? In other words, is the data big/sophisticated enough that R can't handle it efficiently using a simple loop?

Putting aside the memory pre-allocation issue and growing object in loops, we will focus in this example on how to possibly avoid apply loops, method dispatching or re-evaluating R functions within loops.

A standard/easy way of calculating mean by row would be:

| apply(mtcars, 1, mean) | | | | | |
|------------------------|---------------------|-------------------|------------------|-------------------|--|
| Mazda RX4 | Mazda RX4 Wag | Datsun 710 | Hornet 4 Drive | Hornet Sportabout | |
| Valiant | Duster 360 | 23.59818 | 38.73955 | 53.66455 | |
| 29.90727 | 29.98136 | | | | |
| 35.04909 | 59.72000 | | | | |
| Merc 240D | Merc 230 | Merc 280 | Merc 280C | Merc 450SE | |
| Merc 450SL | Merc 450SLC | | | | |
| 24.63455 | 27.23364 | 31.86000 | 31.78727 | 46.43091 | |
| 46.50000 | 46.35000 | | | | |
| Cadillac Fleetwood | Lincoln Continental | Chrysler Imperial | Fiat 128 | Honda Civic | |
| Toyota Corolla | Toyota Corona | | | | |
| 66.23273 | 66.05855 | 65.97227 | 19.44091 | 17.74227 | |
| 18.81409 | 24.88864 | | | | |
| Dodge Challenger | AMC Javelin | Camaro Z28 | Pontiac Firebird | Fiat X1-9 | |
| Porsche 914-2 | Lotus Europa | | | | |
| 47.24091 | 46.00773 | 58.75273 | 57.37955 | 18.92864 | |
| 24.77909 | 24.88027 | | | | |
| Ford Pantera L | Ferrari Dino | Maserati Bora | Volvo 142E | | |
| 60.97182 | 34.50818 | 63.15545 | 26.26273 | | |

But can we do better? Let's see what happened here:

- First, we converted a data.frame to a matrix. (Note that this happens within the apply function.) This is both inefficient and dangerous. A matrix can't hold several column types at a time. Hence, such conversion will probably lead to loss of information and sometimes to misleading results (compare apply(iris, 2, class) with str(iris) or with sapply(iris, class)).
- Second of all, we performed an operation repetitively, one time for each row. Meaning, we had to evaluate some R function nrow(mtcars) times. In this specific case, mean is not a computationally expensive function, hence R could likely easily handle it even for a big data set, but what would happen if we need to calculate the standard deviation by row (which involves an expensive square root operation)? Which brings us to the next point:

我们多次调用了R函数，但也许已经有这个操作的编译版本？

实际上我们可以简单地这样做：

rowMeans(mtcars)

| | 马自达 RX4 | 马自达 RX4 Wagon | 达特桑 710 | 霍内特 4 驱动 | 霍内特 Sportabout |
|------|------------|----------------|----------------|--------------|----------------|
| | Valiant | Duster 360 | 29.98136 | 23.59818 | 38.73955 |
| | 29.90727 | 35.04909 | 59.72000 | | 53.66455 |
| 梅赛德斯 | 240D | 梅赛德斯 230 | 梅赛德斯 280 | 梅赛德斯 280C | 梅赛德斯 450SE |
| | 梅赛德斯 450SL | 梅赛德斯 450SLC | 27.23364 | 31.86000 | 31.78727 |
| | 24.63455 | 46.50000 | 46.35000 | | 46.43091 |
| 凯迪拉克 | Fleetwood | 林肯 Continental | 克莱斯勒 Imperial | 菲亚特 128 | 本田 Civic |
| | 丰田 Corolla | 丰田 Corona | 66.05855 | 65.97227 | 19.44091 |
| | 66.23273 | 18.81409 | 24.88864 | | 17.74227 |
| 道奇 | Challenger | AMC Javelin | 雪佛兰 Camaro Z28 | 庞蒂克 Firebird | 菲亚特 X1-9 |
| 保时捷 | 914-2 | 莲花 Europa | 47.24091 | 58.75273 | 57.37955 |
| | 47.24091 | 24.77909 | 24.88027 | | 18.92864 |
| 福特 | 潘特拉L | 法拉利迪诺 | 玛莎拉蒂博拉 | 沃尔沃142E | 60.97182 |
| | 60.97182 | 34.50818 | 63.15545 | | 26.26273 |

这不涉及按行操作，因此不会重复评估 R 函数。然而，我们仍然将data.frame转换为matrix。虽然rowMeans具有错误处理机制，且不会在无法处理的数据集上运行，但它仍然存在效率成本。

rowMeans(鸢尾花数据集)

rowMeans(鸢尾花数据集) 错误：'x' 必须是 数值型

但是，我们还能做得更好吗？我们可以尝试不用矩阵转换和错误处理，而是用另一种方法，这将允许我们将mtcars作为向量使用（因为data.frame本质上是一个list，而list是一个vector）。

Reduce(`+`, mtcars)/ncol(mtcars)

```
[1] 29.90727 29.98136 23.59818 38.73955 53.66455 35.04909 59.72000 24.63455 27.23364 31.86000  
31.78727 46.43091 46.50000 46.35000 66.23273 66.05855  
[17] 65.97227 19.44091 17.74227 18.81409 24.88864 47.24091 46.00773 58.75273 57.37955 18.92864  
24.77909 24.88027 60.97182 34.50818 63.15545 26.26273
```

为了可能的速度提升，我们失去了列名和错误处理（包括NA处理）。

另一个例子是按组计算均值，使用基础R我们可以尝试

aggregate(. ~ cyl, mtcars, mean)

```
cyl      mpg     disp       hp     drat      wt     qsec      vs      am     gear     carb  
1 4 26.66364 105.1364 82.63636 4.070909 2.285727 19.13727 0.9090909 0.7272727 4.090909 1.545455  
2 6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714 0.5714286 0.4285714 3.857143 3.428571  
3 8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214 0.0000000 0.1428571 3.285714 3.500000
```

不过，我们基本上是在循环中评估一个R函数，但循环现在隐藏在一个内部的C函数中（无论是C循环还是R循环，这点影响不大）。

我们能避免它吗？R中有一个编译函数叫做rowsum，因此我们可以这样做：

```
rowsum(mtcars[-2], mtcars$cyl)/table(mtcars$cyl)  
mpg     disp       hp     drat      wt     qsec      vs      am     gear     carb  
4 26.66364 105.1364 82.63636 4.070909 2.285727 19.13727 0.9090909 0.7272727 4.090909 1.545455
```

3. We evaluated the R function many times, but maybe there already is a compiled version of this operation?

Indeed we could simply do:

| | Mazda RX4 | Mazda RX4 Wag | Datsun 710 | Hornet 4 Drive | Hornet Sportabout |
|------|------------|----------------|----------------|----------------|-------------------|
| | Valiant | Duster 360 | 29.98136 | 23.59818 | 38.73955 |
| | 29.90727 | 35.04909 | 59.72000 | | 53.66455 |
| 梅赛德斯 | 240D | 梅赛德斯 230 | 梅赛德斯 280 | 梅赛德斯 280C | 梅赛德斯 450SE |
| | 梅赛德斯 450SL | 梅赛德斯 450SLC | 27.23364 | 31.86000 | 31.78727 |
| | 24.63455 | 46.50000 | 46.35000 | | 46.43091 |
| 凯迪拉克 | Fleetwood | 林肯 Continental | 克莱斯勒 Imperial | 菲亚特 128 | 本田 Civic |
| | 丰田 Corolla | 丰田 Corona | 66.05855 | 65.97227 | 19.44091 |
| | 66.23273 | 18.81409 | 24.88864 | | 17.74227 |
| 道奇 | Challenger | AMC Javelin | 雪佛兰 Camaro Z28 | 庞蒂克 Firebird | 菲亚特 X1-9 |
| 保时捷 | 914-2 | 莲花 Europa | 47.24091 | 58.75273 | 57.37955 |
| | 47.24091 | 24.77909 | 24.88027 | | 18.92864 |
| 福特 | 潘特拉L | 法拉利迪诺 | 玛莎拉蒂博拉 | 沃尔沃142E | Volvo 142E |
| | 60.97182 | 34.50818 | 63.15545 | | 26.26273 |

This involves no by row operations and therefore no repetitive evaluation of R functions. **However**, we still converted a **data.frame** to a **matrix**. Though **rowMeans** has an error handling mechanism and it won't run on a data set that it can't handle, it's still has an efficiency cost.

rowMeans(iris)

```
Error in rowMeans(iris) : 'x' must be numeric
```

But still, can we do better? We could try instead of a matrix conversion with error handling, a different method that will allow us to use **mtcars** as a vector (because a **data.frame** is essentially a **list** and a **list** is a **vector**).

Reduce(`+`, mtcars)/ncol(mtcars)

```
[1] 29.90727 29.98136 23.59818 38.73955 53.66455 35.04909 59.72000 24.63455 27.23364 31.86000  
31.78727 46.43091 46.50000 46.35000 66.23273 66.05855  
[17] 65.97227 19.44091 17.74227 18.81409 24.88864 47.24091 46.00773 58.75273 57.37955 18.92864  
24.77909 24.88027 60.97182 34.50818 63.15545 26.26273
```

Now for possible speed gain, we lost column names and error handling (including NA handling).

Another example would be calculating mean by group, using base R we could try

aggregate(. ~ cyl, mtcars, mean)

```
cyl      mpg     disp       hp     drat      wt     qsec      vs      am     gear     carb  
1 4 26.66364 105.1364 82.63636 4.070909 2.285727 19.13727 0.9090909 0.7272727 4.090909 1.545455  
2 6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714 0.5714286 0.4285714 3.857143 3.428571  
3 8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214 0.0000000 0.1428571 3.285714 3.500000
```

Still, we are basically evaluating an R function in a loop, but the loop is now hidden in an internal C function (it matters little whether it is a C or an R loop).

Could we avoid it? Well there is a compiled function in R called **rowsum**, hence we could do:

rowsum(mtcars[-2], mtcars\$cyl)/table(mtcars\$cyl)

```
mpg     disp       hp     drat      wt     qsec      vs      am     gear     carb  
4 26.66364 105.1364 82.63636 4.070909 2.285727 19.13727 0.9090909 0.7272727 4.090909 1.545455
```

```
6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714 0.5714286 0.4285714 3.857143 3.428571  
8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214 0.0000000 0.1428571 3.285714 3.500000
```

虽然我们也必须先转换成矩阵。

此时我们可能会质疑当前的数据结构是否最合适。使用`data.frame`是最佳实践吗？还是应该切换到`matrix`数据结构以提高效率？

随着我们开始每次评估昂贵的函数，按行操作将变得越来越昂贵（即使是在矩阵中）。让我们以按行计算方差为例进行考虑。

假设我们有一个矩阵m：

```
set.seed(100)  
m <- matrix(sample(1e2), 10)  
  
m  
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]  
[1,] 8 33 39 86 71 100 81 68 89 84  
[2,] 12 16 57 80 32 82 69 11 41 92  
[3,] 62 91 53 13 42 31 60 70 98 79  
[4,] 66 94 29 67 45 59 20 96 64 1  
[5,] 36 63 76 6 10 48 85 75 99 2  
[6,] 18 4 27 19 44 56 37 95 26 40  
[7,] 3 24 21 25 52 51 83 28 49 17  
[8,] 46 5 22 43 47 74 35 97 77 65  
[9,] 55 54 78 34 50 90 30 61 14 58  
[10,] 88 73 38 15 9 72 7 93 23 87
```

可以简单地执行：

```
apply(m, 1, var)  
[1] 871.6556 957.5111 699.2111 941.4333 1237.3333 641.8222 539.7889 759.4333 500.4889  
1255.6111
```

另一方面，也可以通过遵循方差的公式来完全向量化此操作

```
RowVar <- function(x) {  
  rowSums((x - rowMeans(x))^2)/(dim(x)[2] - 1)  
}  
RowVar(m)  
[1] 871.6556 957.5111 699.2111 941.4333 1237.3333 641.8222 539.7889 759.4333 500.4889  
1255.6111
```

```
6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714 0.5714286 0.4285714 3.857143 3.428571  
8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214 0.0000000 0.1428571 3.285714 3.500000
```

Though we had to convert to a matrix first too.

At this point we may question whether our current data structure is the most appropriate one. Is a `data.frame` is the best practice? Or should one just switch to a `matrix` data structure in order to gain efficiency?

By row operations will get more and more expensive (even in matrices) as we start to evaluate expensive functions each time. Lets us consider a variance calculation by row example.

Lets say we have a matrix m:

```
set.seed(100)  
m <- matrix(sample(1e2), 10)  
  
m  
[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]  
[1,] 8 33 39 86 71 100 81 68 89 84  
[2,] 12 16 57 80 32 82 69 11 41 92  
[3,] 62 91 53 13 42 31 60 70 98 79  
[4,] 66 94 29 67 45 59 20 96 64 1  
[5,] 36 63 76 6 10 48 85 75 99 2  
[6,] 18 4 27 19 44 56 37 95 26 40  
[7,] 3 24 21 25 52 51 83 28 49 17  
[8,] 46 5 22 43 47 74 35 97 77 65  
[9,] 55 54 78 34 50 90 30 61 14 58  
[10,] 88 73 38 15 9 72 7 93 23 87
```

One could simply do:

```
apply(m, 1, var)  
[1] 871.6556 957.5111 699.2111 941.4333 1237.3333 641.8222 539.7889 759.4333 500.4889  
1255.6111
```

On the other hand, one could also completely vectorize this operation by following the formula of variance

```
RowVar <- function(x) {  
  rowSums((x - rowMeans(x))^2)/(dim(x)[2] - 1)  
}  
RowVar(m)  
[1] 871.6556 957.5111 699.2111 941.4333 1237.3333 641.8222 539.7889 759.4333 500.4889  
1255.6111
```

第66章：缺失值

当我们不知道变量的取值时，我们说该值缺失，用NA表示。

第66.1节：检查缺失数据

anyNA报告是否存在任何缺失值；而is.na则逐元素报告缺失值：

```
vec <- c(1, 2, 3, NA, 5)

anyNA(vec)
# [1] TRUE
is.na(vec)
# [1] FALSE FALSE FALSE TRUE FALSE
```

is.na 返回一个逻辑向量，在算术运算中被强制转换为整数值 (FALSE=0, TRUE=1)。

我们可以用它来找出有多少缺失值：

```
sum(is.na(vec))
# [1] 1
```

扩展这种方法，我们可以对数据框使用colSums和is.na来统计每列的缺失值数量：

```
colSums(is.na(airquality))
#   Ozone Solar.R   Wind   Temp Month Day
#     37      7      0      0      0     0
```

naniar包（目前在github上，但未在CRAN发布）提供了更多用于探索缺失值的工具。

第66.2节：读取和写入包含NA值的数据

使用read.*函数读取表格数据集时，R会自动查找类似于“NA”的缺失值。然而，缺失值并不总是用NA表示。有时点(.)、连字符(-)或字符值（例如：empty）表示该值为NA。可以使用read.*函数的na.strings参数告诉R哪些符号/字符需要被视为NA值：

```
read.csv("name_of_csv_file.csv", na.strings = "-")
```

也可以指定多个符号需要被读取为NA：

```
read.csv('missing.csv', na.strings = c('.','-'))
```

同样，可以使用write.csv的na参数用自定义字符串来表示NA。其他用于读取和写入表格的工具也有类似的选项。

第66.3节：使用不同类别的NA

符号NA表示一个logical类型的缺失值：

```
class(NA)
#[1] "logical"
```

这很方便，因为它可以很容易地被强制转换为其他原子向量类型，因此通常是你唯一需要的NA

Chapter 66: Missing values

When we don't know the value a variable takes, we say its value is missing, indicated by NA.

Section 66.1: Examining missing data

anyNA reports whether any missing values are present; while **is.na** reports missing values elementwise:

```
vec <- c(1, 2, 3, NA, 5)

anyNA(vec)
# [1] TRUE
is.na(vec)
# [1] FALSE FALSE FALSE TRUE FALSE
```

is.na returns a logical vector that is coerced to integer values under arithmetic operations (with FALSE=0, TRUE=1).

We can use this to find out how many missing values there are:

```
sum(is.na(vec))
# [1] 1
```

Extending this approach, we can use **colSums** and **is.na** on a data frame to count NAs per column:

```
colSums(is.na(airquality))
#   Ozone Solar.R   Wind   Temp Month Day
#     37      7      0      0      0     0
```

The [naniar package](#) (currently on github but not CRAN) offers further tools for exploring missing values.

Section 66.2: Reading and writing data with NA values

When reading tabular datasets with the **read.*** functions, R automatically looks for missing values that look like "NA". However, missing values are not always represented by NA. Sometimes a dot (.) , a hyphen(-) or a character-value (e.g.: empty) indicates that a value is NA. The **na.strings** parameter of the **read.*** function can be used to tell R which symbols/characters need to be treated as NA values:

```
read.csv("name_of_csv_file.csv", na.strings = "-")
```

It is also possible to indicate that more than one symbol needs to be read as NA:

```
read.csv('missing.csv', na.strings = c('.','-'))
```

Similarly, NAs can be written with customized strings using the **na** argument to **write.csv**. Other tools for reading and writing tables have similar options.

Section 66.3: Using NAs of different classes

The symbol NA is for a **logical** missing value:

```
class(NA)
#[1] "logical"
```

This is convenient, since it can easily be coerced to other atomic vector types, and is therefore usually the only NA

你将需要：

```
x <- c(1, NA, 1)
class(x[2])
#[1] "numeric"
```

如果您确实需要另一种类型的单个NA值，请使用NA_character_、NA_integer_、NA_real_或NA_complex_。对于复杂的缺失值，通常使用NA_integer_进行子集操作；例如，要获取缺失值的日期：

```
class(Sys.Date()[NA_integer_])
# [1] "Date"
```

第66.4节：TRUE/FALSE和/或NA

NA 是一种逻辑类型，且逻辑运算符与 NA 运算时，如果结果不确定将返回 NA。如下，NA OR TRUE 计算结果为 TRUE，因为至少一方为 TRUE，然而 NA OR FALSE 返回 NA，因为我们不知道 NA 是 TRUE 还是 FALSE

```
NA | TRUE
# [1] TRUE
# TRUE | TRUE 是 TRUE, FALSE | TRUE 也是 TRUE。
```

```
NA | FALSE
# [1] NA
# TRUE | FALSE 是 TRUE, 但 FALSE | FALSE 是 FALSE。
```

```
NA & TRUE
# [1] NA
# TRUE 和 TRUE 是 TRUE, 但 FALSE 和 TRUE 是 FALSE。
```

```
NA & FALSE
# [1] FALSE
# TRUE 和 FALSE 是 FALSE, FALSE 和 FALSE 也是 FALSE。
```

如果你想基于包含NA的某些列对数据集进行子集选择，这些属性会很有用。

```
df <- data.frame(v1=0:9,
                   v2=c(rep(1:2, each=4), NA, NA),
                   v3=c(NA, letters[2:10]))
```

```
df[df$v2 == 1 & !is.na(df$v2), ]
#   v1 v2   v3
#1  0  1 <NA>
#2  1  1    b
#3  2  1    c
#4  3  1    d
```

```
df[df$v2 == 1, ]
  v1 v2   v3
#1  0  1 <NA>
#2  1  1    b
#3  2  1    c
#4  3  1    d
#NA  NA NA <NA>
#NA.1 NA NA <NA>
```

you will need:

```
x <- c(1, NA, 1)
class(x[2])
#[1] "numeric"
```

If you do need a single NA value of another type, use NA_character_, NA_integer_, NA_real_ or NA_complex_. For missing values of fancy classes, subsetting with NA_integer_ usually works; for example, to get a missing-value Date:

```
class(Sys.Date()[NA_integer_])
# [1] "Date"
```

Section 66.4: TRUE/FALSE and/or NA

NA is a logical type and a logical operator with an NA will return NA if the outcome is ambiguous. Below, NA OR TRUE evaluates to TRUE because at least one side evaluates to TRUE, however NA OR FALSE returns NA because we do not know whether NA would have been TRUE or FALSE

```
NA | TRUE
# [1] TRUE
# TRUE | TRUE is TRUE and FALSE | TRUE is also TRUE.
```

```
NA | FALSE
# [1] NA
# TRUE | FALSE is TRUE but FALSE | FALSE is FALSE.
```

```
NA & TRUE
# [1] NA
# TRUE & TRUE is TRUE but FALSE & TRUE is FALSE.
```

```
NA & FALSE
# [1] FALSE
# TRUE & FALSE is FALSE and FALSE & FALSE is also FALSE.
```

These properties are helpful if you want to subset a data set based on some columns that contain NA.

```
df <- data.frame(v1=0:9,
                   v2=c(rep(1:2, each=4), NA, NA),
                   v3=c(NA, letters[2:10]))
```

```
df[df$v2 == 1 & !is.na(df$v2), ]
#   v1 v2   v3
#1  0  1 <NA>
#2  1  1    b
#3  2  1    c
#4  3  1    d
```

```
df[df$v2 == 1, ]
  v1 v2   v3
#1  0  1 <NA>
#2  1  1    b
#3  2  1    c
#4  3  1    d
#NA  NA NA <NA>
#NA.1 NA NA <NA>
```

第67章：层级线性模型

第67.1节：基本模型拟合

抱歉：由于我不知道哪个渠道可以讨论或提供改进请求的反馈，我就把我的问题放在这里。如果有更合适的地方，请随时指出！@DataTx 说这是“完全不清楚、不完整，或者格式严重有问题”。既然我没看到有什么大的格式问题（:-）），如果能稍微多给点关于这里期望如何改进清晰度或完整性的指导，以及为什么这里的内容无法挽救，那会很有帮助。

在 R 中拟合分层（也称“混合”或“多层”）线性模型的主要包是 `nlme`（较旧）和 `lme4`（较新）。这些包在许多细节上有所不同，但通常会得到非常相似的拟合模型。

```
library(nlme)
library(lme4)
m1.nlme <- lme(Reaction~Days, random=~Days|Subject, data=sleepstudy, method="REML")
m1.lme4 <- lmer(Reaction~Days+(Days|Subject), data=sleepstudy, REML=TRUE)
all.equal(fixef(m1.nlme), fixef(m1.lme4))
## [1] TRUE
```

需要考虑的差异：

- 公式语法略有不同
- `nlme` 仍然有较好的文档支持（例如 Pinheiro 和 Bates 2000 年的《S-PLUS 中的混合效应模型》；但也请参见 Bates 等人 2015 年《统计软件杂志》/vignette("lmer", package="lme4") 关于 `lme4` 的介绍）
- `lme4` 速度更快，且更容易拟合交叉随机效应
- `nlme` 开箱即用提供线性混合模型的 p 值，`lme4` 需要额外的包支持，例如 `lmerTest` 或 `afex`
- `nlme` 允许对异方差性或残差相关性（空间/时间/系统发育）进行建模

非官方的 [GLMM FAQ](#) 提供了更多信息，尽管它主要关注于 广义 线性混合模型 (GLMMs)。

Chapter 67: Hierarchical Linear Modeling

Section 67.1: basic model fitting

apologies: since I don't know of a channel for discussing/providing feedback on requests for improvement, I'm going to put my question here. Please feel free to point out a better place for this! @DataTx states that this is "completely unclear, incomplete, or has severe formatting problems". Since I don't see any big formatting problems (:-)), a little bit more guidance about what's expected here for improving clarity or completeness, and why what's here is unsalvageable, would be useful.

The primary packages for fitting hierarchical (alternatively "mixed" or "multilevel") linear models in R are `nlme` (older) and `lme4` (newer). These packages differ in many minor ways but should generally result in very similar fitted models.

```
library(nlme)
library(lme4)
m1.nlme <- lme(Reaction~Days, random=~Days|Subject, data=sleepstudy, method="REML")
m1.lme4 <- lmer(Reaction~Days+(Days|Subject), data=sleepstudy, REML=TRUE)
all.equal(fixef(m1.nlme), fixef(m1.lme4))
## [1] TRUE
```

Differences to consider:

- formula syntax is slightly different
- `nlme` is (still) somewhat better documented (e.g. Pinheiro and Bates 2000 *Mixed-effects models in S-PLUS*; however, see Bates et al. 2015 *Journal of Statistical Software*/vignette("lmer", package="lme4") for `lme4`)
- `lme4` is faster and allows easier fitting of crossed random effects
- `nlme` provides p-values for linear mixed models out of the box, `lme4` requires add-on packages such as `lmerTest` or `afex`
- `nlme` allows modeling of heteroscedasticity or residual correlations (in space/time/phylogeny)

The unofficial [GLMM FAQ](#) provides more information, although it is focused on *generalized* linear mixed models (GLMMs).

第68章：*apply函数族 (函数式)

第68.1节：使用内置函数式

内置函数式：lapply()、sapply() 和 mapply()

R自带内置函数式，其中最著名的可能是apply函数族。以下是一些最常用apply函数的描述：

- lapply() = 以列表作为参数，并对列表应用指定的函数。
- sapply() = 与 lapply() 相同，但尝试将输出简化为向量或矩阵。
 - vapply() = sapply() 的变体，必须指定输出对象的类型。
- mapply() = 类似于 lapply()，但可以将多个向量作为输入传递给指定函数。可以简化为 sapply()。
 - Map() 是 mapply() 的别名，参数 SIMPLIFY = FALSE。

lapply()

lapply() 可以用于两种不同的迭代：

- `lapply(variable, FUN)`
- `lapply(seq_along(variable), FUN)`

```
# 两种计算 x 平均值的方法
set.seed(1)
df <- data.frame(x = rnorm(25), y = rnorm(25))
lapply(df, mean)
lapply(seq_along(df), function(x) mean(df[[x]]))
```

sapply()

sapply() 会尝试将输出解析为向量或矩阵。

```
# 两个示例展示 sapply() 的不同输出
sapply(letters, print) ## 生成一个向量
x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))
sapply(x, quantile) ## 生成一个矩阵
```

mapply()

mapply() 的工作方式与 lapply() 类似，但它可以接受多个向量作为输入（因此 m 代表多变量）。

```
mapply(sum, 1:5, 10:6, 3) # 3 会被 mapply "回收"
```

第68.2节：合并多个 `data.frames` (`lapply`、`mapply`)

在本练习中，我们将生成四个自助法线性回归模型，并将这些模型的汇总合并到一个数据框中。

Chapter 68: *apply family of functions (functionals)

Section 68.1: Using built-in functionals

Built-in functionals: lapply(), sapply(), and mapply()

R comes with built-in functionals, of which perhaps the most well-known are the apply family of functions. Here is a description of some of the most common apply functions:

- `lapply()` = takes a list as an argument and applies the specified function to the list.
- `sapply()` = the same as `lapply()` but attempts to simplify the output to a vector or a matrix.
 - `vapply()` = a variant of `sapply()` in which the output object's type must be specified.
- `mapply()` = like `lapply()` but can pass multiple vectors as input to the specified function. Can be simplified like `sapply()`.
 - `Map()` is an alias to `mapply()` with `SIMPLIFY = FALSE`.

lapply()

`lapply()` can be used with two different iterations:

- `lapply(variable, FUN)`
- `lapply(seq_along(variable), FUN)`

```
# Two ways of finding the mean of x
set.seed(1)
df <- data.frame(x = rnorm(25), y = rnorm(25))
lapply(df, mean)
lapply(seq_along(df), function(x) mean(df[[x]]))
```

sapply()

`sapply()` will attempt to resolve its output to either a vector or a matrix.

```
# Two examples to show the different outputs of sapply()
sapply(letters, print) ## produces a vector
x <- list(a = 1:10, beta = exp(-3:3), logic = c(TRUE,FALSE,FALSE,TRUE))
sapply(x, quantile) ## produces a matrix
```

mapply()

`mapply()` works much like `lapply()` except it can take multiple vectors as input (hence the m for multivariate).

```
mapply(sum, 1:5, 10:6, 3) # 3 will be "recycled" by mapply
```

Section 68.2: Combining multiple `data.frames` (`lapply`、`mapply`)

In this exercise, we will generate four bootstrap linear regression models and combine the summaries of these models into a single data frame.

```
library(broom)
```

```
#* 创建自助法数据集
BootData <- lapply(1:4,
  function(i) mtcars[sample(1:nrow(mtcars),
    size = nrow(mtcars),
    replace = TRUE), ])

#* 拟合模型
模型 <- lapply(BootData,
  function(BD) lm(mpg ~ qsec + wt + factor(am),
    data = BD))

#* 将输出整理成 data.frame
整理后 <- lapply(模型,
  tidy)

#* 给整理列表中的每个元素命名
整理后 <- setNames(整理后, paste0("Boot", seq_along(整理后)))
```

此时，我们可以采取两种方法将名称插入到 data.frame 中。

```
#* 使用 `lapply` 将元素名称插入摘要中
#* 需要将 names 属性传递给 `lapply` 并在应用函数中引用 `整理后`。
描述_lapply <-
lapply(names(整理后),
  function(nm) cbind(nm, 整理后[[nm]]))

合并_lapply <- do.call("rbind", 描述_lapply)

#* 使用 `mapply` 将元素名称插入摘要中
#* 允许我们将名称和元素作为单独参数传递。
Described_mapply <-
mapply(
  function(nm, dframe) cbind(nm, dframe),
  names(Tidied),
  Tidied,
  SIMPLIFY = FALSE)

Combined_mapply <- do.call("rbind", Described_mapply)
```

如果你喜欢magrittr风格的管道操作，你可以在一个链式操作中完成整个任务（尽管如果你需要任何中间对象，比如模型对象本身，这样做可能不太明智）：

```
library(magrittr)
library(broom)
Combined <- lapply(1:4,
  function(i) mtcars[sample(1:nrow(mtcars),
    size = nrow(mtcars),
    replace = TRUE), ]) %>%
  lapply(function(BD) lm(mpg ~ qsec + wt + factor(am), data = BD)) %>%
  lapply(tidy) %>%
  setNames(paste0("Boot", seq_along(.))) %>%
  mapply(function(nm, dframe) cbind(nm, dframe),
    nm = names(.),
    dframe = .,
    SIMPLIFY = FALSE) %>%
  do.call("rbind", .)
```

```
library(broom)
```

```
#* Create the bootstrap data sets
BootData <- lapply(1:4,
  function(i) mtcars[sample(1:nrow(mtcars),
    size = nrow(mtcars),
    replace = TRUE), ])

#* Fit the models
Models <- lapply(BootData,
  function(BD) lm(mpg ~ qsec + wt + factor(am),
    data = BD))

#* Tidy the output into a data.frame
Tidied <- lapply(Models,
  tidy)

#* Give each element in the Tidied list a name
Tidied <- setNames(Tidied, paste0("Boot", seq_along(Tidied)))
```

At this point, we can take two approaches to inserting the names into the data.frame.

```
#* Insert the element name into the summary with `lapply`
#* Requires passing the names attribute to `lapply` and referencing `Tidied` within
#* the applied function.
Described_lapply <-
lapply(names(Tidied),
  function(nm) cbind(nm, Tidied[[nm]]))

Combined_lapply <- do.call("rbind", Described_lapply)

#* Insert the element name into the summary with `mapply`
#* Allows us to pass the names and the elements as separate arguments.
Described_mapply <-
mapply(
  function(nm, dframe) cbind(nm, dframe),
  names(Tidied),
  Tidied,
  SIMPLIFY = FALSE)

Combined_mapply <- do.call("rbind", Described_mapply)
```

If you're a fan of magrittr style pipes, you can accomplish the entire task in a single chain (though it may not be prudent to do so if you need any of the intermediary objects, such as the model objects themselves):

```
library(magrittr)
library(broom)
Combined <- lapply(1:4,
  function(i) mtcars[sample(1:nrow(mtcars),
    size = nrow(mtcars),
    replace = TRUE), ]) %>%
  lapply(function(BD) lm(mpg ~ qsec + wt + factor(am), data = BD)) %>%
  lapply(tidy) %>%
  setNames(paste0("Boot", seq_along(.))) %>%
  mapply(function(nm, dframe) cbind(nm, dframe),
    nm = names(.),
    dframe = .,
    SIMPLIFY = FALSE) %>%
  do.call("rbind", .)
```

第68.3节：批量文件加载

对于需要以类似流程操作的大量文件，并且文件名结构良好。

首先必须创建一个要访问的文件名向量，有多种方法：

- 使用paste0()手动创建向量

```
files <- paste0("file_", 1:100, ".rds")
```

- 使用list.files()配合正则表达式搜索文件类型，如果目录中有其他同类型文件，则需要了解正则表达式 (regex)。

```
files <- list.files("./", pattern = "\\.rds$", full.names = TRUE)
```

其中X是文件命名格式中部分内容的向量。

lapply会将每个响应作为列表的元素输出。

readRDS 特定于 .rds 文件，并且会根据处理过程的应用而变化。

```
my_file_list <- lapply(files, readRDS)
```

这不一定比 for 循环更快，但允许所有文件作为列表的元素，而无需显式赋值。

最后，我们经常需要一次加载多个包。这个技巧可以通过对所有想要导入的库应用 library() 来轻松实现：

```
lapply(c("jsonlite","stringr","igraph"),library,character.only=TRUE)
```

第68.4节：使用用户自定义函数式

用户自定义函数式

用户可以创建不同复杂度的自定义函数式。以下示例摘自 Hadley Wickham 的 [Functionals](#)：

```
randomise <- function(f) f(runif(1e3))

lapply2 <- function(x, f, ...) {
  out <- vector("list", length(x))
  for (i in seq_along(x)) {
    out[[i]] <- f(x[[i]], ...)
  }
  out
}
```

在第一种情况下，randomise 接受一个参数 f，并在一组均匀随机变量样本上调用它。为了演示等价性，我们在下面调用 set.seed：

```
set.seed(123)
randomise(mean)
```

Section 68.3: Bulk File Loading

for a large number of files which may need to be operated on in a similar process and with well structured file names.

firstly a vector of the file names to be accessed must be created, there are multiple options for this:

- Creating the vector manually with paste0()

```
files <- paste0("file_", 1:100, ".rds")
```

- Using `list.files()` with a regex search term for the file type, requires knowledge of regular expressions (regex) if other files of same type are in the directory.

```
files <- list.files("./", pattern = "\\.rds$", full.names = TRUE)
```

where X is a vector of part of the files naming format used.

`lapply` will output each response as element of a list.

`readRDS` is specific to .rds files and will change depending on the application of the process.

```
my_file_list <- lapply(files, readRDS)
```

This is not necessarily faster than a for loop from testing but allows all files to be an element of a list without assigning them explicitly.

Finally, we often need to load multiple packages at once. This trick can do it quite easily by applying `library()` to all libraries that we wish to import:

```
lapply(c("jsonlite", "stringr", "igraph"), library, character.only=TRUE)
```

Section 68.4: Using user-defined functionals

User-defined functionals

Users can create their own functionals to varying degrees of complexity. The following examples are from [Functionals](#) by Hadley Wickham:

```
randomise <- function(f) f(runif(1e3))

lapply2 <- function(x, f, ...) {
  out <- vector("list", length(x))
  for (i in seq_along(x)) {
    out[[i]] <- f(x[[i]], ...)
  }
  out
}
```

In the first case, randomise accepts a single argument f, and calls it on a sample of Uniform random variables. To demonstrate equivalence, we call `set.seed` below:

```
set.seed(123)
randomise(mean)
```

```
# [1] 0.4972778
```

```
set.seed(123)
mean(runif(1e3))
#[1] 0.4972778
```

```
set.seed(123)
randomise(max)
#[1] 0.9994045
```

```
set.seed(123)
max(runif(1e3))
#[1] 0.9994045
```

第二个例子是对 base::lapply 的重新实现，它使用函数式编程将操作 (f) 应用于列表 (x) 中的每个元素。参数 ... 允许用户向 f 传递额外参数，例如 mean 函数中的 na.rm 选项：

```
lapply(list(c(1, 3, 5), c(2, NA, 6)), mean)
# [[1]]
# [1] 3
#
# [[2]]
# [1] NA
```

```
lapply2(list(c(1, 3, 5), c(2, NA, 6)), mean)
# [[1]]
# [1] 3
#
# [[2]]
# [1] NA
```

```
lapply(list(c(1, 3, 5), c(2, NA, 6)), mean, na.rm = TRUE)
# [[1]]
# [1] 3
#
# [[2]]
# [1] 4
```

```
lapply2(list(c(1, 3, 5), c(2, NA, 6)), mean, na.rm = TRUE)
# [[1]]
# [1] 3
#
# [[2]]
# [1] 4
```

```
# [1] 0.4972778
```

```
set.seed(123)
mean(runif(1e3))
#[1] 0.4972778
```

```
set.seed(123)
randomise(max)
#[1] 0.9994045
```

```
set.seed(123)
max(runif(1e3))
#[1] 0.9994045
```

The second example is a re-implementation of base::**lapply**, which uses functionals to apply an operation (f) to each element in a list (x). The ... parameter allows the user to pass additional arguments to f, such as the na.rm option in the **mean** function:

```
lapply(list(c(1, 3, 5), c(2, NA, 6)), mean)
# [[1]]
# [1] 3
#
# [[2]]
# [1] NA
```

```
lapply2(list(c(1, 3, 5), c(2, NA, 6)), mean)
# [[1]]
# [1] 3
#
# [[2]]
# [1] NA
```

```
lapply(list(c(1, 3, 5), c(2, NA, 6)), mean, na.rm = TRUE)
# [[1]]
# [1] 3
#
# [[2]]
# [1] 4
```

```
lapply2(list(c(1, 3, 5), c(2, NA, 6)), mean, na.rm = TRUE)
# [[1]]
# [1] 3
#
# [[2]]
# [1] 4
```

第69章：文本挖掘

第69.1节：抓取数据以构建N-gram词云

以下示例利用 tm 文本挖掘包从网络抓取和挖掘文本数据，以构建带有符号阴影和排序的词云。

```
require(RWeka)
require(tau)
require(tm)
require(tm.plugin.webmining)
require(wordcloud)

# 抓取谷歌财经 -----
googlefinance <- WebCorpus(GoogleFinanceSource("NASDAQ:LFVN"))

# 抓取谷歌新闻 -----
lv.googlenews <- WebCorpus(GoogleNewsSource("LifeVantage"))
p.googlenews <- WebCorpus(GoogleNewsSource("Protandim"))
ts.googlenews <- WebCorpus(GoogleNewsSource("TrueScience"))

# 抓取纽约时报 -----
lv.nytimes <- WebCorpus(NYTimesSource(query = "LifeVantage", appid = nytimes_appid))
p.nytimes <- WebCorpus(NYTimesSource("Protandim", appid = nytimes_appid))
ts.nytimes <- WebCorpus(NYTimesSource("TrueScience", appid = nytimes_appid))

# 抓取路透社 -----
lv.reutersnews <- WebCorpus(ReutersNewsSource("LifeVantage"))
p.reutersnews <- WebCorpus(ReutersNewsSource("Protandim"))
ts.reutersnews <- WebCorpus(ReutersNewsSource("TrueScience"))

# 抓取雅虎财经 -----
lv.yahoofinance <- WebCorpus(YahooFinanceSource("LFVN"))

# 抓取雅虎新闻 -----
lv.yahoonews <- WebCorpus(YahooNewsSource("LifeVantage"))
p.yahoonews <- WebCorpus(YahooNewsSource("Protandim"))
ts.yahoonews <- WebCorpus(YahooNewsSource("TrueScience"))

# 抓取雅虎即时新闻 -----
lv.yahooinplay <- WebCorpus(YahooInplaySource("LifeVantage"))

# 结果文本挖掘 -----
corpus <- c(googlefinance, lv.googlenews, p.googlenews, ts.googlenews, lv.yahoofinance,
lv.yahoonews, p.yahoonews,
ts.yahoonews, lv.yahooinplay) #lv.nytimes, p.nytimes, ts.nytimes, lv.reutersnews, p.reutersnews,
ts.reutersnews

inspect(corpus)
wordlist <- c("lfvn", "lifevantage", "protandim", "truescience", "company", "fiscal", "nasdaq")

ds0.1g <- tm_map(corpus, content_transformer(tolower))
ds1.1g <- tm_map(ds0.1g, content_transformer(removeWords), wordlist)
ds1.1g <- tm_map(ds1.1g, content_transformer(removeWords), stopwords("english"))
ds2.1g <- tm_map(ds1.1g, stripWhitespace)
ds3.1g <- tm_map(ds2.1g, removePunctuation)
ds4.1g <- tm_map(ds3.1g, stemDocument)

tdm.1g <- TermDocumentMatrix(ds4.1g)
dtm.1g <- DocumentTermMatrix(ds4.1g)
```

Chapter 69: Text mining

Section 69.1: Scraping Data to build N-gram Word Clouds

The following example utilizes the tm text mining package to scrape and mine text data from the web to build word clouds with symbolic shading and ordering.

```
require(RWeka)
require(tau)
require(tm)
require(tm.plugin.webmining)
require(wordcloud)

# Scrape Google Finance -----
googlefinance <- WebCorpus(GoogleFinanceSource("NASDAQ:LFVN"))

# Scrape Google News -----
lv.googlenews <- WebCorpus(GoogleNewsSource("LifeVantage"))
p.googlenews <- WebCorpus(GoogleNewsSource("Protandim"))
ts.googlenews <- WebCorpus(GoogleNewsSource("TrueScience"))

# Scrape NYTimes -----
lv.nytimes <- WebCorpus(NYTimesSource(query = "LifeVantage", appid = nytimes_appid))
p.nytimes <- WebCorpus(NYTimesSource("Protandim", appid = nytimes_appid))
ts.nytimes <- WebCorpus(NYTimesSource("TrueScience", appid = nytimes_appid))

# Scrape Reuters -----
lv.reutersnews <- WebCorpus(ReutersNewsSource("LifeVantage"))
p.reutersnews <- WebCorpus(ReutersNewsSource("Protandim"))
ts.reutersnews <- WebCorpus(ReutersNewsSource("TrueScience"))

# Scrape Yahoo! Finance -----
lv.yahoofinance <- WebCorpus(YahooFinanceSource("LFVN"))

# Scrape Yahoo! News -----
lv.yahoonews <- WebCorpus(YahooNewsSource("LifeVantage"))
p.yahoonews <- WebCorpus(YahooNewsSource("Protandim"))
ts.yahoonews <- WebCorpus(YahooNewsSource("TrueScience"))

# Scrape Yahoo! Inplay -----
lv.yahooinplay <- WebCorpus(YahooInplaySource("LifeVantage"))

# Text Mining the Results -----
corpus <- c(googlefinance, lv.googlenews, p.googlenews, ts.googlenews, lv.yahoofinance,
lv.yahoonews, p.yahoonews,
ts.yahoonews, lv.yahooinplay) #lv.nytimes, p.nytimes, ts.nytimes, lv.reutersnews, p.reutersnews,
ts.reutersnews

inspect(corpus)
wordlist <- c("lfvn", "lifevantage", "protandim", "truescience", "company", "fiscal", "nasdaq")

ds0.1g <- tm_map(corpus, content_transformer(tolower))
ds1.1g <- tm_map(ds0.1g, content_transformer(removeWords), wordlist)
ds1.1g <- tm_map(ds1.1g, content_transformer(removeWords), stopwords("english"))
ds2.1g <- tm_map(ds1.1g, stripWhitespace)
ds3.1g <- tm_map(ds2.1g, removePunctuation)
ds4.1g <- tm_map(ds3.1g, stemDocument)

tdm.1g <- TermDocumentMatrix(ds4.1g)
dtm.1g <- DocumentTermMatrix(ds4.1g)
```

```

findFreqTerms(tdm.1g, 40)
findFreqTerms(tdm.1g, 60)
findFreqTerms(tdm.1g, 80)
findFreqTerms(tdm.1g, 100)

findAssocs(tdm.1g, "skin", .75)
findAssocs(tdm.1g, "scienc", .5)
findAssocs(tdm.1g, "product", .75)

tdm89.1g <- removeSparseTerms(tdm.1g, 0.89)
tdm9.1g <- removeSparseTerms(tdm.1g, 0.9)
tdm91.1g <- removeSparseTerms(tdm.1g, 0.91)
tdm92.1g <- removeSparseTerms(tdm.1g, 0.92)

tdm2.1g <- tdm92.1g

# 创建一个布尔矩阵 (计算包含术语的文档数量, 而非术语的原始数量)
tdm3.1g <- inspect(tdm2.1g)
tdm3.1g[tdm3.1g>=1] <- 1

# 转换为术语-术语邻接矩阵
termMatrix.1gram <- tdm3.1g %*% t(tdm3.1g)

# 查看编号为5到10的术语
termMatrix.1gram[5:10,5:10]
termMatrix.1gram[1:10,1:10]

# 创建词云以可视化文本数据 -----
notsparse <- tdm2.1g
m = as.matrix(notsparse)
v = sort(rowSums(m),decreasing=TRUE)
d = data.frame(word = names(v),freq=v)

# 创建词云
pal = brewer.pal(9,"BuPu")
wordcloud(words = d$word,
          freq = d$freq,
          scale = c(3,.8),
          random.order = F,
          colors = pal)

```

```

findFreqTerms(tdm.1g, 40)
findFreqTerms(tdm.1g, 60)
findFreqTerms(tdm.1g, 80)
findFreqTerms(tdm.1g, 100)

findAssocs(tdm.1g, "skin", .75)
findAssocs(tdm.1g, "scienc", .5)
findAssocs(tdm.1g, "product", .75)

tdm89.1g <- removeSparseTerms(tdm.1g, 0.89)
tdm9.1g <- removeSparseTerms(tdm.1g, 0.9)
tdm91.1g <- removeSparseTerms(tdm.1g, 0.91)
tdm92.1g <- removeSparseTerms(tdm.1g, 0.92)

tdm2.1g <- tdm92.1g

# Creates a Boolean matrix (counts # docs w/terms, not raw # terms)
tdm3.1g <- inspect(tdm2.1g)
tdm3.1g[tdm3.1g>=1] <- 1

# Transform into a term-term adjacency matrix
termMatrix.1gram <- tdm3.1g %*% t(tdm3.1g)

# inspect terms numbered 5 to 10
termMatrix.1gram[5:10,5:10]
termMatrix.1gram[1:10,1:10]

# Create a WordCloud to Visualize the Text Data -----
notsparse <- tdm2.1g
m = as.matrix(notsparse)
v = sort(rowSums(m),decreasing=TRUE)
d = data.frame(word = names(v),freq=v)

# Create the word cloud
pal = brewer.pal(9,"BuPu")
wordcloud(words = d$word,
          freq = d$freq,
          scale = c(3,.8),
          random.order = F,
          colors = pal)

```



注意使用了 `random.order` 和来自 `RColorBrewer` 的顺序调色板，这使得程序员可以通过为词语的顺序和颜色赋予意义，从而在词云中捕捉更多信息。

以上是 1-gram 的情况。

我们可以大幅提升到 n-gram 词云，在此过程中，我们将看到如何通过转换我们的 TDM，使几乎任何文本挖掘分析都足够灵活以处理 n-gram。

在 R 中处理 n-gram 的初始难点是，最流行的文本挖掘包 `tm` 本身不支持对二元组或 n 元组的分词。分词是将一个词、词的一部分或一组词（或符号）表示为称为 token 的单一数据元素的过程。

幸运的是，我们有一些技巧可以让我们继续使用带有升级分词器的 `tm`。实现这一点的方法不止一种。我们可以使用 `tau` 包中的 `textcnt()` 函数编写自己的简单分词器：

```
tokenize_ngrams <- function(x, n=3)
return(rownames(as.data.frame(unclass(textcnt(x,method="string",n=n)))))
```

或者我们可以在 `tm` 中调用 RWeka 的分词器：

```
# BigramTokenizer
BigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))
```

从这里开始，你可以像处理1-gram情况一样继续：

```
# 创建n-gram词云 -----
tdm.ng <- TermDocumentMatrix(ds5.1g, control = list(tokenize = BigramTokenizer))
dtm.ng <- DocumentTermMatrix(ds5.1g, control = list(tokenize = BigramTokenizer))

# 尝试在不同稀疏度水平下移除稀疏项
tdm89.ng <- removeSparseTerms(tdm.ng, 0.89)
tdm9.ng <- removeSparseTerms(tdm.ng, 0.9)
tdm91.ng <- removeSparseTerms(tdm.ng, 0.91)
tdm92.ng <- removeSparseTerms(tdm.ng, 0.92)
```



Note the use of `random.order` and a sequential pallet from `RColorBrewer`, which allows the programmer to capture more information in the cloud by assigning meaning to the order and coloring of terms.

Above is the 1-gram case.

We can make a major leap to n-gram word clouds and in doing so we'll see how to make almost any text-mining analysis flexible enough to handle n-grams by transforming our TDM.

The initial difficulty you run into with n-grams in R is that `tm`, the most popular package for text mining, does not inherently support tokenization of bi-grams or n-grams. Tokenization is the process of representing a word, part of a word, or group of words (or symbols) as a single data element called a token.

Fortunately, we have some hacks which allow us to continue using `tm` with an upgraded tokenizer. There's more than one way to achieve this. We can write our own simple tokenizer using the `textcnt()` function from `tau`:

```
tokenize_ngrams <- function(x, n=3)
return(rownames(as.data.frame(unclass(textcnt(x,method="string",n=n)))))
```

or we can invoke RWeka's tokenizer within `tm`:

```
# BigramTokenizer
BigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))
```

From this point you can proceed much as in the 1-gram case:

```
# Create an n-gram Word Cloud -----
tdm.ng <- TermDocumentMatrix(ds5.1g, control = list(tokenize = BigramTokenizer))
dtm.ng <- DocumentTermMatrix(ds5.1g, control = list(tokenize = BigramTokenizer))

# Try removing sparse terms at a few different levels
tdm89.ng <- removeSparseTerms(tdm.ng, 0.89)
tdm9.ng <- removeSparseTerms(tdm.ng, 0.9)
tdm91.ng <- removeSparseTerms(tdm.ng, 0.91)
tdm92.ng <- removeSparseTerms(tdm.ng, 0.92)
```

```
notsparse <- tdm91.ng
m = as.matrix(notsparse)
v = sort(rowSums(m),decreasing=TRUE)
d = data.frame(word = names(v), freq=v)

# 创建词云
pal = brewer.pal(9,"BuPu")
wordcloud(words = d$word,
freq = d$freq,
scale = c(3,.8),
random.order = F,
colors = pal)
```



上面的示例是经Hack-R数据科学博客许可转载的。更多评论可在原文中找到。

```

notsparse <- tdm91.ng
m = as.matrix(notsparse)
v = sort(rowSums(m),decreasing=TRUE)
d = data.frame(word = names(v), freq=v)

# Create the word cloud
pal = brewer.pal(9,"BuPu")
wordcloud(words = d$word,
          freq = d$freq,
          scale = c(3,.8),
          random.order = F,
          colors = pal)

```



The example above is [reproduced](#) with permission from Hack-R's data science blog. Additional commentary may be found in the original article.

第70章：方差分析 (ANOVA)

第70.1节：aov()的基本用法

方差分析 (aov) 用于确定两个或多个组的均值是否存在显著差异。假设响应变量彼此独立，且在每个组内服从正态分布，组内方差相等。

为了完成分析，数据必须是长格式（参见数据重塑主题）。aov()是对lm()函数的封装，使用Wilkinson-Rogers公式表示法y~f，其中y是响应（因变量），f是表示组别的因子（分类）变量。如果f是数值型而非因子变量，aov()将以ANOVA格式报告线性回归结果，这可能会让缺乏经验的用户感到意外。

aov()函数使用类型I（顺序）平方和。这种平方和类型按顺序检验所有（主效应和交互效应）。结果是第一个检验的效应也会被分配与模型中其他效应共享的方差。为了使此类模型的结果可靠，数据应当是平衡的（所有组大小相同）。

当类型I平方和的假设不成立时，类型II或类型III平方和可能适用。

类型II平方和在控制其他主效应的重叠方差后检验每个主效应，但假设主效应之间无交互作用。

最后，类型III平方和在控制其他所有主效应和所有交互作用后检验每个主效应。当存在交互作用时，类型III平方和是必需的。

类型II和类型III平方和在Anova()函数中实现。

以mtcars数据集为例。

```
mtCarsAnovaModel <- aov(wt ~ factor(cyl), data=mtcars)
```

查看ANOVA模型摘要：

```
summary(mtCarsAnovaModel)
```

还可以提取底层lm()模型的系数：

```
coefficients(mtCarsAnovaModel)
```

第70.2节：Anova()的基本用法

处理不平衡设计和/或非正交对比时，需要使用II型或III型平方和。car包中的Anova()函数实现了这些。II型平方和假设主效应之间无交互作用。如果假设存在交互作用，则适用III型平方和。

Anova()函数是对lm()函数的封装。

以mtcars数据集为例，演示在检验交互作用时II型和III型的区别。

```
> 方差分析(lm(wt ~ factor(cyl)*factor(am), data=mtcars), type = 2)  
方差分析表 (类型 II 检验)
```

Chapter 70: ANOVA

Section 70.1: Basic usage of aov()

Analysis of Variance (aov) is used to determine if the means of two or more groups differ significantly from each other. Responses are assumed to be independent of each other, Normally distributed (within each group), and the within-group variances are assumed equal.

In order to complete the analysis data must be in long format (see reshaping data topic). **aov()** is a wrapper around the **lm()** function, using Wilkinson-Rogers formula notation $y \sim f$ where y is the response (independent) variable and f is a factor (categorical) variable representing group membership. If f is numeric rather than a factor variable, **aov()** will report the results of a linear regression in ANOVA format, which may surprise inexperienced users.

The **aov()** function uses Type I (sequential) Sum of Squares. This type of Sum of Squares tests all of the (main and interaction) effects sequentially. The result is that the first effect tested is also assigned shared variance between it and other effects in the model. For the results from such a model to be reliable, data should be balanced (all groups are of the same size).

When the assumptions for Type I Sum of Squares do not hold, Type II or Type III Sum of Squares may be applicable. Type II Sum of Squares test each main effect after every other main effect, and thus controls for any overlapping variance. However, Type II Sum of Squares assumes no interaction between the main effects.

Lastly, Type III Sum of Squares tests each main effect after every other main effect and every interaction. This makes Type III Sum of Squares a necessity when an interaction is present.

Type II and Type III Sums of Squares are implemented in the Anova() function.

Using the **mtcars** data set as an example.

```
mtCarsAnovaModel <- aov(wt ~ factor(cyl), data=mtcars)
```

To view summary of ANOVA model:

```
summary(mtCarsAnovaModel)
```

One can also extract the coefficients of the underlying lm() model:

```
coefficients(mtCarsAnovaModel)
```

Section 70.2: Basic usage of Anova()

When dealing with an unbalanced design and/or non-orthogonal contrasts, Type II or Type III Sum of Squares are necessary. The Anova() function from the car package implements these. Type II Sum of Squares assumes no interaction between main effects. If interactions are assumed, Type III Sum of Squares is appropriate.

The Anova() function wraps around the lm() function.

Using the **mtcars** data sets as an example, demonstrating the difference between Type II and Type III when an interaction is tested.

```
> Anova(lm(wt ~ factor(cyl)*factor(am), data=mtcars), type = 2)  
Anova Table (Type II tests)
```

```
响应: wt
平方和 自由度 F 值 Pr(>F)
factor(cyl) 7.2278 2 11.5266 0.0002606 ***
factor(am) 3.2845 1 10.4758 0.0032895 **
factor(cyl):factor(am) 0.0668 2 0.1065 0.8993714
残差 8.1517 26
---
显著性代码: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '' 1
```

```
> 方差分析(lm(wt ~ factor(cyl)*factor(am), data=mtcars), 类型 = 3)
方差分析表 (类型 III 检验)
```

```
响应: wt
平方和 自由度 F 值 Pr(>F)
(截距) 25.8427 1 82.4254 1.524e-09 ***
factor(cyl) 4.0124 2 6.3988 0.005498 **
factor(am) 1.7389 1 5.5463 0.026346 *
factor(cyl):factor(am) 0.0668 2 0.1065 0.899371
残差 8.1517 26
---
显著性代码: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '' 1
```

```
Response: wt
Sum Sq Df F value Pr(>F)
factor(cyl) 7.2278 2 11.5266 0.0002606 ***
factor(am) 3.2845 1 10.4758 0.0032895 **
factor(cyl):factor(am) 0.0668 2 0.1065 0.8993714
Residuals 8.1517 26
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '' 1
```

```
> Anova(lm(wt ~ factor(cyl)*factor(am), data=mtcars), type = 3)
Anova Table (Type III tests)
```

```
Response: wt
Sum Sq Df F value Pr(>F)
(Intercept) 25.8427 1 82.4254 1.524e-09 ***
factor(cyl) 4.0124 2 6.3988 0.005498 **
factor(am) 1.7389 1 5.5463 0.026346 *
factor(cyl):factor(am) 0.0668 2 0.1065 0.899371
Residuals 8.1517 26
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '' 1
```

第71章：栅格和图像分析

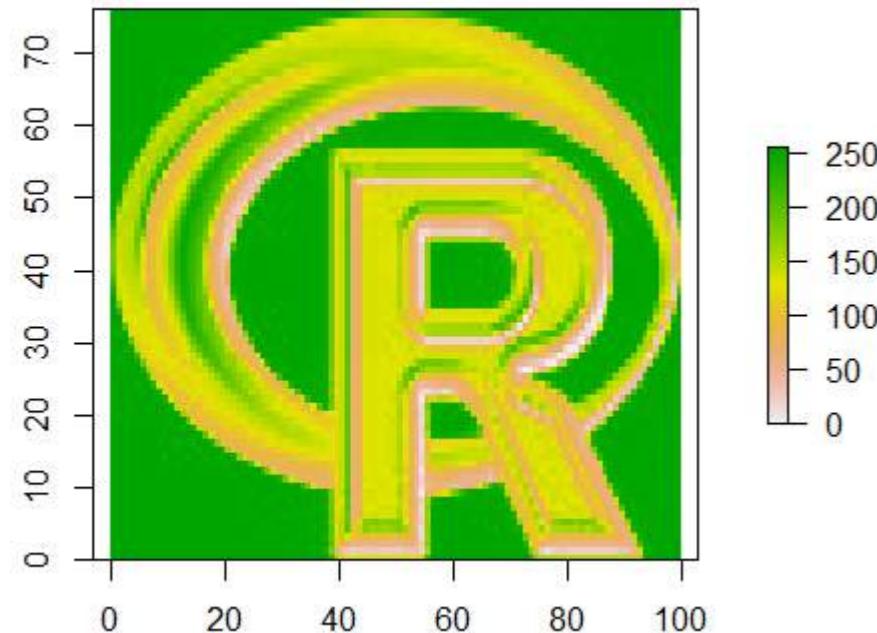
另见栅格图像的输入/输出

第71.1节：计算灰度共生矩阵纹理

灰度共生矩阵 (Haralick 等, 1973) 纹理是图像分析中一种强大的图像特征。glcm包提供了一个易于使用的函数，用于计算R中RasterLayer对象的纹理特征。

```
library(glcm)
library(raster)

r <- raster("C:/Program Files/R/R-3.2.3/doc/html/logo.jpg")
plot(r)
```



计算单方向的GLCM纹理

```
rglcm <- glcm(r,
                window = c(9,9),
                shift = c(1,1),
                statistics = c("mean", "variance", "homogeneity", "contrast",
                             "dissimilarity", "entropy", "second_moment"))
plot(rglcm)
```

Chapter 71: Raster and Image Analysis

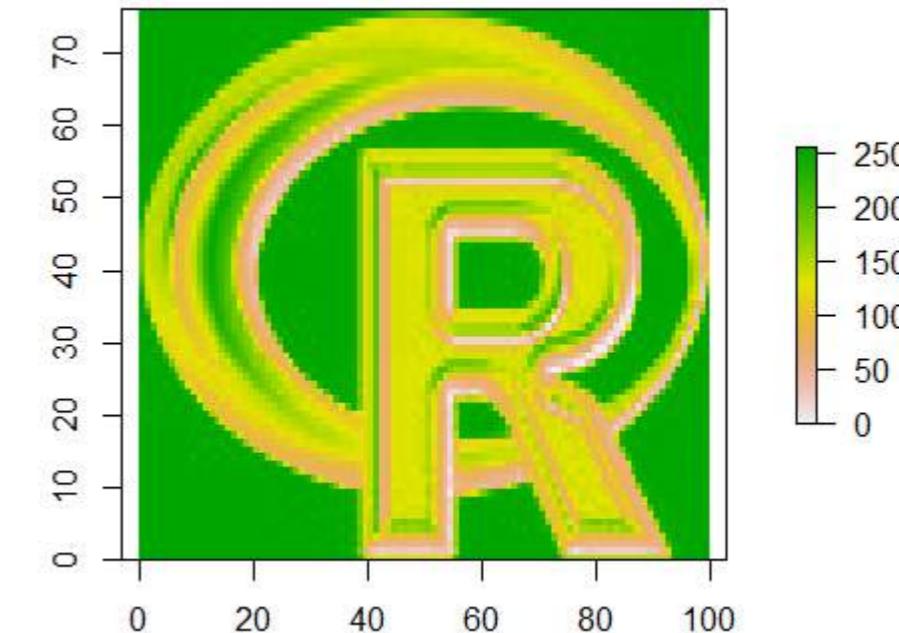
See also I/O for Raster Images

Section 71.1: Calculating GLCM Texture

[Gray Level Co-Occurrence Matrix](#) (Haralick et al. 1973) texture is a powerful image feature for image analysis. The `glcm` package provides a easy-to-use function to calculate such textrual features for `RasterLayer` objects in R.

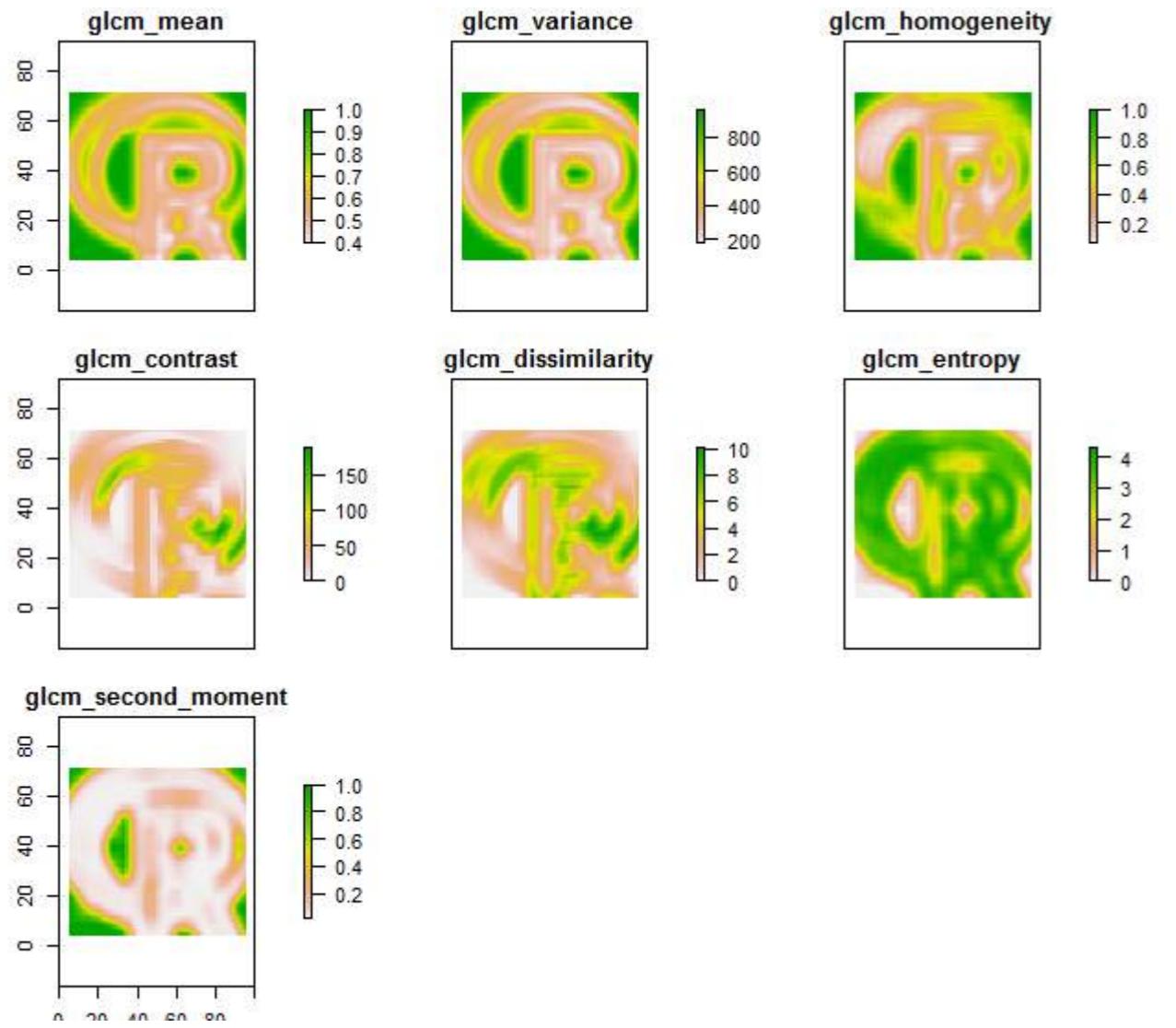
```
library(glcm)
library(raster)

r <- raster("C:/Program Files/R/R-3.2.3/doc/html/logo.jpg")
plot(r)
```



Calculating GLCM textures in one direction

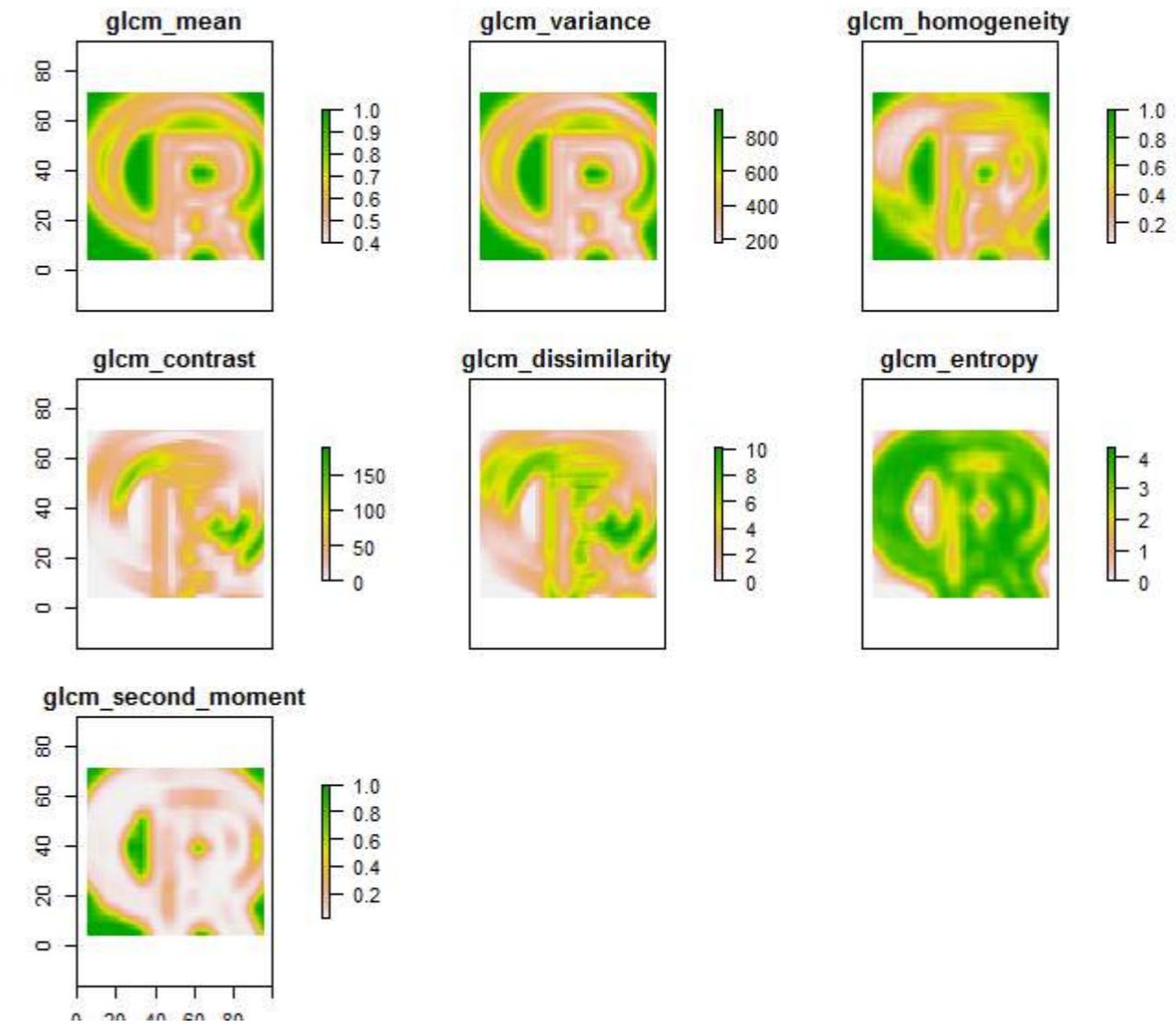
```
rglcm <- glcm(r,
                window = c(9,9),
                shift = c(1,1),
                statistics = c("mean", "variance", "homogeneity", "contrast",
                             "dissimilarity", "entropy", "second_moment"))
plot(rglcm)
```



计算旋转不变纹理特征

纹理特征也可以在所有4个方向（ 0° 、 45° 、 90° 和 135° ）计算，然后合并为一个旋转不变的纹理。关键参数是shift：

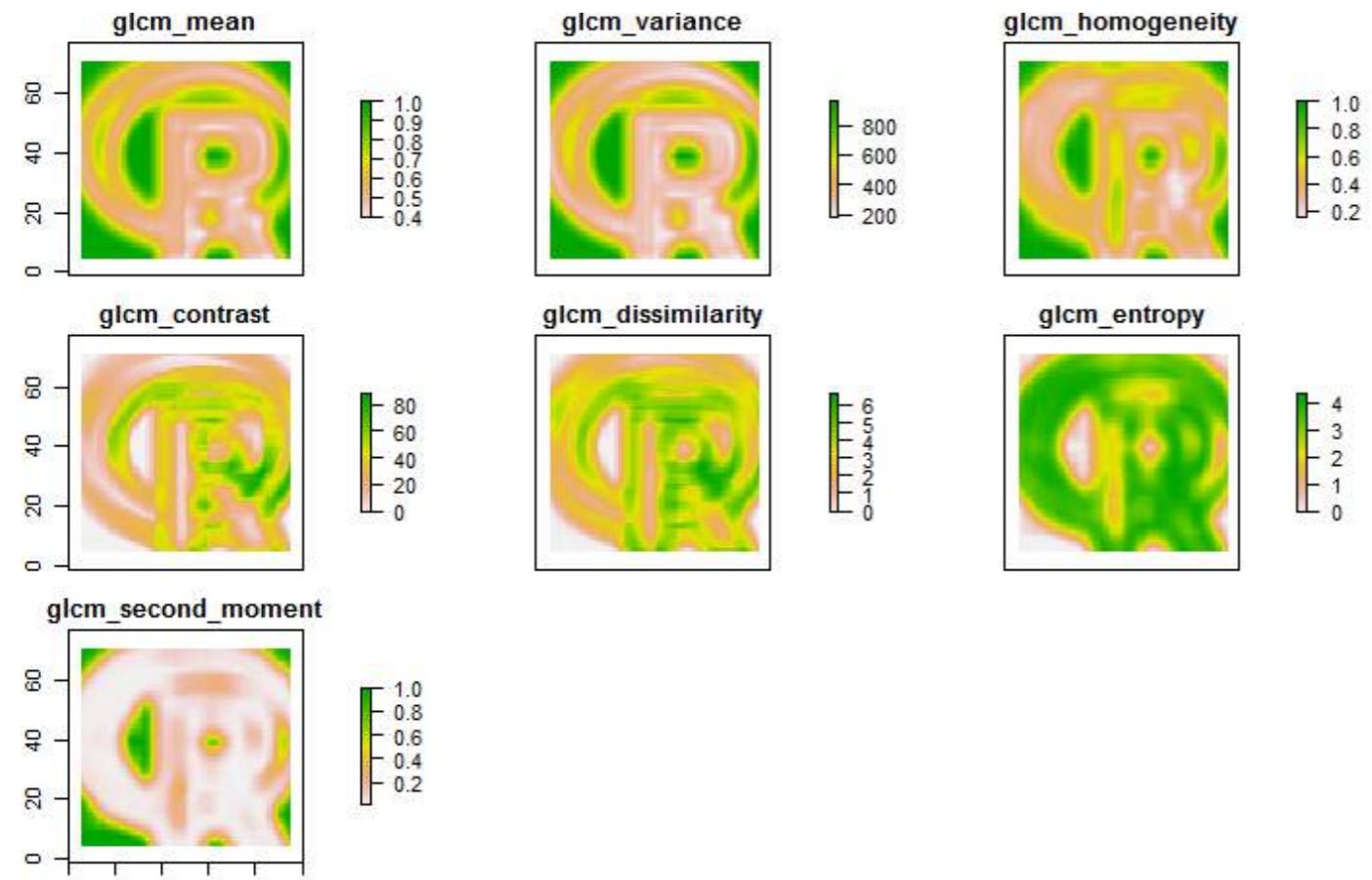
```
rglcm1 <- glcm(r,
                  window = c(9,9),
                  shift=list(c(0,1), c(1,1), c(1,0), c(1,-1)),
                  statistics = c("mean", "variance", "homogeneity", "contrast",
                                "dissimilarity", "entropy", "second_moment"))
)
plot(rglcm1)
```



Calculation rotation-invariant texture features

The textural features can also be calculated in all 4 directions (0° , 45° , 90° and 135°) and then combined to one rotation-invariant texture. The key for this is the shift parameter:

```
rglcm1 <- glcm(r,
                  window = c(9,9),
                  shift=list(c(0,1), c(1,1), c(1,0), c(1,-1)),
                  statistics = c("mean", "variance", "homogeneity", "contrast",
                                "dissimilarity", "entropy", "second_moment"))
)
plot(rglcm1)
```

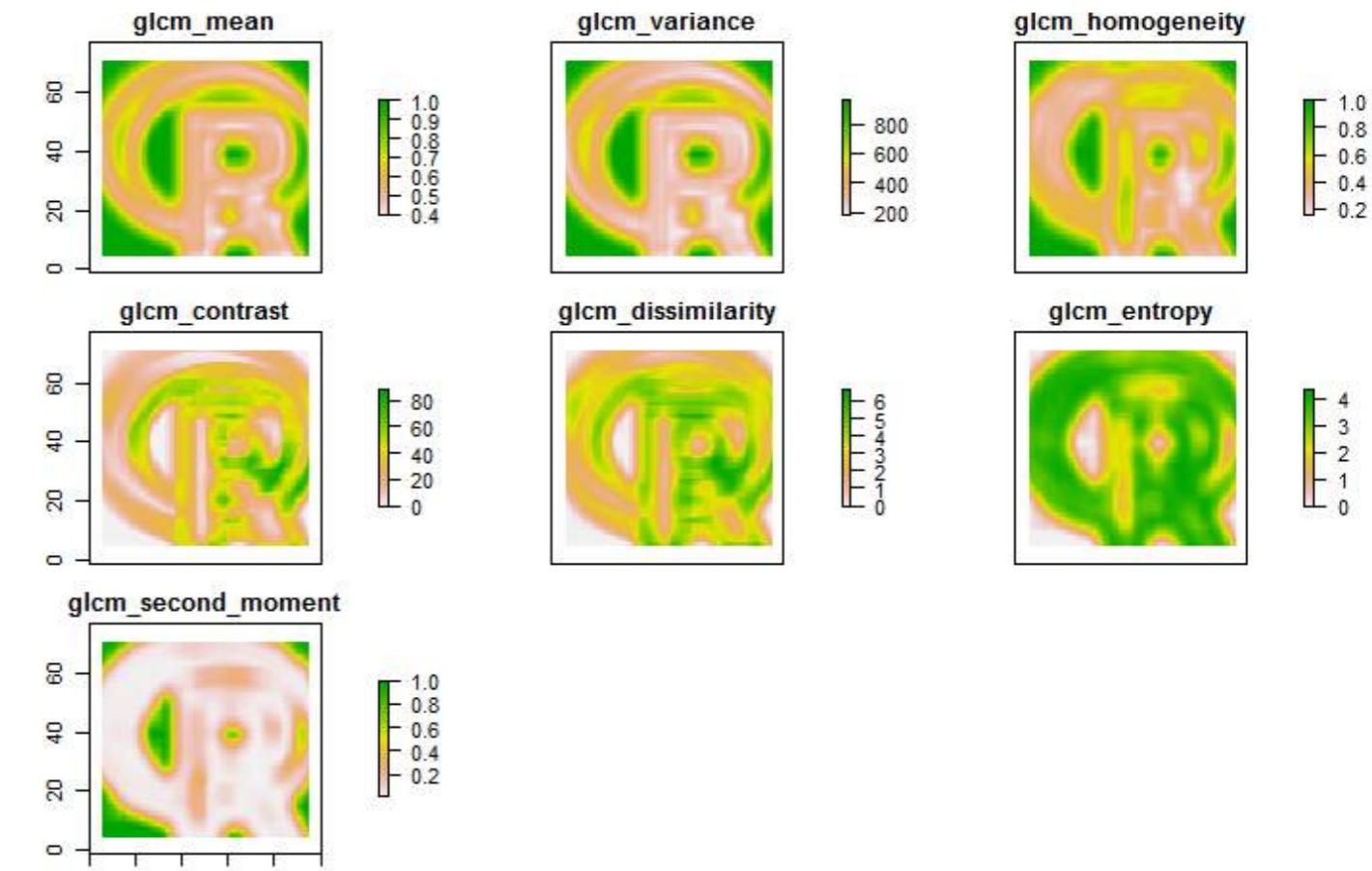
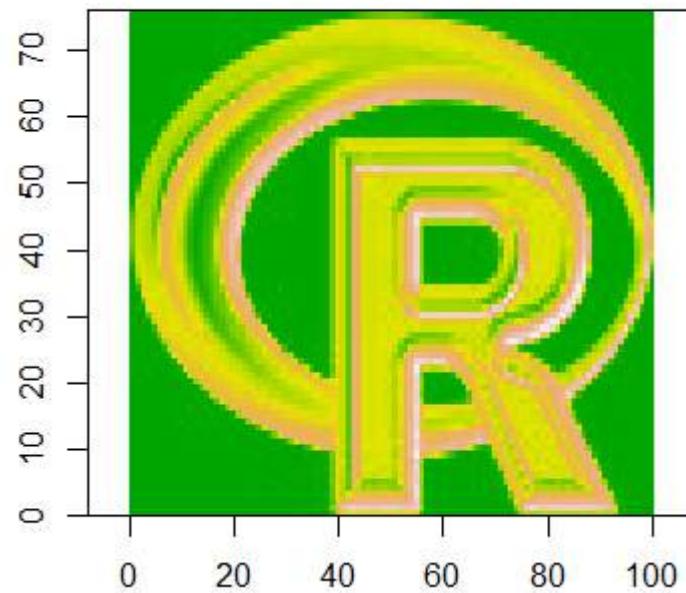


第71.2节：数学形态学

包mmand提供了用于计算n维数组数学形态学的函数。
通过一些变通方法，这些函数也可以用于栅格图像的计算。

```
library(raster)
library(mmand)

r <- raster("C:/Program Files/R/R-3.2.3/doc/html/logo.jpg")
plot(r)
```

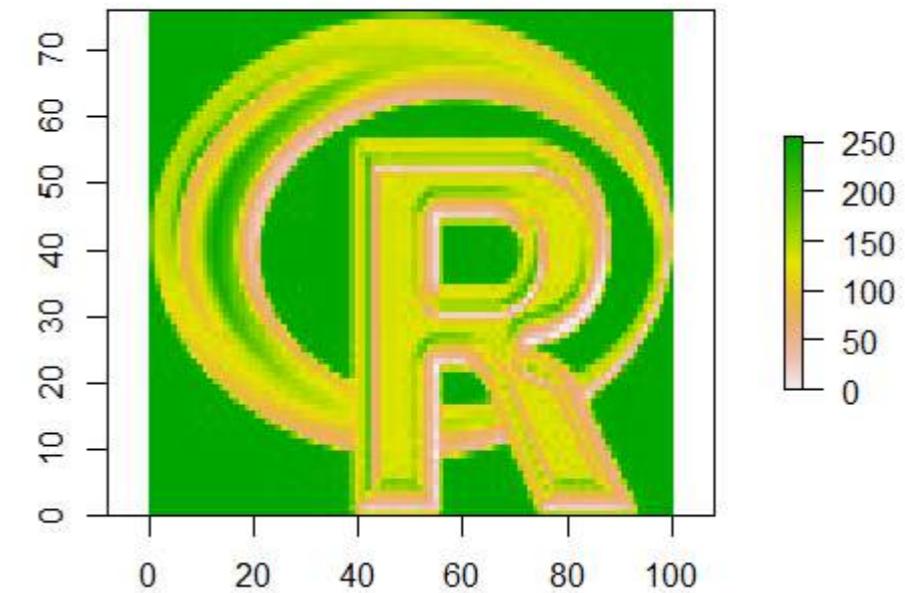


Section 71.2: Mathematical Morphologies

The package mmand provides functions for the calculation of Mathematical Morphologies for n-dimensional arrays.
With a little workaround, these can also be calculated for raster images.

```
library(raster)
library(mmand)

r <- raster("C:/Program Files/R/R-3.2.3/doc/html/logo.jpg")
plot(r)
```



首先，需要设置一个大小（例如9x9）和形状类型（例如disc、box或diamond）的核（移动窗口）

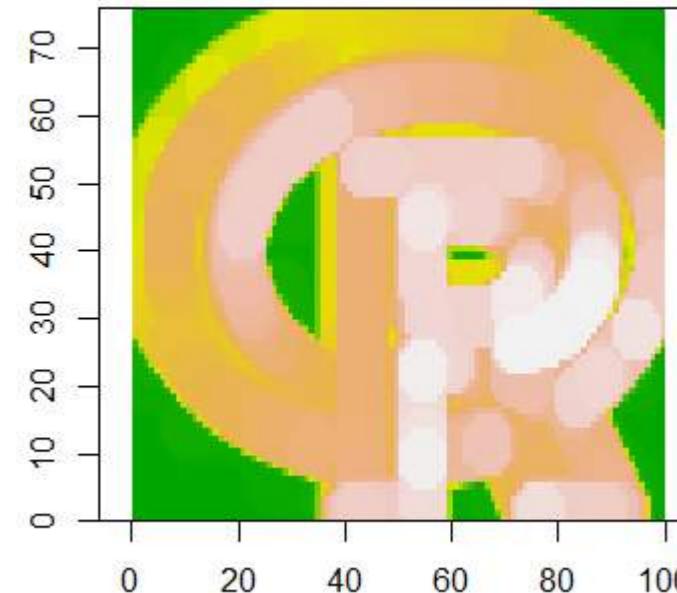
```
sk <- shapeKernel(c(9,9), type="disc")
```

随后，栅格图层必须转换为数组，该数组用作 erode()函数的输入。

```
rArr <- as.array(r, transpose = TRUE)  
rErode <- erode(rArr, sk)  
rErode <- setValues(r, as.vector(aperm(rErode)))
```

除了 erode()，形态学函数 dilate()、 opening()和 closing()也可以这样应用。

```
plot(rErode)
```



At first, a kernel (moving window) has to be set with a size (e.g. 9x9) and a shape type (e.g. disc, **box** or diamond)

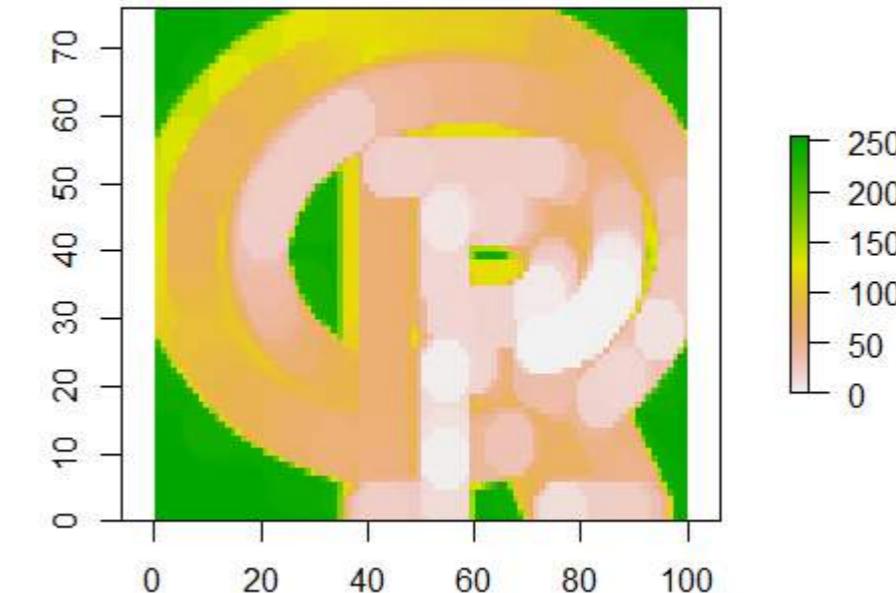
```
sk <- shapeKernel(c(9,9), type="disc")
```

Afterwards, the raster layer has to be converted into an array which is used as input for the erode() function.

```
rArr <- as.array(r, transpose = TRUE)  
rErode <- erode(rArr, sk)  
rErode <- setValues(r, as.vector(aperm(rErode)))
```

Besides erode(), also the morphological functions dilate(), opening() and closing() can be applied like this.

```
plot(rErode)
```



第72章：生存分析

第72.1节：使用

randomForestSRC的随机森林生存分析

正如随机森林算法可以应用于回归和分类任务一样，它也可以扩展到生存分析。

下面的示例中，拟合了一个生存模型，并使用CRAN上的包 randomForestSRC 进行预测、评分和性能分析。

```
require(randomForestSRC)

set.seed(130948) #其他种子给出类似的结果

y <- rnorm(1000, mean = x1, sd = .3)
data <- data.frame(x1 = x1, y = y)
head(data)
```

| | x1 | y |
|---|-----------|------------|
| 1 | 0.9604353 | 1.3549648 |
| 2 | 0.3771234 | 0.2961592 |
| 3 | 0.7844242 | 0.6942191 |
| 4 | 0.9860443 | 1.5348900 |
| 5 | 0.1942237 | 0.4629535 |
| 6 | 0.7442532 | -0.0672639 |

```
(modRFSRC <- rfsrc(y ~ x1, data = data, ntree=500, nodesize = 5))
```

```
样本量: 1000
树的数量: 500
最小终端节点大小: 5
终端节点平均数量: 208.258
每次分裂尝试的变量数: 1
变量总数: 1
分析方法: RF-R
模型类型: 回归
分裂规则: 均方误差(mse)
解释的方差百分比: 32.08
错误率: 0.11
```

```
x1new <- runif(10000)
ynew <- rnorm(10000, 均值 = x1new, 标准差 = .3)
newdata <- data.frame(x1 = x1new, y = ynew)

survival.results <- predict(modRFSRC, newdata = newdata)
survival.results
```

```
测试样本量 (predict) 数据: 10000
生长树的数量: 500
生长终端节点平均数量: 208.258
生长变量总数: 1
分析方法: RF-R
模型类型: 回归
解释的方差百分比: 34.97
测试集错误率: 0.11
```

Chapter 72: Survival analysis

Section 72.1: Random Forest Survival Analysis with randomForestSRC

Just as the [random forest](#) algorithm may be applied to regression and classification tasks, it can also be extended to survival analysis.

In the example below a survival model is fit and used for prediction, scoring, and performance analysis using the package [randomForestSRC](#) [from CRAN](#).

```
require(randomForestSRC)

set.seed(130948) #Other seeds give similar comparative results
x1 <- runif(1000)
y <- rnorm(1000, mean = x1, sd = .3)
data <- data.frame(x1 = x1, y = y)
head(data)
```

| | x1 | y |
|---|-----------|------------|
| 1 | 0.9604353 | 1.3549648 |
| 2 | 0.3771234 | 0.2961592 |
| 3 | 0.7844242 | 0.6942191 |
| 4 | 0.9860443 | 1.5348900 |
| 5 | 0.1942237 | 0.4629535 |
| 6 | 0.7442532 | -0.0672639 |

```
(modRFSRC <- rfsrc(y ~ x1, data = data, ntree=500, nodesize = 5))
```

```
Sample size: 1000
Number of trees: 500
Minimum terminal node size: 5
Average no. of terminal nodes: 208.258
No. of variables tried at each split: 1
Total no. of variables: 1
Analysis: RF-R
Family: regr
Splitting rule: mse
% variance explained: 32.08
Error rate: 0.11
```

```
x1new <- runif(10000)
ynew <- rnorm(10000, mean = x1new, sd = .3)
newdata <- data.frame(x1 = x1new, y = ynew)

survival.results <- predict(modRFSRC, newdata = newdata)
survival.results
```

```
Sample size of test (predict) data: 10000
Number of grow trees: 500
Average no. of grow terminal nodes: 208.258
Total no. of grow variables: 1
Analysis: RF-R
Family: regr
% variance explained: 34.97
Test set error rate: 0.11
```

第72.2节：介绍 - 使用survival包对参数生存模型进行基本拟合和绘图

survival 是R中最常用的生存分析包。使用内置的 lung 数据集，我们可以通过 survreg() 函数拟合回归模型，开始生存分析，创建曲线 survfit()，并通过调用该包的 predict 方法对新数据绘制预测生存曲线。

下面的示例中，我们绘制了两条预测曲线，并在两组新数据中改变 sex 变量，以可视化其影响：

```
require(survival)
s <- with(lung, Surv(time, status))

sWei <- survreg(s ~ as.factor(sex) + age + ph.ecog + wt.loss + ph.karno, dist='weibull', data=lung)

fitKM <- survfit(s ~ sex, data=lung)
plot(fitKM)

lines(predict(sWei, newdata = list(sex      = 1,
                                    age       = 1,
                                    ph.ecog   = 1,
                                    ph.karno  = 90,
                                    wt.loss   = 2),
              type = "quantile",
              p    = seq(.01, .99, by = .01)),
      seq(.99, .01, by     =-.01),
      col = "blue"))

lines(predict(sWei, newdata = list(sex      = 2,
                                    age       = 1,
                                    ph.ecog   = 1,
                                    ph.karno  = 90,
                                    wt.loss   = 2),
              type = "quantile",
              p    = seq(.01, .99, by = .01)),
      seq(.99, .01, by     =-.01),
      col = "red")
```

Section 72.2: Introduction - basic fitting and plotting of parametric survival models with the survival package

survival is the most commonly used package for survival analysis in R. Using the built-in lung dataset we can get started with Survival Analysis by fitting a regression model with the survreg() function, creating a curve with survfit(), and plotting predicted survival curves by calling the predict method for this package with new data.

In the example below we plot 2 predicted curves and vary sex between the 2 sets of new data, to visualize its effect:

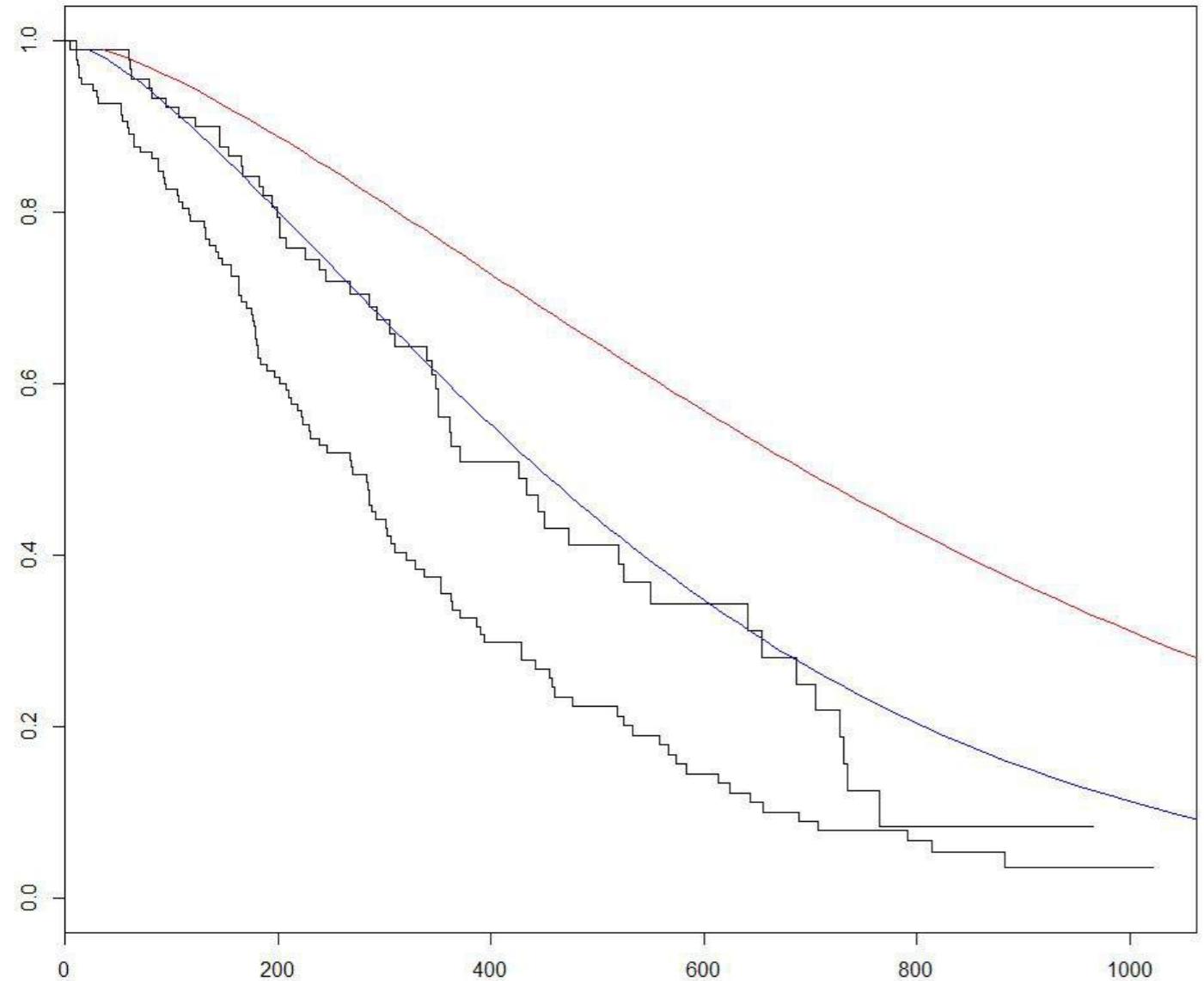
```
require(survival)
s <- with(lung, Surv(time, status))

sWei <- survreg(s ~ as.factor(sex) + age + ph.ecog + wt.loss + ph.karno, dist='weibull', data=lung)

fitKM <- survfit(s ~ sex, data=lung)
plot(fitKM)

lines(predict(sWei, newdata = list(sex      = 1,
                                    age       = 1,
                                    ph.ecog   = 1,
                                    ph.karno  = 90,
                                    wt.loss   = 2),
              type = "quantile",
              p    = seq(.01, .99, by = .01)),
      seq(.99, .01, by     =-.01),
      col = "blue"))

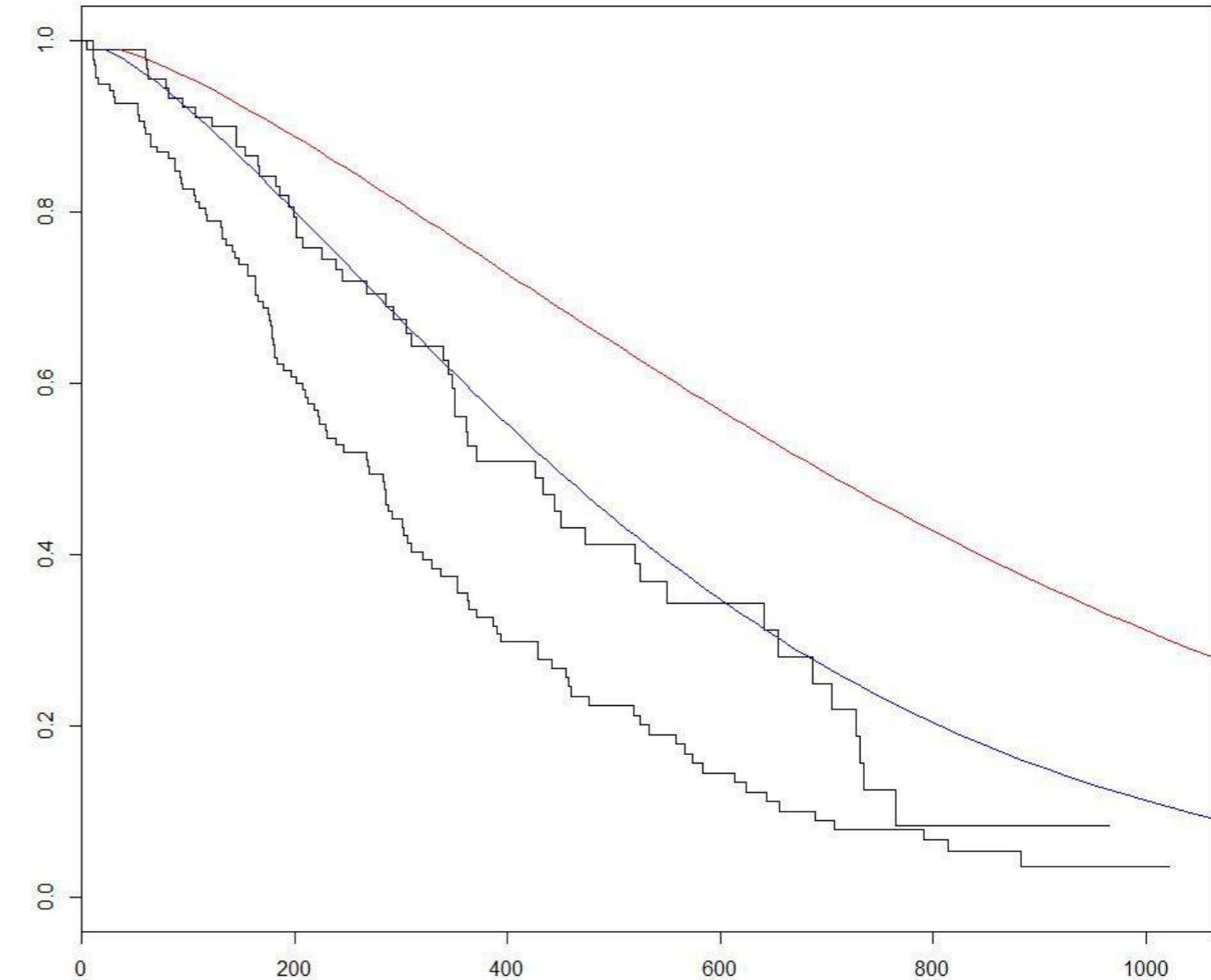
lines(predict(sWei, newdata = list(sex      = 2,
                                    age       = 1,
                                    ph.ecog   = 1,
                                    ph.karno  = 90,
                                    wt.loss   = 2),
              type = "quantile",
              p    = seq(.01, .99, by = .01)),
      seq(.99, .01, by     =-.01),
      col = "red")
```



第72.3节：使用survminer进行Kaplan-Meier生存曲线估计和风险集表

基础绘图

```
install.packages('survminer')
source("https://bioconductor.org/biocLite.R")
biocLite("RTCGA.clinical") # 示例数据
library(RTCGA.clinical)
survivalTCGA(BRCA.clinical, OV.clinical,
             extract.cols = "admin.disease_code") -> BRCAOV.survInfo
library(survival)
fit <- survfit(Surv(times, patient.vital_status) ~ admin.disease_code,
               data = BRCAOV.survInfo)
library(survminer)
ggsurvplot(fit, risk.table = TRUE)
```

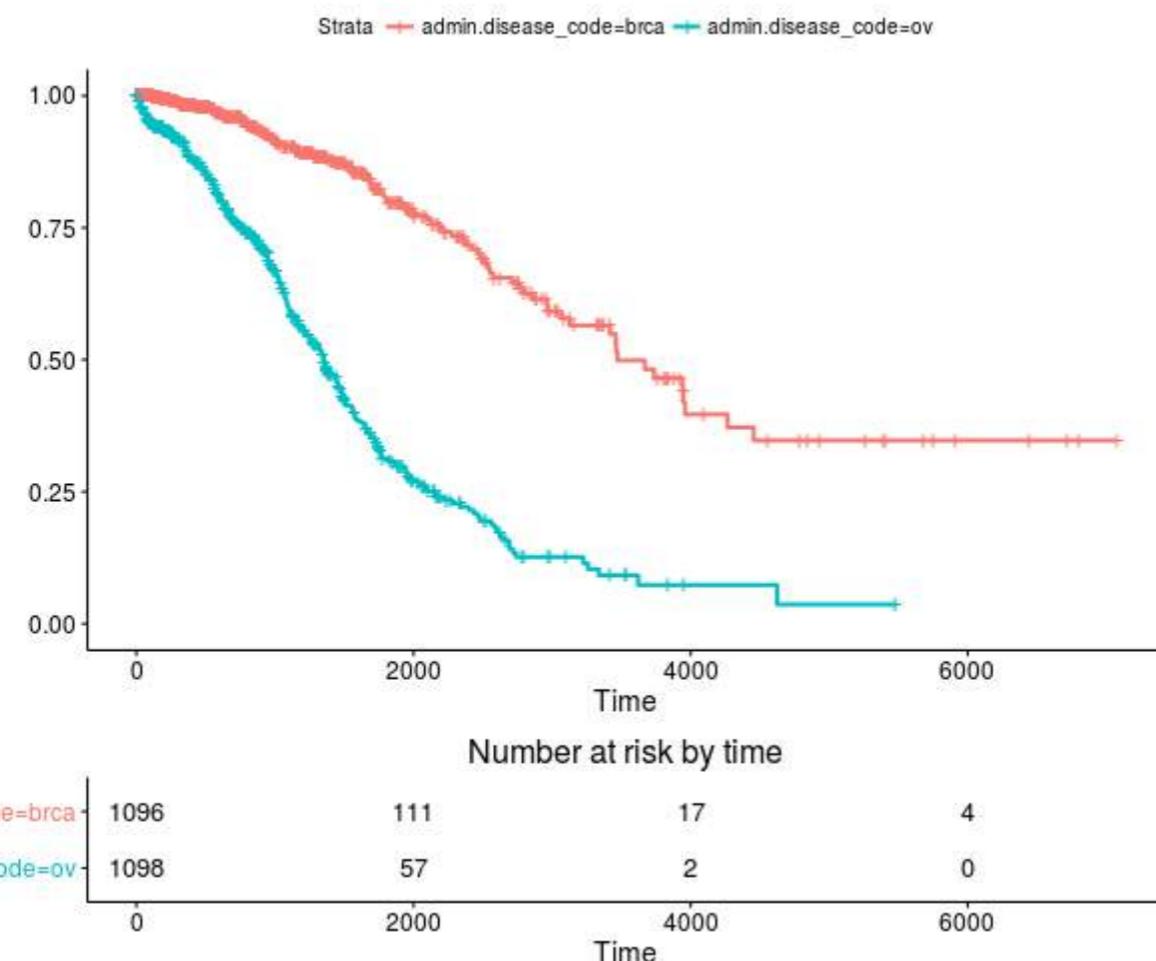


Section 72.3: Kaplan Meier estimates of survival curves and risk set tables with survminer

Base plot

```
install.packages('survminer')
source("https://bioconductor.org/biocLite.R")
biocLite("RTCGA.clinical") # data for examples
library(RTCGA.clinical)
survivalTCGA(BRCA.clinical, OV.clinical,
             extract.cols = "admin.disease_code") -> BRCAOV.survInfo
library(survival)
fit <- survfit(Surv(times, patient.vital_status) ~ admin.disease_code,
               data = BRCAOV.survInfo)
library(survminer)
ggsurvplot(fit, risk.table = TRUE)
```

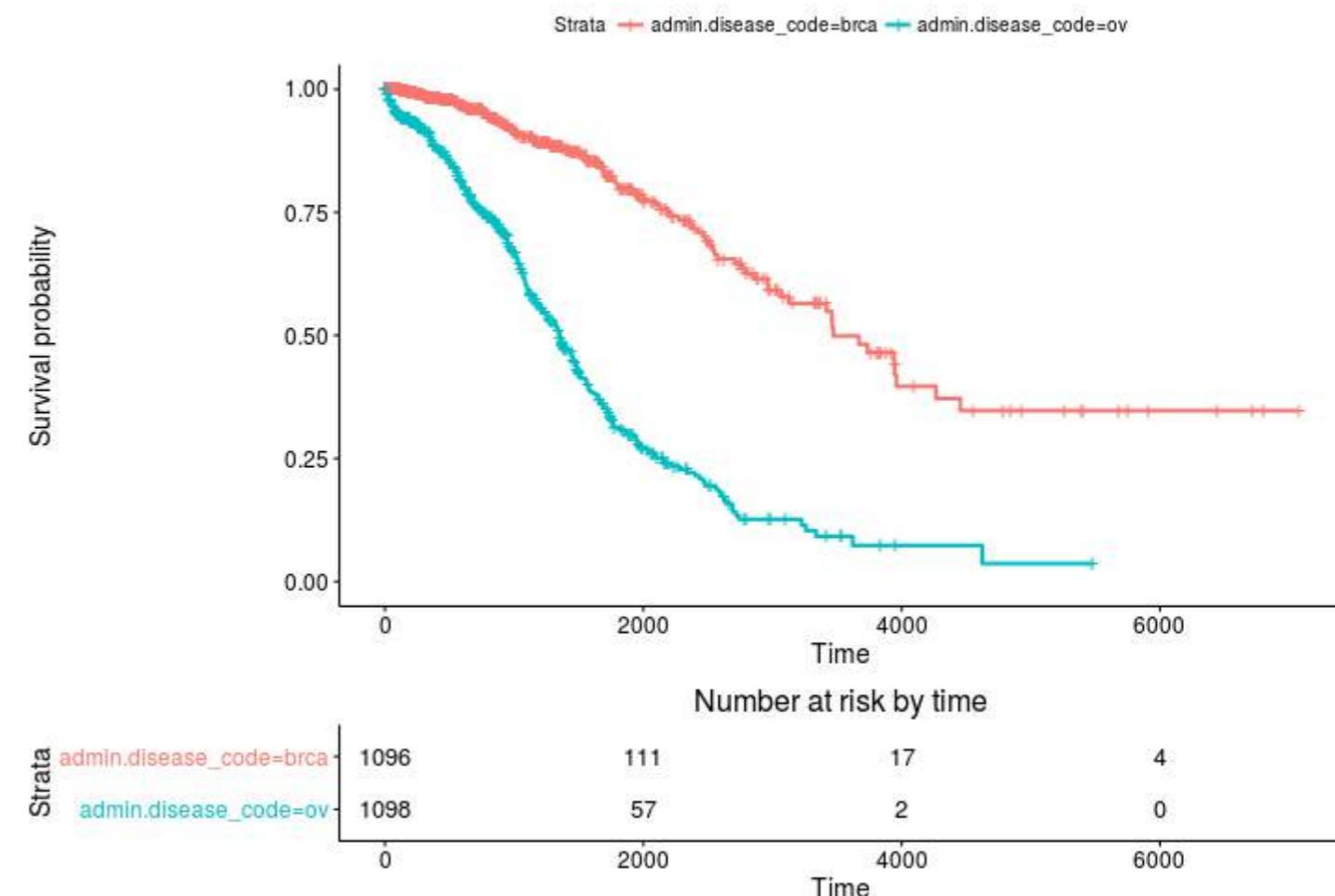
Survival probability



更高级

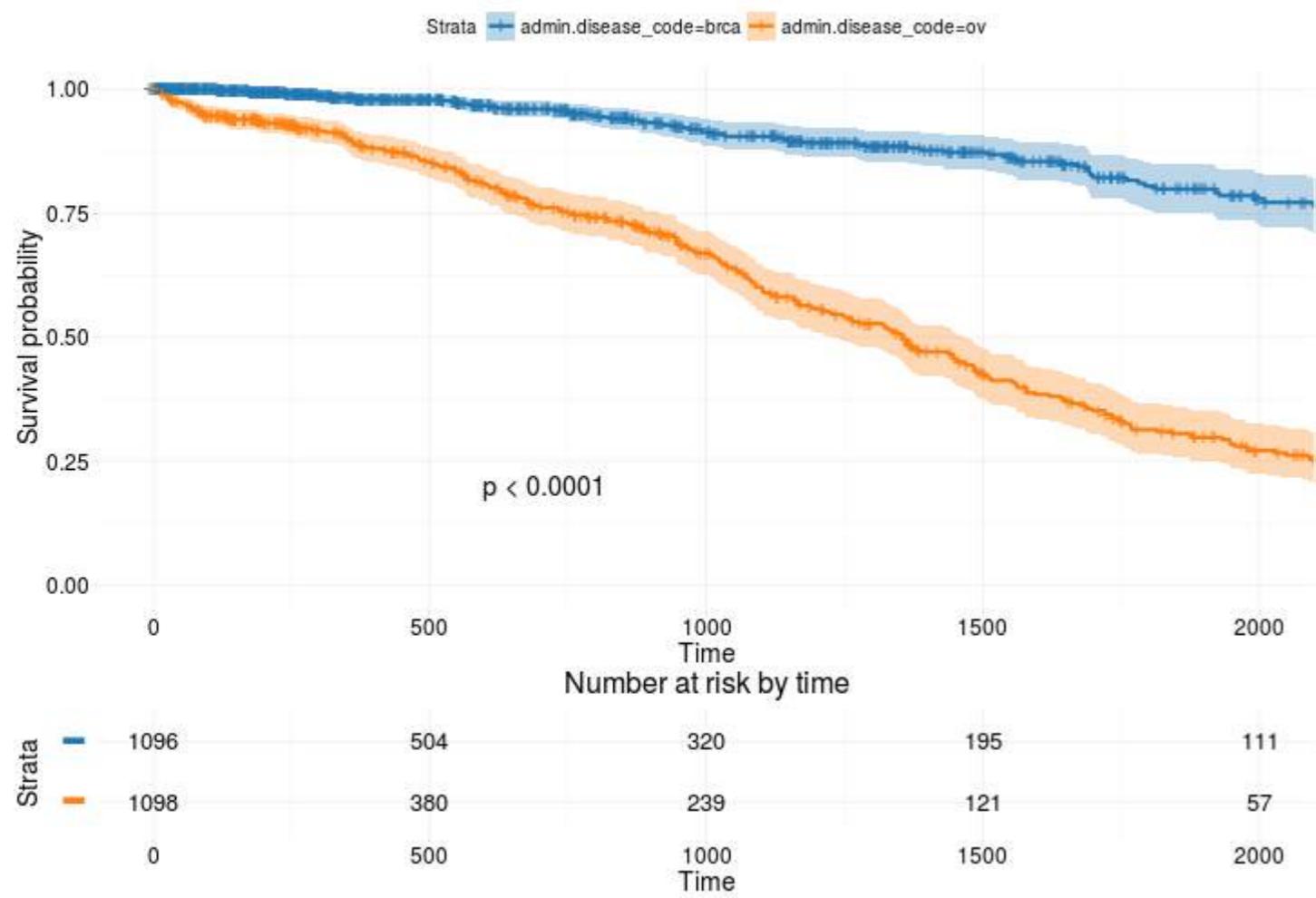
```
ggsurvplot(
  fit,          # 具有计算统计的survfit对象。
  risk.table = TRUE,    # 显示风险表。
  pval = TRUE,      # 显示对数秩检验的p值。
  conf.int = TRUE,   # 显示生存曲线点估计的置信区间。
  xlim = c(0,2000),  # 显示更窄的X轴，但不影响生存估计。
  break.time.by = 500,  # 将X轴按500的时间间隔分段。
  ggtheme = theme_RTCGA(), # 使用主题自定义图表和风险表。
  risk.table.y.text.col = T, # 风险表文本注释着色。
  risk.table.y.text = FALSE # 在风险表图例的文本注释中显示条形而非名称。
)
```

Survival probability



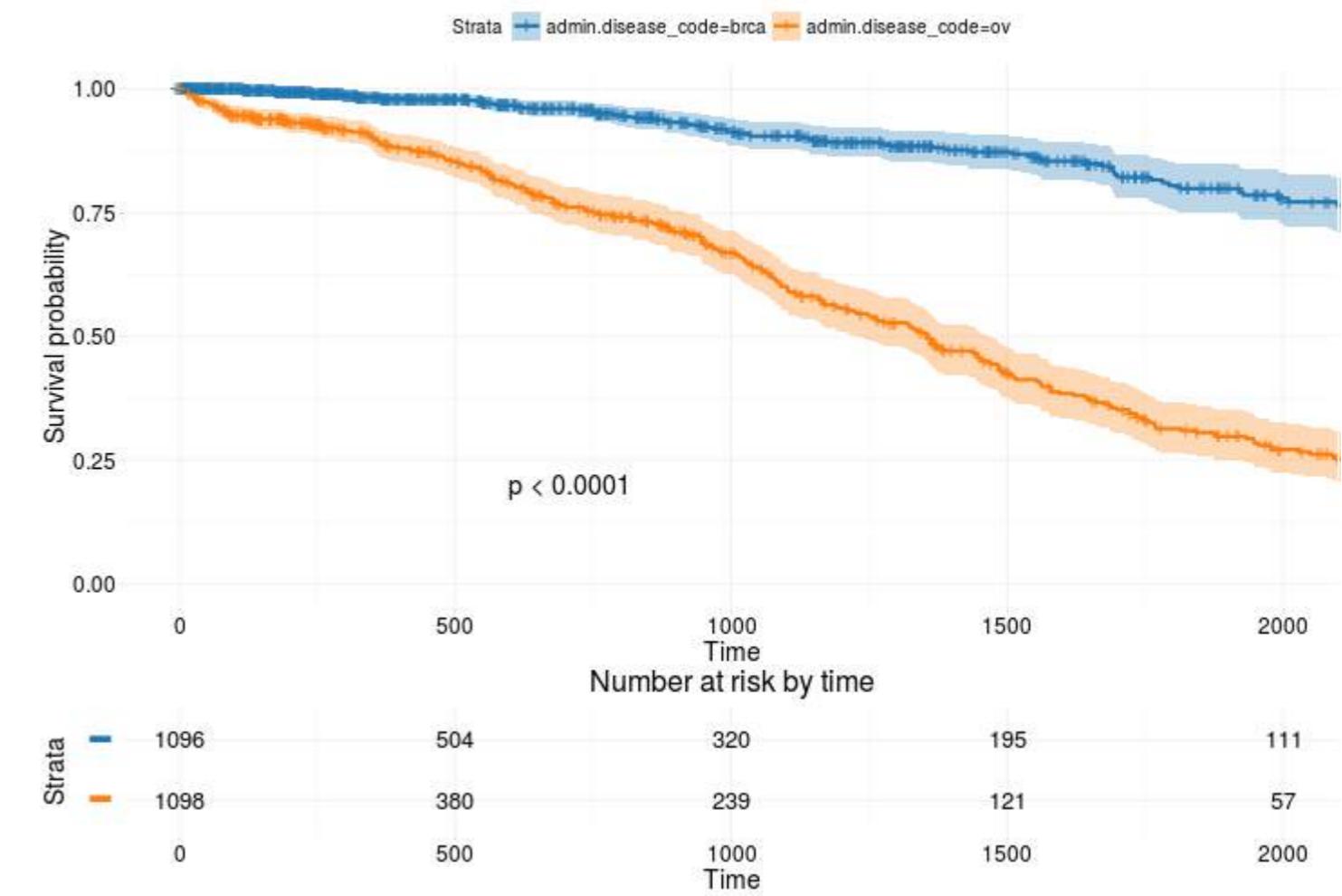
More advanced

```
ggsurvplot(
  fit,          # survfit object with calculated statistics.
  risk.table = TRUE,    # show risk table.
  pval = TRUE,      # show p-value of log-rank test.
  conf.int = TRUE,   # show confidence intervals for
                    # point estimates of survival curves.
  xlim = c(0,2000),  # present narrower X axis, but not affect
                    # survival estimates.
  break.time.by = 500,  # break X axis in time intervals by 500.
  ggtheme = theme_RTCGA(), # customize plot and risk table with a theme.
  risk.table.y.text.col = T, # colour risk table text annotations.
  risk.table.y.text = FALSE # show bars instead of names in text annotations
                           # in legend of risk table
)
```



基于

<http://r-addict.com/2016/05/23/Informative-Survival-Plots.html>



Based on

<http://r-addict.com/2016/05/23/Informative-Survival-Plots.html>

第73章：容错/弹性代码

参数

| | 详细信息 |
|---------|---|
| expr | 如果“try部分”成功完成，tryCatch 将返回最后计算的表达式。因此，在一切顺利且没有条件（即警告或错误）的情况下，实际返回的值是readLines的返回值。注意，你不需要通过return显式声明返回值，因为“try部分”的代码并未被包裹在函数环境中（与下面警告和错误的条件处理器不同）。 |
| 警告/错误/等 | 为你要显式处理的所有条件提供/定义一个处理函数。据我理解，你可以为任何类型的条件提供处理器（不仅仅是警告和错误，还包括自定义条件；参见simple Condition及相关内容），只要相应处理函数的名称与相应条件的类匹配（详见tryCatch文档的Details部分）。 |
| finally | 这里写入所有无论“try部分”表达式是否成功或是否出现任何条件都应执行的内容。如果你想执行多个表达式，则需要用大括号包裹它们，否则你也可以直接写成finally = <表达式>（即与“try部分”相同的逻辑）。 |

第73.1节：使用tryCatch()

我们定义了一个健壮的函数版本，用于从给定URL读取HTML代码。这里的“健壮”是指我们希望它能处理出现错误（error）或不完全按计划进行（警告）的情况。错误和警告的统称是条件

使用tryCatch定义函数

```
readUrl <- function(url) {  
  out <- tryCatch(  
  
    #####  
    # 尝试部分：定义您想要“尝试”的表达式 #  
    #####  
  
    {  
      # 仅作强调：  
      # 如果你想在“try部分”使用多个R表达式  
      # 那么你必须使用大括号。  
      # 否则，只需写你想尝试的单个表达式即可  
  
      message("这是'try'部分")  
      readLines(con = url, warn = FALSE)  
    },  
  
    #####  
    # 条件处理部分：定义你希望如何处理条件 #  
    #####  
  
    # 当发生警告时的处理器：  
    warning = function(cond) {  
      message(paste("读取URL时发生警告：" , url))  
      message("以下是原始警告信息：" )  
      message(cond)  
  
      # 选择当发生此类条件时的返回值  
      return(NULL)  
    },  
  
    # 发生错误时的处理程序：  
    error = function(cond) {
```

Chapter 73: Fault-tolerant/resilient code

Parameter

| | Details |
|---------|--|
| expr | In case the "try part" was completed successfully tryCatch will return the last evaluated expression . Hence, the actual value being returned in case everything went well and there is no condition (i.e. a warning or an error) is the return value of readLines . Note that you don't need to explicitly state the return value via return as code in the "try part" is not wrapped insided a function environment (unlike that for the condition handlers for warnings and error below) |
| finally | Provide/define a handler function for all the conditions that you want to handle explicitly. AFAIU, you can provide handlers for <i>any</i> type of conditions (not just warnings and errors, but also warning/error/etc custom conditions; see simpleCondition and friends for that) as long as the name of the respective handler function matches the class of the respective condition (see the Details part of the doc for tryCatch).

Here goes everything that should be executed at the very end, regardless if the expression in the "try part" succeeded or if there was any condition. If you want more than one expression to be executed, then you need to wrap them in curly brackets, otherwise you could just have written finally = <expression> (i.e. the same logic as for "try part"). |

Section 73.1: Using tryCatch()

We're defining a robust version of a function that reads the HTML code from a given URL. Robust in the sense that we want it to handle situations where something either goes wrong (error) or not quite the way we planned it to (warning). The umbrella term for errors and warnings is *condition*

Function definition using tryCatch

```
readUrl <- function(url) {  
  out <- tryCatch(  
  
    #####  
    # Try part: define the expression(s) you want to "try" #  
    #####  
  
    {  
      # Just to highlight:  
      # If you want to use more than one R expression in the "try part"  
      # then you'll have to use curly brackets.  
      # Otherwise, just write the single expression you want to try and  
  
      message("This is the 'try' part")  
      readLines(con = url, warn = FALSE)  
    },  
  
    #####  
    # Condition handler part: define how you want conditions to be handled #  
    #####  
  
    # Handler when a warning occurs:  
    warning = function(cond) {  
      message(paste("Reading the URL caused a warning:" , url))  
      message("Here's the original warning message:")  
      message(cond)  
  
      # Choose a return value when such a type of condition occurs  
      return(NULL)  
    },  
  
    # Handler when an error occurs:  
    error = function(cond) {
```

```

message(paste("这似乎是一个无效的URL:", url))
message("以下是原始错误信息:")
message(cond)

# 选择当发生此类条件时的返回值
return(NA)
},

#####
##### 最后部分：定义在所有操作尝试和/或处理后应执行的内容 #####
#####

finally = {
  message(paste("处理后的URL:", url))message(
    "结束时的一些信息")
}

return(out)
}

```

测试内容

让我们定义一个包含一个无效URL元素的URL向量

```

urls <- c(
  "http://stat.ethz.ch/R-manual/R-devel/library/base/html/connections.html",
  "http://en.wikipedia.org/wiki/Xz",
  "我不是URL"
)

```

并将其作为输入传递给我们上面定义的函数

```

y <- lapply(urls, readUrl)
# 处理后的URL : http://stat.ethz.ch/R-manual/R-devel/library/base/html/connections.html
# 末尾的一些信息
#
# 处理后的URL : http://en.wikipedia.org/wiki/Xz
# 末尾的一些信息
#
# URL似乎不存在：我不是URL
# 这是原始错误信息：
# 无法打开连接
# 处理的 URL：我不是 URL
# 末尾的一些信息
#
# 警告信息：
# 在文件(con, "r")中：无法打开文件 '我不是 URL'：没有此类文件或目录

```

正在调查输出

```

length(y)
# [1] 3

head(y[[1]])
# [1] "<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">"
# [2] "<html><head><title>R: 操作连接的函数</title>"
# [3] "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=utf-8\">"
# [4] "<link rel=\"stylesheet\" type=\"text/css\" href=\"R.css\">"
# [5] "</head><body>"

```

```

message(paste("This seems to be an invalid URL:", url))
message("Here's the original error message:")
message(cond)

# Choose a return value when such a type of condition occurs
return(NA)
},

#####
##### 最后部分：定义在所有操作尝试和/或处理后应执行的内容 #####
#####

finally = {
  message(paste("Processed URL:", url))
  message("Some message at the end\n")
}
return(out)
}

```

Testing things out

Let's define a vector of URLs where one element isn't a valid URL

```

urls <- c(
  "http://stat.ethz.ch/R-manual/R-devel/library/base/html/connections.html",
  "http://en.wikipedia.org/wiki/Xz",
  "I'm no URL"
)

```

And pass this as input to the function we defined above

```

y <- lapply(urls, readUrl)
# Processed URL: http://stat.ethz.ch/R-manual/R-devel/library/base/html/connections.html
# Some message at the end
#
# Processed URL: http://en.wikipedia.org/wiki/Xz
# Some message at the end
#
# URL does not seem to exist: I'm no URL
# Here's the original error message:
# cannot open the connection
# Processed URL: I'm no URL
# Some message at the end
#
# Warning message:
# In file(con, "r") : cannot open file 'I'm no URL': No such file or directory

```

Investigating the output

```

length(y)
# [1] 3

head(y[[1]])
# [1] "<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">"
# [2] "<html><head><title>R: Functions to Manipulate Connections</title>"
# [3] "<meta http-equiv=\"Content-Type\" content=\"text/html; charset=utf-8\">"
# [4] "<link rel=\"stylesheet\" type=\"text/css\" href=\"R.css\">"
# [5] "</head><body>"

```

```
# [6] ""
```

```
y[[3]]  
# [1] NA
```

```
# [6] ""
```

```
y[[3]]  
# [1] NA
```

第74章：可复现的R语言

“可重复性”指的是其他人（也许是未来的你）能够重复你所执行的步骤并获得相同的结果。参见可重复研究任务视图。

第74.1节：数据可重复性

dput() 和 dget()

共享（最好是小型）数据框的最简单方法是使用基本函数 `dput()`。它会以纯文本形式导出一个R对象。

注意：在制作下面的示例数据之前，请确保你处于一个可以写入的空文件夹中。运行 `getwd()`，如果需要更改文件夹，请阅读 `?setwd`。

```
dput(mtcars, file = 'df.txt')
```

然后，任何人都可以使用 `dget()` 函数将精确的R对象加载到他们的全局环境中。

```
df <- dget('df.txt')
```

对于较大的R对象，有多种方法可以实现可重复保存。参见输入和输出。

第74.2节：包的可重复性

包的可复现性是在复现某些R代码时非常常见的问题。当各种包被更新时，它们之间的一些相互连接可能会断裂。解决该问题的理想方案是在你的计算机上重现R代码编写者机器的镜像，时间点为代码编写时的日期。这时就用到了checkpoint包。

从2014年9月17日起，该包的作者每天都会将整个CRAN包仓库复制到他们自己的镜像仓库——微软R归档网络。因此，为了避免在创建可复现的R项目时出现包的可复现性问题，你只需要：

1. 确保你所有的包（以及R版本）都是最新的。
2. 在你的代码中包含`checkpoint::checkpoint('YYYY-MM-DD')`这一行。

`checkpoint`会在你的R_home目录（“~”）下创建一个目录`.checkpoint`。它会将项目中使用的所有包安装到这个技术目录中。这意味着，`checkpoint`会扫描你项目目录下所有的.R文件，查找所有的`library()`或`require()`调用，并以指定日期CRAN上存在的形式安装所有所需的包。

优点 你可以摆脱包的可复现性问题。

缺点 对于每个指定的日期，你必须下载并安装你想复现的某个项目中使用的所有包。这可能会花费相当长的时间。

Chapter 74: Reproducible R

With 'Reproducibility' we mean that someone else (perhaps you in the future) can repeat the steps you performed and get the same result. See the [Reproducible Research Task View](#).

Section 74.1: Data reproducibility

`dput()` and `dget()`

The easiest way to share a (preferable small) data frame is to use a basic function `dput()`. It will export an R object in a plain text form.

Note: Before making the example data below, make sure you're in an empty folder you can write to. Run `getwd()` and read `?setwd` if you need to change folders.

```
dput(mtcars, file = 'df.txt')
```

Then, anyone can load the precise R object to their GlobalEnvironment using the `dget()` function.

```
df <- dget('df.txt')
```

For larger R objects, there are a number of ways of saving them reproducibly. See Input and output .

Section 74.2: Package reproducibility

Package reproducibility is a very common issue in reproducing some R code. When various packages get updated, some interconnections between them may break. The ideal solution for the problem is to reproduce the image of the R code writer's machine on your computer at the date when the code was written. And here comes `checkpoint` package.

Starting from 2014-09-17, the authors of the package make daily copies of the whole CRAN package repository to their own mirror repository -- Microsoft R Archived Network. So, to avoid package reproduciblity issues when creating a reproducible R project, all you need is to:

1. Make sure that all your packages (and R version) are up-to-date.
2. Include `checkpoint::checkpoint('YYYY-MM-DD')` line in your code.

`checkpoint` will create a directory `.checkpoint` in your R_home directory ("~"). To this technical directory it will install all the packages, that are used in your project. That means, `checkpoint` looks through all the .R files in your project directory to pick up all the `library()` or `require()` calls and install all the required packages in the form they existed at CRAN on the specified date.

PRO You are freed from the package reproducibility issue.

CONTRA For each specified date you have to download and install all the packages that are used in a certain project that you aim to reproduce. That may take quite a while.

第75章：傅里叶级数与变换

傅里叶变换将时间函数（信号）分解为构成它的频率，类似于一个和弦可以表示为其组成音符的振幅（或响度）。时间函数的傅里叶变换本身是一个频率的复值函数，其绝对值表示原函数中该频率的含量，复数的幅角表示该频率基本正弦波的相位偏移。

傅里叶变换被称为原始信号的频域表示。傅里叶变换一词既指频域表示，也指将频域表示与时间函数关联的数学运算。傅里叶变换不限于时间函数，但为了统一表述，原函数的定义域通常称为时间域。对于许多实际感兴趣的函数，可以定义一个逆运算：逆傅里叶变换，也称为傅里叶合成，将所有不同频率的贡献组合起来，恢复原始的时间函数。

在一个域（时间域或频率域）中执行的线性运算，在另一个域中有对应的运算，有时更容易执行。时间域中的微分运算对应于频率的乘法，因此某些微分方程在频率域中更容易分析。

此外，时域中的卷积对应于频域中的普通乘法。具体来说，这意味着任何线性时不变系统，例如应用于信号的电子滤波器，都可以相对简单地表示为对频率的操作。因此，通过将时域函数转换到频域，执行所需操作，然后将结果转换回时域，通常可以实现显著的简化。

谐波分析是对频域和时域之间关系的系统研究，包括在其中一种域中“更简单”的函数或操作类型，并且与现代数学的几乎所有领域都有深刻联系。

在时域中局部化的函数，其傅里叶变换在频域中则是分散的，反之亦然。关键的例子是高斯函数，它在概率论和统计学中具有重要意义，同时也用于研究表现正态分布的物理现象（例如扩散）。经过适当归一化后，高斯函数在傅里叶变换下保持不变。约瑟夫·傅里叶在研究热传导时引入了傅里叶变换，高斯函数作为热方程的解出现。

傅里叶变换可以形式上定义为一个不定黎曼积分，使其成为一种积分变换，尽管这种定义不适用于许多需要更复杂积分理论的应用。

例如，许多相对简单的应用使用狄拉克 δ 函数，可以形式上将其视为函数，但其合理性需要更为复杂的数学视角。傅里叶变换还可以推广到欧几里得空间中多个变量的函数，将三维空间的函数映射为三维动量的函数（或将时空函数映射为四维动量的函数）。

这一思想使得空间傅里叶变换在波动研究中非常自然，在量子力学中尤为重要，因为能够将波动解表示为空间函数或动量函数，有时两者兼有。一般来说，适用傅里叶方法的函数是复值的，可能还是向量值的。更进一步的推广是对群上的函数进行傅里叶变换，除了最初定义在 \mathbb{R} 或 \mathbb{R}^n （作为加法群）上的傅里叶变换外，还包括离散时间傅里叶变换（DTFT，群 = \mathbb{Z} ）、离散傅里叶变换（DFT，群 = \mathbb{Z} 模N）以及傅里叶级数或圆形傅里叶变换（群 = S_1 ，单位圆≈端点相连的闭合有限区间）。后者常用于处理周期函数。

Chapter 75: Fourier Series and Transformations

The Fourier transform decomposes a function of time (a signal) into the frequencies that make it up, similarly to how a musical chord can be expressed as the amplitude (or loudness) of its constituent notes. The Fourier transform of a function of time itself is a complex-valued function of frequency, whose absolute value represents the amount of that frequency present in the original function, and whose complex argument is the phase offset of the basic sinusoid in that frequency.

The Fourier transform is called the frequency domain representation of the original signal. The term Fourier transform refers to both the frequency domain representation and the mathematical operation that associates the frequency domain representation to a function of time. The Fourier transform is not limited to functions of time, but in order to have a unified language, the domain of the original function is commonly referred to as the time domain. For many functions of practical interest one can define an operation that reverses this: the inverse Fourier transformation, also called Fourier synthesis, of a frequency domain representation combines the contributions of all the different frequencies to recover the original function of time.

Linear operations performed in one domain (time or frequency) have corresponding operations in the other domain, which are sometimes easier to perform. The operation of differentiation in the time domain corresponds to multiplication by the frequency, so some differential equations are easier to analyze in the frequency domain. Also, convolution in the time domain corresponds to ordinary multiplication in the frequency domain. Concretely, this means that any linear time-invariant system, such as an electronic filter applied to a signal, can be expressed relatively simply as an operation on frequencies. So significant simplification is often achieved by transforming time functions to the frequency domain, performing the desired operations, and transforming the result back to time.

Harmonic analysis is the systematic study of the relationship between the frequency and time domains, including the kinds of functions or operations that are "simpler" in one or the other, and has deep connections to almost all areas of modern mathematics.

Functions that are localized in the time domain have Fourier transforms that are spread out across the frequency domain and vice versa. The critical case is the Gaussian function, of substantial importance in probability theory and statistics as well as in the study of physical phenomena exhibiting normal distribution (e.g., diffusion), which with appropriate normalizations goes to itself under the Fourier transform. Joseph Fourier introduced the transform in his study of heat transfer, where Gaussian functions appear as solutions of the heat equation.

The Fourier transform can be formally defined as an improper Riemann integral, making it an integral transform, although this definition is not suitable for many applications requiring a more sophisticated integration theory.

For example, many relatively simple applications use the Dirac delta function, which can be treated formally as if it were a function, but the justification requires a mathematically more sophisticated viewpoint. The Fourier transform can also be generalized to functions of several variables on Euclidean space, sending a function of 3-dimensional space to a function of 3-dimensional momentum (or a function of space and time to a function of 4-momentum).

This idea makes the spatial Fourier transform very natural in the study of waves, as well as in quantum mechanics, where it is important to be able to represent wave solutions either as functions either of space or momentum and sometimes both. In general, functions to which Fourier methods are applicable are complex-valued, and possibly vector-valued. Still further generalization is possible to functions on groups, which, besides the original Fourier transform on \mathbb{R} or \mathbb{R}^n (viewed as groups under addition), notably includes the discrete-time Fourier transform (DTFT, group = \mathbb{Z}), the discrete Fourier transform (DFT, group = \mathbb{Z} mod N) and the Fourier series or circular Fourier transform (group = S_1 , the unit circle ≈ closed finite interval with endpoints identified). The latter is routinely

快速傅里叶变换（FFT）是一种计算离散傅里叶变换（DFT）的算法。

第75.1节：傅里叶级数

约瑟夫·傅里叶证明了任何周期波都可以表示为简单正弦波的叠加。这个叠加称为傅里叶级数。傅里叶级数仅在线性系统中成立。如果存在例如溢出效应（输出在某个阈值后不随输入增加而变化），则引入非线性效应，破坏正弦波形和叠加原理。

```
# 正弦波
xs <- seq(-2*pi,2*pi,pi/100)
wave.1 <- sin(3*xs)
wave.2 <- sin(10*xs)
par(mfrow = c(1, 2))
plot(xs, wave.1, type="l", ylim=c(-1,1)); abline(h=0,lty=3)
plot(xs, wave.2, type="l", ylim=c(-1,1)); abline(h=0,lty=3)

# 复合波
wave.3 <- 0.5 * wave.1 + 0.25 * wave.2
plot(xs, wave.3, type="l"); title("示例复合波"); abline(h=0,lty=3)
```

employed to handle periodic functions. The Fast Fourier transform (FFT) is an algorithm for computing the DFT.

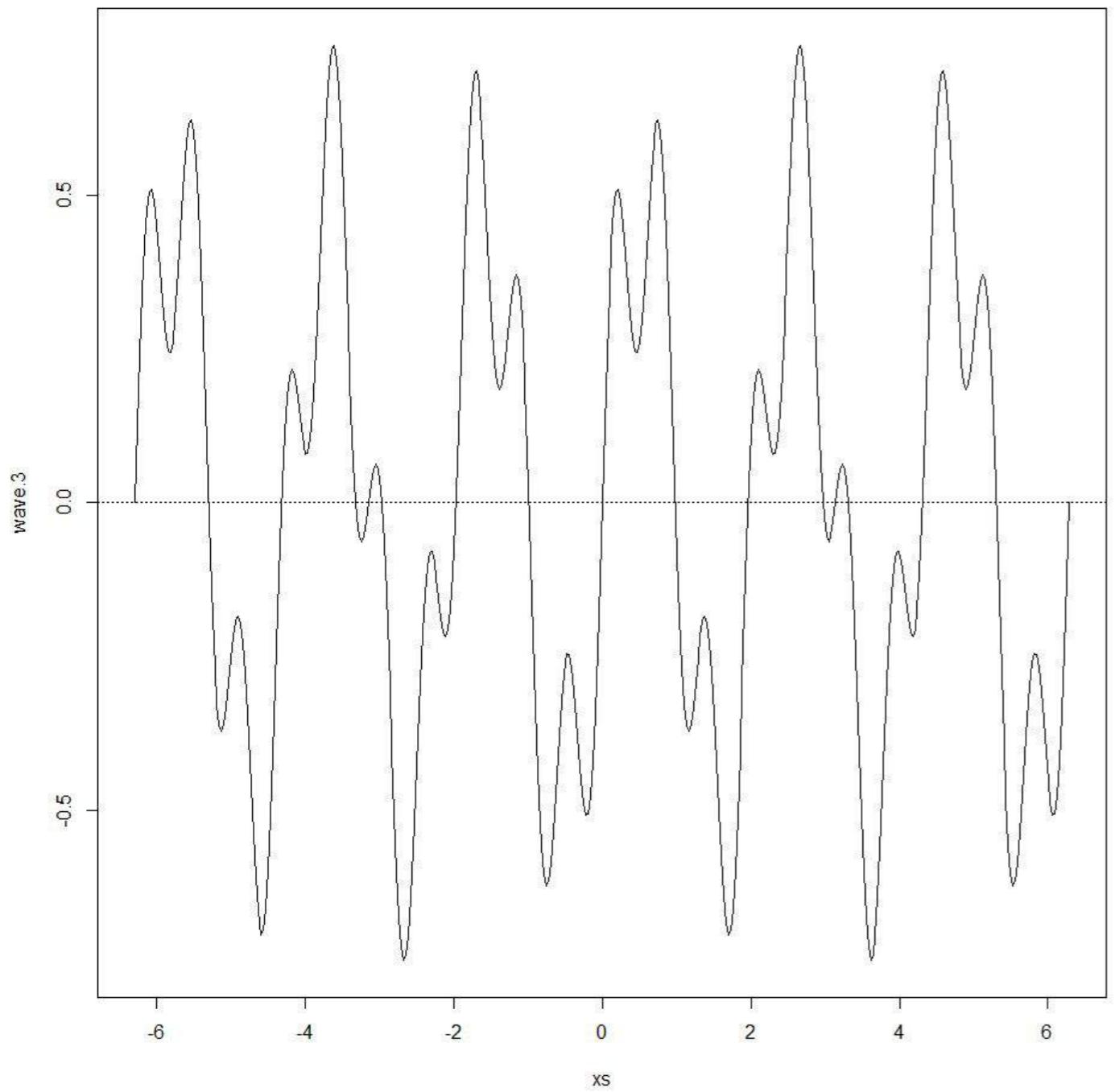
Section 75.1: Fourier Series

Joseph Fourier showed that any periodic wave can be represented by a sum of simple sine waves. This sum is called the Fourier Series. The Fourier Series only holds while the system is linear. If there is, eg, some overflow effect (a threshold where the output remains the same no matter how much input is given), a non-linear effect enters the picture, breaking the sinusoidal wave and the superposition principle.

```
# Sine waves
xs <- seq(-2*pi,2*pi,pi/100)
wave.1 <- sin(3*xs)
wave.2 <- sin(10*xs)
par(mfrow = c(1, 2))
plot(xs, wave.1, type="l", ylim=c(-1,1)); abline(h=0,lty=3)
plot(xs, wave.2, type="l", ylim=c(-1,1)); abline(h=0,lty=3)

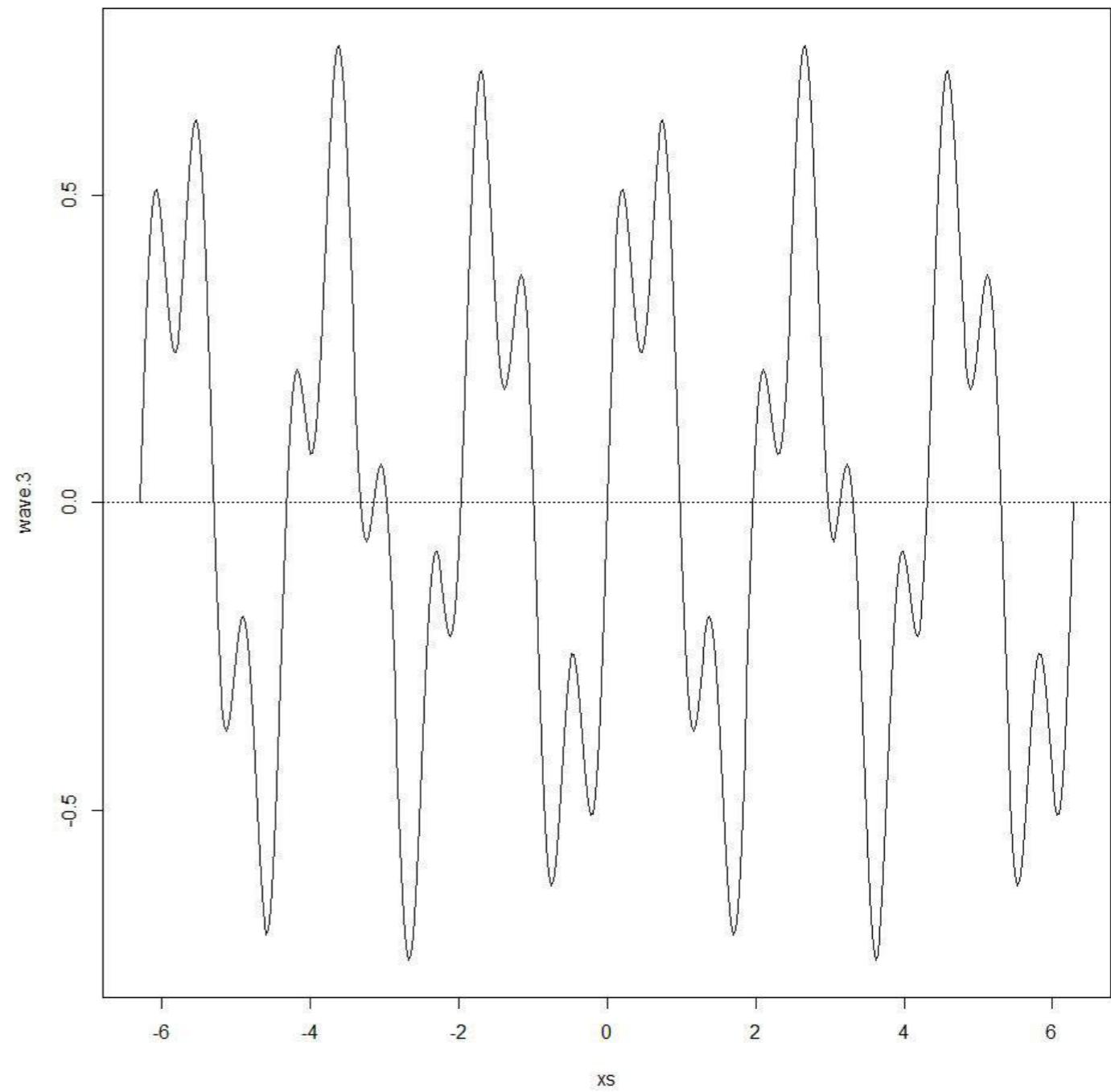
# Complex Wave
wave.3 <- 0.5 * wave.1 + 0.25 * wave.2
plot(xs, wave.3, type="l"); title("Eg complex wave"); abline(h=0,lty=3)
```

Eg complex wave



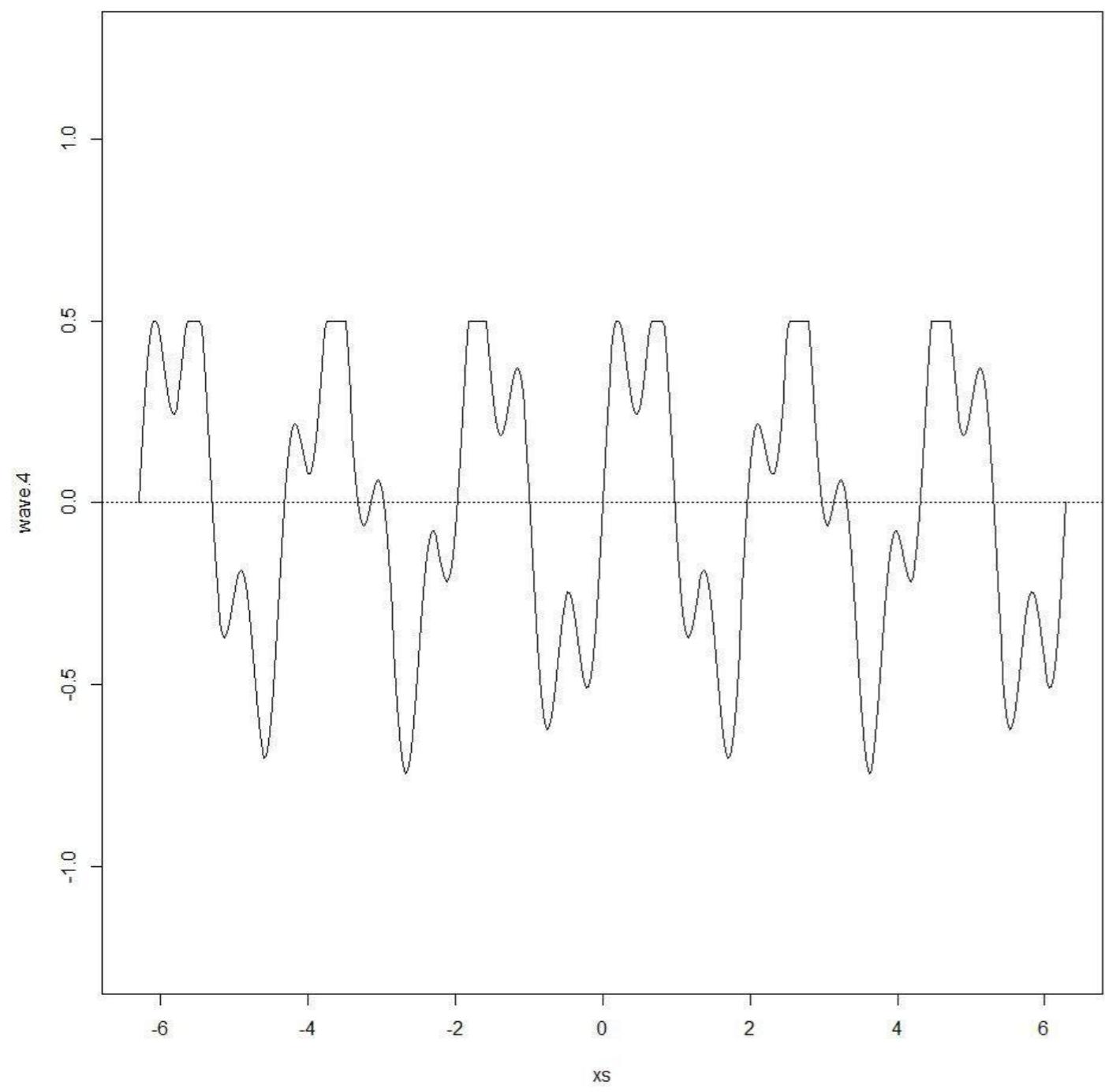
```
wave.4 <- wave.3  
wave.4[wave.3>0.5] <- 0.5  
plot(xs,wave.4,type="l",ylim=c(-1.25,1.25))  
title("溢出, 非线性复合波")  
abline(h=0,lty=3)
```

Eg complex wave



```
wave.4 <- wave.3  
wave.4[wave.3>0.5] <- 0.5  
plot(xs,wave.4,type="l",ylim=c(-1.25,1.25))  
title("overflowed, non-linear complex wave")  
abline(h=0,lty=3)
```

overflowed, non-linear complex wave

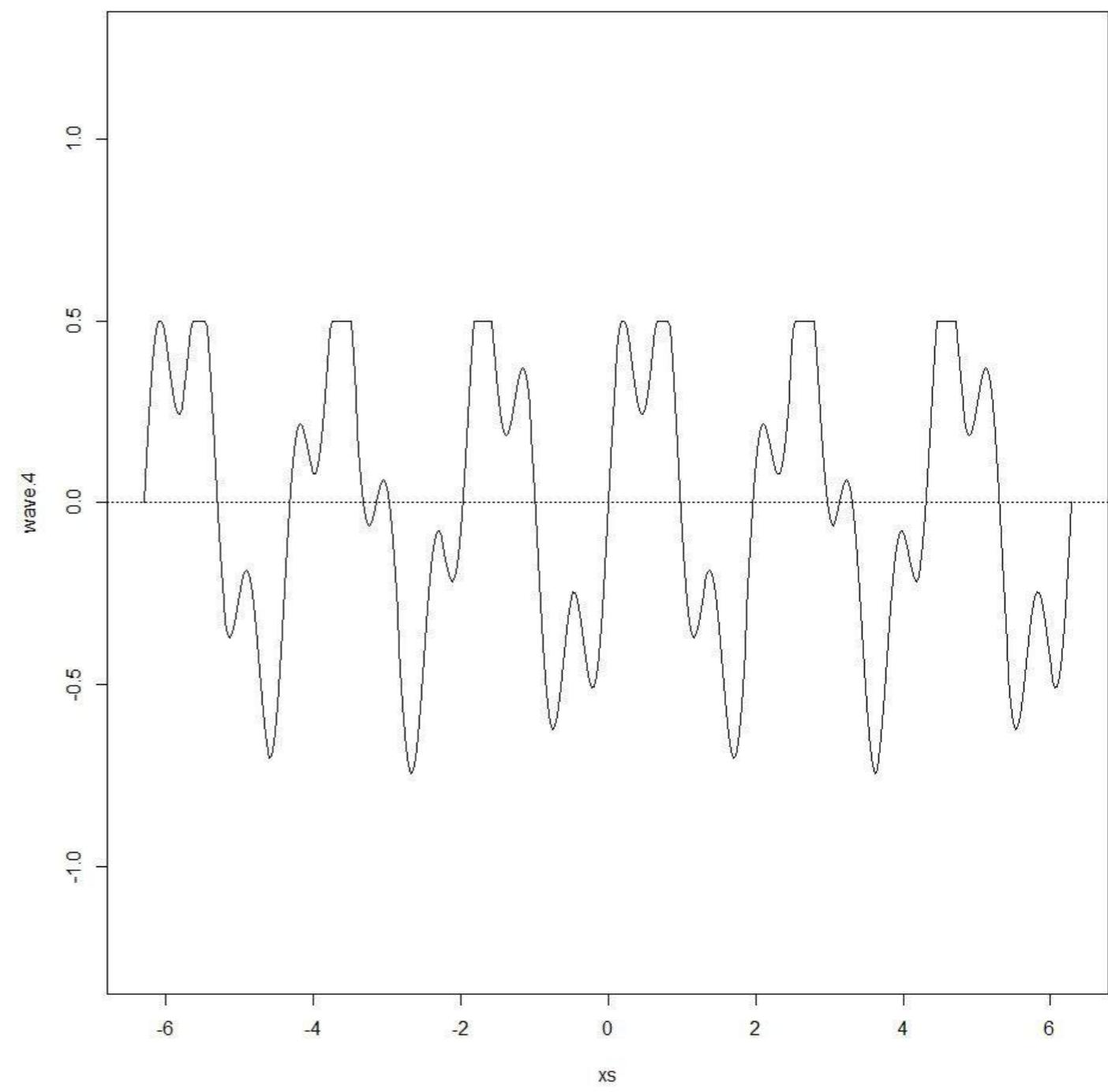


此外，傅里叶级数仅适用于波是周期性的，即它们具有重复的模式（非周期波由傅里叶变换处理，见下文）。周期波具有频率 f 和波长 λ （波长是介质中一个周期的起点和终点之间的距离， $\lambda = v/f_0$ ，其中 v 是波速），这些由重复的模式定义。非周期波没有频率或波长。

一些概念：

- 基本周期 T 是所有采样的周期，即从第一个采样点到最后一个采样点的时间间隔
- 采样率 sr 是在一段时间内采集的样本数量（也称为采集频率）。为了简化，我们将样本之间的时间间隔设为相等。这个时间间隔称为采样间隔 si ，即基本周期时间除以样本数 N 。所以， $si=T/N$
- 基本频率 f_0 是 $1/T$ 。基本频率是重复模式的频率，或者说是波长的长度。在之前的波形中，基本频率是 12π 。

overflowed, non-linear complex wave



Also, the Fourier Series only holds if the waves are periodic, ie, they have a repeating pattern (non periodic waves are dealt by the Fourier Transform, see below). A periodic wave has a frequency f and a wavelength λ (a wavelength is the distance in the medium between the beginning and end of a cycle, $\lambda=v/f_0$, where v is the wave velocity) that are defined by the repeating pattern. A non-periodic wave does not have a frequency or wavelength.

Some concepts:

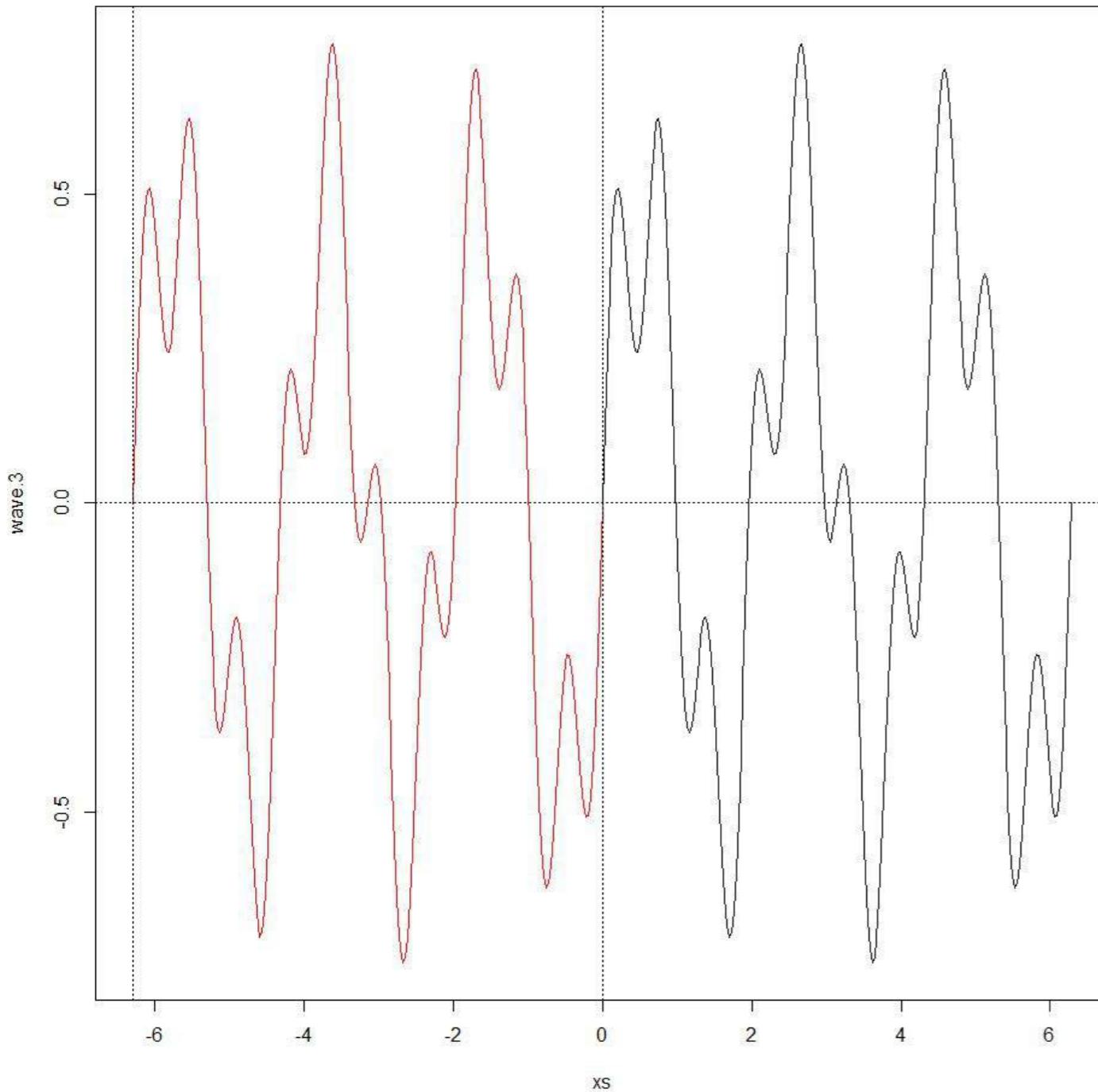
- The fundamental period, T , is the period of all the samples taken, the time between the first sample and the last
- The sampling rate, sr , is the number of samples taken over a time period (aka acquisition frequency). For simplicity we will make the time interval between samples equal. This time interval is called the sample interval, si , which is the fundamental period time divided by the number of samples N . So, $si=T/N$
- The fundamental frequency, f_0 , which is $1/T$. The fundamental frequency is the frequency of the repeating pattern or how long the wavelength is. In the previous waves, the fundamental frequency was 12π . The

波形分量的频率必须是基本频率的整数倍。f0 称为第一谐波，第二谐波是 $2*f0$ ，第三谐波是 $3*f0$ ，依此类推。

```
repeat.xs      <- seq(-2*pi,0,pi/100)
wave.3.repeat <- 0.5*sin(3*repeat.xs) + 0.25*sin(10*repeat.xs)
plot(xs, wave.3, type="l")

title("Repeating pattern")
points(repeat.xs, wave.3.repeat, type="l", col="red");
abline(h=0, v=c(-2*pi,0), lty=3)
```

Repeating pattern



这里有一个用于绘制给定傅里叶级数轨迹的 R 函数：

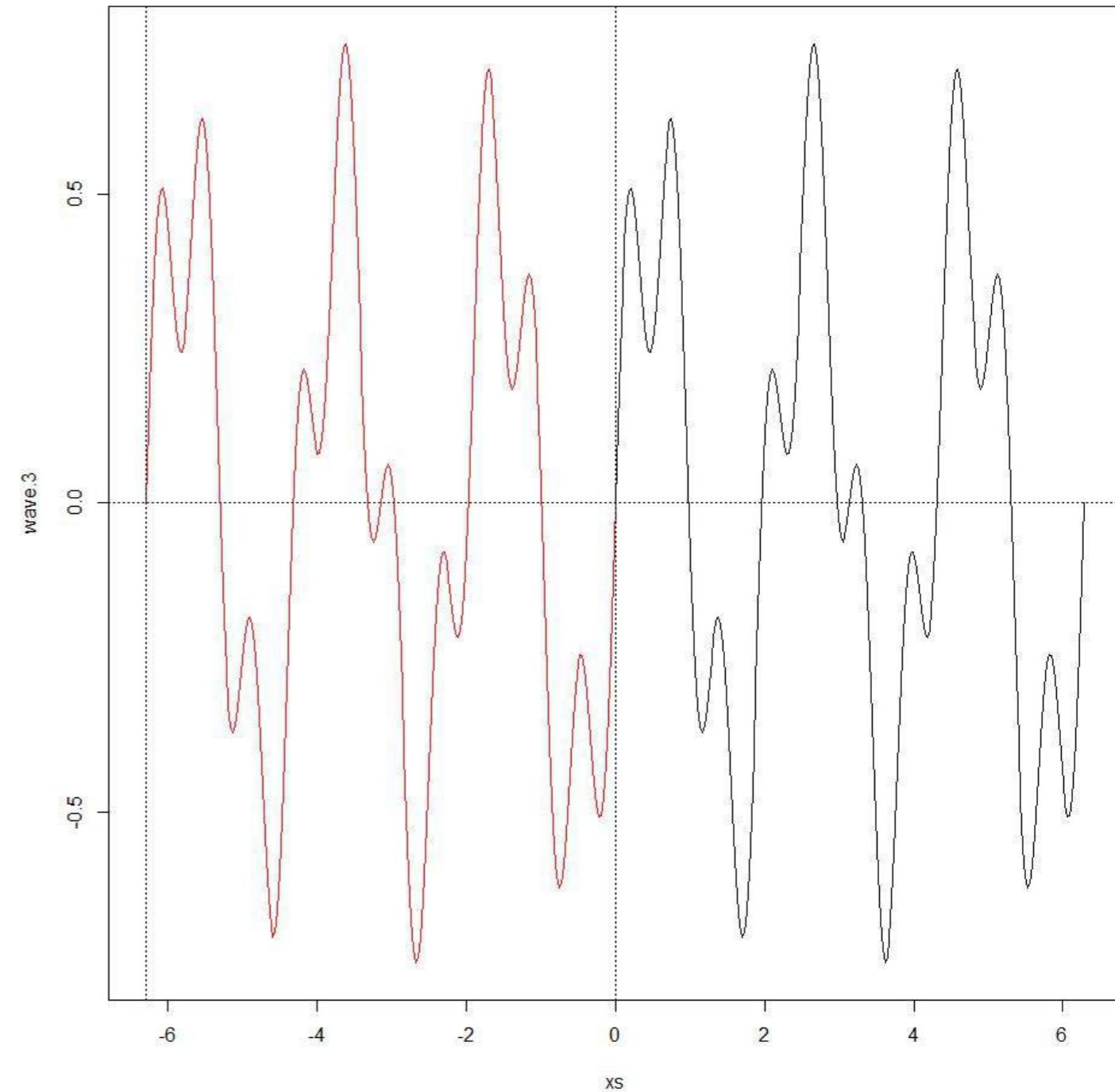
```
plot.fourier <- function(fourier.series, f.0, ts) {
```

frequencies of the wave components must be integer multiples of the fundamental frequency. f0 is called the first harmonic, the second harmonic is $2*f0$, the third is $3*f0$, etc.

```
repeat.xs      <- seq(-2*pi,0,pi/100)
wave.3.repeat <- 0.5*sin(3*repeat.xs) + 0.25*sin(10*repeat.xs)
plot(xs, wave.3, type="l")

title("Repeating pattern")
points(repeat.xs, wave.3.repeat, type="l", col="red");
abline(h=0, v=c(-2*pi,0), lty=3)
```

Repeating pattern



Here's a R function for plotting trajectories given a fourier series:

```
plot.fourier <- function(fourier.series, f.0, ts) {
```

```
w <- 2*pi*f.0 轨迹 <- sapply(ts, function(t) fourier.series(t,w))
  plot(ts, 轨迹, type="l", xlab="时间", ylab="f(t)");
  abline(h=0,lty=3)}
```

```
w <- 2*pi*f.0 trajectory <- sapply(ts, function(t) fourier.series(t,w))
  plot(ts, trajectory, type="l", xlab="time", ylab="f(t)");
  abline(h=0,lty=3)}
```

第76章：.Rprofile

第76.1节：.Rprofile - 执行的第一段代码

.Rprofile 是一个包含R代码的文件，当你从包含该文件的目录启动R时会执行该文件中的代码。.Rprofile 文件名相似的 Rprofile.site 位于R的主目录中，每次你从任何目录加载R时都会默认执行。 Rprofile.site 和更大程度上 .Rprofile 可以用来初始化R会话，设置个人偏好和你定义的各种实用函数。

重要提示：如果你使用RStudio，可以在每个RStudio项目目录中拥有单独的 .Rprofile 文件。

以下是一些你可能会包含在 .Rprofile 文件中的代码示例。

设置你的R主目录

```
# 设置 R_home  
Sys.setenv(R_USER="c:/R_home") # 仅为示例目录  
# 但不要将此与 $R_HOME 环境变量混淆。
```

设置页面大小选项

```
options(paperSize="a4")  
options(editor="notepad")  
options(pager="internal")
```

设置默认帮助类型

```
options(help_type="html")
```

设置站点库

```
.Library.site <- file.path(chartr("\\", "/", R.home()), "site-library")
```

设置 CRAN 镜像

```
local({r <- getOption("repos")  
r["CRAN"] <- "http://my.local.cran"  
options(repos=r)})
```

设置库的位置

这将使您在每次更新R版本时无需重新安装所有软件包。

```
# 库位置  
.libPaths("c:/R_home/Rpackages/win")
```

自定义快捷方式或函数

有时为一个较长的R表达式设置快捷方式是很有用的。一个常见的例子是设置一个活动绑定，以便访问最后一个顶层表达式的结果，而无需输入.Last.value：

```
makeActiveBinding(".", function(){.Last.value}, .GlobalEnv)
```

因为.Rprofile只是一个R文件，所以它可以包含任意的R代码。

预加载最有用的软件包

这是一种不好的做法，通常应避免，因为它将软件包加载代码与实际使用这些软件包的脚本分离开来。

另请参见

Chapter 76: .Rprofile

Section 76.1: .Rprofile - the first chunk of code executed

.Rprofile is a file containing R code that is executed when you launch R from the directory containing the .Rprofile file. The similarly named Rprofile.site, located in R's home directory, is executed by default every time you load R from any directory. Rprofile.site and to a greater extend .Rprofile can be used to initialize an R session with personal preferences and various utility functions that you have defined.

Important note: if you use RStudio, you can have a separate .Rprofile in every RStudio project directory.

Here are some examples of code that you might include in an .Rprofile file.

Setting your R home directory

```
# set R_home  
Sys.setenv(R_USER="c:/R_home") # just an example directory  
# but don't confuse this with the $R_HOME environment variable.
```

Setting page size options

```
options(paperSize="a4")  
options(editor="notepad")  
options(pager="internal")
```

set the default help type

```
options(help_type="html")
```

set a site library

```
.Library.site <- file.path(chartr("\\", "/", R.home()), "site-library")
```

Set a CRAN mirror

```
local({r <- getOption("repos")  
r["CRAN"] <- "http://my.local.cran"  
options(repos=r)})
```

Setting the location of your library

This will allow you to not have to install all the packages again with each R version update.

```
# library location  
.libPaths("c:/R_home/Rpackages/win")
```

Custom shortcuts or functions

Sometimes it is useful to have a shortcut for a long R expression. A common example of this setting an active binding to access the last top-level expression result without having to type out .Last.value:

```
makeActiveBinding(".", function(){.Last.value}, .GlobalEnv)
```

Because .Rprofile is just an R file, it can contain any arbitrary R code.

Pre-loading the most useful packages

This is bad practice and should generally be avoided because it separates package loading code from the scripts where those packages are actually used.

See Also

请参见help(Startup)了解所有不同的启动脚本及其他相关内容。特别是，可以加载两个系统范围的Profile文件。第一个R profile可能包含全局设置，另一个文件Profile.site可能包含系统管理员为所有用户制定的本地选项。这两个文件都位于R安装目录的\${RHOME}/etc目录中。该目录还包含全局文件Renviron和Renviron.site，这两个文件都可以通过用户主目录中的本地文件~/Renviron进行补充。

第76.2节：.Rprofile示例

启动

```
# 启动时加载库 setwidth - 自动设置宽度。  
.First <- function() {  
  library(setwidth)  
  # 如果是256色终端 - 使用库 colorout.  
  if (Sys.getenv("TERM") %in% c("xterm-256color", "screen-256color")) {  
    library("colorout")  
  }  
}
```

选项

```
# 选择默认的 CRAN 镜像用于安装包。  
options(repos=c(CRAN="https://cran.gis-lab.info/"))  
  
# 打印最多1000个元素。  
options(max.print=1000)  
  
# 不使用科学计数法。  
options(scipen=10)  
  
# 菜单中不显示图形。  
options(menu.graphics=FALSE)  
  
# 包名自动补全。  
utils::rc.settings(ipck=TRUE)
```

自定义函数

```
# 用于屏蔽已定义函数的隐形环境  
.env = new.env()  
  
# 退出R时不询问是否保存。  
.env$q <- function (save="no", ...) {  
  quit(save=save, ...)  
}  
  
# 附加环境以启用函数。  
attach(.env, warn.conflicts=FALSE)
```

See [help\(Startup\)](#) for all the different startup scripts, and further aspects. In particular, two system-wide Profile files can be loaded as well. The first, Rprofile, may contain global settings, the other file Profile.site may contain local choices the system administrator can make for all users. Both files are found in the \${RHOME}/etc directory of the R installation. This directory also contains global files Renviron and Renviron.site which both can be complemented with a local file ~/Renviron in the user's home directory.

Section 76.2: .Rprofile example

Startup

```
# Load library setwidth on start - to set the width automatically.  
.First <- function() {  
  library(setwidth)  
  # If 256 color terminal - use library colorout.  
  if (Sys.getenv("TERM") %in% c("xterm-256color", "screen-256color")) {  
    library("colorout")  
  }  
}
```

Options

```
# Select default CRAN mirror for package installation.  
options(repos=c(CRAN="https://cran.gis-lab.info/"))  
  
# Print maximum 1000 elements.  
options(max.print=1000)  
  
# No scientific notation.  
options(scipen=10)  
  
# No graphics in menus.  
options(menu.graphics=FALSE)  
  
# Auto-completion for package names.  
utils::rc.settings(ipck=TRUE)
```

Custom Functions

```
# Invisible environment to mask defined functions  
.env = new.env()  
  
# Quit R without asking to save.  
.env$q <- function (save="no", ...) {  
  quit(save=save, ...)  
}  
  
# Attach the environment to enable functions.  
attach(.env, warn.conflicts=FALSE)
```

第77章：dplyr

第77.1节：dplyr的单表动词

dplyr在R中引入了一种数据操作语法。它提供了一个一致的接口来处理数据，无论数据存储在哪里：data.frame、data.table，还是数据库。dplyr的关键部分是使用Rcpp编写的，这使得它在处理内存中数据时非常快速。

dplyr的理念是拥有小而专注的函数。五个简单的函数（`filter`、`arrange`、`SELECT`、`mutate`和`summarise`）可以用来揭示描述数据的新方法。当与`group_by`结合使用时，这些函数可以用来计算分组的汇总统计。

语法共性

所有这些函数都有类似的语法：

- 所有这些函数的第一个参数始终是数据框
- 列可以直接使用裸变量名引用（即不使用\$）
- 这些函数不会修改原始数据本身，即它们没有副作用。因此，结果应始终保存到一个对象中。

我们将使用内置的mtcars数据集来探索dplyr的单表动词。在将mtcars的类型转换为tbl_df之前（因为这样打印更整洁），我们使用ibble包中的rownames_to_column函数将数据集的rownames作为一列添加进去。

```
library(dplyr) # 本文档使用版本0.5.0编写

mtcars_tbl <- as_data_frame(tibble::rownames_to_column(mtcars, "cars"))

# 查看数据结构
head(mtcars_tbl)

# 一个6行12列的tibble
#   汽车 英里每加仑 气缸数 排量 马力 后桥比 重量 四分之一英里加速时间 发动机形状 变速箱类型 齿轮数 化油器数
#   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1 马自达 RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4
#2 马自达 RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4
#3 达特桑 710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1
#4 霍尼特 4 驱动 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1
#5 霍尼特 运动版 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2
#6 瓦里安特 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1
```

filter

`filter` 帮助筛选符合特定条件的行。第一个参数是data.frame的名称，第二个（及后续）参数是筛选数据的条件（这些条件应当计算为TRUE或FALSE）

筛选所有气缸数为4的汽车 - cyl :

```
filter(mtcars_tbl, cyl == 4)

# 一个 tibble: 11 x 12
#   汽车 英里每加仑 气缸数 排量 马力 后桥比 重量 四分之一英里加速时间 发动机形状 变速箱类型 齿轮数 化油器数
#   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1 达特桑 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1
#2 梅赛德斯 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2
```

Chapter 77: dplyr

Section 77.1: dplyr's single table verbs

`dplyr` introduces a grammar of data manipulation in R. It provides a consistent interface to work with data no matter where it is stored: data.frame, data.table, or a database. The key pieces of dplyr are written using Rcpp, which makes it very fast for working with in-memory data.

dplyr's philosophy is to have small functions that do one thing well. The five simple functions (`filter`, `arrange`, `SELECT`, `mutate`, and `summarise`) can be used to reveal new ways to describe data. When combined with `group_by`, these functions can be used to calculate group wise summary statistics.

Syntax commonalities

All these functions have a similar syntax:

- The first argument to all these functions is always a data frame
- Columns can be referred directly using bare variable names (i.e., without using \$)
- These functions do not modify the original data itself, i.e., they don't have side effects. Hence, the results should always be saved to an object.

We will use the built-in `mtcars` dataset to explore dplyr's single table verbs. Before converting the type of `mtcars` to `tbl_df` (since it makes printing cleaner), we add the `rownames` of the dataset as a column using `rownames_to_column` function from the `tidyverse` package.

```
library(dplyr) # This documentation was written using version 0.5.0
```

```
mtcars_tbl <- as_data_frame(tibble::rownames_to_column(mtcars, "cars"))

# examine the structure of data
head(mtcars_tbl)
```

```
# A tibble: 6 x 12
#   cars   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1 Mazda RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4
#2 Mazda RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4
#3 Datsun 710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1
#4 Hornet 4 Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1
#5 Hornet Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2
#6 Valiant 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1
```

filter

`filter` helps subset rows that match certain criteria. The first argument is the name of the `data.frame` and the second (and subsequent) arguments are the criteria that filter the data (these criteria should evaluate to either TRUE or FALSE)

Subset all cars that have 4 cylinders - cyl:

```
filter(mtcars_tbl, cyl == 4)
```

```
# A tibble: 11 x 12
#   cars   mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
#   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1 Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1
#2 Merc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2
```

```
#3 梅赛德斯 230 22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2
#4 菲亚特 128 32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1
#5 本田 思域 30.4 4 75.7 52 4.93 1.615 18.52 1 1 4 2
# ... 还有6行
```

我们可以传入多个用逗号分隔的条件。筛选气缸数为4或6的汽车 - cyl，且齿轮数为5 - gear：

```
filter(mtcars_tbl, cyl == 4 | cyl == 6, gear == 5)
```

```
# 一个 tibble: 3 x 12
#   汽车 英里每加仑 气缸数 排量 马力 后桥比 重量 四分之一英里加速时间 引擎形状 变速器类型 齿轮数 化油器数
#   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1 保时捷 914-2 26.0 4 120.3 91 4.43 2.140 16.7 0 1 5 2
#2 莲花 Europa 30.4 4 95.1 113 3.77 1.513 16.9 1 1 5 2
#3 法拉利 Dino 19.7 6 145.0 175 3.62 2.770 15.5 0 1 5 6
```

filter 根据条件选择行，若要按位置选择行，请使用 slice。slice 只接受两个参数：第一个是 data.frame，第二个是整数行值。

选择第6行到第9行：

```
slice(mtcars_tbl, 6:9)
```

```
# 一个包含4行12列的表格
#   汽车 英里每加仑 气缸数 排量 马力 后桥比 重量 四分之一英里加速时间 发动机形状 变速箱类型 齿轮数 化油器数
#   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1 Valiant 18.1 6 225.0 105 2.76 3.46 20.22 1 0 3 1
#2 Duster 360 14.3 8 360.0 245 3.21 3.57 15.84 0 0 3 4
#3 Merc 240D 24.4 4 146.7 62 3.69 3.19 20.00 1 0 4 2
#4 Merc 230 22.8 4 140.8 95 3.92 3.15 22.90 1 0 4 2
```

或者：

```
slice(mtcars_tbl, -c(1:5, 10:n()))
```

这将产生与 slice(mtcars_tbl, 6:9) 相同的输出

n() 表示当前组中的观测数量

arrange

arrange 用于按指定变量对数据进行排序。就像前面的动词（以及所有其他函数）一样 (dplyr)，第一个参数是一个 data.frame，后续参数用于对数据进行排序。如果传入多个变量，数据首先按第一个变量排序，然后按第二个变量排序，依此类推。

按马力 - hp 排序数据

```
arrange(mtcars_tbl, hp)
```

```
# 一个包含32行12列的tibble
#   汽车 英里每加仑 气缸数 排量 马力 后桥比 重量 1/4英里加速时间 引擎形状 变速箱类型 齿轮数 化油器数
#   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1 本田思域 30.4 4 75.7 52 4.93 1.615 18.52 1 1 4 2
#2 梅赛德斯240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2
#3 丰田卡罗拉 33.9 4 71.1 65 4.22 1.835 19.90 1 1 4 1
#4 菲亚特128 32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1
#5 菲亚特X1-9 27.3 4 79.0 66 4.08 1.935 18.90 1 1 4 1
```

```
#3 Merc 230 22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2
#4 Fiat 128 32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1
#5 Honda Civic 30.4 4 75.7 52 4.93 1.615 18.52 1 1 4 2
# ... with 6 more rows
```

We can pass multiple criteria separated by a comma. To subset the cars which have either 4 or 6 cylinders - cyl and have 5 gears - gear:

```
filter(mtcars_tbl, cyl == 4 | cyl == 6, gear == 5)
```

```
# A tibble: 3 x 12
#   cars mpg cyl disp hp drat wt qsec vs am gear carb
#   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1 Porsche 914-2 26.0 4 120.3 91 4.43 2.140 16.7 0 1 5 2
#2 Lotus Europa 30.4 4 95.1 113 3.77 1.513 16.9 1 1 5 2
#3 Ferrari Dino 19.7 6 145.0 175 3.62 2.770 15.5 0 1 5 6
```

filter selects rows based on criteria, to select rows by position, use slice. slice takes only 2 arguments: the first one is a data.frame and the second is integer row values.

To select rows 6 through 9:

```
slice(mtcars_tbl, 6:9)
```

```
# A tibble: 4 x 12
#   cars mpg cyl disp hp drat wt qsec vs am gear carb
#   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1 Valiant 18.1 6 225.0 105 2.76 3.46 20.22 1 0 3 1
#2 Duster 360 14.3 8 360.0 245 3.21 3.57 15.84 0 0 3 4
#3 Merc 240D 24.4 4 146.7 62 3.69 3.19 20.00 1 0 4 2
#4 Merc 230 22.8 4 140.8 95 3.92 3.15 22.90 1 0 4 2
```

Or:

```
slice(mtcars_tbl, -c(1:5, 10:n()))
```

This results in the same output as slice(mtcars_tbl, 6:9)

n() represents the number of observations in the current group

arrange

arrange 用于按指定变量对数据进行排序。就像前面的动词（以及所有其他函数）一样 (dplyr)，第一个参数是一个 data.frame，后续参数用于对数据进行排序。如果传入多个变量，数据首先按第一个变量排序，然后按第二个变量排序，依此类推。

To order the data by horsepower - hp

```
arrange(mtcars_tbl, hp)
```

```
# A tibble: 32 x 12
#   cars mpg cyl disp hp drat wt qsec vs am gear carb
#   <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1 Honda Civic 30.4 4 75.7 52 4.93 1.615 18.52 1 1 4 2
#2 Merc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2
#3 Toyota Corolla 33.9 4 71.1 65 4.22 1.835 19.90 1 1 4 1
#4 Fiat 128 32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1
#5 Fiat X1-9 27.3 4 79.0 66 4.08 1.935 18.90 1 1 4 1
```

```
#6 保时捷914-2 26.0    4 120.3   91 4.43 2.140 16.70    0   1   5   2
# ... 还有26行
```

按英里每加仑 - mpg降序排列数据，随后按气缸数 - cyl排序：

```
arrange(mtcars_tbl, desc(mpg), cyl)
```

```
# 一个包含32行12列的tibble
#       汽车 英里每加仑 气缸数 排量 马力 后桥比 重量 1/4英里加速时间 发动机形状 变速箱类型 齿轮数 化油器数
# <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1 丰田卡罗拉 33.9    4 71.1   65 4.22 1.835 19.90    1   1   4   1
#2 菲亚特128 32.4    4 78.7   66 4.08 2.200 19.47    1   1   4   1
#3 本田思域 30.4    4 75.7   52 4.93 1.615 18.52    1   1   4   2
#4 莲花欧洲 30.4    4 95.1   113 3.77 1.513 16.90   1   1   5   2
#5 菲亚特X1-9 27.3    4 79.0   66 4.08 1.935 18.90   1   1   4   1
#6 保时捷914-2 26.0    4 120.3   91 4.43 2.140 16.70    0   1   5   2
# ... 还有26行
```

select

SELECT用于选择变量的子集。要仅从mtcars_tbl中选择mpg、disp、wt、qsec和vs：

```
SELECT(mtcars_tbl, mpg, disp, wt, qsec, vs)
```

```
# 一个 tibble: 32 x 5
#       mpg   disp   wt   qsec   vs
# <dbl> <dbl> <dbl> <dbl> <dbl>
#1 21.0 160.0 2.620 16.46    0
#2 21.0 160.0 2.875 17.02    0
#3 22.8 108.0 2.320 18.61    1
#4 21.4 258.0 3.215 19.44    1
#5 18.7 360.0 3.440 17.02    0
#6 18.1 225.0 3.460 20.22    1
# ... 还有 26 行
```

: 符号可用于选择连续的列。要选择从 cars 到 disp 以及从 vs 到

carb 的列：

```
SELECT(mtcars_tbl, cars:disp, vs:carb)
```

```
# 一个 tibble: 32 x 8
#       cars   mpg   cyl   disp   vs   am   gear   carb
# <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1 Mazda RX4 21.0   6 160.0    0   1   4   4
#2 Mazda RX4 Wag 21.0   6 160.0    0   1   4   4
#3 Datsun 710 22.8   4 108.0    1   1   4   1
#4 Hornet 4 Drive 21.4   6 258.0    1   0   3   1
#5 Hornet Sportabout 18.7   8 360.0    0   0   3   2
#6 Valiant 18.1   6 225.0    1   0   3   1
# ... 还有 26 行
```

或者 SELECT(mtcars_tbl, -(hp:qsec))

对于包含多列的数据集，通过名称选择多列可能很繁琐。为了简化操作，有许多辅助函数（例如 starts_with()，ends_with()，contains()，matches()，num_range()，one_of() 和 everything()）可以在 **SELECT** 中使用。想了解更多用法，请参见 ?select_helpers 和 ?select。

注意：在 **SELECT()** 中直接引用列时，我们使用裸列名，但在

```
#6 Porsche 914-2 26.0    4 120.3   91 4.43 2.140 16.70    0   1   5   2
# ... with 26 more rows
```

To arrange the data by miles per gallon - mpg in descending order, followed by number of cylinders - cyl:

```
arrange(mtcars_tbl, desc(mpg), cyl)
```

```
# A tibble: 32 x 12
#       cars   mpg   cyl   disp   hp   drat   wt   qsec   vs   am   gear   carb
# <chr> <dbl> <dbl>
#1 Toyota Corolla 33.9    4 71.1   65 4.22 1.835 19.90    1   1   4   1
#2 Fiat 128 32.4    4 78.7   66 4.08 2.200 19.47    1   1   4   1
#3 Honda Civic 30.4    4 75.7   52 4.93 1.615 18.52    1   1   4   2
#4 Lotus Europa 30.4    4 95.1   113 3.77 1.513 16.90    1   1   5   2
#5 Fiat X1-9 27.3    4 79.0   66 4.08 1.935 18.90    1   1   4   1
#6 Porsche 914-2 26.0    4 120.3   91 4.43 2.140 16.70    0   1   5   2
# ... with 26 more rows
```

select

SELECT 是用于选择只有子集的变量。要选择只有 mpg、disp、wt、qsec 和 vs 从 mtcars_tbl:

```
SELECT(mtcars_tbl, mpg, disp, wt, qsec, vs)
```

```
# 一个 tibble: 32 x 5
#       mpg   disp   wt   qsec   vs
# <dbl> <dbl> <dbl> <dbl> <dbl>
#1 21.0 160.0 2.620 16.46    0
#2 21.0 160.0 2.875 17.02    0
#3 22.8 108.0 2.320 18.61    1
#4 21.4 258.0 3.215 19.44    1
#5 18.7 360.0 3.440 17.02    0
#6 18.1 225.0 3.460 20.22    1
# ... WITH 26 more ROWS
```

: notation can be used to select consecutive columns. To select columns from cars through disp and vs through carb:

```
SELECT(mtcars_tbl, cars:disp, vs:carb)
```

```
# 一个 tibble: 32 x 8
#       cars   mpg   cyl   disp   vs   am   gear   carb
# <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1 Mazda RX4 21.0   6 160.0    0   1   4   4
#2 Mazda RX4 Wag 21.0   6 160.0    0   1   4   4
#3 Datsun 710 22.8   4 108.0    1   1   4   1
#4 Hornet 4 Drive 21.4   6 258.0    1   0   3   1
#5 Hornet Sportabout 18.7   8 360.0    0   0   3   2
#6 Valiant 18.1   6 225.0    1   0   3   1
# ... WITH 26 more ROWS
```

或者 **SELECT(mtcars_tbl, -(hp:qsec))**

For datasets that contain several columns, it can be tedious to select several columns by name. To make life easier, there are a number of helper functions (such as starts_with(), ends_with(), contains(), matches(), num_range(), one_of(), and everything()) that can be used in **SELECT**. To learn more about how to use them, see ?select_helpers and ?select.

Note: While referring to columns directly in **SELECT()**, we use bare column names, but quotes should be used while

引用辅助函数中的列。

选择时重命名列：

```
SELECT(mtcars_tbl, cylinders = cyl, displacement = disp)
```

```
# 一个 tibble: 32 行 2 列
#   cylinders displacement
#       <dbl>        <dbl>
#1       6        160.0
#2       6        160.0
#3       4        108.0
#4       6        258.0
#5       8        360.0
#6       6        225.0
# ... 还有 26 行
```

如预期，这会删除所有其他变量。

要重命名列而不删除其他变量，请使用 `rename`：

```
rename(mtcars_tbl, cylinders = cyl, displacement = disp)
```

```
# 一个 tibble: 32 行 12 列
#   cars     mpg cylinders displacement      hp drat      wt qsec vs
#   <chr>    <dbl>        <dbl>        <dbl> <dbl> <dbl> <dbl> <dbl>
#1 Mazda RX4 21.0        6        160.0    110 3.90 2.620 16.46 0
#2 Mazda RX4 Wag 21.0      6        160.0    110 3.90 2.875 17.02 0
#3 Datsun 710 22.8        4        108.0    93 3.85 2.320 18.61 1
#4 Hornet 4 Drive 21.4      6        258.0    110 3.08 3.215 19.44 1
#5 Hornet Sportabout 18.7      8        360.0    175 3.15 3.440 17.02 0
#6 Valiant 18.1        6        225.0    105 2.76 3.460 20.22 1
# ... 还有 26 行, 和 3 个变量: am <dbl>, gear <dbl>, carb <dbl>
```

mutate

`mutate` 可用于向数据中添加新列。像 `dplyr` 中的所有其他函数一样，`mutate` 不会将新创建的列添加到原始数据中。新列会被添加到 `data.frame` 的末尾。

```
mutate(mtcars_tbl, weight_ton = wt/2, weight_pounds = weight_ton * 2000)
```

```
# 一个包含32行14列的tibble
#   汽车 英里每加仑 气缸数 排量 马力 后桥比 重量 1/4英里加速时间 发动机形状 变速箱类型 齿轮数 化油器数 重量_吨
#   <chr> <dbl> <dbl>
#1 马自达 RX4 21.0 6 160.0 110 3.90 2.620 16.46 0 1 4 4 1.3100
#2 马自达 RX4 旅行版 21.0 6 160.0 110 3.90 2.875 17.02 0 1 4 4 1.4375
#3 达特桑 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1 1.1600
#4 霍尼特 4 驱 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1 1.6075
#5 Hornet Sportabout 18.7 8 360.0 175 3.15 3.440 17.02 0 0 3 2 1.7200
#6 Valiant 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1 1.7300
# ... 还有26行
```

referring to columns in helper functions.

To rename columns while selecting:

```
SELECT(mtcars_tbl, cylinders = cyl, displacement = disp)
```

```
# A tibble: 32 x 2
#   cylinders displacement
#       <dbl>        <dbl>
#1       6        160.0
#2       6        160.0
#3       4        108.0
#4       6        258.0
#5       8        360.0
#6       6        225.0
# ... WITH 26 more ROWS
```

As expected, this drops all other variables.

To rename columns without dropping other variables, use `rename`:

```
rename(mtcars_tbl, cylinders = cyl, displacement = disp)
```

```
# A tibble: 32 x 12
#   cars     mpg cylinders displacement      hp drat      wt qsec vs
#   <chr>    <dbl>        <dbl>        <dbl> <dbl> <dbl> <dbl> <dbl>
#1 Mazda RX4 21.0        6        160.0    110 3.90 2.620 16.46 0
#2 Mazda RX4 Wag 21.0      6        160.0    110 3.90 2.875 17.02 0
#3 Datsun 710 22.8        4        108.0    93 3.85 2.320 18.61 1
#4 Hornet 4 Drive 21.4      6        258.0    110 3.08 3.215 19.44 1
#5 Hornet Sportabout 18.7      8        360.0    175 3.15 3.440 17.02 0
#6 Valiant 18.1        6        225.0    105 2.76 3.460 20.22 1
# ... with 26 more rows, and 3 more variables: am <dbl>, gear <dbl>, carb <dbl>
```

mutate

`mutate` can be used to add new columns to the data. Like all other functions in `dplyr`, `mutate` doesn't add the newly created columns to the original data. Columns are added at the end of the `data.frame`.

```
mutate(mtcars_tbl, weight_ton = wt/2, weight_pounds = weight_ton * 2000)
```

```
# A tibble: 32 x 14
#   weight_pounds
#   <dbl>
#1 1.3100
#2 1.4375
#3 1.1600
#4 1.6075
#5 1.7200
#6 1.7300
# ... with 26 more rows
```

注意 在创建weight_ton时使用了weight_pounds。与基础R不同，mutate允许我们引用刚创建的列以用于后续操作。

若只保留新创建的列，请使用transmute代替mutate：

```
transmute(mtcars_tbl, weight_ton = wt/2, weight_pounds = weight_ton * 2000)
```

```
# 一个32行2列的tibble  
#   weight_ton    weight_pounds  
#   <dbl>        <dbl>  
#1  1.3100      2620  
#2  1.4375      2875  
#3  1.1600      2320  
#4  1.6075      3215  
#5  1.7200      3440  
#6  1.7300      3460  
# ... 还有26行
```

summarise

summarise 通过将多个值合并为单个值来计算变量的汇总统计量。它可以计算多种统计量，并且我们可以在同一语句中为这些汇总列命名。

计算数据集中所有汽车的mpg和disp的平均值和标准差：

```
summarise(mtcars_tbl, mean_mpg = mean(mpg), sd_mpg = sd(mpg),  
          mean_disp = mean(disp), sd_disp = sd(disp))
```

```
# 一个 tibble: 1 x 4  
#   mean_mpg    sd_mpg  mean_disp    sd_disp  
#   <dbl>       <dbl>     <dbl>       <dbl>  
#1 20.09062  6.026948 230.7219 123.9387
```

group_by

group_by 可用于对数据执行分组操作。当上述定义的动词应用于该分组数据时，它们会自动分别应用于每个组。

按cyl计算mpg的平均值和标准差：

```
by_cyl <- group_by(mtcars_tbl, cyl)  
summarise(by_cyl, mean_mpg = mean(mpg), sd_mpg = sd(mpg))
```

```
# 一个 tibble: 3 x 3  
#   cyl  mean_mpg  sd_mpg  
#   <dbl>     <dbl>     <dbl>  
#1  4    26.66364 4.509828  
#2  6    19.74286 1.453567  
#3  8    15.10000 2.560048
```

综合应用

我们从cars中选择列 hp和 gear，按 cyl排序，并按 mpg从高到低排序，按 gear分组数据，最后只筛选出 mpg > 20且 hp > 75的汽车子集

```
selected <- SELECT(mtcars_tbl, cars:hp, gear)  
ordered <- arrange(selected, cyl, DESC(mpg))  
by_cyl <- group_by(ordered, gear)  
FILTER(by_cyl, mpg > 20, hp > 75)
```

Note the use of weight_ton while creating weight_pounds. Unlike base R, mutate allows us to refer to columns that we just created to be used for a subsequent operation.

To retain only the newly created columns, use transmute instead of mutate:

```
transmute(mtcars_tbl, weight_ton = wt/2, weight_pounds = weight_ton * 2000)
```

```
# A tibble: 32 x 2  
#   weight_ton    weight_pounds  
#   <dbl>        <dbl>  
#1  1.3100      2620  
#2  1.4375      2875  
#3  1.1600      2320  
#4  1.6075      3215  
#5  1.7200      3440  
#6  1.7300      3460  
# ... with 26 more rows
```

summarise

summarise 计算变量的汇总统计量，将多个值合并为单个值。它可以计算多种统计量，并且我们可以在同一语句中为这些汇总列命名。

To calculate the *mean* and *standard deviation* of mpg and disp of all cars in the dataset:

```
summarise(mtcars_tbl, mean_mpg = mean(mpg), sd_mpg = sd(mpg),  
          mean_disp = mean(disp), sd_disp = sd(disp))
```

```
# A tibble: 1 x 4  
#   mean_mpg    sd_mpg  mean_disp    sd_disp  
#   <dbl>       <dbl>     <dbl>       <dbl>  
#1 20.09062  6.026948 230.7219 123.9387
```

group_by

group_by 可以在分组数据上执行操作。当上述定义的动词应用于该分组数据时，它们会自动分别应用于每个组。

To find *mean* and *sd* of mpg by cyl:

```
by_cyl <- group_by(mtcars_tbl, cyl)  
summarise(by_cyl, mean_mpg = mean(mpg), sd_mpg = sd(mpg))
```

```
# 一个 tibble: 3 x 3  
#   cyl  mean_mpg  sd_mpg  
#   <dbl>     <dbl>     <dbl>  
#1  4    26.66364 4.509828  
#2  6    19.74286 1.453567  
#3  8    15.10000 2.560048
```

Putting it all together

We select columns from cars through hp and gear, order the rows by cyl and from highest to lowest mpg, group the data by gear, and finally subset only those cars have mpg > 20 and hp > 75

```
selected <- SELECT(mtcars_tbl, cars:hp, gear)  
ordered <- arrange(selected, cyl, DESC(mpg))  
by_cyl <- group_by(ordered, gear)  
FILTER(by_cyl, mpg > 20, hp > 75)
```

SOURCE: LOCAL DATA frame [9 x 6]

Groups: gear [3]

```
#       cars  mpg cyl disp hp gear
#       <chr> <dbl> <dbl> <dbl> <dbl> <dbl>
#1 Lotus Europa 30.4   4 95.1 113   5
#2 Porsche 914-2 26.0   4 120.3 91    5
#3 Datsun 710 22.8   4 108.0 93    4
#4 Merc 230 22.8   4 140.8 95    4
#5 Toyota Corona 21.5   4 120.1 97    3
# ... WITH 4 more ROWS
```

也许我们对中间结果不感兴趣，我们可以通过嵌套函数调用来实现与上述相同的结果：

```
filter(
group_by(
arrange(
select(
  mtcars_tbl, cars:hp
), cyl, desc(mpg)
), 气缸数
), 英里每加仑 > 20, 马力 > 75
)
```

这可能有点难以阅读。因此，dplyr 操作可以使用管道符 `%>%` 链接。上述代码相当于：

```
mtcars_tbl %>%
select(汽车:马力) %>%
arrange(气缸数, 降序(英里每加仑)) %>%
group_by(气缸数) %>%
filter(英里每加仑 > 20, 马力 > 75)
```

汇总多列

dplyr 提供了 `summarise_all()` 用于对所有（非分组）列应用函数。

查找每列的不同值数量：

```
mtcars_tbl %>%
summarise_all(n_distinct)

# 一个 tibble: 1 x 12
#   汽车数 英里每加仑 气缸数 排量 马力 后桥比 重量 四分之一英里时间 引擎形状 变速器类型 齿轮数 化油器数
#   <int> <int>
#1  32    25     3    27    22    22    29    30     2     2     3     6
```

要按气缸数查找每列的不同值数量：

```
mtcars_tbl %>%
group_by(气缸数) %>%
summarise_all(n_distinct)

# 一个 tibble: 3 x 12
#   气缸数 汽车数 英里每加仑 排量 马力 后桥比 重量 四分之一英里时间 引擎形状 变速器类型 齿轮数 化油器数
#   <dbl> <int> <int> <int> <int> <int> <int> <int> <int> <int> <int>
#1     4    11     9    11    10    10    11    11     2     2     3     2
#2     6     7     6     5     4     5     6     7     2     2     3     3
```

SOURCE: LOCAL DATA frame [9 x 6]

Groups: gear [3]

```
#       cars  mpg cyl disp hp gear
#       <chr> <dbl> <dbl> <dbl> <dbl> <dbl>
#1 Lotus Europa 30.4   4 95.1 113   5
#2 Porsche 914-2 26.0   4 120.3 91    5
#3 Datsun 710 22.8   4 108.0 93    4
#4 Merc 230 22.8   4 140.8 95    4
#5 Toyota Corona 21.5   4 120.1 97    3
# ... WITH 4 more ROWS
```

Maybe we are not interested the intermediate results, we can achieve the same result as above by wrapping the function calls:

```
filter(
group_by(
arrange(
select(
  mtcars_tbl, cars:hp
), cyl, desc(mpg)
), cyl
), mpg > 20, hp > 75
)
```

This can be a little difficult to read. So, dplyr operations can be chained using the pipe `%>%` operator. The above code translates to:

```
mtcars_tbl %>%
select(cars:hp) %>%
arrange(cyl, desc(mpg)) %>%
group_by(cyl) %>%
filter(mpg > 20, hp > 75)
```

summarise multiple columns

dplyr provides `summarise_all()` to apply functions to all (non-grouping) columns.

To find the number of distinct values for each column:

```
mtcars_tbl %>%
summarise_all(n_distinct)

# A tibble: 1 x 12
#   cars  mpg cyl disp hp drat  wt  qsec vs am gear carb
#   <dbl> <dbl>
#1  32    25     3    27    22    22    29    30     2     2     3     6
```

To find the number of distinct values for each column by cyl:

```
mtcars_tbl %>%
group_by(cyl) %>%
summarise_all(n_distinct)

# A tibble: 3 x 12
#   cyl cars  mpg cyl disp hp drat  wt  qsec vs am gear carb
#   <dbl> <int> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
#1     4    11     9    11    10    10    11    11     2     2     3     2
#2     6     7     6     5     4     5     6     7     2     2     3     3
```

```
#3 8 14 12 11 9 11 13 14 1 2 2 4
```

注意，我们只需添加group_by语句，其余代码保持不变。输出现在包含三行——每个唯一的气缸数对应一行。

要对特定的多个列进行summarise，使用 summarise_at

```
mtcars_tbl %>%
  group_by(气缸数) %>%
  summarise_at(c("mpg", "disp", "hp"), mean)

# 一个 tibble: 3 x 4
#   气缸数    英里每加仑    排量    马力
#   <dbl>      <dbl>     <dbl>     <dbl>
#1     4 26.66364 105.1364 82.63636
#2     6 19.74286 183.3143 122.28571
#3     8 15.10000 353.1000 209.21429
```

辅助函数 (?select_helpers) 可以用来代替列名以选择特定列

要应用多个函数，可以将函数名作为字符向量传入：

```
mtcars_tbl %>%
  group_by(cyl) %>%
  summarise_at(c("mpg", "disp", "hp"),
               c("mean", "sd"))
```

或者将它们包裹在funs中：

```
mtcars_tbl %>%
  group_by(cyl) %>%
  summarise_at(c("mpg", "disp", "hp"),
               funs(mean, sd))

# 一个 tibble: 3 行 7 列
#   cyl mpg_mean disp_mean hp_mean mpg_sd disp_sd hp_sd
#   <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
#1     4 26.66364 105.1364 82.63636 4.509828 26.87159 20.93453
#2     6 19.74286 183.3143 122.28571 1.453567 41.56246 24.26049
#3     8 15.10000 353.1000 209.21429 2.560048 67.77132 50.97689
```

列名现在会附加函数名以保持区分。若要更改此行为，传入要附加的名称：

```
mtcars_tbl %>%
  group_by(cyl) %>%
  summarise_at(c("mpg", "disp", "hp"),
               c(Mean = "mean", SD = "sd"))

mtcars_tbl %>%
  group_by(气缸数) %>%
  summarise_at(c("mpg", "disp", "hp"),
               funs(Mean = mean, SD = sd))
```

```
# 一个 tibble: 3 行 7 列
#   气缸数 mpg_均值 disp_均值 马力_均值 mpg_标准差 disp_标准差 马力_标准差
#   <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
#1     4 26.66364 105.1364 82.63636 4.509828 26.87159 20.93453
```

```
#3 8 14 12 11 9 11 13 14 1 2 2 4
```

Note that we just had to add the group_by statement and the rest of the code is the same. The output now consists of three rows - one for each unique value of cyl.

To summarise specific multiple columns, use summarise_at

```
mtcars_tbl %>%
  group_by(cyl) %>%
  summarise_at(c("mpg", "disp", "hp"), mean)

# A tibble: 3 x 4
#   cyl    mpg    disp    hp
#   <dbl>  <dbl>  <dbl>  <dbl>
#1     4 26.66364 105.1364 82.63636
#2     6 19.74286 183.3143 122.28571
#3     8 15.10000 353.1000 209.21429
```

helper functions (?select_helpers) can be used in place of column names to select specific columns

To apply multiple functions, either pass the function names as a character vector:

```
mtcars_tbl %>%
  group_by(cyl) %>%
  summarise_at(c("mpg", "disp", "hp"),
               c("mean", "sd"))
```

or wrap them inside funs:

```
mtcars_tbl %>%
  group_by(cyl) %>%
  summarise_at(c("mpg", "disp", "hp"),
               funs(mean, sd))

# A tibble: 3 x 7
#   cyl mpg_mean disp_mean hp_mean mpg_sd disp_sd hp_sd
#   <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
#1     4 26.66364 105.1364 82.63636 4.509828 26.87159 20.93453
#2     6 19.74286 183.3143 122.28571 1.453567 41.56246 24.26049
#3     8 15.10000 353.1000 209.21429 2.560048 67.77132 50.97689
```

Column names are now be appended with function names to keep them distinct. In order to change this, pass the name to be appended with the function:

```
mtcars_tbl %>%
  group_by(cyl) %>%
  summarise_at(c("mpg", "disp", "hp"),
               c(Mean = "mean", SD = "sd"))

mtcars_tbl %>%
  group_by(cyl) %>%
  summarise_at(c("mpg", "disp", "hp"),
               funs(Mean = mean, SD = sd))
```

```
# A tibble: 3 x 7
#   cyl mpg_Mean disp_Mean hp_Mean mpg_SD disp_SD hp_SD
#   <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>
#1     4 26.66364 105.1364 82.63636 4.509828 26.87159 20.93453
```

```
#2 6 19.74286 183.3143 122.28571 1.453567 41.56246 24.26049  
#3 8 15.10000 353.1000 209.21429 2.560048 67.77132 50.97689
```

要有条件地选择列，使用 `summarise_if`：

按 `cyl` 分组，取所有 `numeric` 列的 `mean`：

```
mtcars_tbl %>%  
group_by(cyl) %>%  
summarise_if(is.numeric, mean)  
  
# 一个 tibble: 3 x 11  
#   气缸数   mpg   排量   马力   后桥比   重量   1/4英里加速时间  
#   <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>  
#1     4 26.66364 105.1364 82.63636 4.070909 2.285727 19.13727  
#2     6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714  
#3     8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214  
# ... 还有4个变量: vs <dbl>, am <dbl>, gear <dbl>,  
#   carb <dbl>
```

但是，有些变量是离散的，对这些变量取 `mean` 没有意义。

按 `cyl` 分组，只对连续变量取 `mean`：

```
mtcars_tbl %>%  
group_by(气缸数) %>%  
summarise_if(function(x) is.numeric(x) & n_distinct(x) > 6, mean)  
  
# 一个 tibble: 3 x 7  
#   气缸数   mpg   排量   马力   后桥比   重量   1/4英里加速时间  
#   <dbl>    <dbl>    <dbl>    <dbl>    <dbl>    <dbl>  
#1     4 26.66364 105.1364 82.63636 4.070909 2.285727 19.13727  
#2     6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714  
#3     8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214
```

第77.2节：使用 `%>%`（管道）操作符进行聚合

管道（`%>%`）操作符可以与 `dplyr` 函数结合使用。在此示例中，我们使用 `mtcars` 数据集（有关更多信息，请参见 `help("mtcars")`）来展示如何汇总数据框，并通过应用函数的结果向数据中添加变量。

```
library(dplyr)  
library(magrittr)  
df <- mtcars  
df$cars <- rownames(df) # 仅将汽车名称添加到df中  
df <- df[, c(ncol(df), 1:(ncol(df)-1))] # 并将名称放在第一列
```

1. 汇总数据

为了计算统计量，我们使用 `summarize` 和相应的函数。在本例中，使用 `n()` 来计算案例数量。

```
df %>%  
summarize(count=n(), mean_mpg = mean(mpg, na.rm = TRUE),  
min_weight = min(wt), max_weight = max(wt))  
  
# count mean_mpg min_weight max_weight
```

```
#2 6 19.74286 183.3143 122.28571 1.453567 41.56246 24.26049  
#3 8 15.10000 353.1000 209.21429 2.560048 67.77132 50.97689
```

To select columns conditionally, use `summarise_if`:

Take the `mean` of all columns that are `numeric` grouped by `cyl`:

```
mtcars_tbl %>%  
group_by(cyl) %>%  
summarise_if(is.numeric, mean)  
  
# A tibble: 3 x 11  
#   cyl   mpg   disp   hp   drat   wt   qsec  
#   <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  
#1     4 26.66364 105.1364 82.63636 4.070909 2.285727 19.13727  
#2     6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714  
#3     8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214  
# ... with 4 more variables: vs <dbl>, am <dbl>, gear <dbl>,  
#   carb <dbl>
```

However, some variables are discrete, and `mean` of these variables doesn't make sense.

To take the `mean` of only continuous variables by `cyl`:

```
mtcars_tbl %>%  
group_by(cyl) %>%  
summarise_if(function(x) is.numeric(x) & n_distinct(x) > 6, mean)  
  
# A tibble: 3 x 7  
#   cyl   mpg   disp   hp   drat   wt   qsec  
#   <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  <dbl>  
#1     4 26.66364 105.1364 82.63636 4.070909 2.285727 19.13727  
#2     6 19.74286 183.3143 122.28571 3.585714 3.117143 17.97714  
#3     8 15.10000 353.1000 209.21429 3.229286 3.999214 16.77214
```

Section 77.2: Aggregating with `%>%` (pipe) operator

The pipe (`%>%`) operator could be used in combination with `dplyr` functions. In this example we use the `mtcars` dataset (see `help("mtcars")` for more information) to show how to summarize a data frame, and to add variables to the data with the result of the application of a function.

```
library(dplyr)  
library(magrittr)  
df <- mtcars  
df$cars <- rownames(df) # just add the cars names to the df  
df <- df[, c(ncol(df), 1:(ncol(df)-1))] # and place the names in the first column
```

1. Summarize the data

To compute statistics we use `summarize` and the appropriate functions. In this case `n()` is used for counting the number of cases.

```
df %>%  
summarize(count=n(), mean_mpg = mean(mpg, na.rm = TRUE),  
min_weight = min(wt), max_weight = max(wt))  
  
# count mean_mpg min_weight max_weight
```

```
#1 32 20.09062 1.513 5.424
```

2. 按组计算统计量

可以按数据的分组计算统计量。本例中按气缸数和前进档数分组计算

```
df %>%
group_by(气缸数, 变速器齿轮数) %>%
summarize(计数=n(), 平均每加仑英里数 = mean(每加仑英里数, na.rm = TRUE),
最小重量 = min(重量), 最大重量 = max(重量))
```

```
# 来源: 本地数据框 [8 x 6]
# 分组: 气缸数 [?]
#
#   气缸数 变速器齿轮数 计数 平均每加仑英里数 最小重量 最大重量
#   <dbl> <dbl> <int> <dbl> <dbl>
#1     4      3       1  21.500  2.465
#2     4      4       8  26.925  1.615
#3     4      5       2  28.200  1.513
#4     6      3       2  19.750  3.215
#5     6      4       4  19.750  2.620
#6     6      5       1  19.700  2.770
#7     8      3      12  15.050  3.435
#8     8      5       2  15.400  3.170
```

```
#1 32 20.09062 1.513 5.424
```

2. Compute statistics by group

It is possible to compute the statistics by groups of the data. In this case by *Number of cylinders* and *Number of forward gears*

```
df %>%
group_by(cyl, gear) %>%
summarize(count=n(), mean_mpg = mean(mpg, na.rm = TRUE),
min_weight = min(wt), max_weight = max(wt))
```

```
# Source: local data frame [8 x 6]
# Groups: cyl [?]
#
#   cyl  gear count mean_mpg min_weight max_weight
#   <dbl> <dbl> <int> <dbl> <dbl>
#1     4      3       1  21.500  2.465
#2     4      4       8  26.925  1.615
#3     4      5       2  28.200  1.513
#4     6      3       2  19.750  3.215
#5     6      4       4  19.750  2.620
#6     6      5       1  19.700  2.770
#7     8      3      12  15.050  3.435
#8     8      5       2  15.400  3.170
```

第77.3节：子集观察（行）

dplyr::filter() - 选择满足逻辑条件的数据框中的行子集：

```
dplyr::filter(鸢尾花数据集, 莺片长度>7)
#   莺片长度 莺片宽度 花瓣长度 花瓣宽度 物种
# 1     7.1      3.0      5.9      2.1 维吉尼卡
# 2     7.6      3.0      6.6      2.1 维吉尼卡
# 3     7.3      2.9      6.3      1.8 维吉尼卡
# 4     7.2      3.6      6.1      2.5 维吉尼卡
# 5     7.7      3.8      6.7      2.2 维吉尼卡
# 6     7.7      2.6      6.9      2.3 维吉尼卡
# 7     7.7      2.8      6.7      2.0 维吉尼卡
# 8     7.2      3.2      6.0      1.8 维吉尼卡
# 9     7.2      3.0      5.8      1.6 维吉尼卡
# 10    7.4      2.8      6.1      1.9 维吉尼卡
# 11    7.9      3.8      6.4      2.0 维吉尼卡
# 12    7.7      3.0      6.1      2.3 维吉尼卡
```

dplyr::distinct() - 删除重复行：

```
distinct(鸢尾花数据集, 莺片长度, .keep_all = TRUE)
#   莺片长度 莺片宽度 花瓣长度 花瓣宽度 物种
# 1     5.1      3.5      1.4      0.2 山鸢尾
# 2     4.9      3.0      1.4      0.2 山鸢尾
# 3     4.7      3.2      1.3      0.2 山鸢尾
# 4     4.6      3.1      1.5      0.2 山鸢尾
# 5     5.0      3.6      1.4      0.2 山鸢尾
# 6     5.4      3.9      1.7      0.4 山鸢尾
# 7     4.4      2.9      1.4      0.2 山鸢尾
# 8     4.8      3.4      1.6      0.2 山鸢尾
# 9     4.3      3.0      1.1      0.1 山鸢尾
# 10    5.8      4.0      1.2      0.2 山鸢尾
# 11    5.7      4.4      1.5      0.4 山鸢尾
# 12    5.2      3.5      1.5      0.2 山鸢尾
```

Section 77.3: Subset Observation (Rows)

dplyr::filter() - Select a subset of rows in a data frame that meet a logical criteria:

```
dplyr::filter(iris, Sepal.Length>7)
#   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
# 1          7.1        3.0        5.9        2.1 virginica
# 2          7.6        3.0        6.6        2.1 virginica
# 3          7.3        2.9        6.3        1.8 virginica
# 4          7.2        3.6        6.1        2.5 virginica
# 5          7.7        3.8        6.7        2.2 virginica
# 6          7.7        2.6        6.9        2.3 virginica
# 7          7.7        2.8        6.7        2.0 virginica
# 8          7.2        3.2        6.0        1.8 virginica
# 9          7.2        3.0        5.8        1.6 virginica
# 10         7.4        2.8        6.1        1.9 virginica
# 11         7.9        3.8        6.4        2.0 virginica
# 12         7.7        3.0        6.1        2.3 virginica
```

dplyr::distinct() - Remove duplicate rows:

```
distinct(iris, Sepal.Length, .keep_all = TRUE)
#   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
# 1          5.1        3.5        1.4        0.2 setosa
# 2          4.9        3.0        1.4        0.2 setosa
# 3          4.7        3.2        1.3        0.2 setosa
# 4          4.6        3.1        1.5        0.2 setosa
# 5          5.0        3.6        1.4        0.2 setosa
# 6          5.4        3.9        1.7        0.4 setosa
# 7          4.4        2.9        1.4        0.2 setosa
# 8          4.8        3.4        1.6        0.2 setosa
# 9          4.3        3.0        1.1        0.1 setosa
# 10         5.8        4.0        1.2        0.2 setosa
# 11         5.7        4.4        1.5        0.4 setosa
# 12         5.2        3.5        1.5        0.2 setosa
```

```

# 13 5.5 4.2 1.4 0.2 山鸢尾
# 14 4.5 2.3 1.3 0.3 山鸢尾
# 15 5.3 3.7 1.5 0.2 山鸢尾
# 16 7.0 3.2 4.7 1.4 变色鸢尾
# 17 6.4 3.2 4.5 1.5 变色鸢尾
# 18 6.9 3.1 4.9 1.5 变色鸢尾
# 19 6.5 2.8 4.6 1.5 变色鸢尾
# 20 6.3 3.3 4.7 1.6 变色鸢尾
# 21 6.6 2.9 4.6 1.3 变色鸢尾
# 22 5.9 3.0 4.2 1.5 变色鸢尾
# 23 6.0 2.2 4.0 1.0 变色鸢尾
# 24 6.1 2.9 4.7 1.4 变色鸢尾
# 25 5.6 2.9 3.6 1.3 变色鸢尾
# 26 6.7 3.1 4.4 1.4 变色鸢尾
# 27 6.2 2.2 4.5 1.5 变色鸢尾
# 28 6.8 2.8 4.8 1.4 变色鸢尾
# 29 7.1 3.0 5.9 2.1 维吉尼卡
# 30 7.6 3.0 6.6 2.1 维吉尼卡
# 31 7.3 2.9 6.3 1.8 维吉尼卡
# 32 7.2 3.6 6.1 2.5 维吉尼卡
# 33 7.7 3.8 6.7 2.2 维吉尼卡
# 34 7.4 2.8 6.1 1.9 维吉尼卡
# 35 7.9 3.8 6.4 2.0 维吉尼卡

```

第77.4节：dplyr中NSE和字符串变量的示例

dplyr 使用非标准求值 (NSE) , 这就是为什么我们通常可以直接使用变量名而不加引号的原因。但是, 有时在数据管道中, 我们需要从其他来源获取变量名, 例如 Shiny选择框。对于像SELECT这样的函数, 我们可以直接使用select_来使用字符串变量进行选择

```

variable1 <- "Sepal.Length"
variable2 <- "Sepal.Width"
iris %>%
  select_(variable1, variable2) %>%
  head(n=5)
# Sepal.Length Sepal.Width
# 1 5.1 3.5
# 2 4.9 3.0
# 3 4.7 3.2
# 4 4.6 3.1
# 5 5.0 3.6

```

但是如果我想使用其他功能, 比如summarize或filter, 我们需要使用lazyeval

包中的interp函数

```

variable1 <- "Sepal.Length"
variable2 <- "Sepal.Width"
variable3 <- "Species"
iris %>%
  select_(变量1, 变量2, 变量3) %>%
  group_by_(变量3) %>%
  summarize_(均值1 = lazeval::interp(~mean(var), var = as.name(变量1)), 均值2 =
  lazeval::interp(~mean(var), var = as.name(变量2)))
#   物种 均值1 均值2
#   <fctr> <dbl> <dbl>
# 1 山鸢尾 5.006 3.428
# 2 变色鸢尾 5.936 2.770
# 3 维吉尼西亚鸢尾 6.588 2.974

```

```

# 13 5.5 4.2 1.4 0.2 setosa
# 14 4.5 2.3 1.3 0.3 setosa
# 15 5.3 3.7 1.5 0.2 setosa
# 16 7.0 3.2 4.7 1.4 versicolor
# 17 6.4 3.2 4.5 1.5 versicolor
# 18 6.9 3.1 4.9 1.5 versicolor
# 19 6.5 2.8 4.6 1.5 versicolor
# 20 6.3 3.3 4.7 1.6 versicolor
# 21 6.6 2.9 4.6 1.3 versicolor
# 22 5.9 3.0 4.2 1.5 versicolor
# 23 6.0 2.2 4.0 1.0 versicolor
# 24 6.1 2.9 4.7 1.4 versicolor
# 25 5.6 2.9 3.6 1.3 versicolor
# 26 6.7 3.1 4.4 1.4 versicolor
# 27 6.2 2.2 4.5 1.5 versicolor
# 28 6.8 2.8 4.8 1.4 versicolor
# 29 7.1 3.0 5.9 2.1 virginica
# 30 7.6 3.0 6.6 2.1 virginica
# 31 7.3 2.9 6.3 1.8 virginica
# 32 7.2 3.6 6.1 2.5 virginica
# 33 7.7 3.8 6.7 2.2 virginica
# 34 7.4 2.8 6.1 1.9 virginica
# 35 7.9 3.8 6.4 2.0 virginica

```

Section 77.4: Examples of NSE and string variables in dplyr

dplyr uses Non-Standard Evaluation(NSE), which is why we normally can use the variable names without quotes. However, sometimes during the data pipeline, we need to get our variable names from other sources such as a Shiny selection box. In case of functions like SELECT, we can just use select_ to use a string variable to select

```

variable1 <- "Sepal.Length"
variable2 <- "Sepal.Width"
iris %>%
  select_(variable1, variable2) %>%
  head(n=5)
# Sepal.Length Sepal.Width
# 1 5.1 3.5
# 2 4.9 3.0
# 3 4.7 3.2
# 4 4.6 3.1
# 5 5.0 3.6

```

But if we want to use other features such as summarize or filter we need to use interp function from lazeval package

```

variable1 <- "Sepal.Length"
variable2 <- "Sepal.Width"
variable3 <- "Species"
iris %>%
  select_(variable1, variable2, variable3) %>%
  group_by_(variable3) %>%
  summarize_(mean1 = lazeval::interp(~mean(var), var = as.name(variable1)), mean2 =
  lazeval::interp(~mean(var), var = as.name(variable2)))
#   Species mean1 mean2
#   <fctr> <dbl> <dbl>
# 1 setosa 5.006 3.428
# 2 versicolor 5.936 2.770
# 3 virginica 6.588 2.974

```

第78章：caret

caret 是一个R包，帮助处理机器学习问题所需的数据。它代表分类和回归训练。当为真实数据集构建模型时，除了实际的学习算法之外，还需要执行一些任务，比如清理数据、处理不完整的观测值，在测试集上验证我们的模型，以及比较不同的模型。

caret 在这些场景中提供帮助，与实际使用的学习算法无关。

第78.1节：预处理

caret中的预处理通过 `preProcess()` 函数完成。给定一个矩阵或数据框类型的对象 `x`，`preProcess()` 对训练数据应用转换，然后可以将其应用于测试数据。

`preProcess()` 函数的核心是 `method` 参数。方法操作按以下顺序应用：

1. 零方差滤波器
2. 近零方差过滤器
3. Box-Cox/Yeo-Johnson/指数变换
4. 中心化
5. 缩放
6. 范围
7. 缺失值填补

8. 主成分分析 (PCA)

9. 独立成分分析 (ICA)

10. 空间符号变换

下面，我们以mtcars数据集为例，执行中心化、缩放和空间符号变换。

```
auto_index <- createDataPartition(mtcars$mpg, p = .8,
                                    list = FALSE,
                                    times = 1)

mt_train <- mtcars[auto_index, ]
mt_test <- mtcars[-auto_index,]

process_mtcars <- preProcess(mt_train, method = c("center", "scale", "spatialSign"))

mtcars_train_transf <- predict(process_mtcars, mt_train)
mtcars_test_transf <- predict(process_mtcars, mt_test)
```

Chapter 78: caret

caret is an R package that aids in data processing needed for machine learning problems. It stands for classification and regression training. When building models for a real dataset, there are some tasks other than the actual learning algorithm that need to be performed, such as cleaning the data, dealing with incomplete observations, validating our model on a test set, and compare different models.

caret helps in these scenarios, independent of the actual learning algorithms used.

Section 78.1: Preprocessing

Pre-processing in caret is done through the `preProcess()` function. Given a matrix or data frame type object `x`, `preProcess()` applies transformations on the training data which can then be applied to testing data.

The heart of the `preProcess()` function is the `method` argument. Method operations are applied in this order:

1. Zero-variance filter
2. Near-zero variance filter
3. Box-Cox/Yeo-Johnson/exponential transformation
4. Centering
5. Scaling
6. Range
7. Imputation
8. PCA
9. ICA
10. Spatial Sign

Below, we take the mtcars data set and perform centering, scaling, and a spatial sign transform.

```
auto_index <- createDataPartition(mtcars$mpg, p = .8,
                                    list = FALSE,
                                    times = 1)

mt_train <- mtcars[auto_index, ]
mt_test <- mtcars[-auto_index,]

process_mtcars <- preProcess(mt_train, method = c("center", "scale", "spatialSign"))

mtcars_train_transf <- predict(process_mtcars, mt_train)
mtcars_test_transf <- predict(process_mtcars, mt_test)
```

第79章：压缩档案中的文件提取与列出

第79.1节：从.zip档案中提取文件

解压zip档案是通过unzip函数完成的，该函数来自utils包（包含在基础R中）。

```
unzip(zipfile = "bar.zip", exdir = "./foo")
```

这将把"bar.zip"中的所有文件提取到"foo"目录中，如有必要该目录将被创建。波浪号扩展会自动基于你的工作目录完成。或者，你也可以传入zipfile的完整路径名。

Chapter 79: Extracting and Listing Files in Compressed Archives

Section 79.1: Extracting files from a .zip archive

Unzipping a zip archive is done with `unzip` function from the `utils` package (which is included in base R).

```
unzip(zipfile = "bar.zip", exdir = "./foo")
```

This will extract all files in `"bar.zip"` to the `"foo"` directory, which will be created if necessary. Tilde expansion is done automatically from your working directory. Alternatively, you can pass the whole path name to the zipfile.

第80章：使用R的概率分布

第80.1节：R中不同分布的概率密度函数（PDF）和概率质量函数（PMF）

二项分布的概率质量函数（PMF）

假设一个公平的骰子掷了10次。掷出恰好两个六的概率是多少？

你可以使用dbinom函数来回答这个问题：

```
> dbinom(2, 10, 1/6)
[1] 0.29071
```

泊松分布的概率质量函数（PMF）

某餐厅某天订购的三明治数量已知服从均值为20的泊松分布。明天恰好订购十八个三明治的概率是多少？

你可以使用 dpois 函数来回答这个问题：

```
> dpois(18, 20)
[1] 0.08439355
```

正态分布的概率密度函数（PDF）

要计算均值为5、标准差为2的正态分布在 $x=2.5$ 处的概率密度函数值，使用命令：

```
> dnorm(2.5, mean=5, sd=2)
[1] 0.09132454
```

Chapter 80: Probability Distributions with R

Section 80.1: PDF and PMF for different distributions in R

PMF FOR THE BINOMIAL DISTRIBUTION

Suppose that a fair die is rolled 10 times. What is the probability of throwing exactly two sixes?

You can answer the question using the dbinom function:

```
> dbinom(2, 10, 1/6)
[1] 0.29071
```

PMF FOR THE POISSON DISTRIBUTION

The number of sandwich ordered in a restaurant on a given day is known to follow a Poisson distribution with a mean of 20. What is the probability that exactly eighteen sandwich will be ordered tomorrow?

You can answer the question with the dpois function:

```
> dpois(18, 20)
[1] 0.08439355
```

PDF FOR THE NORMAL DISTRIBUTION

To find the value of the pdf at $x=2.5$ for a normal distribution with a mean of 5 and a standard deviation of 2, use the command:

```
> dnorm(2.5, mean=5, sd=2)
[1] 0.09132454
```

第81章：使用 knitr 在 LaTeX 中调用 R

| 选项 | 详情 |
|----------------------|------------------------------------|
| echo | (TRUE/FALSE) - 是否在输出文件中包含 R 源代码 |
| message | (TRUE/FALSE) - 是否在输出文件中包含 R 源执行的消息 |
| warning | (TRUE/FALSE) - 是否在输出文件中包含 R 源执行的警告 |
| error | (TRUE/FALSE) - 是否在输出文件中包含 R 源执行的错误 |
| cache | (TRUE/FALSE) - 是否缓存 R 源执行的结果 |
| fig.width(numeric) | R 源执行生成图形的宽度 |
| fig.height (numeric) | R 源执行生成图形的高度 |

第81.1节：使用 Knitr 和代码外部化在 LaTeX 中使用 R

Knitr 是一个 R 包，允许我们将 R 代码与 LaTeX 代码混合使用。实现这一点的一种方式是使用外部代码块。外部代码块允许我们在 R 开发环境中开发/测试 R 脚本，然后将结果包含在报告中。这是一种强大的组织技术。下面演示了这种方法。

```
# r-noweb-file.Rnw
\documentclass{article}

<<echo=FALSE, cache=FALSE>>=
knitr::opts_chunk$set(echo=FALSE, cache=TRUE)
knitr::read_chunk('r-file.R')
@
```

\begin{document}
这是一个Rnw文件 (R noweb)。它包含了LaTeX和R的组合。

如上所述，我们调用了read_chunk命令，可以引用r-file.R脚本中的代码段。

```
<<Chunk1>>=
@
\end{document}
```

使用这种方法时，我们将代码保存在一个单独的R文件中，如下所示。

```
## r-file.R
## 注意特定的注释风格：单个井号后跟四个短横线

# ---- Chunk1 ----

print("这是外部文件中的R代码")

x <- seq(1:10)
y <- rev(seq(1:10))
plot(x,y)
```

第81.2节：使用Knitr和内联代码块在LaTeX中使用R

Knitr是一个R包，允许我们将R代码与LaTeX代码交织使用。实现这一点的一种方式是内联代码块。下面演示了这种方法。

```
# r-noweb-file.Rnw
\documentclass{article}
```

Chapter 81: R in LaTeX with knitr

| Option | Details |
|----------------------|---|
| echo | (TRUE/FALSE) - whether to include R source code in the output file |
| message | (TRUE/FALSE) - whether to include messages from the R source execution in the output file |
| warning | (TRUE/FALSE) - whether to include warnings from the R source execution in the output file |
| error | (TRUE/FALSE) - whether to include errors from the R source execution in the output file |
| cache | (TRUE/FALSE) - whether to cache the results of the R source execution |
| fig.width (numeric) | width of the plot generated by the R source execution |
| fig.height (numeric) | height of the plot generated by the R source execution |

Section 81.1: R in LaTeX with Knitr and Code Externalization

Knitr is an R package that allows us to intermingle R code with LaTeX code. One way to achieve this is external code chunks. External code chunks allow us to develop/test R Scripts in an R development environment and then include the results in a report. It is a powerful organizational technique. This approach is demonstrated below.

```
# r-noweb-file.Rnw
\documentclass{article}

<<echo=FALSE, cache=FALSE>>=
knitr::opts_chunk$set(echo=FALSE, cache=TRUE)
knitr::read_chunk('r-file.R')
@

\begin{document}
This is an Rnw file (R noweb). It contains a combination of LaTeX and R.

One we have called the read\_\_chunk command above we can reference sections of code in the r-file.R script.

<<Chunk1>>=
@
\end{document}
```

When using this approach we keep our code in a separate R file as shown below.

```
## r-file.R
## note the specific comment style of a single pound sign followed by four dashes

# ---- Chunk1 ----

print("This is R Code in an external file")

x <- seq(1:10)
y <- rev(seq(1:10))
plot(x,y)
```

Section 81.2: R in LaTeX with Knitr and Inline Code Chunks

Knitr is an R package that allows us to intermingle R code with LaTeX code. One way to achieve this is inline code chunks. This approach is demonstrated below.

```
# r-noweb-file.Rnw
\documentclass{article}
```

```
\begin{document}  
这是一个Rnw文件 (R noweb)。它包含了LateX和R的组合。
```

```
<<my-label>>=  
print("这是一个R代码块")  
x <- seq(1:10)  
@
```

以上是一个内部代码块。

我们可以像这样在LaTeX代码中内联访问任何代码块中创建的数据。

数组x的长度是\Expr{length(x)}。

```
\end{document}
```

第81.3节：使用Knitr和内部代码块在LaTeX中使用R

Knitr 是一个 R 包，允许我们将 R 代码与 LaTeX 代码交织在一起。实现这一点的一种方法是内部代码块。下面演示了这种方法。

```
# r-noweb-file.Rnw  
\documentclass{article}  
\begin{document}  
这是一个Rnw文件 (R noweb)。它包含了LateX和R的组合。
```

```
<<code-chunk-label>>=  
print("这是一个R代码块")  
x <- seq(1:10)  
y <- seq(1:10)  
plot(x,y) # 布朗运动  
@
```

```
\end{document}
```

```
\begin{document}  
This is an Rnw file (R noweb). It contains a combination of LateX and R.
```

```
<<my-label>>=  
print("This is an R Code Chunk")  
x <- seq(1:10)  
@
```

Above is an internal code chunk.

We can access data created in any code chunk inline with our LaTeX code like this.

The length of array x is \Expr{length(x)}.

```
\end{document}
```

Section 81.3: R in LaTex with Knitr and Internal Code Chunks

Knitr is an R package that allows us to intermingle R code with LaTeX code. One way to achieve this is internal code chunks. This approach is demonstrated below.

```
# r-noweb-file.Rnw  
\documentclass{article}  
\begin{document}  
This is an Rnw file (R noweb). It contains a combination of LateX and R.
```

```
<<code-chunk-label>>=  
print("This is an R Code Chunk")  
x <- seq(1:10)  
y <- seq(1:10)  
plot(x,y) # Brownian motion  
@
```

```
\end{document}
```

第82章：R语言中的网页爬取

第82.1节：使用RCurl包的标准爬取方法

我们尝试提取IMDb排行榜电影及评分

```
R> library(RCurl)
R> library(XML)
R> url <- "http://www.imdb.com/chart/top"
R> top <- getURL(url)
R> parsed_top <- htmlParse(top, encoding = "UTF-8")
R> top_table <- readHTMLTable(parsed_top)[[1]]
R> head(top_table[1:10, 1:3])
```

排名 & 片名 IMDb 评分
1 1. 肖申克的救赎 (1994) 9.2
2 2. 教父 (1972) 9.2
3 3. 教父: 第二部 (1974) 9.0
4 4. 蝙蝠侠：黑暗骑士 (2008) 8.9
5 5. 低俗小说 (1994) 8.9
6 6. 黄金三镖客 (1966) 8.9
7 7. 辛德勒的名单 (1993) 8.9
8 8. 十二怒汉 (1957) 8.9
9 9. 《指环王》：《王者归来》 (2003) 8.9
10 10. 搏击俱乐部 (1999) 8.8

Chapter 82: Web Crawling in R

Section 82.1: Standard scraping approach using the RCurl package

We try to extract imdb top chart movies and ratings

```
R> library(RCurl)
R> library(XML)
R> url <- "http://www.imdb.com/chart/top"
R> top <- getURL(url)
R> parsed_top <- htmlParse(top, encoding = "UTF-8")
R> top_table <- readHTMLTable(parsed_top)[[1]]
R> head(top_table[1:10, 1:3])
```

| Rank | Title | IMDb | Rating |
|------|---|------|--------|
| 1 | 1. The Shawshank Redemption (1994) | 9.2 | |
| 2 | 2. The Godfather (1972) | 9.2 | |
| 3 | 3. The Godfather: Part II (1974) | 9.0 | |
| 4 | 4. The Dark Knight (2008) | 8.9 | |
| 5 | 5. Pulp Fiction (1994) | 8.9 | |
| 6 | 6. The Good, the Bad and the Ugly (1966) | 8.9 | |
| 7 | 7. Schindler's List (1993) | 8.9 | |
| 8 | 8. 12 Angry Men (1957) | 8.9 | |
| 9 | 9. The Lord of the Rings: The Return of the King (2003) | 8.9 | |
| 10 | 10. Fight Club (1999) | 8.8 | |

第83章：使用RMarkdown创建报告

第83.1节：包含参考文献目录

可以通过YAML选项`bibliography`:轻松包含`bibtex`目录。可以通过`biblio-style`:添加参考文献的特定样式
参考文献会被添加到文档末尾。

```
---
标题: "包含参考文献目录"
作者: "约翰·多伊"
输出: pdf_document
bibliography: references.bib
---

# 摘要

@R_Core_Team_2016

# 参考文献
```

Including Bibliography
John Doe

Abstract

R Core Team (2016)

References

R Core Team. 2016. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. <http://www.R-project.org/>.

第83.2节：包含LaTeX导言命令

在RMarkdown文档中包含LaTeX导言命令（例如`\usepackage`）有两种可能的方法。

1. 使用YAML选项`header-includes`:

```
---
标题: "在 RMarkdown 中包含 LaTeX 前导命令"
header-includes:
- \renewcommand{\familydefault}{cmss}
- \usepackage[cm, slantedGreek]{sfmath}
- \usepackage[T1]{fontenc}
输出: pdf_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE, external=T)
```

# 第一节
```

如您所见，这段文字使用了计算机现代字体！

Chapter 83: Creating reports with RMarkdown

Section 83.1: Including bibliographies

A bibtex catalogue can easily be included with the YAML option `bibliography`:. A certain style for the bibliography can be added with `biblio-style`:. The references are added at the end of the document.

```
---
title: "Including Bibliography"
author: "John Doe"
output: pdf_document
bibliography: references.bib
---

# Abstract

@R_Core_Team_2016

# References
```

Including Bibliography
John Doe

Abstract

R Core Team (2016)

References

R Core Team. 2016. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. <http://www.R-project.org/>.

Section 83.2: Including LaTeX Preamble Commands

There are two possible ways of including LaTeX preamble commands (e.g. `\usepackage`) in a RMarkdown document.

1. Using the YAML option `header-includes`:

```
---
title: "Including LaTeX Preamble Commands in RMarkdown"
header-includes:
- \renewcommand{\familydefault}{cmss}
- \usepackage[cm, slantedGreek]{sfmath}
- \usepackage[T1]{fontenc}
output: pdf_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE, external=T)
```

# Section 1

As you can see, this text uses the Computer Modern Font!
```

Section 1

As you can see, this text uses the Computer Modern Font!

2. 使用includes、in_header包含外部命令

```
---  
标题: "在 RMarkdown 中包含 LaTeX 前导命令"  
输出:  
pdf_document:  
包括:  
in_header: includes.tex  
---
```

```
```{r setup, include=FALSE}  
knitr::opts_chunk$set(echo = TRUE, external=T)
```
```

第一节

正如你所见，这段文字使用了计算机现代字体！

这里，includes.tex 的内容是我们用 header-includes 包含的相同三个命令。

编写一个全新的模板

第三种可能的选择是编写你自己的 LaTeX 模板，并用 template 包含它。但这涵盖的结构远比仅仅前言部分要多得多。

```
---  
title: "我的模板"  
author: "马丁·施梅尔策"  
output:  
pdf_document:  
template: myTemplate.tex  
---
```

第83.3节：打印表格

有几个包允许以 HTML 或 LaTeX 表格的形式输出数据结构。它们主要在灵活性上有所不同。

这里我使用的包有：

- knitr
- xtable
- pandoc

用于HTML文档

Section 1

As you can see, this text uses the Computer Modern Font!

2. Including External Commands with includes, in_header

```
---  
title: "Including LaTeX Preamble Commands in RMarkdown"  
output:  
pdf_document:  
includes:  
in_header: includes.tex  
---
```

```
```{r setup, include=FALSE}  
knitr::opts_chunk$set(echo = TRUE, external=T)
```
```

Section 1

As you can see, this text uses the Computer Modern Font!

Here, the content of includes.tex are the same three commands we included with header-includes.

Writing a whole new template

A possible third option is to write your own LaTeX template and include it with template. But this covers a lot more of the structure than only the preamble.

```
---  
title: "My Template"  
author: "Martin Schmelzer"  
output:  
pdf_document:  
template: myTemplate.tex  
---
```

Section 83.3: Printing tables

There are several packages that allow the output of data structures in form of HTML or LaTeX tables. They mostly differ in flexibility.

Here I use the packages:

- knitr
- xtable
- pandoc

For HTML documents

```
---  
标题: "打印表格"  
作者: "马丁·施梅尔策"  
日期: "2016年7月29日"  
output: html_document  
---
```

```
```{r setup, include=FALSE}  
knitr::opts_chunk$set(echo = TRUE)
library(knitr)
library(xtable)
```

```
library(pander)
df <- mtcars[1:4,1:4]
```
```

```
# 使用 `kable` 打印表格  
```{r, 'kable'}  
kable(df)
```
```

```
# 使用 `xtable` 打印表格  
```{r, 'xtable', results='asis'}  
print(xtable(df), type="html")
```
```

```
# 使用 `pander` 打印表格  
```{r, 'pander'}  
pander(df)
```
```

Printing Tables

Martin Schmelzer
29.Juli.2016

Print tables using kable

| | mpg | cyl | disp | hp |
|----------------|------|-----|------|-----|
| Mazda RX4 | 21.0 | 6 | 160 | 110 |
| Mazda RX4 Wag | 21.0 | 6 | 160 | 110 |
| Datsun 710 | 22.8 | 4 | 108 | 93 |
| Hornet 4 Drive | 21.0 | 6 | 258 | 110 |

Print tables using xtable

| | mpg | cyl | disp | hp |
|----------------|------|-----|------|-----|
| Mazda RX4 | 21.0 | 6 | 160 | 110 |
| Mazda RX4 Wag | 21.0 | 6 | 160 | 110 |
| Datsun 710 | 22.8 | 4 | 108 | 93 |
| Hornet 4 Drive | 21.0 | 6 | 258 | 110 |

Print tables using pander

| | mpg | cyl | disp | hp |
|----------------|------|-----|------|-----|
| Mazda RX4 | 21 | 6 | 160 | 110 |
| Mazda RX4 Wag | 21 | 6 | 160 | 110 |
| Datsun 710 | 22.8 | 4 | 108 | 93 |
| Hornet 4 Drive | 21.0 | 6 | 258 | 110 |

对于 PDF 文档

```
---  
标题: "打印表格"  
作者: "马丁·施梅尔策"  
日期: "2016年7月29日"  
输出: pdf_document  
---
```

```
```{r setup, include=FALSE}  
knitr::opts_chunk$set(echo = TRUE)
```

```

title: "Printing Tables"
author: "Martin Schmelzer"
date: "29 Juli 2016"
output: html_document

```

```
```{r setup, include=FALSE}  
knitr::opts_chunk$set(echo = TRUE)  
library(knitr)  
library(xtable)  
library(pander)  
df <- mtcars[1:4,1:4]  
```
```

```
Print tables using `kable`
```{r, 'kable'}  
kable(df)  
```
```

```
Print tables using `xtable`
```{r, 'xtable', results='asis'}  
print(xtable(df), type="html")  
```
```

```
Print tables using `pander`
```{r, 'pander'}  
pander(df)  
```
```

## Printing Tables

Martin Schmelzer  
29.Juli.2016

### Print tables using kable

|                | mpg  | cyl | disp | hp  |
|----------------|------|-----|------|-----|
| Mazda RX4      | 21.0 | 6   | 160  | 110 |
| Mazda RX4 Wag  | 21.0 | 6   | 160  | 110 |
| Datsun 710     | 22.8 | 4   | 108  | 93  |
| Hornet 4 Drive | 21.0 | 6   | 258  | 110 |

### Print tables using xtable

|                | mpg  | cyl | disp | hp  |
|----------------|------|-----|------|-----|
| Mazda RX4      | 21.0 | 6   | 160  | 110 |
| Mazda RX4 Wag  | 21.0 | 6   | 160  | 110 |
| Datsun 710     | 22.8 | 4   | 108  | 93  |
| Hornet 4 Drive | 21.0 | 6   | 258  | 110 |

### Print tables using pander

|                | mpg  | cyl | disp | hp  |
|----------------|------|-----|------|-----|
| Mazda RX4      | 21   | 6   | 160  | 110 |
| Mazda RX4 Wag  | 21   | 6   | 160  | 110 |
| Datsun 710     | 22.8 | 4   | 108  | 93  |
| Hornet 4 Drive | 21.0 | 6   | 258  | 110 |

## For PDF documents

```

title: "Printing Tables"
author: "Martin Schmelzer"
date: "29 Juli 2016"
output: pdf_document

```

```
```{r setup, include=FALSE}  
knitr::opts_chunk$set(echo = TRUE)
```

```
library(knitr)
library(xtable)
library(pander)
df <- mtcars[1:4,1:4]
```
```
# 使用 `kable` 打印表格
```{r, 'kable'}
kable(df)
```
```
使用 `xtable` 打印表格
```{r, 'xtable', results='asis'}
print(xtable(df, caption="我的表格"))
```
```
# 使用 `pander` 打印表格
```{r, 'pander'}
pander(df)
```
``
```

| Printing Tables | | | | |
|---------------------------------|------|-----|-------|-------|
| Martin Schröder | | | | |
| 29 July 2018 | | | | |
| Print tables using kable | | | | |
| knitr(kable) | | | | |
| | mpg | cyl | disp | hp |
| Mazda RX4 | 21.0 | 6 | 160.0 | 110.0 |
| Mazda RX4 Wag | 21.0 | 6 | 160.0 | 110.0 |
| Datsun 710 | 22.8 | 4 | 108.0 | 93.0 |
| Hornet 4 Drive | 21.0 | 6 | 160.0 | 110.0 |

| Print tables using xtable | | | | |
|---|-------|------|--------|--------|
| print(xtable(tab), caption = "My Table") | | | | |
| Unless told otherwise in R 3.3.1 by mtable 1.6-2 package 29 July 2018 10:28:34 2018 | | | | |
| | mpg | cyl | disp | hp |
| Mazda RX4 | 21.00 | 6.00 | 160.00 | 110.00 |
| Mazda RX4 Wag | 21.00 | 6.00 | 160.00 | 110.00 |
| Datsun 710 | 22.80 | 4.00 | 108.00 | 93.00 |
| Hornet 4 Drive | 21.00 | 6.00 | 160.00 | 110.00 |

| Table 2: My Table | | | | |
|---|------|-----|------|-----|
| print(tab) | | | | |
| unless told otherwise in R 3.3.1 by mtable 1.6-2 package 29 July 2018 10:28:34 2018 | | | | |
| | mpg | cyl | disp | hp |
| Mazda RX4 | 21 | 6 | 160 | 110 |
| Mazda RX4 Wag | 21 | 6 | 160 | 110 |
| Datsun 710 | 22.8 | 4 | 108 | 93 |
| Hornet 4 Drive | 21.0 | 6 | 160 | 110 |

如何阻止 xtable 在每个表格前打印注释？

```
options(xtable.comment = FALSE)
```

第 83.4 节：基本 R-markdown 文档结构

R-markdown 代码块

R-markdown 是一个带有嵌入式 R 代码块的 markdown 文件，这些代码块称为chunks。R 代码块有两种类型 inline 和 block。

内联代码块使用以下语法添加：

2*2

它们会被计算并将输出结果插入到相应位置。

块状代码块有不同的语法：

```
library(knitr)
library(xtable)
library(pander)
df <- mtcars[1:4,1:4]
```
```
# Print tables using `kable`
```{r, 'kable'}
kable(df)
```
```
Print tables using `xtable`
```{r, 'xtable', results='asis'}
print(xtable(df, caption="My Table"))
```
```
# Print tables using `pander`
```{r, 'pander'}
pander(df)
```

Printing Tables																													
Martin Schmalzried																													
29 July 2010																													
<b>Print tables using <code>table</code></b>																													
<code>knitr::kable()</code>																													
<table border="1"> <thead> <tr> <th></th><th>mpg</th><th>cyl</th><th>disp</th><th>hp</th></tr> </thead> <tbody> <tr> <td>Mazda RX4</td><td>21.0</td><td>6</td><td>160</td><td>110</td></tr> <tr> <td>Mazda RX4 Wag</td><td>21.0</td><td>6</td><td>160</td><td>110</td></tr> <tr> <td>Datsun 710</td><td>22.8</td><td>4</td><td>108</td><td>93</td></tr> <tr> <td>Hornet 4 Drive</td><td>18.0</td><td>6</td><td>126</td><td>108</td></tr> </tbody> </table>						mpg	cyl	disp	hp	Mazda RX4	21.0	6	160	110	Mazda RX4 Wag	21.0	6	160	110	Datsun 710	22.8	4	108	93	Hornet 4 Drive	18.0	6	126	108
	mpg	cyl	disp	hp																									
Mazda RX4	21.0	6	160	110																									
Mazda RX4 Wag	21.0	6	160	110																									
Datsun 710	22.8	4	108	93																									
Hornet 4 Drive	18.0	6	126	108																									
<b>Print tables using <code>xtable</code></b>																													
<code>print(xtable(mtcars), caption = "My Table")</code>																													
Useful table generated in R 3.1.1 by xtable 1.8-2 package 2011-07-29 10:18:41 UTC																													
<table border="1"> <thead> <tr> <th></th><th>mpg</th><th>cyl</th><th>disp</th><th>hp</th></tr> </thead> <tbody> <tr> <td>Mazda RX4</td><td>21.0</td><td>6.00</td><td>160.00</td><td>110.00</td></tr> <tr> <td>Mazda RX4 Wag</td><td>21.0</td><td>6.00</td><td>160.00</td><td>110.00</td></tr> <tr> <td>Datsun 710</td><td>22.8</td><td>4.00</td><td>108.00</td><td>93.00</td></tr> <tr> <td>Hornet 4 Drive</td><td>18.0</td><td>6.00</td><td>126.00</td><td>108.00</td></tr> </tbody> </table>						mpg	cyl	disp	hp	Mazda RX4	21.0	6.00	160.00	110.00	Mazda RX4 Wag	21.0	6.00	160.00	110.00	Datsun 710	22.8	4.00	108.00	93.00	Hornet 4 Drive	18.0	6.00	126.00	108.00
	mpg	cyl	disp	hp																									
Mazda RX4	21.0	6.00	160.00	110.00																									
Mazda RX4 Wag	21.0	6.00	160.00	110.00																									
Datsun 710	22.8	4.00	108.00	93.00																									
Hornet 4 Drive	18.0	6.00	126.00	108.00																									
Table 2: My Table																													
<b>Print tables using <code>pander</code></b>																													
<code>knitr::pander()</code>																													
<table border="1"> <thead> <tr> <th></th><th>mpg</th><th>cyl</th><th>disp</th><th>hp</th></tr> </thead> <tbody> <tr> <td>Mazda RX4</td><td>21</td><td>6</td><td>160</td><td>110</td></tr> <tr> <td>Mazda RX4 Wag</td><td>21</td><td>6</td><td>160</td><td>110</td></tr> <tr> <td>Datsun 710</td><td>22.8</td><td>4</td><td>108</td><td>93</td></tr> <tr> <td>Hornet 4 Drive</td><td>18.0</td><td>6</td><td>126</td><td>108</td></tr> </tbody> </table>						mpg	cyl	disp	hp	Mazda RX4	21	6	160	110	Mazda RX4 Wag	21	6	160	110	Datsun 710	22.8	4	108	93	Hornet 4 Drive	18.0	6	126	108
	mpg	cyl	disp	hp																									
Mazda RX4	21	6	160	110																									
Mazda RX4 Wag	21	6	160	110																									
Datsun 710	22.8	4	108	93																									
Hornet 4 Drive	18.0	6	126	108																									

## How can I stop xtable printing the comment ahead of each table?

```
options(xtable.comment = FALSE)
```

## Section 83.4: Basic R-markdown document structure

## R-markdown code chunks

R-markdown is a markdown file with embedded blocks of R code called *chunks*. There are two types of R code chunks: **inline** and **block**.

**Inline** chunks are added using the following syntax:

r 2\*2

They are evaluated and inserted their output answer in place.

**Block** chunks have a different syntax:

```
```{r name, echo=TRUE, include=TRUE, ...}
```

2*2

....

它们带有多种可选项。以下是主要选项（但还有许多其他选项）：

- **echo** (布尔值) 控制代码块内的代码是否包含在文档中
- **include** (布尔值) 控制输出是否包含在文档中
- **fig.width** (数值型) 设置输出图形的宽度
- **fig.height** (数值型) 设置输出图形的高度
- **fig.cap** (字符型) 设置图形标题

它们以简单的 tag=value 格式编写，如上例所示。

R-markdown 文档示例

下面是一个基本的 R-markdown 文件示例，展示了如何在 r-markdown 中嵌入 R 代码块。

```
# 标题 #
```

这是**纯 markdown** 文本。

```
```{r 代码, include=FALSE, echo=FALSE}
```

```
仅声明变量
```

```
income <- 1000
taxes <- 125
```

...

我的收入是 `r income` 美元，我缴纳了 `r taxes` 美元的税款。

下面是 我剩下的钱的 sum：

```
```{r gain, include=TRUE, echo=FALSE}
```

```
gain <- income-taxes
```

增益

...

```
```{r plotOutput, include=TRUE, echo=FALSE, fig.width=6, fig.height=6}
```

```
pie(c(收入, 税收), label=c("收入", "税收"))
```

...

## 将 R-markdown 转换为其他格式

R knitr 包可用于评估 R-markdown 文件中的 R 代码块，并将其转换为常规的 markdown 文件。

将 R-markdown 文件转换为 pdf/html 需要以下步骤：

```
```{r name, echo=TRUE, include=TRUE, ...}
```

2*2

....

And they come with several possible options. Here are the main ones (but there are many others):

- **echo** (boolean) controls whether the code inside chunk will be included in the document
- **include** (boolean) controls whether the output should be included in the document
- **fig.width** (numeric) sets the width of the output figures
- **fig.height** (numeric) sets the height of the output figures
- **fig.cap** (character) sets the figure captions

They are written in a simple tag=value format like in the example above.

R-markdown document example

Below is a basic example of R-markdown file illustrating the way R code chunks are embedded inside r-markdown.

```
# Title #
```

This **is** **plain markdown** text.

```
```{r code, include=FALSE, echo=FALSE}
```

```
Just declare variables
```

```
income <- 1000
taxes <- 125
```

...

My income **is**: `r income` dollars and **I** payed `r taxes` dollars **in** taxes.

Below **is** the **sum** of money **I** will have left:

```
```{r gain, include=TRUE, echo=FALSE}
```

```
gain <- income-taxes
```

gain

...

```
```{r plotOutput, include=TRUE, echo=FALSE, fig.width=6, fig.height=6}
```

```
pie(c(income,taxes), label=c("income", "taxes"))
```

...

## Converting R-markdown to other formats

The R knitr package can be used to evaluate R chunks inside R-markdown file and turn it into a regular markdown file.

The following steps are needed in order to turn R-markdown file into pdf/html:

1. 使用 knitr 将 R-markdown 文件转换为 markdown 文件。
2. 使用专用工具如 pandoc 将得到的 markdown 文件转换为 pdf/html。

除了上述方法, knitr 包还提供了包装函数 `knit2html()` 和 `knit2pdf()`, 可用于直接生成最终文档, 无需手动中间转换为 markdown 格式:

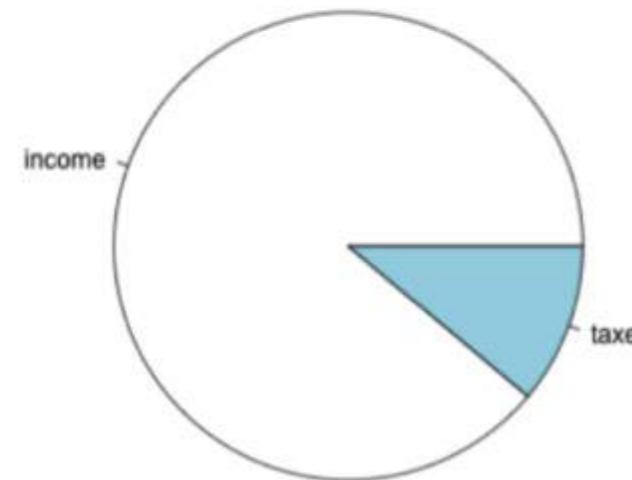
如果上述示例文件保存为`income.Rmd`, 则可以使用以下R命令将其转换为pdf文件:

```
library(knitr)
knit2pdf("income.Rmd", "income.pdf")
```

最终文档将类似于下面的内容。

## Title

This is plain markdown text.  
My income is: 1000 dollars and I payed 125 dollars in taxes.  
Below is the sum of money I will have left:  
  
## [1] 875



1. Convert R-markdown file to markdown file using `knitr`.
2. Convert the obtained markdown file to pdf/html using specialized tools like `pandoc`.

In addition to the above `knitr` package has wrapper functions `knit2html()` and `knit2pdf()` that can be used to produce the final document without the intermediate step of manually converting it to the markdown format:

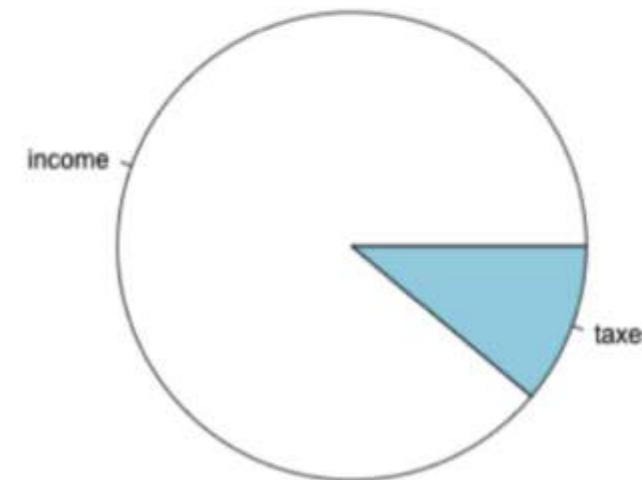
If the above example file was saved as `income.Rmd` it can be converted to a `pdf` file using the following R commands:

```
library(knitr)
knit2pdf("income.Rmd", "income.pdf")
```

The final document will be similar to the one below.

## Title

This is plain markdown text.  
My income is: 1000 dollars and I payed 125 dollars in taxes.  
Below is the sum of money I will have left:  
  
## [1] 875



# 第84章：GPU加速计算

## 第84.1节：gpuR gpuMatrix对象

```
library(gpuR) # gpuMatrix对象 X <- gpuMatrix(rnorm(100), 10, 10) Y <- gpuMatrix(rnorm(100), 10, 10) # 调用操作时将数据传输到GPU # 自动复制回CPU Z <- X %*% Y
```

## 第84.2节：gpuR vclMatrix对象

```
library(gpuR) # vclMatrix对象 X <- vclMatrix(rnorm(100), 10, 10) Y <- vclMatrix(rnorm(100), 10, 10) # 数据始终在GPU上 # 无数据传输 Z <- X %*% Y
```

# Chapter 84: GPU-accelerated computing

## Section 84.1: gpuR gpuMatrix objects

```
library(gpuR) # gpuMatrix objects X <- gpuMatrix(rnorm(100), 10, 10) Y <- gpuMatrix(rnorm(100), 10, 10) # transfer data to GPU when operation called # automatically copied back to CPU Z <- X %*% Y
```

## Section 84.2: gpuR vclMatrix objects

```
library(gpuR) # vclMatrix objects X <- vclMatrix(rnorm(100), 10, 10) Y <- vclMatrix(rnorm(100), 10, 10) # data always on GPU # no data transfer Z <- X %*% Y
```

# 第85章：heatmap和heatmap.2

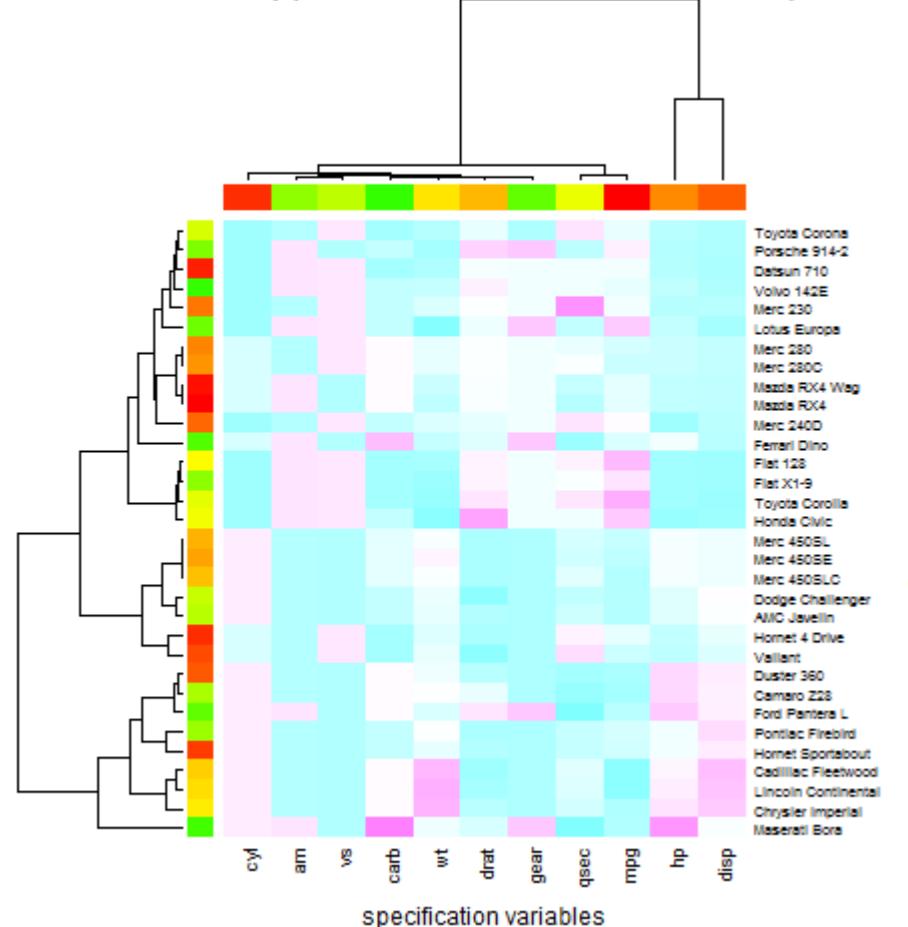
## 第85.1节：官方文档中的示例

stats::heatmap

示例 1 (基本用法)

```
require(graphics); require(grDevices)
x <- as.matrix(mtcars)
rc <- rainbow(nrow(x), start = 0, end = .3)
cc <- rainbow(ncol(x), start = 0, end = .3)
hv <- heatmap(x, col = cm.colors(256), scale = "column",
 RowSideColors = rc, ColSideColors = cc, margins = c(5,10),
 xlab = "规格变量", ylab = "汽车型号",
 main = "heatmap(<Mtcars 数据>, ..., scale = \"column\")")
```

heatmap(<Mtcars data>, ..., scale = "column")



```
utils::str(hv) # 两个重新排序的索引向量
4 项列表
$ rowInd: int [1:32] 31 17 16 15 5 25 29 24 7 6 ...
$ colInd: int [1:11] 2 9 8 11 6 5 10 7 1 4 ...
$ Rowv : NULL
$ Colv : NULL
```

示例 2 (完全无列树状图 (也无重新排序) )

```
heatmap(x, Colv = NA, col = cm.colors(256), scale = "column",
 RowSideColors = rc, margins = c(5,10),
 xlab = "规格变量", ylab = "汽车型号",
 main = "heatmap(<Mtcars 数据>, ..., scale = \"column\")")
```

# Chapter 85: heatmap and heatmap.2

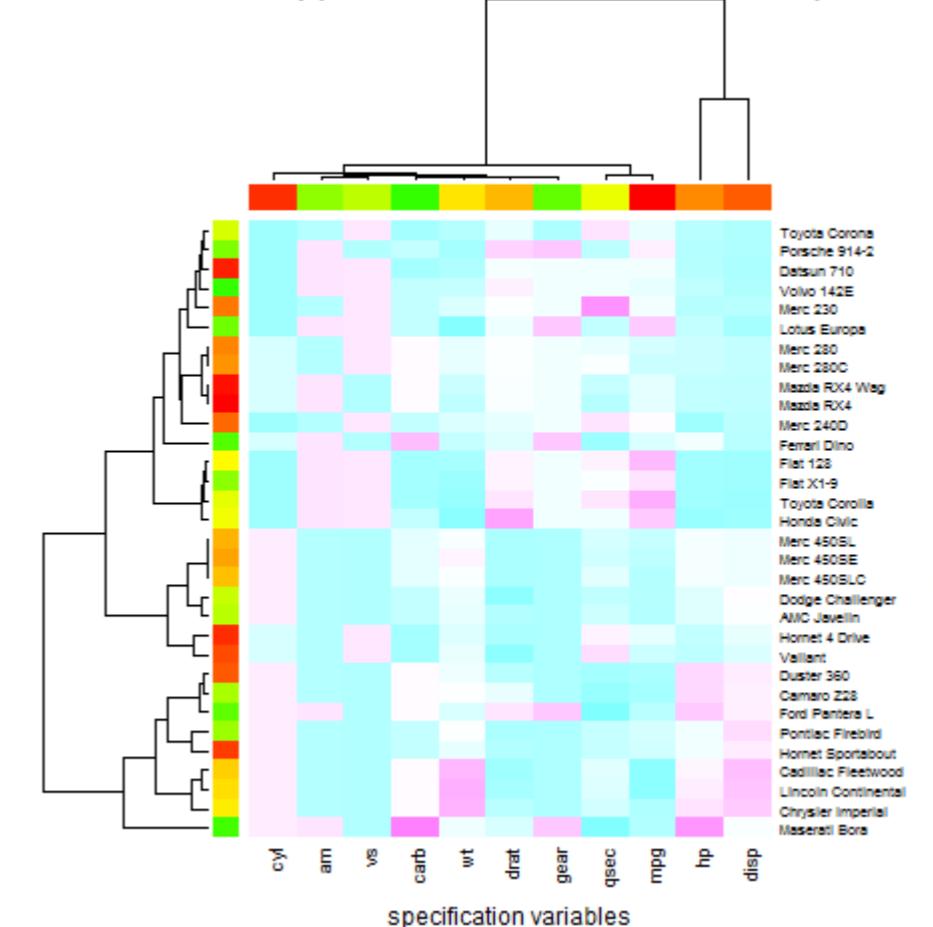
## Section 85.1: Examples from the official documentation

stats::heatmap

Example 1 (Basic usage)

```
require(graphics); require(grDevices)
x <- as.matrix(mtcars)
rc <- rainbow(nrow(x), start = 0, end = .3)
cc <- rainbow(ncol(x), start = 0, end = .3)
hv <- heatmap(x, col = cm.colors(256), scale = "column",
 RowSideColors = rc, ColSideColors = cc, margins = c(5,10),
 xlab = "specification variables", ylab = "Car Models",
 main = "heatmap(<Mtcars data>, ..., scale = \"column\")")
```

heatmap(<Mtcars data>, ..., scale = "column")

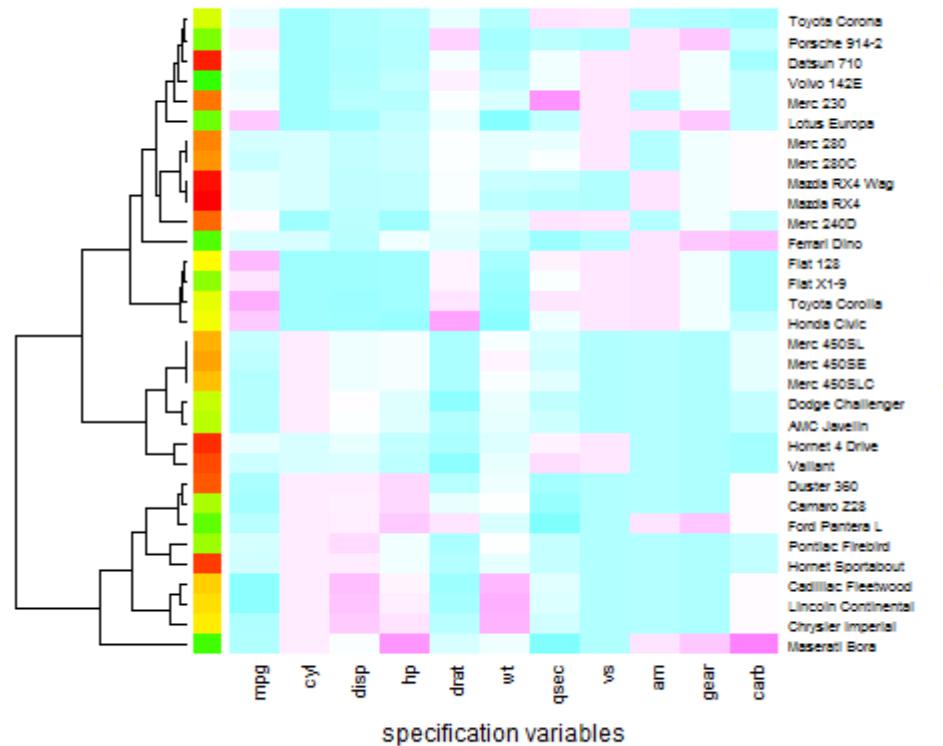


```
utils::str(hv) # the two re-ordering index vectors
List of 4
$ rowInd: int [1:32] 31 17 16 15 5 25 29 24 7 6 ...
$ colInd: int [1:11] 2 9 8 11 6 5 10 7 1 4 ...
$ Rowv : NULL
$ Colv : NULL
```

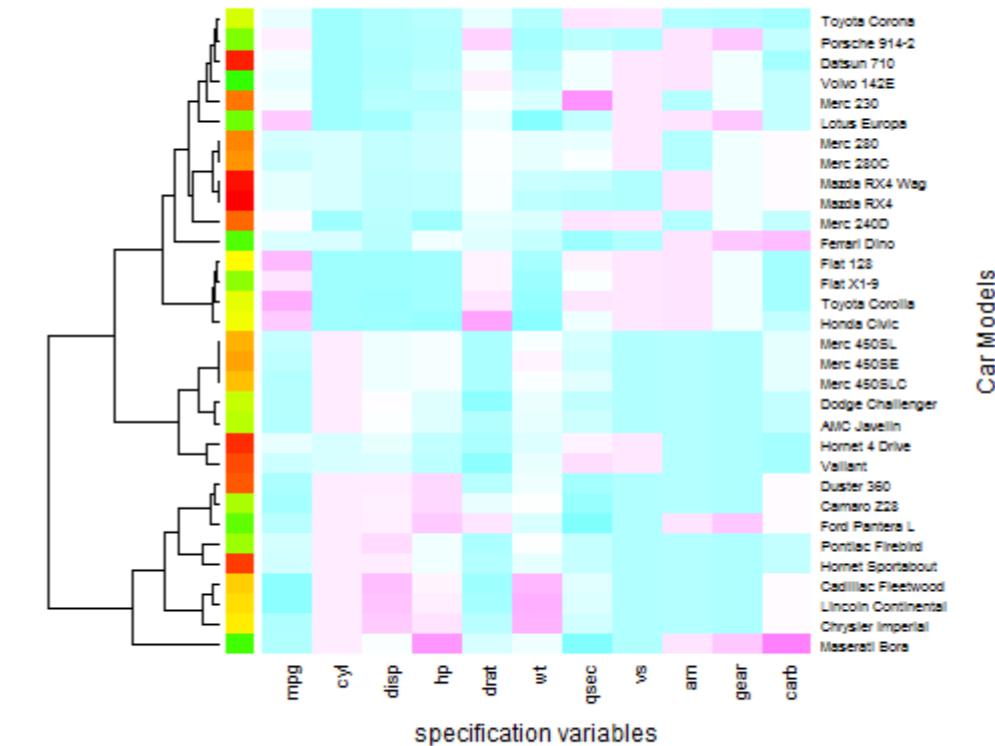
Example 2 (no column dendrogram (nor reordering) at all)

```
heatmap(x, Colv = NA, col = cm.colors(256), scale = "column",
 RowSideColors = rc, margins = c(5,10),
 xlab = "specification variables", ylab = "Car Models",
 main = "heatmap(<Mtcars data>, ..., scale = \"column\")")
```

```
heatmap(<Mtcars data>, ..., scale = "column")
```



```
heatmap(<Mtcars data>, ..., scale = "column")
```



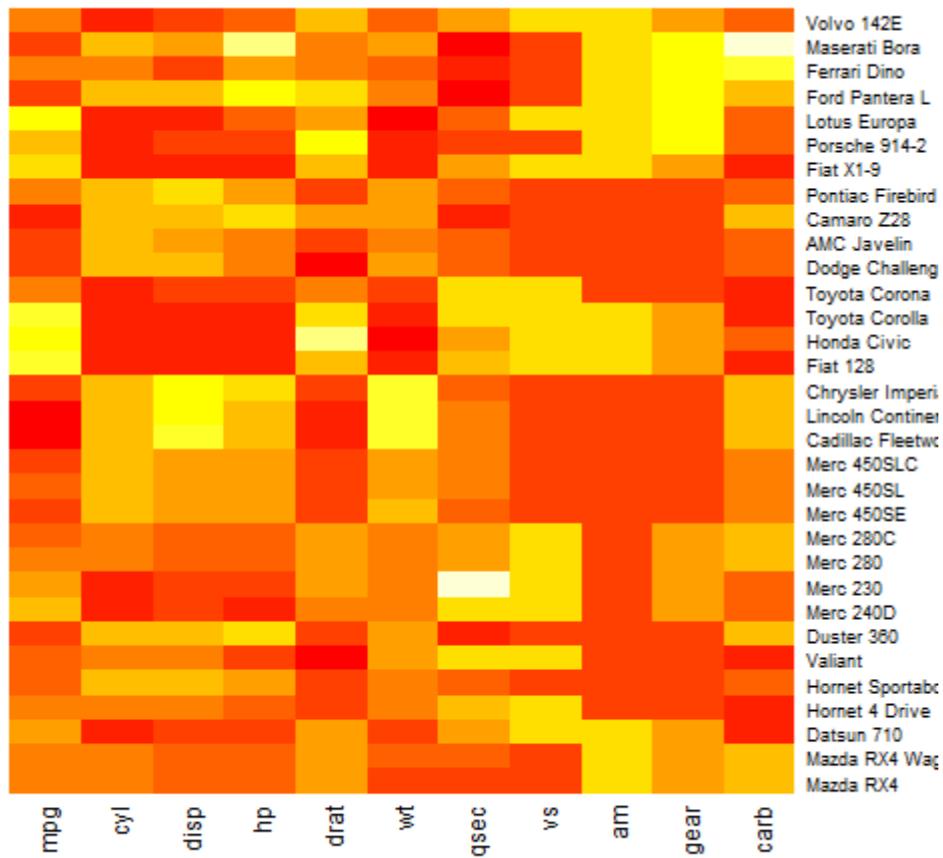
### 示例 3 ("无无")

```
heatmap(x, Rowv = NA, Colv = NA, scale = "column",
 main = "heatmap(*, NA, NA) ~ image(t(x))")
```

### Example 3 ("no nothing")

```
heatmap(x, Rowv = NA, Colv = NA, scale = "column",
 main = "heatmap(*, NA, NA) ~ image(t(x))")
```

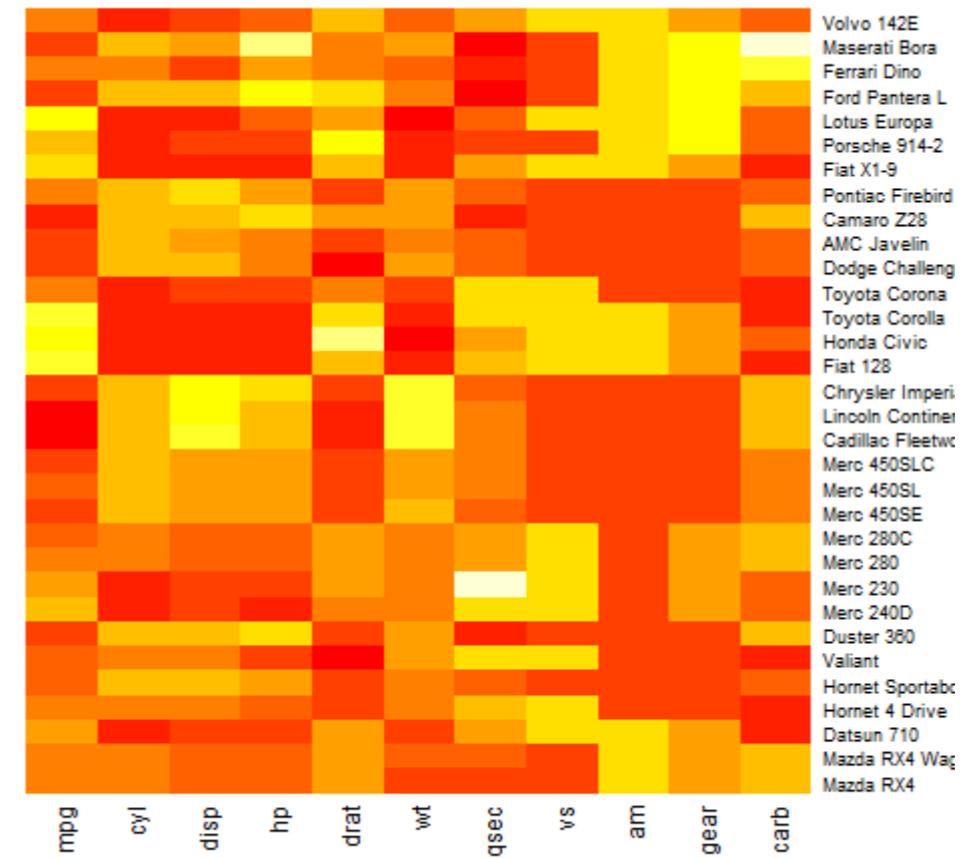
`heatmap(*, NA, NA) ~-= image(t(x))`



#### 示例 4 (使用 `reorder()`)

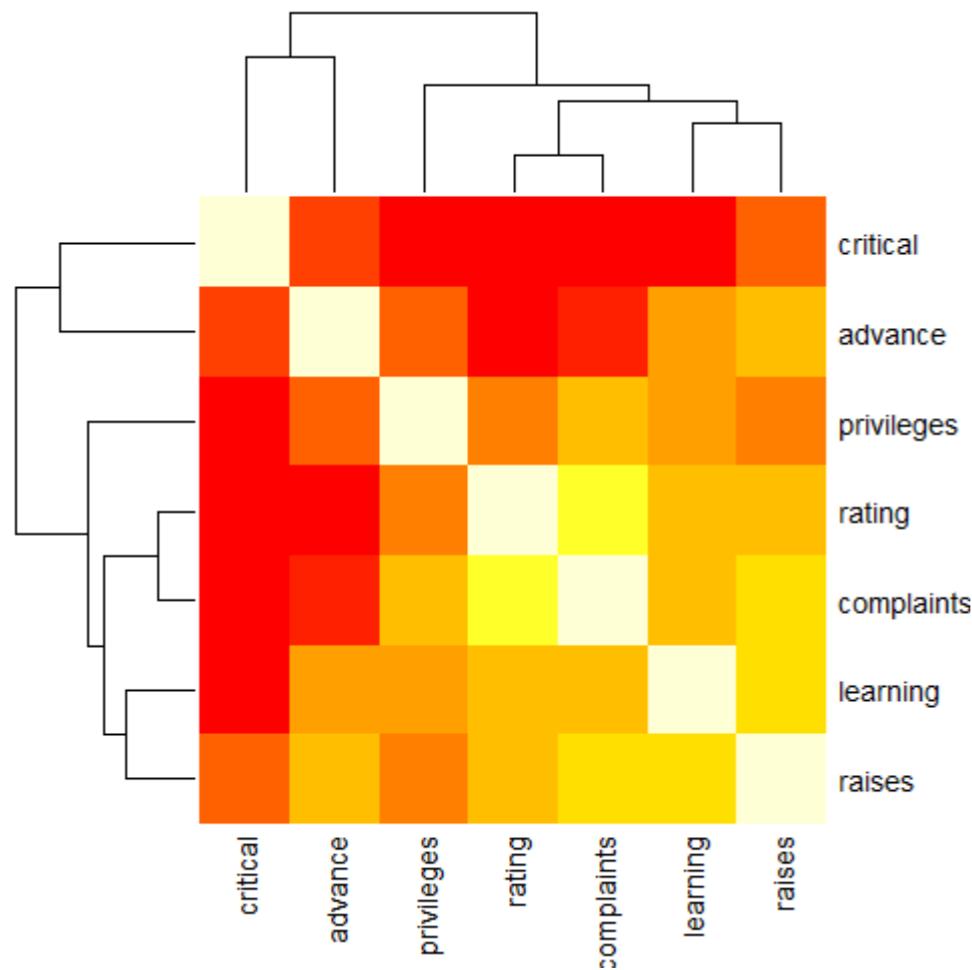
```
round(Ca <- cor(attitude), 2)
评分 投诉 权利 学习 加薪 批评 晋升
评分 1.00 0.83 0.43 0.62 0.59 0.16 0.16
投诉 0.83 1.00 0.56 0.60 0.67 0.19 0.22
权利 0.43 0.56 1.00 0.49 0.45 0.15 0.34
学习 0.62 0.60 0.49 1.00 0.64 0.12 0.53
加薪 0.59 0.67 0.45 0.64 1.00 0.38 0.57
批评 0.16 0.19 0.15 0.12 0.38 1.00 0.28
晋升 0.16 0.22 0.34 0.53 0.57 0.28 1.00
symnum(Ca) # 简单图形
rt cm p l rs cr a
评分 1
投诉 + 1
权利 . . 1
学习 , . . 1
加薪 . , . , 1
批评 . . 1
晋升 . . . 1
attr(,"legend")
[1] 0 '' 0.3 '.' 0.6 ',' 0.8 '+' 0.9 '*' 0.95 'B' 1
heatmap(Ca, symm = TRUE, margins = c(6,6))
```

`heatmap(*, NA, NA) ~-= image(t(x))`



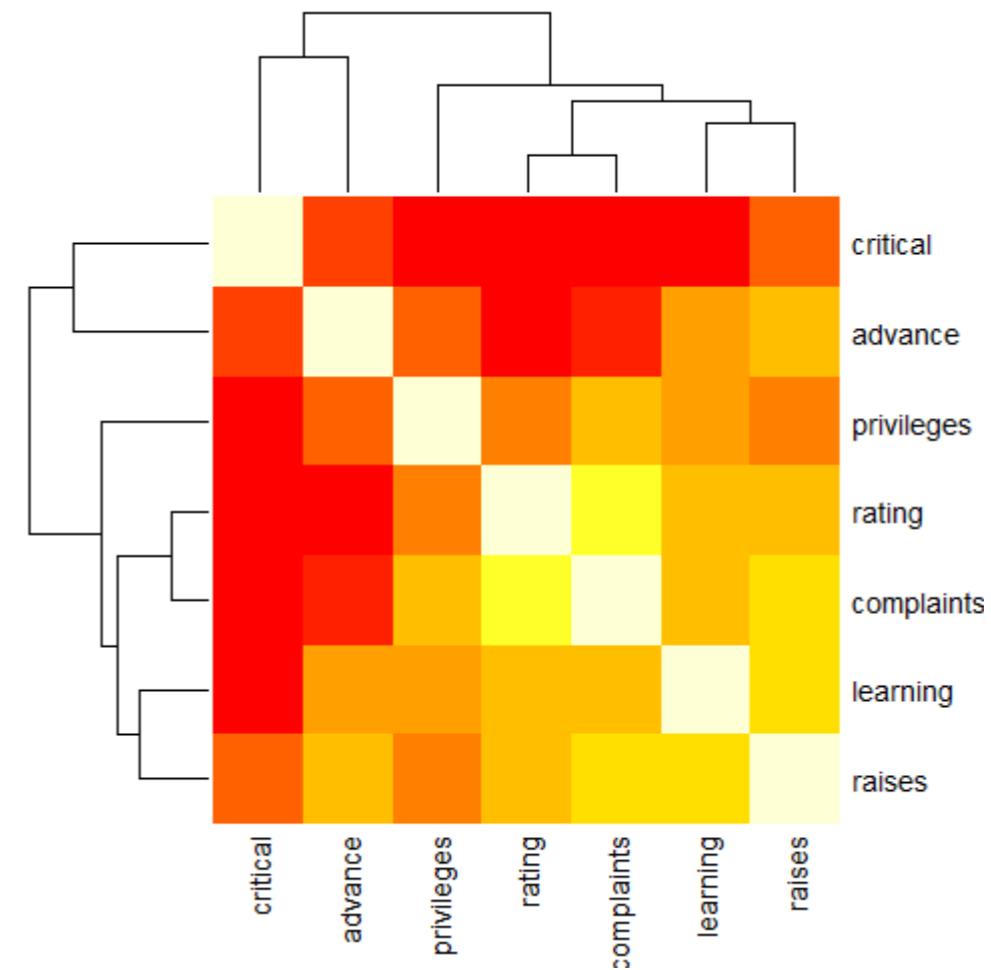
#### Example 4 (with `reorder()`)

```
round(Ca <- cor(attitude), 2)
rating complaints privileges learning raises critical advance
rating 1.00 0.83 0.43 0.62 0.59 0.16 0.16
complaints 0.83 1.00 0.56 0.60 0.67 0.19 0.22
privileges 0.43 0.56 1.00 0.49 0.45 0.15 0.34
learning 0.62 0.60 0.49 1.00 0.64 0.12 0.53
raises 0.59 0.67 0.45 0.64 1.00 0.38 0.57
critical 0.16 0.19 0.15 0.12 0.38 1.00 0.28
advance 0.16 0.22 0.34 0.53 0.57 0.28 1.00
symnum(Ca) # simple graphic
rt cm p l rs cr a
rating 1
complaints + 1
privileges . . 1
learning , . . 1
raises . , . , 1
critical . . 1
advance . . . 1
attr(,"legend")
[1] 0 '' 0.3 '.' 0.6 ',' 0.8 '+' 0.9 '*' 0.95 'B' 1
heatmap(Ca, symm = TRUE, margins = c(6,6))
```



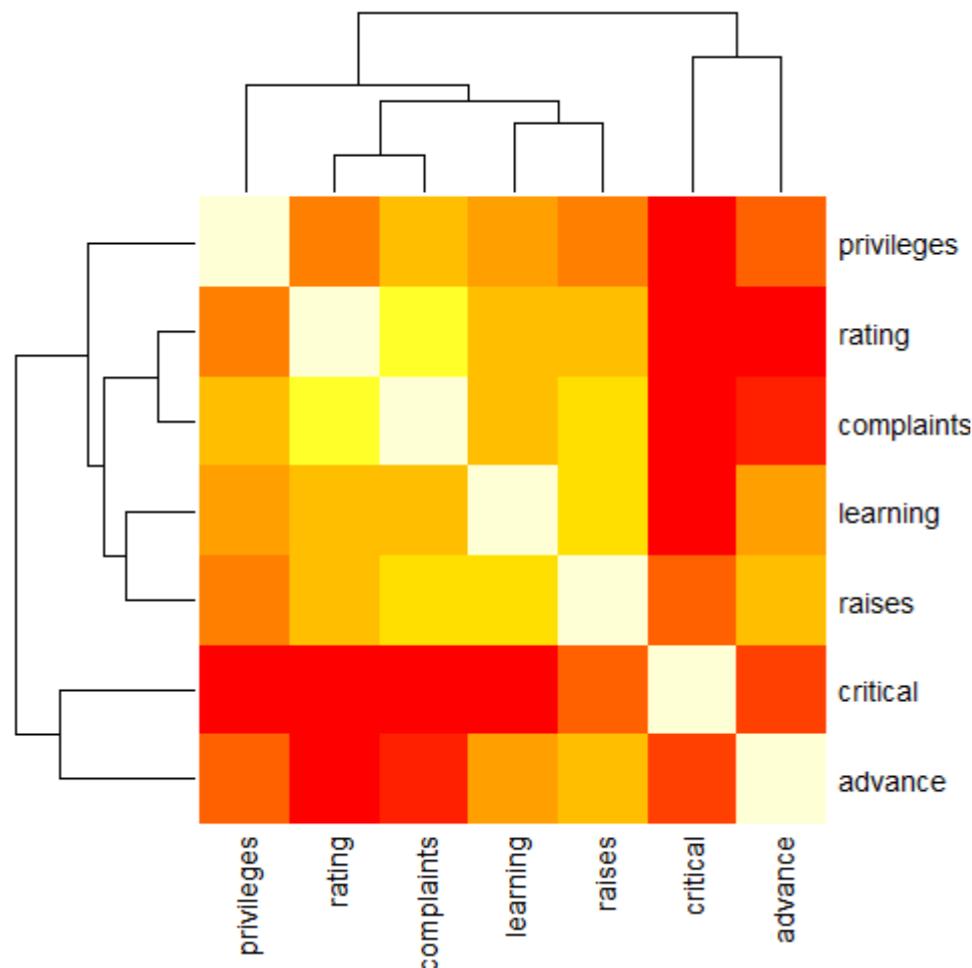
示例 5 (不使用 reorder())

```
heatmap(Ca, 行聚类 = FALSE, 对称 = TRUE, 边距 = c(6,6))
```



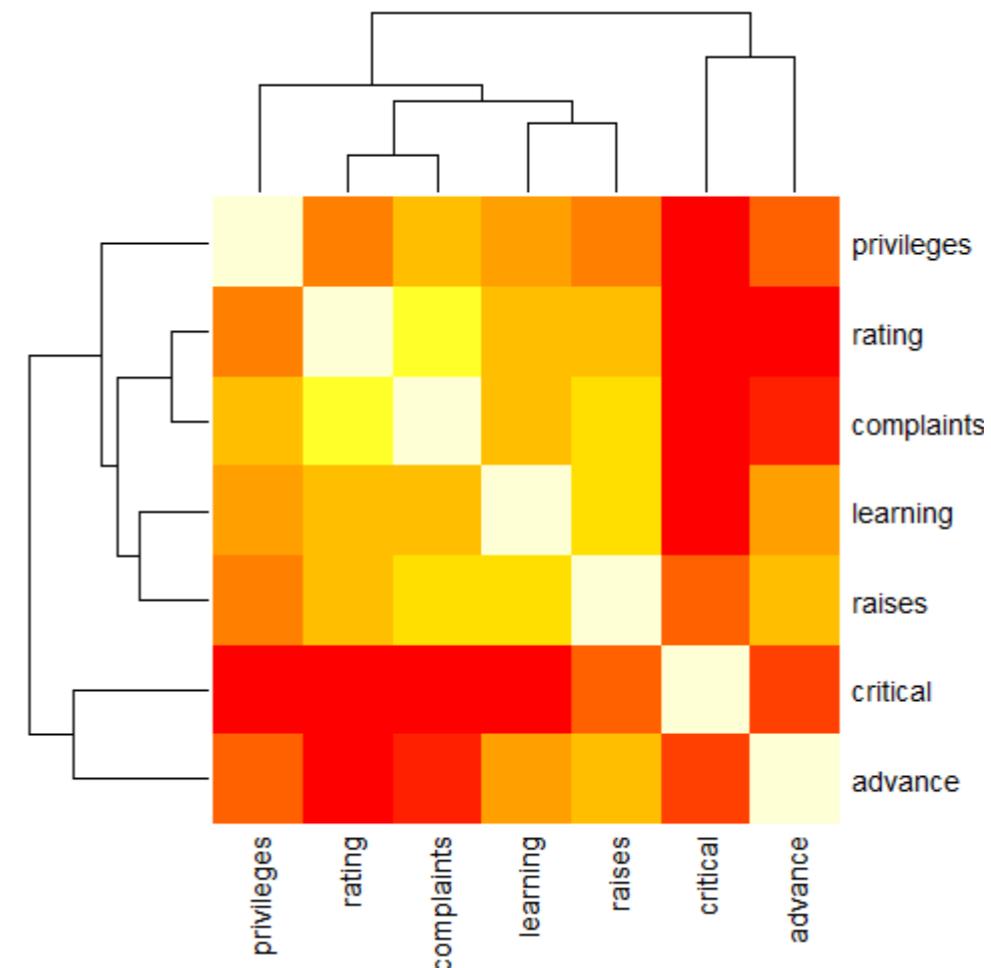
Example 5 (NO reorder())

```
heatmap(Ca, Rowv = FALSE, symm = TRUE, margins = c(6,6))
```



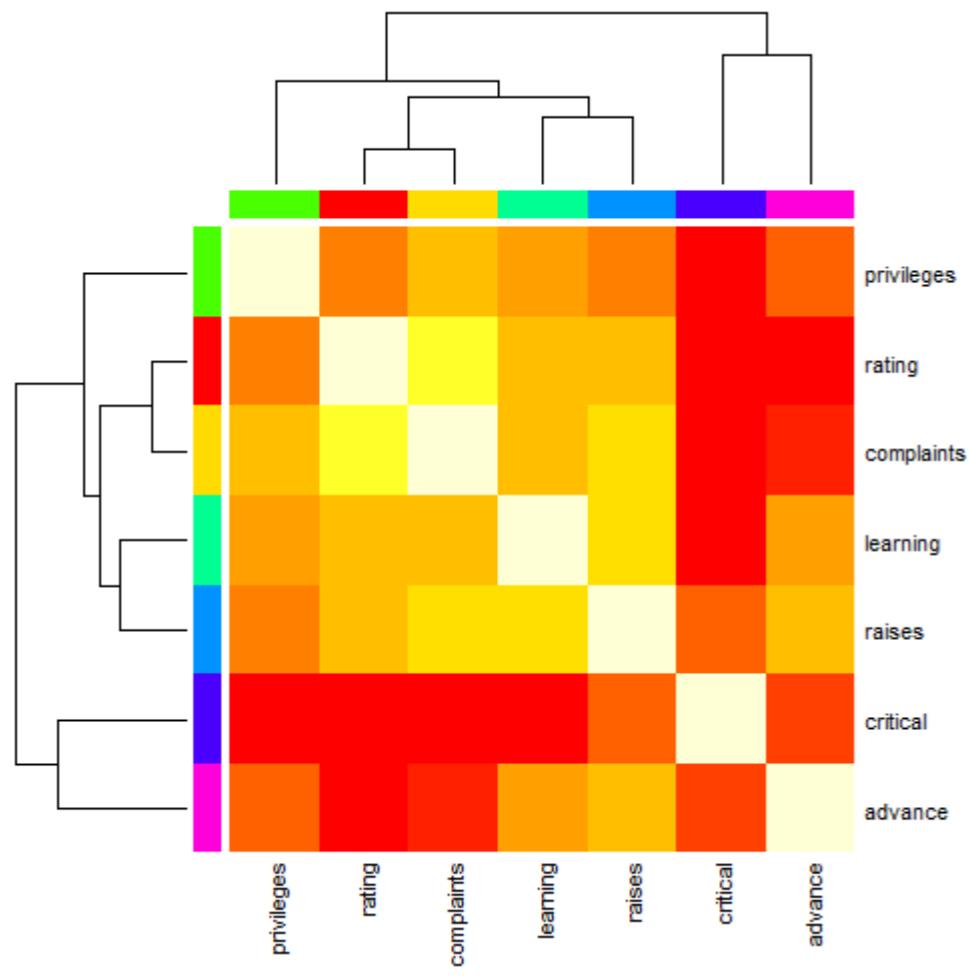
示例 6 (带颜色条的稍微人工示例, 无排序)

```
cc <- rainbow(行数(Ca))
heatmap(Ca, 行聚类 = FALSE, 对称 = TRUE, 行侧颜色 = cc, 列侧颜色 = cc,
 边距 = c(6, 6))
```



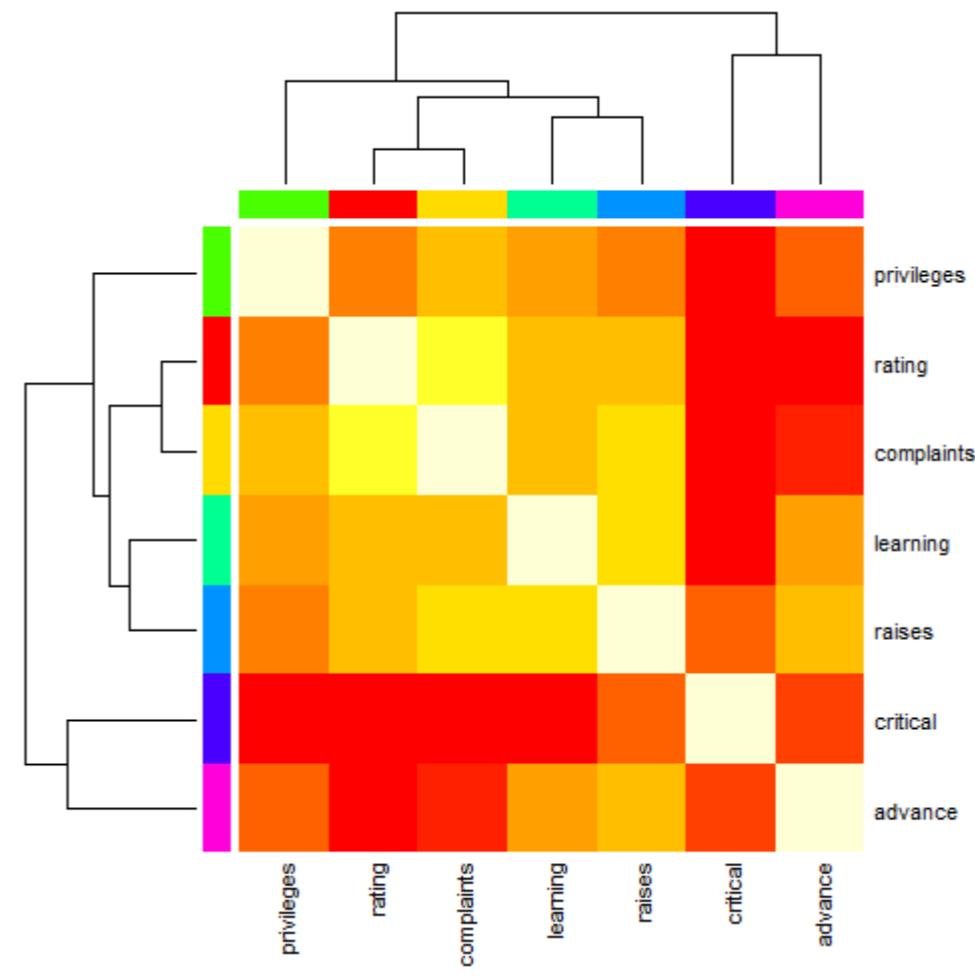
Example 6 (slightly artificial with color bar, without ordering)

```
cc <- rainbow(nrow(Ca))
heatmap(Ca, Rowv = FALSE, symm = TRUE, RowSideColors = cc, ColSideColors = cc,
 margins = c(6, 6))
```



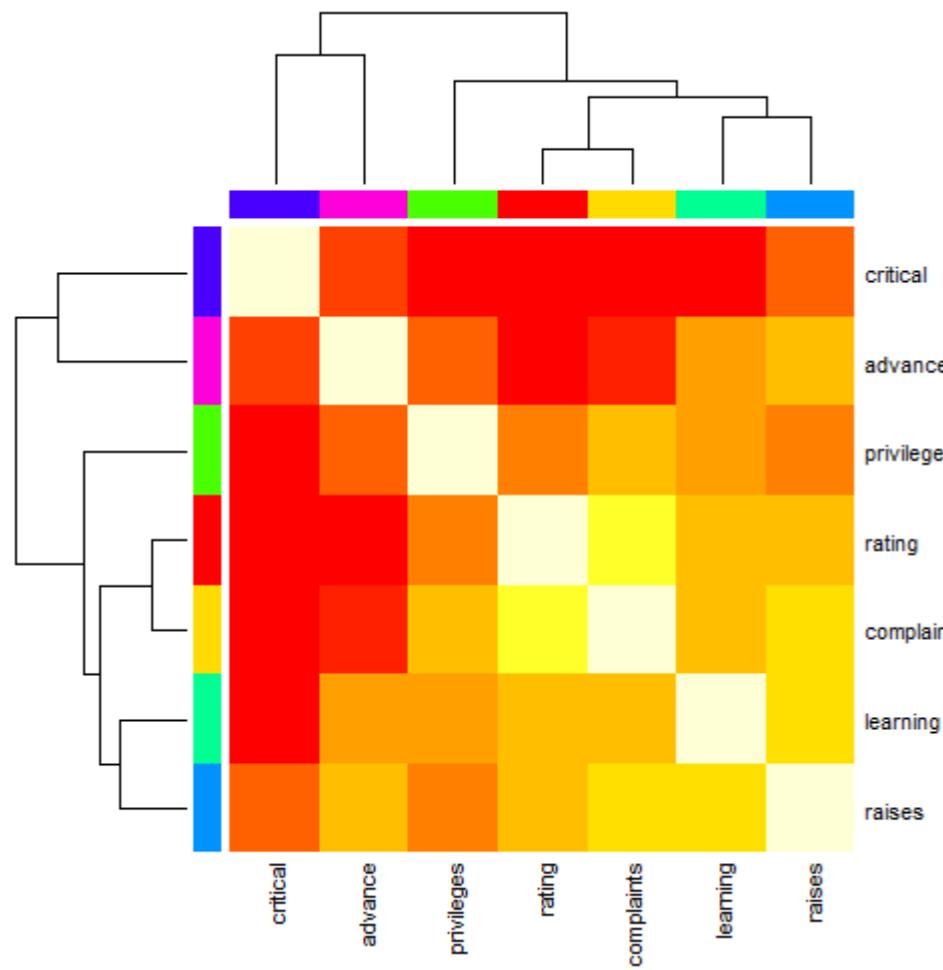
示例 7 (带颜色条的稍微人工示例, 有排序)

```
heatmap(Ca, 对称 = TRUE, 行侧颜色 = cc, 列侧颜色 = cc,
 边距 = c(6, 6))
```



Example 7 (slightly artificial with color bar, with ordering)

```
heatmap(Ca, symm = TRUE, RowSideColors = cc, ColSideColors = cc,
 margins = c(6, 6))
```



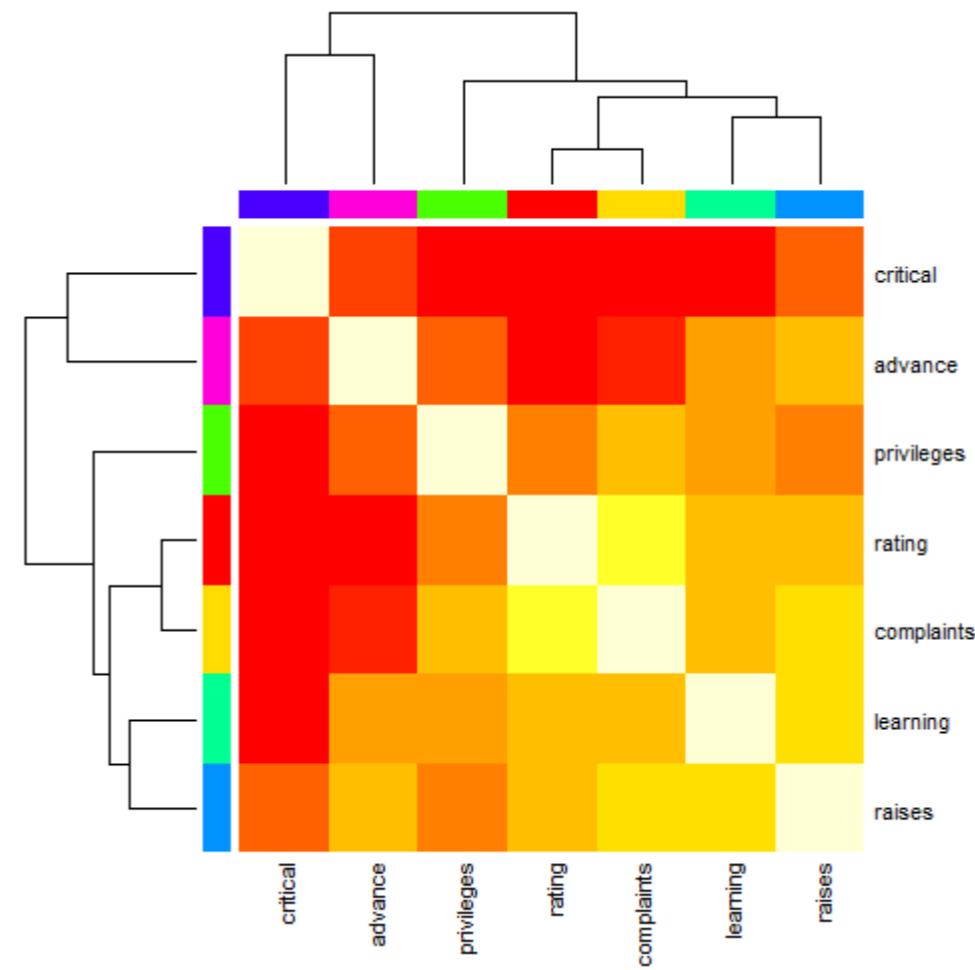
示例 8 (用于变量聚类, 建议使用基于 cor() 的距离)

```

symnum(cU <- cor(USJudgeRatings))
CO I DM DI CF DE PR F O W PH R
CONT 1
INTG 1
DMNR B 1
DILG + + 1
CFMG + + B 1
DECI + + B B 1
PREP + + B B B 1
FAMI + + B * * B 1
ORAL * * B B * B B 1
WRIT * + B * * B B B 1
PHYS , , + + + + + + + 1
RTEN * * * * * B * B B * 1
attr("legend")
[1] 0 ' 0.3 ' 0.6 ' 0.8 '+' 0.9 '*' 0.95 'B' 1

hU <- heatmap(cU, Rowv = FALSE, symm = TRUE, col = topo.colors(16),
 distfun = function(c) as.dist(1 - c), keep.dendro = TRUE)

```



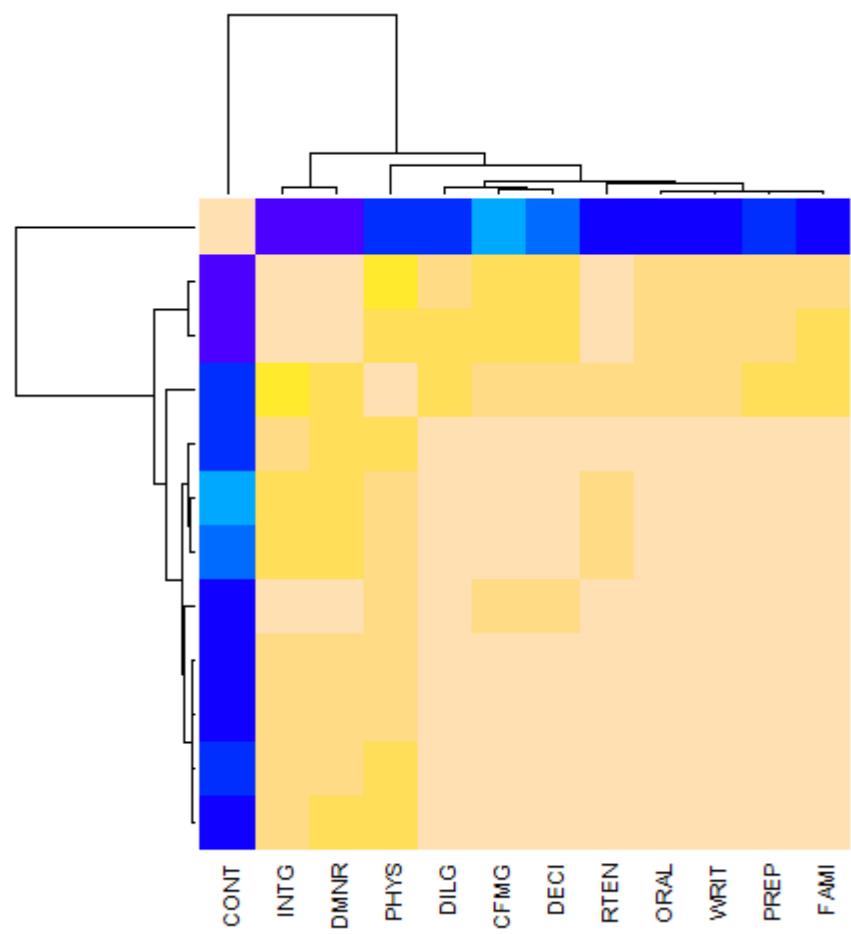
Example 8 (For variable clustering, rather use distance based on cor())

```

symnum(cU <- cor(USJudgeRatings))
CO I DM DI CF DE PR F O W PH R
CONT 1
INTG 1
DMNR B 1
DILG + + 1
CFMG + + B 1
DECI + + B B 1
PREP + + B B B 1
FAMI + + B * * B 1
ORAL * * B B * B B 1
WRIT * + B * * B B B 1
PHYS , , + + + + + + + 1
RTEN * * * * * B * B B * 1
attr("legend")
[1] 0 ' 0.3 ' 0.6 ' 0.8 '+' 0.9 '*' 0.95 'B' 1

hU <- heatmap(cU, Rowv = FALSE, symm = TRUE, col = topo.colors(16),
 distfun = function(c) as.dist(1 - c), keep.dendro = TRUE)

```

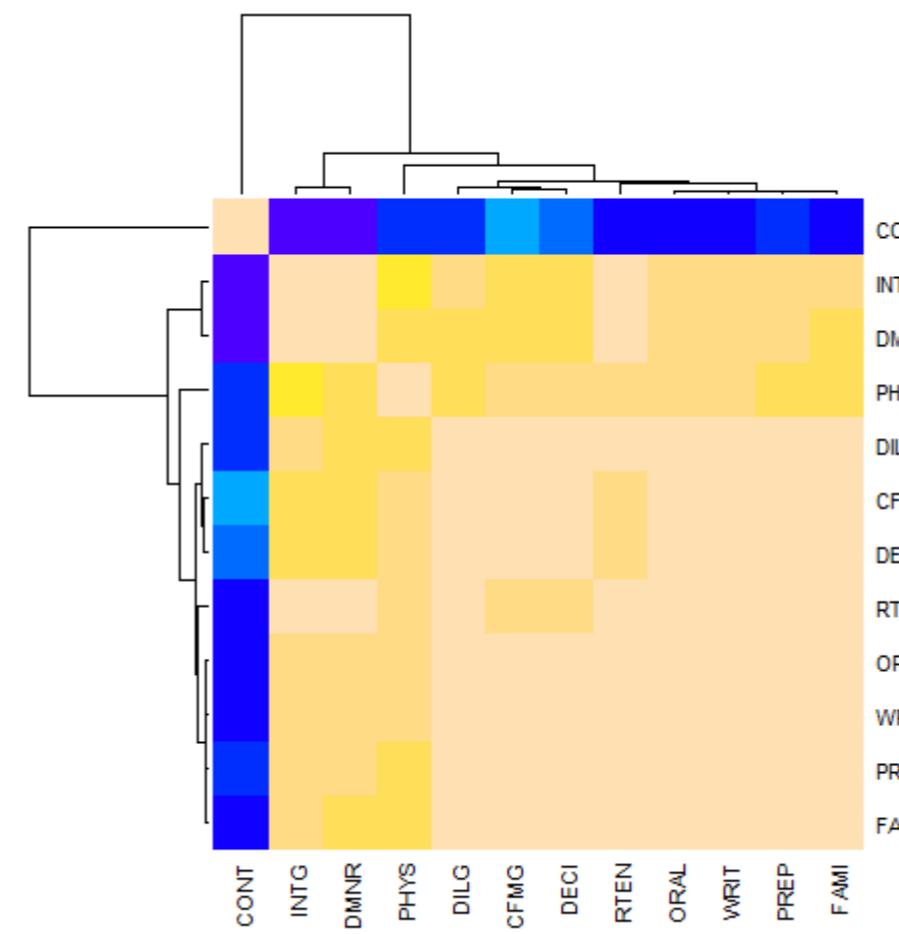


```

具有相同重新排序的相关矩阵：
round(100 * cU[hU[[1]], hU[[2]]])
CONT INTG DMNR PHYS DILG CFMG DECI RTEN ORAL WRIT PREP FAMI
CONT 100 -13 -15 5 1 14 9 -3 -1 -4 1 -3
INTG -13 100 96 74 87 81 80 94 91 91 88 87
DMNR -15 96 100 79 84 81 80 94 91 89 86 84
PHYS 5 74 79 100 81 88 87 91 89 86 85 84
DILG 1 87 84 81 100 96 96 93 95 96 98 96
CFMG 14 81 81 88 96 100 98 93 95 94 96 94
DECI 9 80 80 87 96 98 100 92 95 95 96 94
RTEN -3 94 94 91 93 93 92 100 98 97 95 94
ORAL -1 91 91 89 95 95 95 98 100 99 98 98
WRIT -4 91 89 86 96 94 95 97 99 100 99 99
PREP 1 88 86 85 98 96 96 95 98 99 100 99
FAMI -3 87 84 84 96 94 94 98 99 99 100

列树状图：
utils::str(hU$Colv)
--[树状图, 2个分支, 12个成员, 高度 = 1.15]
|--叶子节点 "CONT"
`-[树状图, 2个分支, 11个成员, 高度 = 0.258]
|--[树状图, 2个分支, 2个成员, 高度 = 0.0354]
| |--叶子节点 "INTG"
| |--叶子节点 "DMNR"
`-[树状图, 2个分支, 9个成员, 高度 = 0.187]
| |--叶子节点 "PHYS"
|-[树状图, 2个分支, 8个成员, 高度 h = 0.075]
|-[树状图, 2个分支, 3个成员, 高度 h = 0.0438]
| |--叶节点 "DILG"
|`-[带有2个分支和2个成员的树状图, h = 0.0189]
| |--叶子 "CFMG"
| |`--leaf "DECI"
|`--leaf "DECI"

```



```

The Correlation matrix with same reordering:
round(100 * cU[hU[[1]], hU[[2]]])
CONT INTG DMNR PHYS DILG CFMG DECI RTEN ORAL WRIT PREP FAMI
CONT 100 -13 -15 5 1 14 9 -3 -1 -4 1 -3
INTG -13 100 96 74 87 81 80 94 91 91 88 87
DMNR -15 96 100 79 84 81 80 94 91 89 86 84
PHYS 5 74 79 100 81 88 87 91 89 86 85 84
DILG 1 87 84 81 100 96 96 93 95 96 98 96
CFMG 14 81 81 88 96 100 98 93 95 94 96 94
DECI 9 80 80 87 96 98 100 92 95 95 96 94
RTEN -3 94 94 91 93 93 92 100 98 97 95 94
ORAL -1 91 91 89 95 95 95 98 100 99 98 98
WRIT -4 91 89 86 96 94 95 97 99 100 99 99
PREP 1 88 86 85 98 96 96 95 98 99 100 99
FAMI -3 87 84 84 96 94 94 98 99 99 100

The column dendrogram:
utils::str(hU$Colv)
--[dendrogram w/ 2 branches and 12 members at h = 1.15]
|--leaf "CONT"
`-[dendrogram w/ 2 branches and 11 members at h = 0.258]
|--[dendrogram w/ 2 branches and 2 members at h = 0.0354]
| |--leaf "INTG"
| |--leaf "DMNR"
`-[dendrogram w/ 2 branches and 9 members at h = 0.187]
| |--leaf "PHYS"
|-[dendrogram w/ 2 branches and 8 members at h = 0.075]
|-[dendrogram w/ 2 branches and 3 members at h = 0.0438]
| |--leaf "DILG"
|`-[dendrogram w/ 2 branches and 2 members at h = 0.0189]
| |--leaf "CFMG"
| |`--leaf "DECI"
|`--leaf "DECI"

```

```

`--[树状图, 2个分支, 5个成员, 高度 h = 0.0584]
|--叶子 "RTEN"
'--[树状图, 2个分支, 4个成员, 高度 h = 0.0187]
|--[树状图, 2个分支, 2个成员, 高度 h = 0.00657]
| |--节点 "ORAL"
| `--leaf "WRIT"
'--[树状图, 包含2个分支和2个成员, 距离h = 0.0101]
|--叶子 "PREP"
`--leaf "FAMI"

```

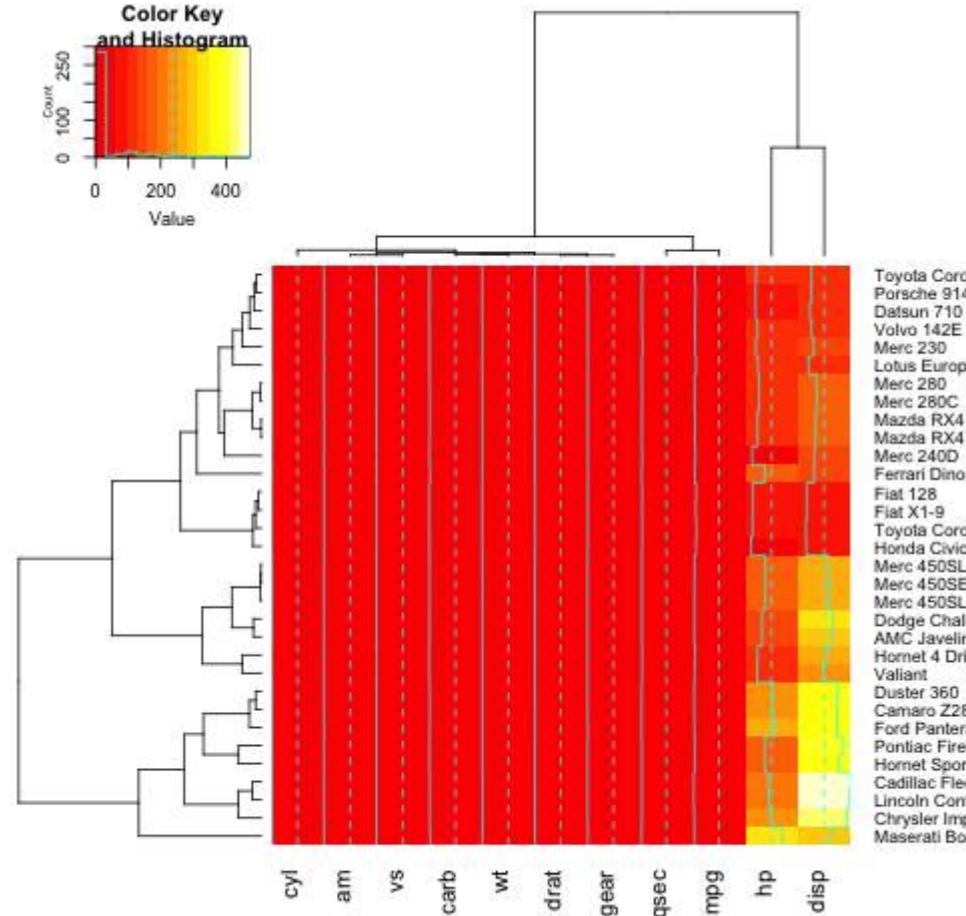
## 第85.2节：heatmap.2中的调节参数

已知：

```
x <- as.matrix(mtcars)
```

可以使用 heatmap.2 —— 一个更近期优化版本的 heatmap，通过加载以下库：

```
require(gplots)
heatmap.2(x)
```



要为热图添加标题、x 轴或 y 轴标签，需要设置 main、xlab 和 ylab：

```
heatmap.2(x, main = "我的主标题：汽车特征概览", xlab="汽车特征", ylab = "汽车品牌")
```

如果想为热图定义自定义颜色调色板，可以使用  
colorRampPalette函数设置col参数：

```

`--[dendrogram w/ 2 branches and 5 members at h = 0.0584]
|--leaf "RTEN"
'--[dendrogram w/ 2 branches and 4 members at h = 0.0187]
|--[dendrogram w/ 2 branches and 2 members at h = 0.00657]
| |--leaf "ORAL"
| `--leaf "WRIT"
'--[dendrogram w/ 2 branches and 2 members at h = 0.0101]
|--leaf "PREP"
`--leaf "FAMI"

```

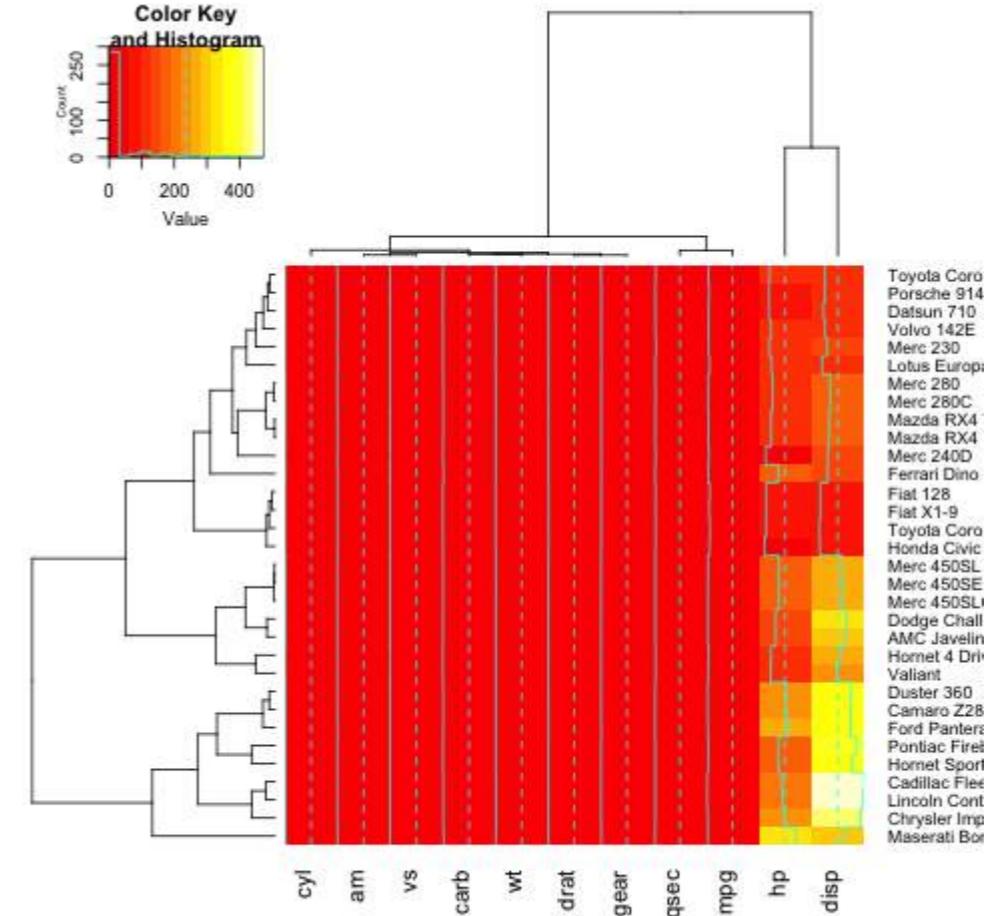
## Section 85.2: Tuning parameters in heatmap.2

Given:

```
x <- as.matrix(mtcars)
```

One can use heatmap.2 - a more recent optimized version of [heatmap](#), by loading the following library:

```
require(gplots)
heatmap.2(x)
```

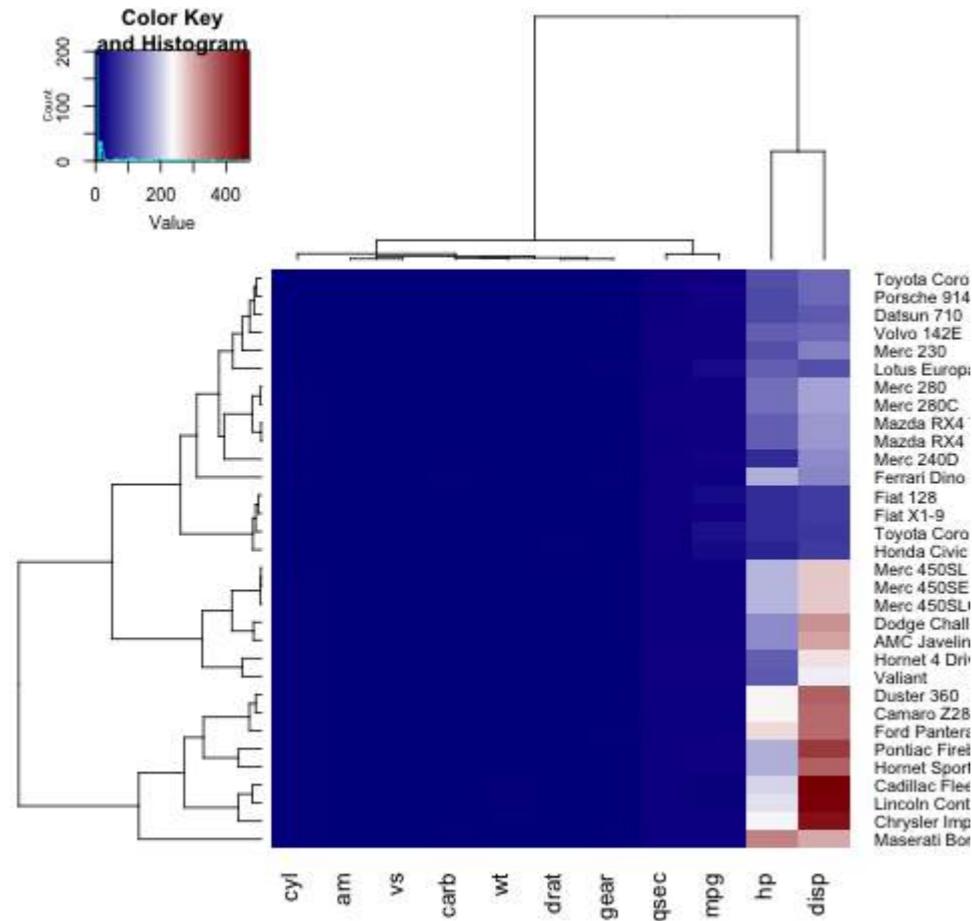


To add a title, x- or y-label to your heatmap, you need to set the main, xlab and ylab:

```
heatmap.2(x, main = "My main title: Overview of car features", xlab="Car features", ylab = "Car brands")
```

If you wish to define your own color palette for your heatmap, you can set the col parameter by using the [colorRampPalette](#) function:

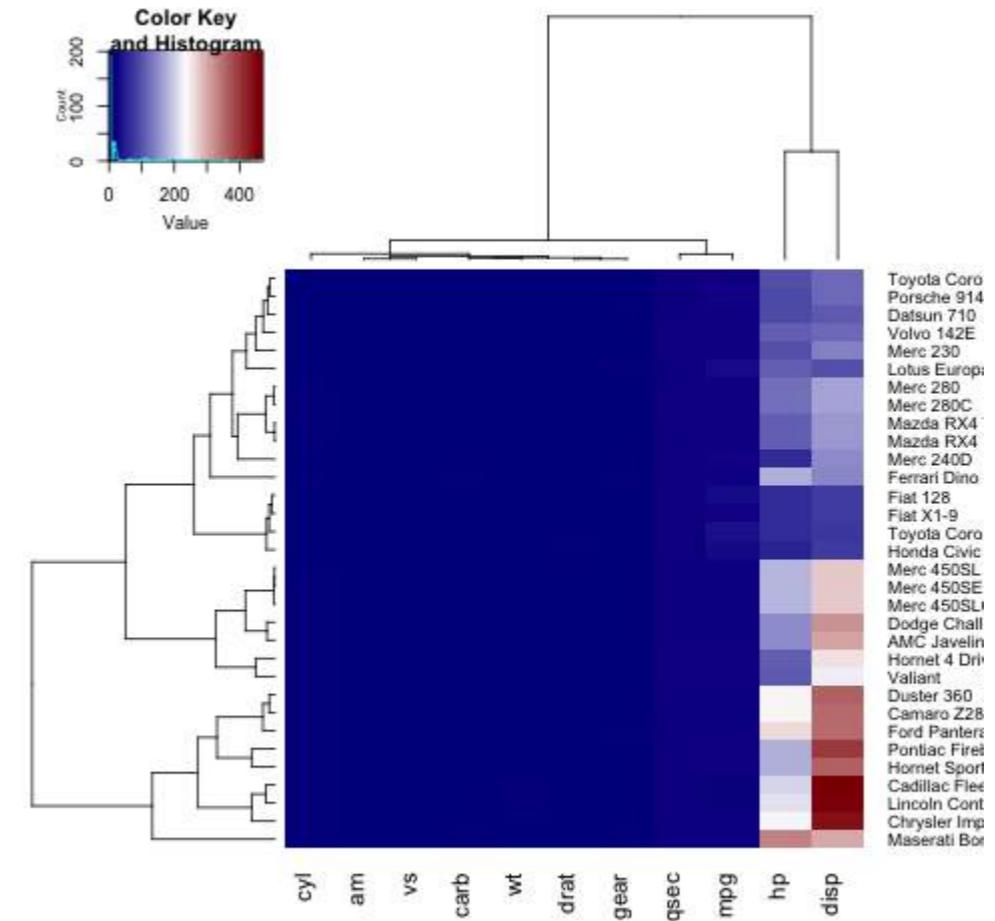
```
heatmap.2(x, trace="none", key=TRUE, Colv=FALSE, dendrogram = "row", col = colorRampPalette(c("darkblue", "white", "darkred"))(100))
```



如你所见，y 轴上的标签（汽车名称）无法完全显示在图中。为了解决这个问题，用户可以调整margins参数：

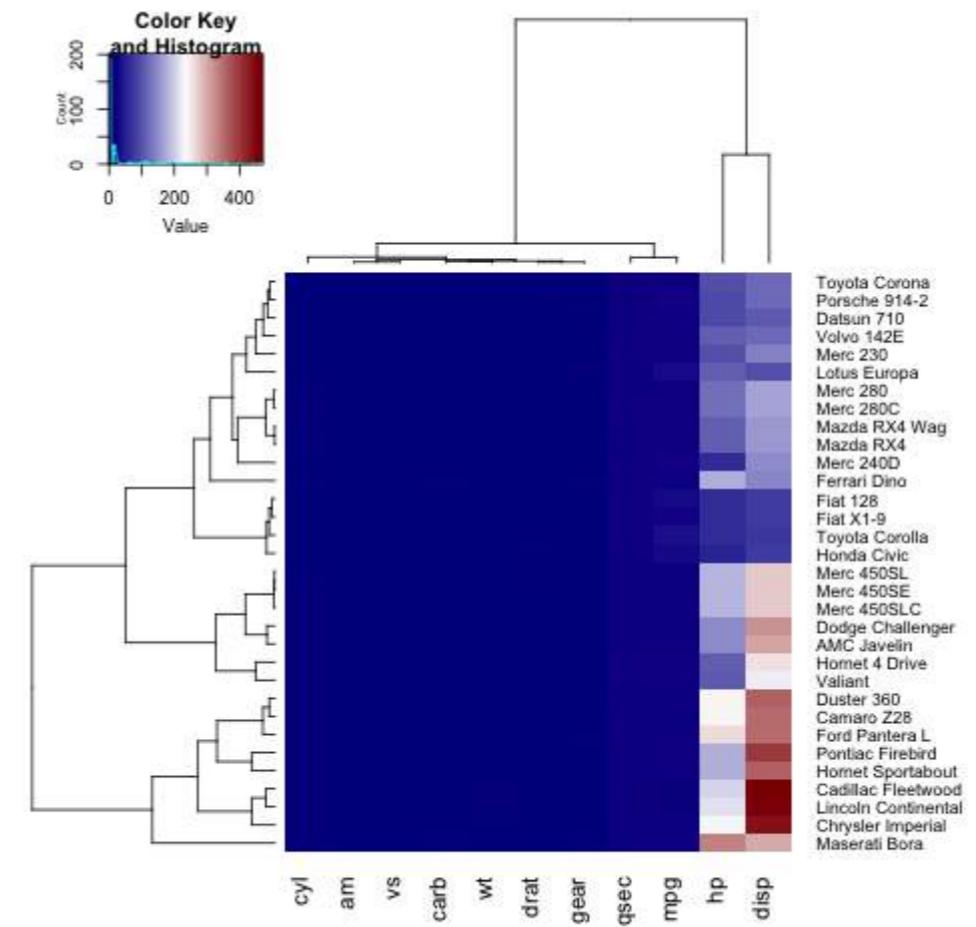
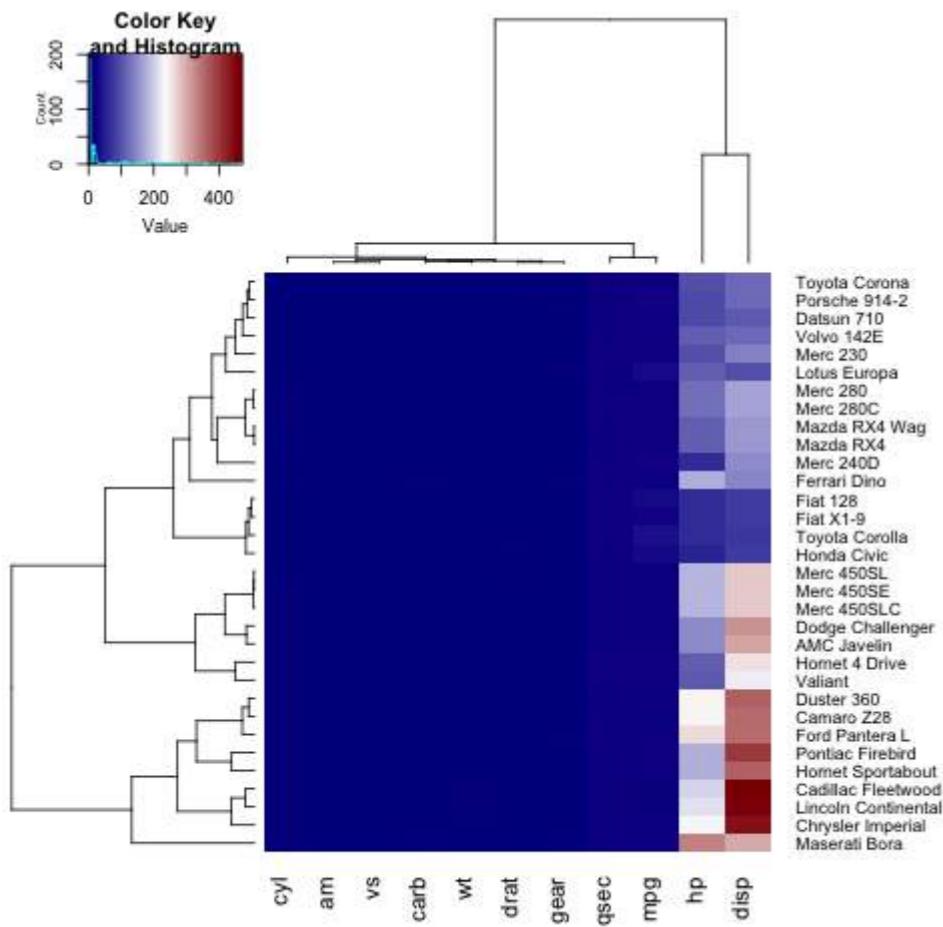
```
heatmap.2(x, trace="none", key=TRUE, col = colorRampPalette(c("darkblue", "white", "darkred"))(100), margins=c(5,8))
```

```
heatmap.2(x, trace="none", key=TRUE, Colv=FALSE, dendrogram = "row", col = colorRampPalette(c("darkblue", "white", "darkred"))(100))
```



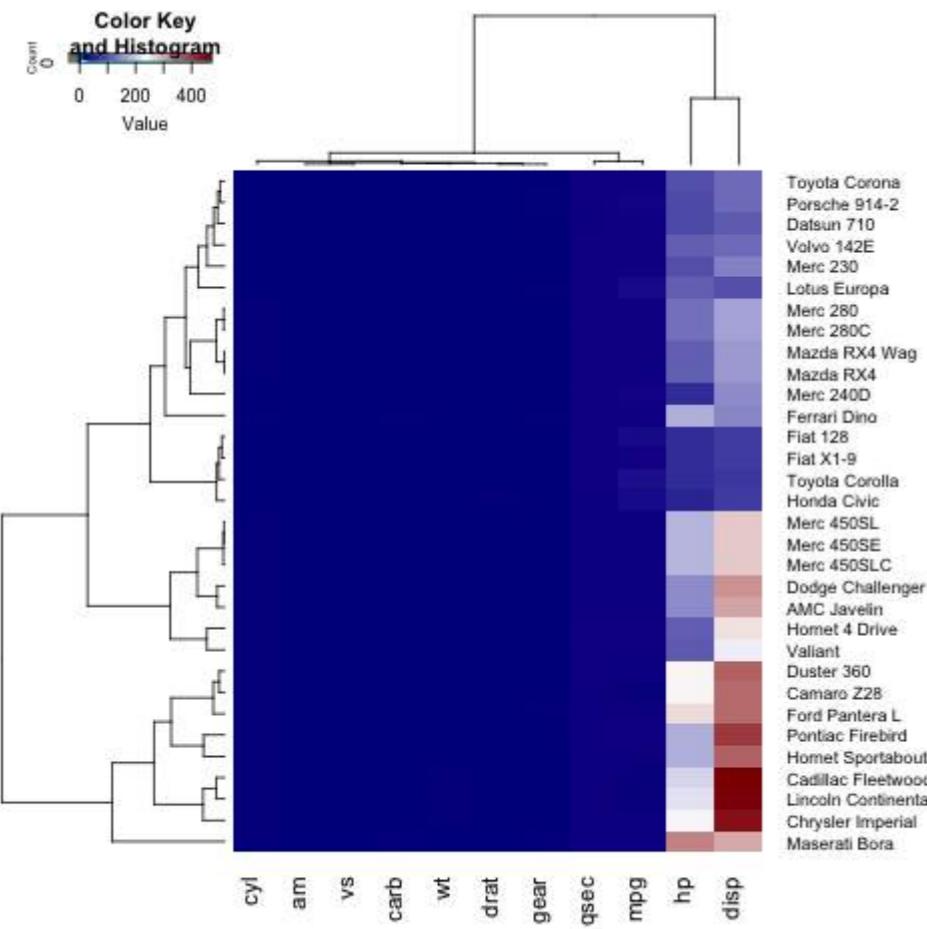
As you can notice, the labels on the y axis (the car names) don't fit in the figure. In order to fix this, the user can tune the margins parameter:

```
heatmap.2(x, trace="none", key=TRUE, col = colorRampPalette(c("darkblue", "white", "darkred"))(100), margins=c(5,8))
```



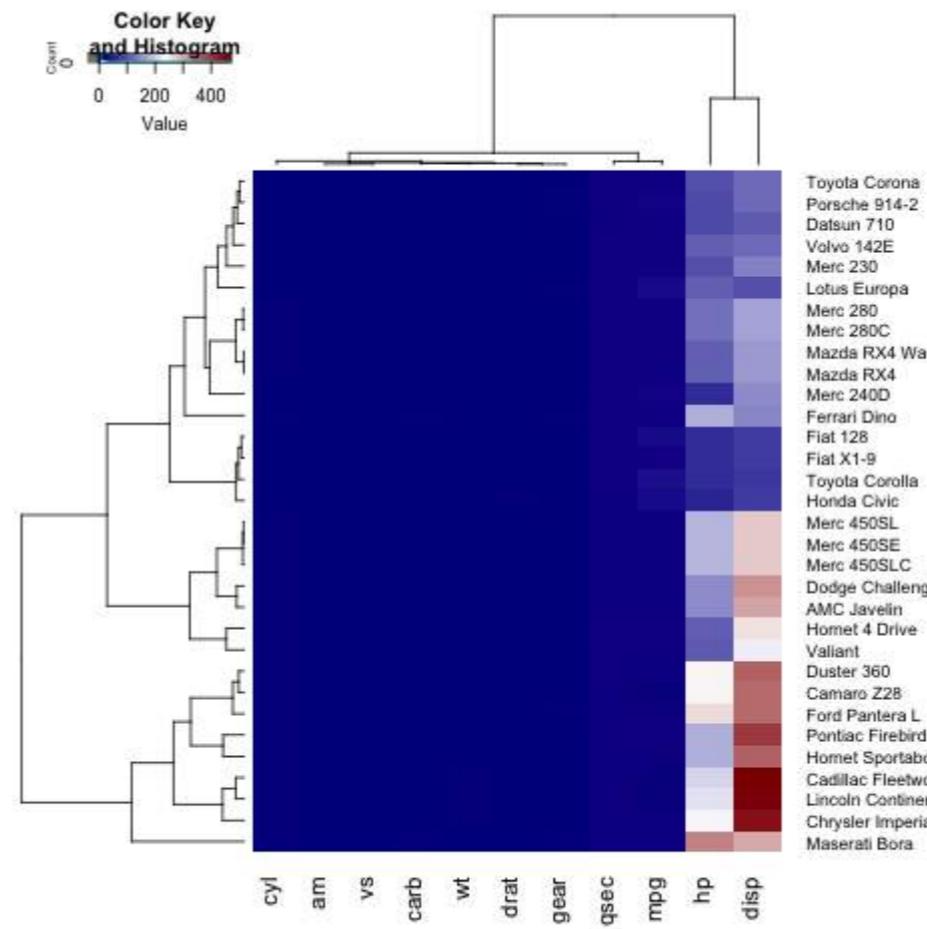
此外，我们可以通过调整hei和wid来改变热图各部分的尺寸（关键直方图、树状图和热图本身）：

Further, we can change the dimensions of each section of our heatmap (the key histogram, the dendograms and the heatmap itself), by tuning hei and wid :



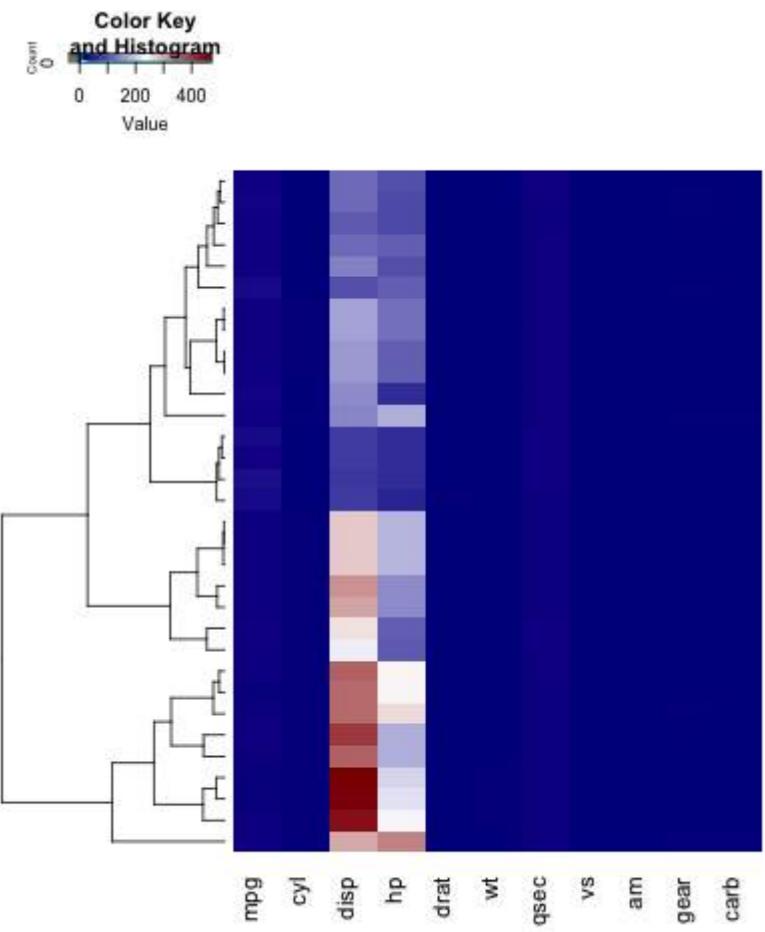
如果我们只想显示行（或列）树状图，需要设置Colv=FALSE（或Rowv=FALSE）并调整dendrogram参数：

```
heatmap.2(x, trace="none", key=TRUE, Colv=FALSE, dendrogram = "row", col =
colorRampPalette(c("darkblue", "white", "darkred"))(100), margins=c(5,8), lwid = c(5,15), lhei =
c(3,15))
```



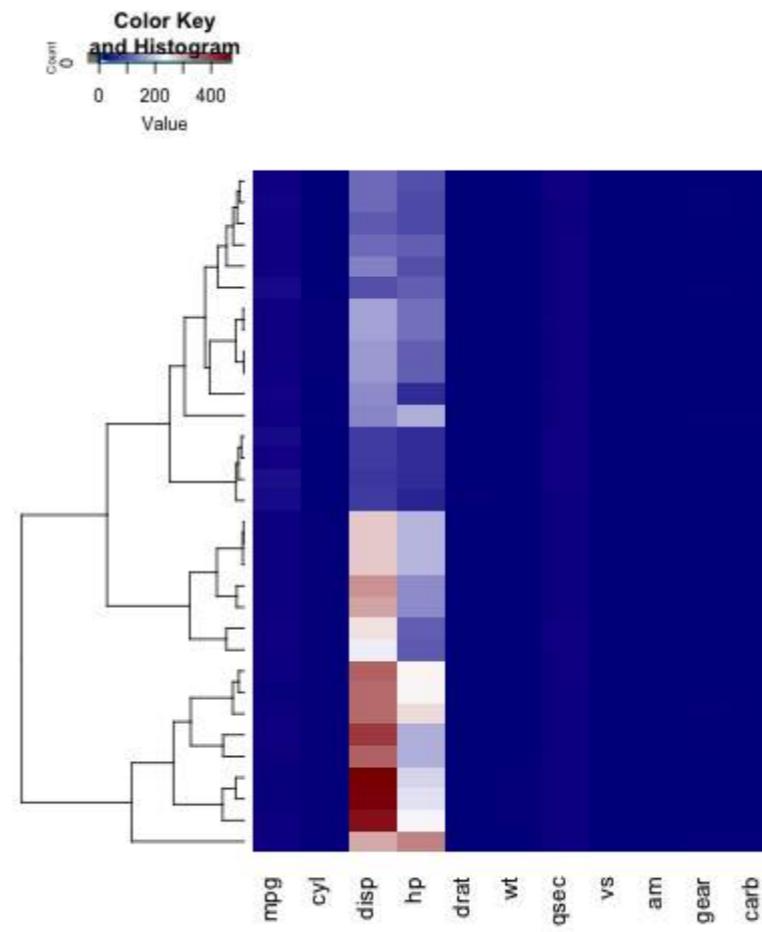
If we only want to show a row(or column) dendrogram, we need to set Colv=FALSE (or Rowv=FALSE) and adjust the dendrogram parameter:

```
heatmap.2(x, trace="none", key=TRUE, Colv=FALSE, dendrogram = "row", col =
colorRampPalette(c("darkblue", "white", "darkred"))(100), margins=c(5,8), lwid = c(5,15), lhei =
c(3,15))
```



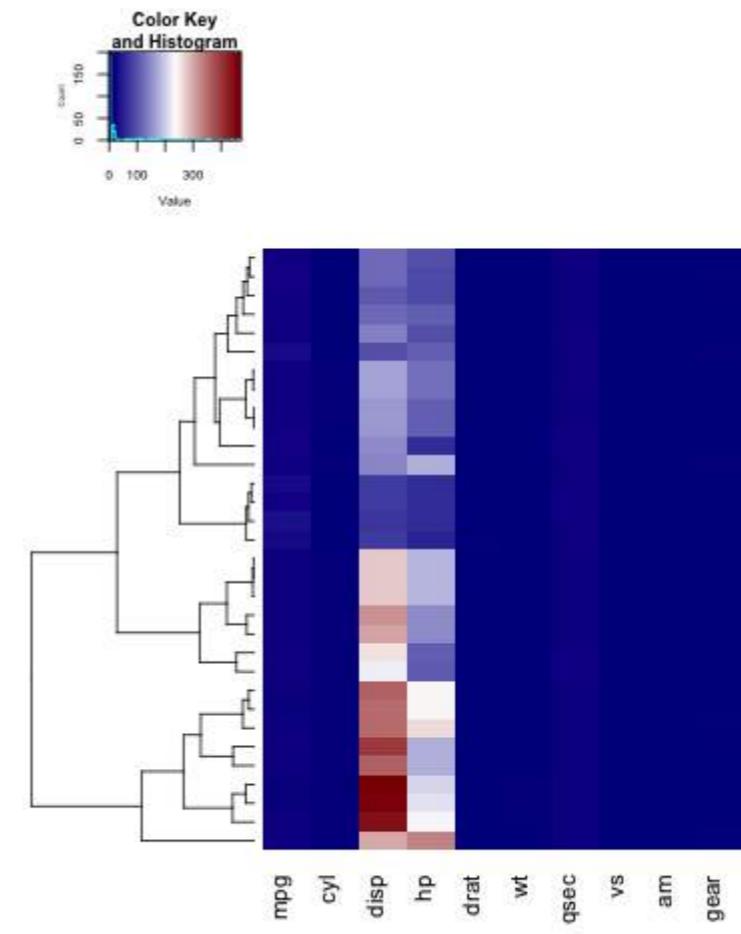
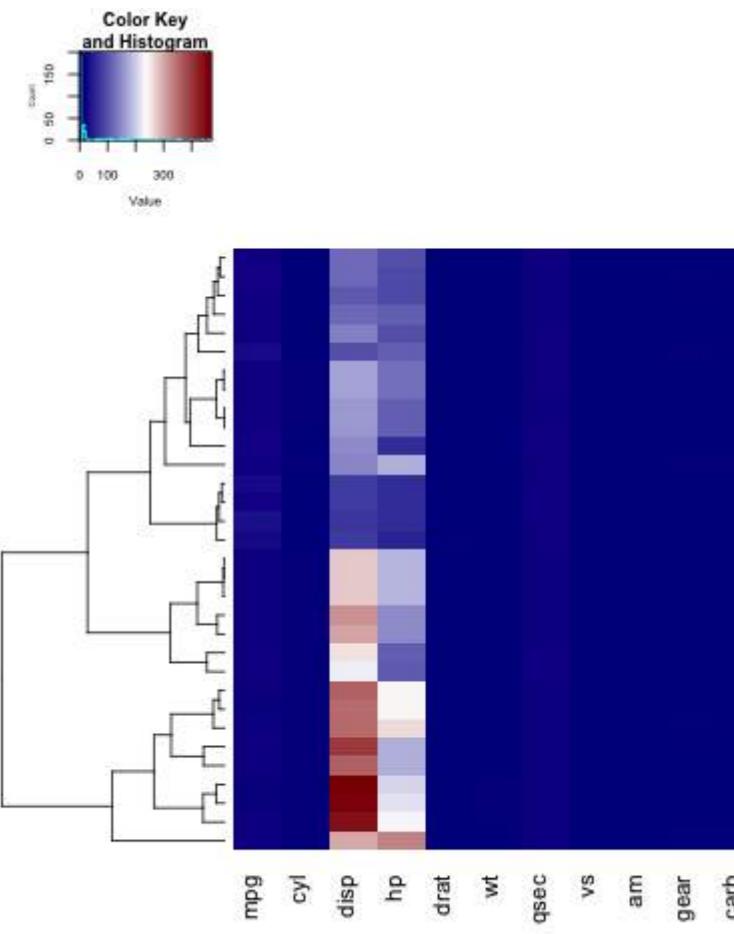
要更改图例标题、标签和坐标轴的字体大小，用户需要在par列表中设置cex.main, cex.lab, cex.axis：

```
par(cex.main=1, cex.lab=0.7, cex.axis=0.7)
heatmap.2(x, trace="none", key=TRUE, Colv=FALSE, dendrogram = "row", col =
colorRampPalette(c("darkblue","white","darkred"))(100), margins=c(5,8), lwid = c(5,15), lhei =
c(5,15))
```



For changing the font size of the legend title, labels and axis, the user needs to set cex.main, cex.lab, cex.axis in the [par](#) list:

```
par(cex.main=1, cex.lab=0.7, cex.axis=0.7)
heatmap.2(x, trace="none", key=TRUE, Colv=FALSE, dendrogram = "row", col =
colorRampPalette(c("darkblue","white","darkred"))(100), margins=c(5,8), lwid = c(5,15), lhei =
c(5,15))
```



# 第86章：使用igraph包进行网络分析

## 第86.1节：简单有向和无向网络图

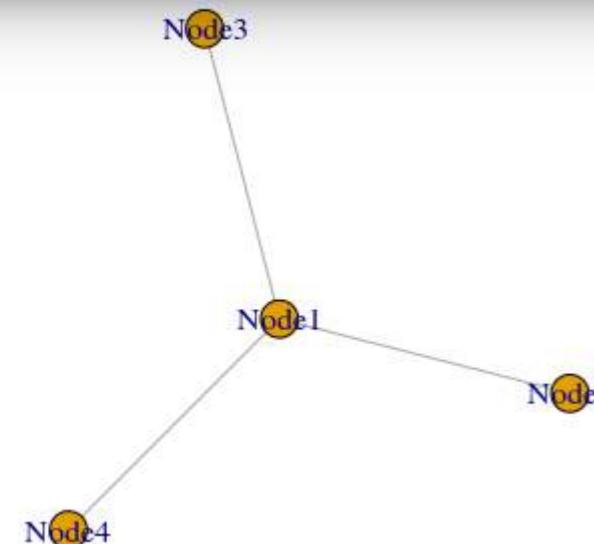
R语言的igraph包是一个非常棒的工具，可以用来简单地模拟真实和虚拟的网络。这个示例旨在演示如何使用R v. 3.2.3中的igraph包创建两个简单的网络图。

### 无向网络

该网络通过以下代码创建：

```
g<-graph.formula(节点1-节点2, 节点1-节点3, 节点4-节点1)
plot(g)
```

```
> g<-graph.formula(Node1-Node2, Node1-Node3, Node4-Node1)
> plot(g)
> █
```



### 有向网络

```
dg<-graph.formula(汤姆-+玛丽, 汤姆-+比尔, 汤姆-+山姆, 苏-+玛丽, 比尔-+苏)
plot(dg)
```

这段代码将生成带箭头的网络：

# Chapter 86: Network analysis with the igraph package

## Section 86.1: Simple Directed and Non-directed Network Graphing

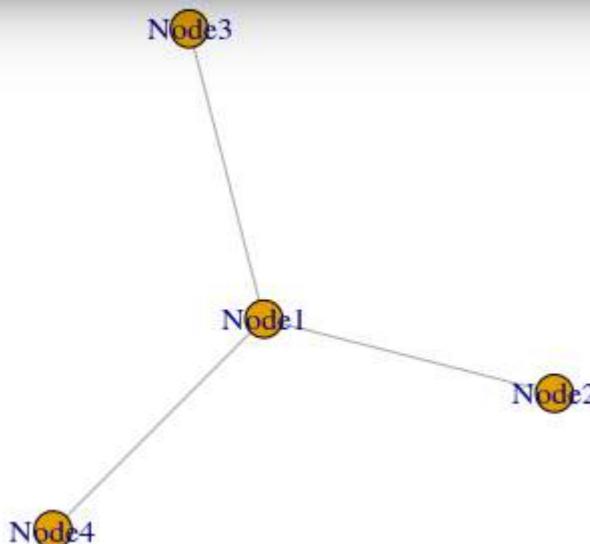
The igraph package for R is a wonderful tool that can be used to model networks, both real and virtual, with simplicity. This example is meant to demonstrate how to create two simple network graphs using the igraph package within R v.3.2.3.

### Non-Directed Network

The network is created with this piece of code:

```
g<-graph.formula(Node1-Node2, Node1-Node3, Node4-Node1)
plot(g)
```

```
> g<-graph.formula(Node1-Node2, Node1-Node3, Node4-Node1)
> plot(g)
> █
```

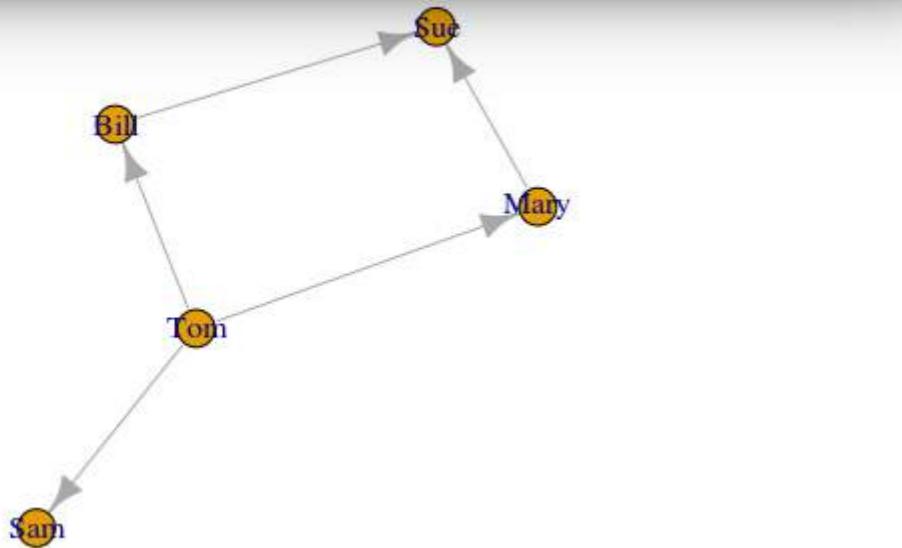


### Directed Network

```
dg<-graph.formula(Tom-+Mary, Tom-+Bill, Tom-+Sam, Sue-+Mary, Bill-+Sue)
plot(dg)
```

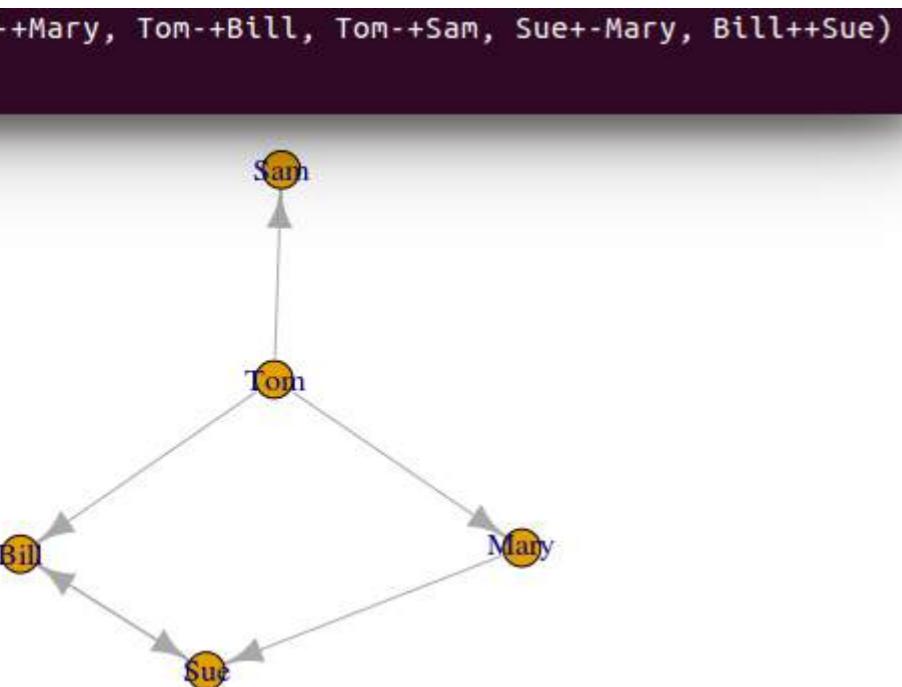
This code will then generate a network with arrows:

```
> dg<-graph.formula(Tom+Mary, Tom+Bill, Tom+Sam, Sue+Mary, Bill+Sue)
> plot(dg)
>
```

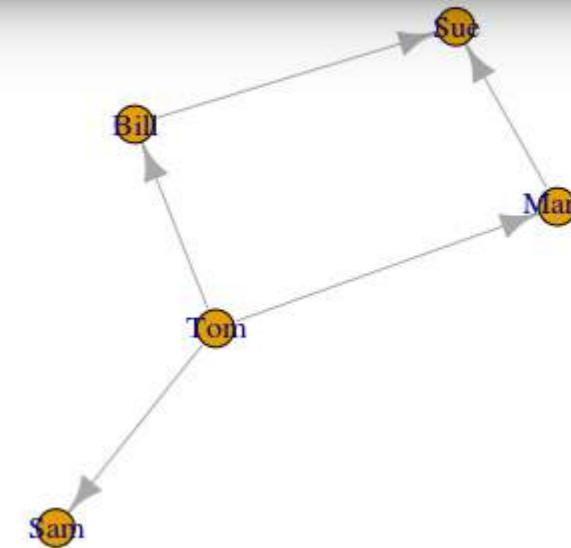


如何制作双向箭头的代码示例：

```
dg<-graph.formula(Tom+Mary, Tom+Bill, Tom+Sam, Sue+Mary, Bill++Sue)
plot(dg)
```

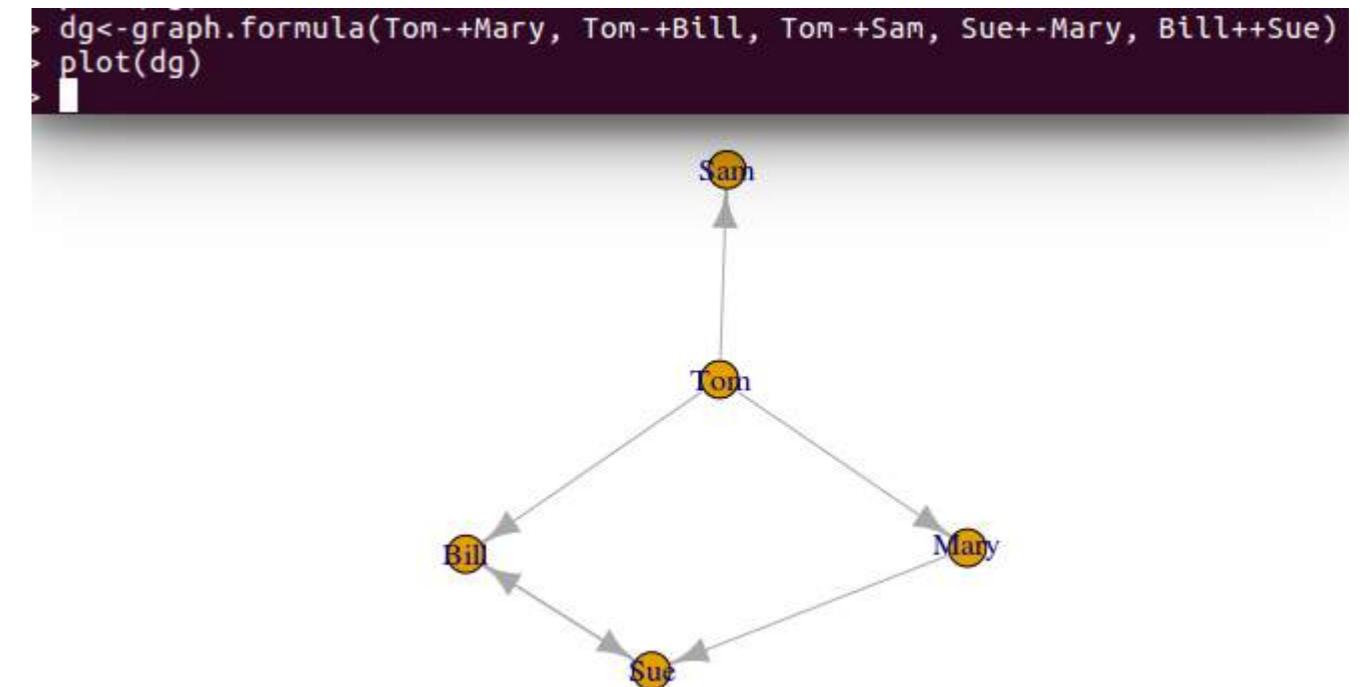


```
> dg<-graph.formula(Tom+Mary, Tom+Bill, Tom+Sam, Sue+Mary, Bill+Sue)
> plot(dg)
>
```



Code example of how to make a double sided arrow:

```
dg<-graph.formula(Tom+Mary, Tom+Bill, Tom+Sam, Sue+Mary, Bill++Sue)
plot(dg)
```



# 第87章：函数式编程

## 第87.1节：内置高阶函数

R拥有一组内置的高阶函数：Map、Reduce、Filter、Find、Position、Negate。

Map将给定函数应用于一个值列表：

```
words <- list("this", "is", "an", "example")
Map(toupper, words)
```

Reduce以递归方式连续将二元函数应用于值列表。

```
Reduce(`*`, 1:10)
```

Filter 给定一个谓词函数和一个值列表，返回一个仅包含谓词函数为 TRUE 的值的过滤列表。

```
Filter(is.character, list(1,"a",2,"b",3,"c"))
```

Find 给定一个谓词函数和一个值列表，返回谓词函数为TRUE的第一个值。

```
Find(is.character, list(1,"a",2,"b",3,"c"))
```

Position 给定一个谓词函数和一个值列表，返回列表中第一个使谓词函数为TRUE的值的位置。

```
Position(is.character, list(1,"a",2,"b",3,"c"))
```

Negate 反转一个谓词函数，使其对原本返回TRUE的值返回FALSE，反之亦然。

```
is.noncharacter <- Negate(is.character)
is.noncharacter("a")
is.noncharacter(mean)
```

# Chapter 87: Functional programming

## Section 87.1: Built-in Higher Order Functions

R has a set of built in higher order functions: [Map](#), [Reduce](#), [Filter](#), [Find](#), [Position](#), [Negate](#).

**Map** applies a given function to a list of values:

```
words <- list("this", "is", "an", "example")
Map(toupper, words)
```

**Reduce** successively applies a binary function to a list of values in a recursive fashion.

```
Reduce(`*`, 1:10)
```

**Filter** given a predicate function and a list of values returns a filtered list containing only values for whom predicate function is TRUE.

```
Filter(is.character, list(1,"a",2,"b",3,"c"))
```

**Find** given a predicate function and a list of values returns the first value for which the predicate function is TRUE.

```
Find(is.character, list(1,"a",2,"b",3,"c"))
```

**Position** given a predicate function and a list of values returns the position of the first value in the list for which the predicate function is TRUE.

```
Position(is.character, list(1,"a",2,"b",3,"c"))
```

**Negate** inverts a predicate function making it return FALSE for values where it returned TRUE and vice versa.

```
is.noncharacter <- Negate(is.character)
is.noncharacter("a")
is.noncharacter(mean)
```

# 第88章：获取用户输入

## 第88.1节：R中的用户输入

有时，用户与程序之间进行交互会很有趣，例如设计用来在R中教授R语言的 `swirl` 包。

可以使用 `readline` 命令请求用户输入：

```
name <- readline(prompt = "你叫什么名字?")
```

用户可以给出任何答案，比如数字、字符、向量，扫描结果是为了确保用户给出了合适的答案。例如：

```
result <- readline(prompt = "1+1 的结果是多少?")
while(result!=2){
 readline(prompt = "答案错误。1+1 的结果是多少?")
}
```

但是，需要注意的是，这段代码可能会陷入无限循环，因为用户输入被保存为字符类型。

我们必须使用 `as.numeric` 将其强制转换为数字：

```
result <- as.numeric(readline(prompt = "1+1 的结果是多少?"))
while(result!=2){
 readline(prompt = "答案错误。1+1 的结果是多少?")
}
```

# Chapter 88: Get user input

## Section 88.1: User input in R

Sometimes it can be interesting to have a cross-talk between the user and the program, one example being the [swirl](#) package that had been designed to teach R in R.

One can ask for user input using the `readline` command:

```
name <- readline(prompt = "What is your name?")
```

The user can then give any answer, such as a number, a character, vectors, and scanning the result is here to make sure that the user has given a proper answer. For example:

```
result <- readline(prompt = "What is the result of 1+1?")
while(result!=2){
 readline(prompt = "Wrong answer. What is the result of 1+1?")
}
```

However, it is to be noted that this code be stuck in a never-ending loop, as user input is saved as a character.

We have to coerce it to a number, using `as.numeric`:

```
result <- as.numeric(readline(prompt = "What is the result of 1+1?"))
while(result!=2){
 readline(prompt = "Wrong answer. What is the result of 1+1?")
}
```

# 第89章：Spark API (SparkR)

## 第89.1节：设置Spark上下文

### 在R中设置Spark上下文

要开始使用Spark的分布式数据框，必须将你的R程序连接到现有的Spark集群。

```
library(SparkR)
sc <- sparkR.init() # 连接到Spark上下文
sqlContext <- sparkRSQl.init(sc) # 连接到SQL上下文
```

[以下是如何将你的IDE连接到Spark集群的信息。](#)

### 获取Spark集群

这里有一个Apache Spark入门主题，包含安装说明。基本上，你可以通过java在本地使用Spark集群（[参见说明](#)），或者使用（非免费）云应用（例如[微软Azure](#)[主题网站]，[IBM](#)）。

## 第89.2节：缓存数据

内容：

缓存可以优化Spark中的计算。缓存将数据存储在内存中，是持久化的一种特殊情况。

[这里解释了在Spark中缓存RDD时会发生什么。](#)

原因：

基本上，缓存会保存一个中间的部分结果——通常是在转换之后——的原始数据。因此，当你使用缓存的RDD时，会直接从内存中访问已经转换过的数据，而无需重新计算之前的转换。

方法：

下面是一个示例，展示如何在多次访问时快速从内存存储中访问大数据（这里是3GB大小的csv文件）：

```
library(SparkR)
下一行是直接导入csv所需的：
Sys.setenv('SPARKR_SUBMIT_ARGS'='"--packages" "com.databricks:spark-csv_2.10:1.4.0" "sparkr-shell"')
sc <- sparkR.init()
sqlContext <- sparkRSQl.init(sc)

加载3GB大小的csv文件：
train <- read.df(sqlContext, "/train.csv", source = "com.databricks.spark.csv", inferSchema =
"true")
cache(train)
system.time(head(train))
输出：耗时：125秒。此操作在此处触发缓存。
system.time(head(train))
输出：耗时：0.2秒 (!!)
```

# Chapter 89: Spark API (SparkR)

## Section 89.1: Setup Spark context

### Setup Spark context in R

To start working with Spaks distributed dataframes, you must connect your R program with an existing Spark Cluster.

```
library(SparkR)
sc <- sparkR.init() # connection to Spark context
sqlContext <- sparkRSQl.init(sc) # connection to SQL context
```

[Here are infos](#) how to connect your IDE to a Spark cluster.

### Get Spark Cluster

There is an Apache Spark introduction topic with install instructions. Basically, you can employ a Spark Cluster locally via java ([see instructions](#)) or use (non-free) cloud applications (e.g. [Microsoft Azure](#) [topic site], [IBM](#)).

## Section 89.2: Cache data

What:

Caching can optimize computation in Spark. Caching stores data in memory and is a special case of persistence. [Here is explained](#) what happens when you cache an RDD in Spark.

Why:

Basically, caching saves an interim partial result - usually after transformations - of your original data. So, when you use the cached RDD, the already transformed data from memory is accessed without recomputing the earlier transformations.

How:

Here is an example how to quickly access large data (*here 3 GB big csv*) from in-memory storage when accessing it more than once:

```
library(SparkR)
next line is needed for direct csv import:
Sys.setenv('SPARKR_SUBMIT_ARGS'='"--packages" "com.databricks:spark-csv_2.10:1.4.0" "sparkr-shell"')
sc <- sparkR.init()
sqlContext <- sparkRSQl.init(sc)

loading 3 GB big csv file:
train <- read.df(sqlContext, "/train.csv", source = "com.databricks.spark.csv", inferSchema =
"true")
cache(train)
system.time(head(train))
output: time elapsed: 125 s. This action invokes the caching at this point.
system.time(head(train))
output: time elapsed: 0.2 s (!!)
```

## 第89.3节：创建RDD（弹性分布式数据集）

从数据框：

```
mtrdd <- createDataFrame(sqlContext, mtcars)
```

从csv：

对于csv文件，需要在启动Spark上下文之前添加csv包到环境中：

```
Sys.setenv('SPARKR_SUBMIT_ARGS'='--packages "com.databricks:spark-csv_2.10:1.4.0" "sparkr-shell"') # 用于csv导入的上下文，读取csv ->
sc <- sparkR.init()
sqlContext <- sparkRSQl.init(sc)
```

然后，你可以通过推断列中数据的模式来加载csv：

```
train <- read.df(sqlContext, "/train.csv", header= "true", source = "com.databricks.spark.csv",
inferSchema = "true")
```

或者通过预先指定数据模式：

```
customSchema <- structType(
 structField("margin", "integer"),
 structField("gross", "integer"),
 structField("name", "string"))

train <- read.df(sqlContext, "/train.csv", header= "true", source = "com.databricks.spark.csv",
schema = customSchema)
```

## Section 89.3: Create RDDs (Resilient Distributed Datasets)

**From dataframe:**

```
mtrdd <- createDataFrame(sqlContext, mtcars)
```

**From csv:**

For csv's, you need to add the [csv package](#) to the environment before initiating the Spark context:

```
Sys.setenv('SPARKR_SUBMIT_ARGS'=' --packages "com.databricks:spark-csv_2.10:1.4.0" "sparkr-shell"') # context for csv import read csv ->
sc <- sparkR.init()
sqlContext <- sparkRSQl.init(sc)
```

Then, you can load the csv either by inferring the data schema of the data in the columns:

```
train <- read.df(sqlContext, "/train.csv", header= "true", source = "com.databricks.spark.csv",
inferSchema = "true")
```

Or by specifying the data schema beforehand:

```
customSchema <- structType(
 structField("margin", "integer"),
 structField("gross", "integer"),
 structField("name", "string"))

train <- read.df(sqlContext, "/train.csv", header= "true", source = "com.databricks.spark.csv",
schema = customSchema)
```

# 第90章：元数据：文档指南

## 第90.1节：风格

### 提示

如果您希望代码可以直接复制粘贴，请删除每行开头的提示符，如R>、>或+。有些文档作者不喜欢让复制粘贴变得容易，这也是可以理解的。

### 控制台输出

控制台输出应与代码清晰区分。常见的方法包括：

- 在输入前包含提示符（如使用控制台时所见）。
- 将所有输出注释掉，每行以#或##开头。
- 按原样打印，依靠前导的[1]使输出与输入区分开来。
- 在代码和控制台输出之间添加空行。

### 赋值

=和<-都可以用于给R对象赋值。适当使用空格，避免写出难以解析的代码，例如 x<-1（在 x <- 1和 x < -1之间存在歧义）

### 代码注释

务必解释代码本身的目的和功能。对于这种解释是采用散文形式还是代码注释，并没有硬性规定。散文可能更易读且允许更长的说明，但代码注释便于复制粘贴。两种方式都要考虑。

### 章节

许多示例足够简短，不需要章节划分，但如果使用章节，请以H1开始。

## 第90.2节：制作好的示例

大部分关于制作好的示例的指导同样适用于文档。

- 保持简洁，直奔主题。复杂和离题会适得其反。
- 包含可运行的代码和解释代码的文字说明。单独一种都不够。
- 不要依赖外部数据源。尽可能生成数据或使用datasets库：

```
library(help = "datasets")
```

在文档环境中还有一些额外的注意事项：

- 在相关情况下，请参考内置文档，如?data.frame。SO文档并非试图取代内置文档。确保新R用户知道内置文档的存在以及如何查找它们非常重要。
- 将适用于多个示例的内容移至备注部分。

# Chapter 90: Meta: Documentation Guidelines

## Section 90.1: Style

### Prompts

If you want your code to be copy-pastable, remove prompts such as R>, >, or + at the beginning of each new line. Some Docs authors prefer to not make copy-pasting easy, and that is okay.

### Console output

Console output should be clearly distinguished from code. Common approaches include:

- Include prompts on input (as seen when using the console).
- Comment out all output, with # or ## starting each line.
- Print as-is, trusting the leading [1] to make the output stand out from the input.
- Add a blank line between code and console output.

### Assignment

= and <- are fine for assigning R objects. Use white space appropriately to avoid writing code that is difficult to parse, such as x<-1 (ambiguous between x <- 1 and x < -1)

### Code comments

Be sure to explain the purpose and function of the code itself. There isn't any hard-and-fast rule on whether this explanation should be in prose or in code comments. Prose may be more readable and allows for longer explanations, but code comments make for easier copy-pasting. Keep both options in mind.

### Sections

Many examples are short enough to not need sections, but if you use them, start with [H1](#).

## Section 90.2: Making good examples

Most of the guidance for [creating good examples](#) for Q&A carries over into the documentation.

- Make it minimal and get to the point. Complications and digressions are counterproductive.
- Include both working code and prose explaining it. Neither one is sufficient on its own.
- Don't rely on external sources for data. Generate data or use the datasets library if possible:

```
library(help = "datasets")
```

There are some additional considerations in the context of Docs:

- Refer to built-in docs like [?data.frame](#) whenever relevant. The SO Docs are not an attempt to replace the built-in docs. It is important to make sure new R users know that the built-in docs exist as well as how to find them.
- Move content that applies to multiple examples to the Remarks section.

# 第91章：输入和输出

## 第91.1节：读取和写入数据框

数据框是R中的表格数据结构。它们可以通过多种方式进行写入或读取。

本示例说明了几种常见情况。有关其他资源，请参见文末的链接。

### 写入

在制作下面的示例数据之前，请确保你处于想要写入的文件夹中。运行`getwd()`以确认当前文件夹，若需更改文件夹，请阅读`?setwd`。

```
set.seed(1)
for (i in 1:3)
 write.table(
 data.frame(id = 1:2, v = sample(letters, 2)),
 file = sprintf("file201%s.csv", i)
)
```

现在，我们有三个格式相似的CSV文件保存在磁盘上。

### 读取

我们有三个格式相似的文件（来自上一节）需要读取。由于这些文件相关，读取后我们应该将它们一起存储在一个列表中：

```
file_names = c("file2011.csv", "file2012.csv", "file2013.csv")
file_contents = lapply(setNames(file_names, file_names), read.table)

$file2011.csv
id v
1 1 g
2 2 j
#
$file2012.csv
id v
1 1 o
2 2 w
#
$file2013.csv
id v
1 1 f
2 2 w
```

要处理这个文件列表，首先用`str(file_contents)`检查结构，然后阅读关于用`?rbind`堆叠列表或用`?lapply`遍历列表的内容。

### 更多资源

查看`?read.table` 和 `?write.table` 以扩展此示例。此外：

- R 二进制格式（用于表格和其他对象）
- 纯文本表格格式
  - 逗号分隔的 CSV 文件
  - 制表符分隔的 TSV 文件

# Chapter 91: Input and output

## Section 91.1: Reading and writing data frames

Data frames are R's tabular data structure. They can be written to or read from in a variety of ways.

This example illustrates a couple common situations. See the links at the end for other resources.

### Writing

*Before making the example data below, make sure you're in a folder you want to write to. Run `getwd()` to verify the folder you're in and read `?setwd` if you need to change folders.*

```
set.seed(1)
for (i in 1:3)
 write.table(
 data.frame(id = 1:2, v = sample(letters, 2)),
 file = sprintf("file201%s.csv", i)
)
```

Now, we have three similarly-formatted CSV files on disk.

### Reading

We have three similarly-formatted files (from the last section) to read in. Since these files are related, we should store them together after reading in, in a `list`:

```
file_names = c("file2011.csv", "file2012.csv", "file2013.csv")
file_contents = lapply(setNames(file_names, file_names), read.table)

$file2011.csv
id v
1 1 g
2 2 j
#
$file2012.csv
id v
1 1 o
2 2 w
#
$file2013.csv
id v
1 1 f
2 2 w
```

To work with this list of files, first examine the structure with `str(file_contents)`, then read about stacking the list with `?rbind` or iterating over the list with `?lapply`.

### Further resources

Check out `?read.table` and `?write.table` to extend this example. Also:

- R binary formats (for tables and other objects)
- Plain-text table formats
  - comma-delimited CSVs
  - tab-delimited TSVs

- 定宽格式
- 与语言无关的二进制表格格式
  - Feather
- 外部表格和电子表格格式
  - SAS
  - SPSS
  - Stata
  - Excel
- 关系数据库表格格式
  - MySQL
  - SQLite
  - PostgreSQL

- Fixed-width formats
- Language-agnostic binary table formats
  - Feather
- Foreign table and spreadsheet formats
  - SAS
  - SPSS
  - Stata
  - Excel
- Relational database table formats
  - MySQL
  - SQLite
  - PostgreSQL

# 第92章：外部表的I/O（Excel, SAS, SPSS, Stata）

## 第92.1节：使用rio导入数据

导入许多常见文件格式数据的一个非常简单的方法是使用 `rio`。该包提供了一个函数 `import()`，封装了许多常用的数据导入函数，从而提供了一个标准接口。它的工作原理很简单，只需将文件名或URL传递给 `import()`：

```
import("example.csv") # 逗号分隔值
import("example.tsv") # 制表符分隔值
import("example.dta") # Stata
import("example.sav") # SPSS
import("example.sas7bdat") # SAS
import("example.xlsx") # Excel
```

`import()`还可以从压缩目录、URL（HTTP或HTTPS）以及剪贴板读取。所有支持的文件格式的完整列表可在 `rio`包的GitHub仓库中找到。

甚至可以指定一些与您尝试读取的特定文件格式相关的其他参数，直接在 `import()`函数中传递它们：

```
import("example.csv", format = ",") # 用逗号作为分隔符的csv文件
import("example.csv", format = ";") # 用分号作为分隔符的csv文件
```

## 第92.2节：读取和写入Stata、SPSS和SAS文件

包`foreign`和`haven`可以用来导入和导出来自多种其他统计软件包的数据文件，如Stata、SPSS和SAS及相关软件。对于每种支持的数据类型，都有一个`read`函数用于导入文件。

```
加载包
library(foreign)
library(haven)
library(readstata13)
library(Hmisc)
```

一些最常见数据类型的示例：

```
使用`foreign`读取Stata文件
read.dta("path\your\data")
使用`haven`读取Stata文件
read_dta("path\your\data")
```

`foreign`包可以读取Stata 7-12版本的stata (.dta)文件。根据开发页面，`read.dta`基本上已冻结，不会更新以支持13及以上版本。对于较新版本的Stata，可以使用`readstata13`包或`haven`。对于`readstata13`，文件是

```
使用`readstata13`读取较新Stata (13+)文件
read.dta13("path\your\data")
```

用于读取SPSS和SAS文件

# Chapter 92: I/O for foreign tables (Excel, SAS, SPSS, Stata)

## Section 92.1: Importing data with rio

A very simple way to import data from many common file formats is with `rio`. This package provides a function `import()` that wraps many commonly used data import functions, thereby providing a standard interface. It works simply by passing a file name or URL to `import()`:

```
import("example.csv") # comma-separated values
import("example.tsv") # tab-separated values
import("example.dta") # Stata
import("example.sav") # SPSS
import("example.sas7bdat") # SAS
import("example.xlsx") # Excel
```

`import()` can also read from compressed directories, URLs (HTTP or HTTPS), and the clipboard. A comprehensive list of all supported file formats is available on the [rio package github repository](#).

It is even possible to specify some further parameters related to the specific file format you are trying to read, passing them directly within the `import()` function:

```
import("example.csv", format = ",") #for csv file where comma is used as separator
import("example.csv", format = ";") #for csv file where semicolon is used as separator
```

## Section 92.2: Read and write Stata, SPSS and SAS files

The packages `foreign` and `haven` can be used to import and export files from a variety of other statistical packages like Stata, SPSS and SAS and related software. There is a `read` function for each of the supported data types to import the files.

```
loading the packages
library(foreign)
library(haven)
library(readstata13)
library(Hmisc)
```

Some examples for the most common data types:

```
reading Stata files with `foreign`
read.dta("path\to\your\data")
reading Stata files with `haven`
read_dta("path\to\your\data")
```

The `foreign` package can read in stata (.dta) files for versions of Stata 7-12. According to the development page, the `read.dta` is more or less frozen and will not be updated for reading in versions 13+. For more recent versions of Stata, you can use either the `readstata13` package or `haven`. For `readstata13`, the files are

```
reading recent Stata (13+) files with `readstata13`
read.dta13("path\to\your\data")
```

For reading in SPSS and SAS files

```

使用`foreign`读取SPSS文件
read.spss("patho\your\data.sav", to.data.frame = TRUE)
使用`haven`读取SPSS文件
read_spss("patho\your\data.sav")
read_sav("patho\your\data.sav")
read_por("patho\your\data.por")

使用 `foreign` 读取 SAS 文件
read.ssd("patho\your\data")
使用 `haven` 读取 SAS 文件
read_sas("patho\your\data")
使用 `Hmisc` 读取原生 SAS 文件
sas.get("patho\your\data") # 需要访问 saslib
读取 SA XPORT 格式 (*.XPT) 文件
sasxport.get("patho\your\data.xpt") # 不需要访问 SAS 可执行文件

```

SASci 包提供了接受 SAS SET 导入代码并构建可用 read.fwf 处理的文本文件的函数。它在导入大型公开发布数据集方面表现非常稳定。支持地址为

<https://github.com/ajdamico/SASci>

要将数据框导出到其他统计软件包，可以使用写入函数 write.foreign()。该函数将写入两个文件，一个包含数据，另一个包含其他软件包读取数据所需的指令。

```

使用 `foreign` 写入 Stata、SPSS 或 SAS 文件
write.foreign(dataframe, datafile, codefile,
 package = c("SPSS", "Stata", "SAS"), ...)
write.foreign(dataframe, "patho\data\file", "patho\instruction\file", package = "Stata")

使用 `foreign` 写入 Stata 文件
write.dta(dataframe, "file", version = 7L,
 convert.dates = TRUE, tz = "GMT",
 convert.factors = c("labels", "string", "numeric", "codes"))

使用 `haven` 写入 Stata 文件
write_dta(dataframe, "patho\your\data")

使用 `readstata13` 写入 Stata 文件
save.dta13(dataframe, file, data.label = NULL, time.stamp = TRUE,
 convert.factors = TRUE, convert.dates = TRUE, tz = "GMT",
 add.rownames = FALSE, compress = FALSE, version = 117,
 convert.underscore = FALSE)

使用 `haven` 写入 SPSS 文件
write_sav(dataframe, "路径o\你的\data")

```

SPSS存储的文件也可以通过read.spss以这种方式读取：

```

foreign::read.spss('data.sav', to.data.frame=TRUE, use.value.labels=FALSE,
 use.missing=TRUE, reencode='UTF-8')
to.data.frame 如果为TRUE：返回一个数据框
use.value.labels 如果为TRUE：将带有值标签的变量转换为具有这些级别的R因子
use.missing 如果为TRUE：用户定义的缺失值信息将用于将相应的值设置为NA。
reencode 字符串将重新编码为当前区域设置。默认值NA表示仅在UTF-8区域设置中执行此操作。

```

```

reading SPSS files with `foreign`
read.spss("path\to\your\data.sav", to.data.frame = TRUE)
reading SPSS files with `haven`
read_spss("path\to\your\data.sav")
read_sav("path\to\your\data.sav")
read_por("path\to\your\data.por")

reading SAS files with `foreign`
read.ssd("path\to\your\data")
reading SAS files with `haven`
read_sas("path\to\your\data")
reading native SAS files with `Hmisc`
sas.get("path\to\your\data") # requires access to saslib
Reading SA XPORT format (*.XPT) files
sasxport.get("path\to\your\data.xpt") # does not require access to SAS executable

```

The SASci package provides functions that will accept SAS SET import code and construct a text file that can be processed with `read.fwf`. It has proved very robust for import of large public-released datasets. Support is at <https://github.com/ajdamico/SASci>

To export data frames to other statistical packages you can use the write functions `write.foreign()`. This will write 2 files, one containing the data and one containing instructions the other package needs to read the data.

```

writing to Stata, SPSS or SAS files with `foreign`
write.foreign(dataframe, datafile, codefile,
 package = c("SPSS", "Stata", "SAS"), ...)
write.foreign(dataframe, "path\to\data\file", "path\to\instruction\file", package = "Stata")

writing to Stata files with `foreign`
write.dta(dataframe, "file", version = 7L,
 convert.dates = TRUE, tz = "GMT",
 convert.factors = c("labels", "string", "numeric", "codes"))

writing to Stata files with `haven`
write_dta(dataframe, "path\to\your\data")

writing to Stata files with `readstata13`
save.dta13(dataframe, file, data.label = NULL, time.stamp = TRUE,
 convert.factors = TRUE, convert.dates = TRUE, tz = "GMT",
 add.rownames = FALSE, compress = FALSE, version = 117,
 convert.underscore = FALSE)

writing to SPSS files with `haven`
write_sav(dataframe, "path\to\your\data")

```

File stored by the SPSS can also be read with `read.spss` in this way:

```

foreign::read.spss('data.sav', to.data.frame=TRUE, use.value.labels=FALSE,
 use.missing=TRUE, reencode='UTF-8')
to.data.frame if TRUE: return a data frame
use.value.labels if TRUE: convert variables with value labels into R factors with those levels
use.missing if TRUE: information on user-defined missing values will be used to set the corresponding values to NA.
reencode character strings will be re-encoded to the current locale. The default, NA, means to do so in a UTF-8 locale, only.

```

## 第92.3节：导入Excel文件

有几个R包可以读取Excel文件，每个包使用不同的语言或资源，

## Section 92.3: Importing Excel files

There are several R packages to read excel files, each of which using different languages or resources, as

总结如下表：

## R 包使用

xlsx	Java
XLconnect	Java
openxlsxC++	C++
readxl	C++
RODBC	ODBC
gdata	Perl

对于使用 Java 或 ODBC 的包，了解您的系统细节非常重要，因为根据您的 R 版本和操作系统，可能存在兼容性问题。例如，如果您使用的是 64 位 R，那么您也必须拥有 64 位 Java 才能使用 xlsx 或 XLconnect。

下面提供了一些使用各个包读取 Excel 文件的示例。请注意，许多包的函数名称相同或非常相似。因此，明确指出包名是很有用的，比如 package::function。该包openxlsx需要事先安装RTools。

### 使用 xlsx 包读取 Excel 文件

`library(xlsx)`

导入时需要指定工作表的索引或名称。

```
xlsx::read.xlsx("Book1.xlsx", sheetIndex=1)
xlsx::read.xlsx("Book1.xlsx", sheetName="Sheet1")
```

### 使用 XLconnect 包读取 Excel 文件

```
library(XLConnect)
wb <- XLConnect::loadWorkbook("Book1.xlsx")
```

```
如果 Book1.xlsx 有一个名为 "Sheet1" 的工作表：
sheet1 <- XLConnect::readWorksheet(wb, "Sheet1")
或者，更通用的方式，直接获取 Book1.xlsx 中的第一个工作表：
sheet1 <- XLConnect::readWorksheet(wb, getSheets(wb)[1])
```

XLConnect 会自动导入嵌入在 Book1.xlsx 中预定义的 Excel 单元格样式。当你希望格式化工作簿对象并导出格式完美的 Excel 文档时，这非常有用。首先，你需要在 Book1.xlsx 中创建所需的单元格格式并保存，例如 myHeader、myBody 和 myPcts。然后，在 R 中加载工作簿（见上文）：

```
Headerstyle <- XLConnect::getCellStyle(wb, "myHeader")
Bodystyle <- XLConnect::getCellStyle(wb, "myBody")
Pctsstyle <- XLConnect::getCellStyle(wb, "myPcts")
```

单元格样式现已保存在您的R环境中。为了将单元格样式分配给数据的特定范围，您需要先定义范围，然后分配样式：

```
Headerrange <- expand.grid(row = 1, col = 1:8)
Bodyrange <- expand.grid(row = 2:6, col = c(1:5, 8))
Pctrange <- expand.grid(row = 2:6, col = c(6, 7))

XLConnect::setCellStyle(wb, sheet = "sheet1", row = Headerrange$row,
 col = Headerrange$col, cellstyle = Headerstyle)
XLConnect::setCellStyle(wb, sheet = "sheet1", row = Bodyrange$row,
```

summarized in the following table:

## R package Uses

xlsx	Java
XLconnect	Java
openxlsx	C++
readxl	C++
RODBC	ODBC
gdata	Perl

For the packages that use Java or ODBC it is important to know details about your system because you may have compatibility issues depending on your R version and OS. For instance, if you are using R 64 bits then you also must have Java 64 bits to use xlsx or XLconnect.

Some examples of reading excel files with each package are provided below. Note that many of the packages have the same or very similar function names. Therefore, it is useful to state the package explicitly, like `package::function`. The package openxlsx requires prior installation of RTools.

### Reading excel files with the xlsx package

`library(xlsx)`

The index or name of the sheet is required to import.

```
xlsx::read.xlsx("Book1.xlsx", sheetIndex=1)
xlsx::read.xlsx("Book1.xlsx", sheetName="Sheet1")
```

### Reading Excel files with the XLconnect package

```
library(XLConnect)
wb <- XLConnect::loadWorkbook("Book1.xlsx")
```

```
Either, if Book1.xlsx has a sheet called "Sheet1":
sheet1 <- XLConnect::readWorksheet(wb, "Sheet1")
Or, more generally, just get the first sheet in Book1.xlsx:
sheet1 <- XLConnect::readWorksheet(wb, getSheets(wb)[1])
```

XLConnect automatically imports the pre-defined Excel cell-styles embedded in Book1.xlsx. This is useful when you wish to format your workbook object and export a perfectly formatted Excel document. Firstly, you will need to create the desired cell formats in Book1.xlsx and save them, for example, as myHeader, myBody and myPcts. Then, after loading the workbook in R (see above):

```
Headerstyle <- XLConnect::getCellStyle(wb, "myHeader")
Bodystyle <- XLConnect::getCellStyle(wb, "myBody")
Pctsstyle <- XLConnect::getCellStyle(wb, "myPcts")
```

The cell styles are now saved in your R environment. In order to assign the cell styles to certain ranges of your data, you need to define the range and then assign the style:

```
Headerrange <- expand.grid(row = 1, col = 1:8)
Bodyrange <- expand.grid(row = 2:6, col = c(1:5, 8))
Pctrange <- expand.grid(row = 2:6, col = c(6, 7))

XLConnect::setCellStyle(wb, sheet = "sheet1", row = Headerrange$row,
 col = Headerrange$col, cellstyle = Headerstyle)
XLConnect::setCellStyle(wb, sheet = "sheet1", row = Bodyrange$row,
```

```
col = Bodyrange$col, cellstyle = Bodystyle)
XLConnect::setCellStyle(wb, sheet = "sheet1", row = Pctrange$row,
col = Pctrange$col, cellstyle = Pctsstyle)
```

注意，XLConnect使用简单，但格式化时可能非常慢。一个更快但更繁琐的格式化选项是openxlsx提供的。

## 使用 openxlsx 包读取 Excel 文件

Excel 文件可以通过openxlsx包导入

```
library(openxlsx)

openxlsx::read.xlsx("spreadsheet1.xlsx", colNames=TRUE, rowNames=TRUE)

#colNames : 如果为TRUE, 数据的第一行将用作列名。
#rowNames : 如果为TRUE, 数据的第一列将用作行名。
```

要读取到R中的工作表，可以通过在 sheet参数中提供其位置来选择：

```
openxlsx::read.xlsx("spreadsheet1.xlsx", sheet = 1)
```

或者通过声明其名称：

```
openxlsx::read.xlsx("spreadsheet1.xlsx", sheet = "Sheet1")
```

此外，openxlsx可以检测读取工作表中的日期列。为了允许自动检测日期，参数detectDates应设置为TRUE：

```
openxlsx::read.xlsx("spreadsheet1.xlsx", sheet = "Sheet1", detectDates= TRUE)
```

## 使用readxl包读取Excel文件

可以使用readxl包将Excel文件导入为R中的数据框。

```
library(readxl)
```

它可以读取.xls和.xlsx文件。

```
readxl::read_excel("spreadsheet1.xls")
readxl::read_excel("spreadsheet2.xlsx")
```

要导入的工作表可以通过编号或名称指定。

```
readxl::read_excel("spreadsheet.xls", sheet = 1)
readxl::read_excel("spreadsheet.xls", sheet = "summary")
```

参数col\_names = TRUE将第一行设置为列名。

```
readxl::read_excel("spreadsheet.xls", sheet = 1, col_names = TRUE)
```

参数col\_types可用于以向量形式指定数据中的列类型。

```
readxl::read_excel("spreadsheet.xls", sheet = 1, col_names = TRUE,
col_types = c("text", "date", "numeric", "numeric"))
```

## 使用RODBC包读取Excel文件

```
col = Bodyrange$col, cellstyle = Bodystyle)
XLConnect::setCellStyle(wb, sheet = "sheet1", row = Pctrange$row,
col = Pctrange$col, cellstyle = Pctsstyle)
```

Note that XLConnect is easy, but can become extremely slow in formatting. A much faster, but more cumbersome formatting option is offered by openxlsx.

## Reading excel files with the openxlsx package

Excel files can be imported with package openxlsx

```
library(openxlsx)

openxlsx::read.xlsx("spreadsheet1.xlsx", colNames=TRUE, rowNames=TRUE)

#colNames: If TRUE, the first row of data will be used as column names.
#rowNames: If TRUE, first column of data will be used as row names.
```

The sheet, which should be read into R can be selected either by providing its position in the sheet argument:

```
openxlsx::read.xlsx("spreadsheet1.xlsx", sheet = 1)
```

or by declaring its name:

```
openxlsx::read.xlsx("spreadsheet1.xlsx", sheet = "Sheet1")
```

Additionally, openxlsx can detect date columns in a read sheet. In order to allow automatic detection of dates, an argument detectDates should be set to TRUE:

```
openxlsx::read.xlsx("spreadsheet1.xlsx", sheet = "Sheet1", detectDates= TRUE)
```

## Reading excel files with the readxl package

Excel files can be imported as a data frame into R using the readxl package.

```
library(readxl)
```

It can read both .xls and .xlsx files.

```
readxl::read_excel("spreadsheet1.xls")
readxl::read_excel("spreadsheet2.xlsx")
```

The sheet to be imported can be specified by number or name.

```
readxl::read_excel("spreadsheet.xls", sheet = 1)
readxl::read_excel("spreadsheet.xls", sheet = "summary")
```

The argument col\_names = TRUE sets the first row as the column names.

```
readxl::read_excel("spreadsheet.xls", sheet = 1, col_names = TRUE)
```

The argument col\_types can be used to specify the column types in the data as a vector.

```
readxl::read_excel("spreadsheet.xls", sheet = 1, col_names = TRUE,
col_types = c("text", "date", "numeric", "numeric"))
```

## Reading excel files with the RODBC package

可以使用ODBC Excel驱动程序读取Excel文件，该驱动程序与Windows的Access数据库引擎（ACE，前称JET）接口。通过RODBC包，R可以连接到该驱动程序并直接查询工作簿。假设工作表在第一行保留列标题，数据以相似类型的有序列排列。注意：此方法仅限于Windows/PC机器，因为JET/ACE是安装的.dll文件，其他操作系统不可用。

#### library(RODBC)

```
xlconn <- odbcDriverConnect('驱动程序={Microsoft Excel 驱动程序 (*.xls, *.xlsx, *.xlsm, *.xlsb)};
DBQ=C:\\路径\\到\\工作簿.xlsx')

df <- sqlQuery(xlconn, "SELECT * FROM [SheetName$]")
close(xlconn)
```

通过这种方法连接SQL引擎，可以像查询数据库表一样查询Excel工作表包括 JOIN 和 UNION 操作。语法遵循 JET/ACE SQL 方言。注意：只能对工作簿执行数据访问DML语句，特别是 SELECT，视为不可更新的查询。

```
joindf <- sqlQuery(xlconn, "SELECT t1.*, t2.* FROM [Sheet1$] t1
INNER JOIN [Sheet2$] t2
ON t1.[ID] = t2.[ID]")

uniondf <- sqlQuery(xlconn, "SELECT * FROM [Sheet1$]
UNION
SELECT * FROM [Sheet2$]")
```

甚至可以通过指向当前工作簿的同一ODBC通道查询其他工作簿：

```
otherwkbkdf <- sqlQuery(xlconn, "SELECT * FROM
[Excel 12.0 Xml;HDR=Yes;
Database=C:\\路径\\到\\其他\\工作簿.xlsx].[Sheet1$];")
```

#### 使用 gdata 包读取 Excel 文件

示例在此

## 第92.4节：Feather文件的导入或导出

Feather是Apache Arrow的一个实现，旨在以语言无关的方式存储数据框，同时保持元数据（例如日期类），提高Python和R之间的互操作性。读取feather文件将生成一个tibble，而不是标准的数据框。

```
library(feather)

path <- "filename.feather"
df <- mtcars

write_feather(df, path)

df2 <- read_feather(path)

head(df2)
一个6行11列的tibble
mpg cyl disp hp drat wt qsec vs am gear carb
<dbl>
1 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4
2 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4
3 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1
```

Excel files can be read using the ODBC Excel Driver that interfaces with Windows' Access Database Engine (ACE), formerly JET. With the RODBC package, R can connect to this driver and directly query workbooks. Worksheets are assumed to maintain column headers in first row with data in organized columns of similar types. **NOTE:** This approach is limited to only Windows/PC machines as JET/ACE are installed .dll files and not available on other operating systems.

#### library(RODBC)

```
xlconn <- odbcDriverConnect('Driver={Microsoft Excel Driver (*.xls, *.xlsx, *.xlsm, *.xlsb)};
DBQ=C:\\Path\\To\\Workbook.xlsx')

df <- sqlQuery(xlconn, "SELECT * FROM [SheetName$]")
close(xlconn)
```

Connecting with an SQL engine in this approach, Excel worksheets can be queried similar to database tables including JOIN and UNION operations. Syntax follows the JET/ACE SQL dialect. **NOTE:** Only data access DML statements, specifically **SELECT** can be run on workbooks, considered not updateable queries.

```
joindf <- sqlQuery(xlconn, "SELECT t1.*, t2.* FROM [Sheet1$] t1
INNER JOIN [Sheet2$] t2
ON t1.[ID] = t2.[ID]")

uniondf <- sqlQuery(xlconn, "SELECT * FROM [Sheet1$]
UNION
SELECT * FROM [Sheet2$]")
```

Even other workbooks can be queried from the same ODBC channel pointing to a current workbook:

```
otherwkbkdf <- sqlQuery(xlconn, "SELECT * FROM
[Excel 12.0 Xml;HDR=Yes;
Database=C:\\Path\\To\\Other\\Workbook.xlsx].[Sheet1$];")
```

#### Reading excel files with the gdata package

example here

## Section 92.4: Import or Export of Feather file

Feather is an implementation of Apache Arrow designed to store data frames in a language agnostic manner while maintaining metadata (e.g. date classes), increasing interoperability between Python and R. Reading a feather file will produce a tibble, not a standard data.frame.

```
library(feather)

path <- "filename.feather"
df <- mtcars

write_feather(df, path)

df2 <- read_feather(path)

head(df2)
A tibble: 6 x 11
mpg cyl disp hp drat wt qsec vs am gear carb
<dbl>
1 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4
2 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4
3 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1
```

```
4 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1
5 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2
6 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1
```

#### head(df)

```
英里每加仑 气缸 排量 马力 驱动比 重量 四分之一英里加速时间 V型发动机 变速箱 齿轮数 化油器数
马自达 RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4
马自达 RX4 Wagon 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4
达特桑 710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1
霍内特 4 驱动 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1
霍内特 运动版 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2
瓦里安特 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1
```

当前文档包含以下警告：

用户注意：Feather 应视为测试版软件。特别是，文件格式可能会在未来一年内发生变化。请勿将 Feather 用于长期数据存储。

```
4 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1
5 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2
6 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1
```

#### head(df)

```
mpg cyl disp hp drat wt qsec vs am gear carb
Mazda RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4
Mazda RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4
Datsun 710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1
Hornet 4 Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1
Hornet Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2
Valiant 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1
```

The current documentation contains this warning:

Note to users: Feather should be treated as alpha software. In particular, the file format is likely to evolve over the coming year. Do not use Feather for long-term data storage.

# 第93章：数据库表的输入/输出

## 第93.1节：从 MySQL 数据库读取数据

### 概述

使用RMySQL包，我们可以轻松查询 MySQL 以及 MariaDB 数据库，并将结果存储在 R 数据框中：

```
library(RMySQL)

mydb <- dbConnect(MySQL(), user='user', password='password', dbname='dbname', host='127.0.0.1')

queryString <- "SELECT * FROM table1 t1 JOIN table2 t2 on t1.id=t2.id"
query <- dbSendQuery(mydb, queryString)
data <- fetch(query, n=-1) # n=-1 表示返回所有结果
```

### 使用限制

也可以定义一个限制，例如只获取前100,000行。为此，只需更改SQL查询以设置所需的限制。所提到的软件包将考虑这些选项。示例：

```
queryString <- "SELECT * FROM table1 limit 100000"
```

## 第93.2节：从MongoDB数据库读取数据

为了将数据从MongoDB数据库加载到R数据框中，使用库MongoLite：

```
使用MongoLite库：
#install.packages("mongolite")
library(jsonlite)
library(mongolite)

以root身份连接到数据库和所需的集合：
db <- mongo(collection = "Tweets", db = "TweetCollector", url =
"mongodb://USERNAME:PASSWORD@HOSTNAME")

读取所需文档，即一个数据框中的推文：
documents <- db$find(limit = 100000, skip = 0, fields = '{ "_id" : false, "Text" : true }')
```

代码连接到服务器 HOSTNAME，使用 USERNAME 和 PASSWORD，尝试打开数据库 TweetCollector 并读取集合 Tweets。查询尝试读取字段，即列 Text。

结果是一个包含所返回数据集列的数据框。在本例中，数据框包含列 Text，例如 documents\$Text。

# Chapter 93: I/O for database tables

## Section 93.1: Reading Data from MySQL Databases

### General

Using the package [RMySQL](#) we can easily query MySQL as well as MariaDB databases and store the result in an R dataframe:

```
library(RMySQL)

mydb <- dbConnect(MySQL(), user='user', password='password', dbname='dbname', host='127.0.0.1')

queryString <- "SELECT * FROM table1 t1 JOIN table2 t2 on t1.id=t2.id"
query <- dbSendQuery(mydb, queryString)
data <- fetch(query, n=-1) # n=-1 to return all results
```

### Using limits

It is also possible to define a limit, e.g. getting only the first 100,000 rows. In order to do so, just change the SQL query regarding the desired limit. The mentioned package will consider these options. Example:

```
queryString <- "SELECT * FROM table1 limit 100000"
```

## Section 93.2: Reading Data from MongoDB Databases

In order to load data from a MongoDB database into an R dataframe, use the library [MongoLite](#):

```
Use MongoLite library:
#install.packages("mongolite")
library(jsonlite)
library(mongolite)

Connect to the database and the desired collection as root:
db <- mongo(collection = "Tweets", db = "TweetCollector", url =
"mongodb://USERNAME:PASSWORD@HOSTNAME")

Read the desired documents i.e. Tweets inside one dataframe:
documents <- db$find(limit = 100000, skip = 0, fields = '{ "_id" : false, "Text" : true }')
```

The code connects to the server HOSTNAME as USERNAME with PASSWORD, tries to open the database TweetCollector and read the collection Tweets. The query tries to read the field i.e. column Text.

The results is a dataframe with columns as the yielded data set. In case of this example, the dataframe contains the column Text, e.g. documents\$Text.

# 第94章：地理数据的输入输出 (shapefile等)

另见地理地图介绍及输入输出

## 第94.1节：导入和导出Shapefile

使用 rgdal 包可以在R中导入和导出shapefile。函数 readOGR 可用于导入shapefile。如果你想导入例如ArcGIS的文件，第一个参数 dsn 是包含shapefile的文件夹路径。layer 是shapefile的名称，不带文件后缀（仅为 map 而非 map.shp）。

```
library(rgdal)
readOGR(dsn = "path\to\the\folder\containing\the\shapefile", layer = "map")
```

要导出shapefile，使用writeOGR函数。第一个参数是R中生成的空间对象。dsn和layer与上述相同。第四个必需参数是用于生成shapefile的驱动程序。函数ogrDrivers()列出所有可用驱动。如果你想导出shapefile到ArcGIS或QGIS，可以使用 driver = "ESRI Shapefile"。

```
writeOGR(Rmap, dsn = "包含 shapefile 的路径文件夹", layer = "map",
 driver = "ESRI Shapefile")
```

tmap 包含一个非常方便的函数 read\_shape()，它是 rgdal::readOGR() 的封装。read\_shape() 函数大大简化了导入 shapefile 的过程。缺点是 tmap 比较庞大。

# Chapter 94: I/O for geographic data (shapefiles, etc.)

See also Introduction to Geographical Maps and Input and Output

## Section 94.1: Import and Export Shapefiles

With the rgdal package it is possible to import and export shapfiles with R. The function readOGR can be used to imports shapfiles. If you want to import a file from e.g. ArcGIS the first argument dsn is the path to the folder which contains the shapefile. layer is the name of the shapefile without the file ending (just map and not map.shp).

```
library(rgdal)
readOGR(dsn = "path\to\the\folder\containing\the\shapefile", layer = "map")
```

To export a shapefile use the writeOGR function. The first argument is the spatial object produced in R. dsn and layer are the same as above. The obligatory 4. argument is the driver used to generate the shapefile. The function ogrDrivers() lists all available drivers. If you want to export a shapfile to ArcGis or QGis you could use driver = "ESRI Shapefile".

```
writeOGR(Rmap, dsn = "path\to\the\folder\containing\the\shapefile", layer = "map",
 driver = "ESRI Shapefile")
```

tmap package has a very convenient function read\_shape(), which is a wrapper for rgdal::readOGR(). The read\_shape() function simplifies the process of importing a shapefile a lot. On the downside, tmap is quite heavy.

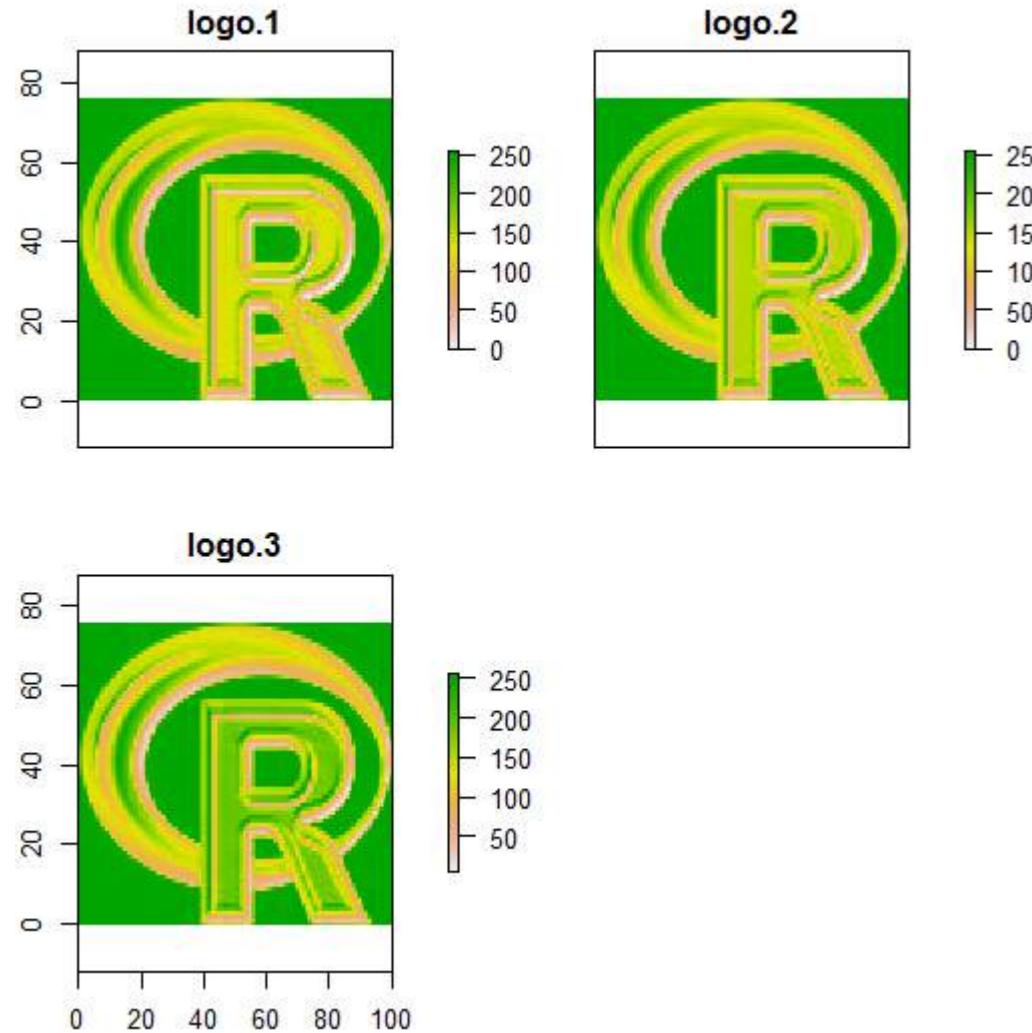
# 第95章：栅格图像的输入输出

另见 棚格和图像分析 以及 输入输出

## 第95.1节：加载多层栅格

R-Logo 是一个多层栅格文件（红、绿、蓝）

```
library(raster)
r <- stack("C:/Program Files/R/R-3.2.3/doc/html/logo.jpg")
plot(r)
```



可以通过 [[ 访问 RasterStack 对象的各个单独图层。

```
plot(r[[1]])
```

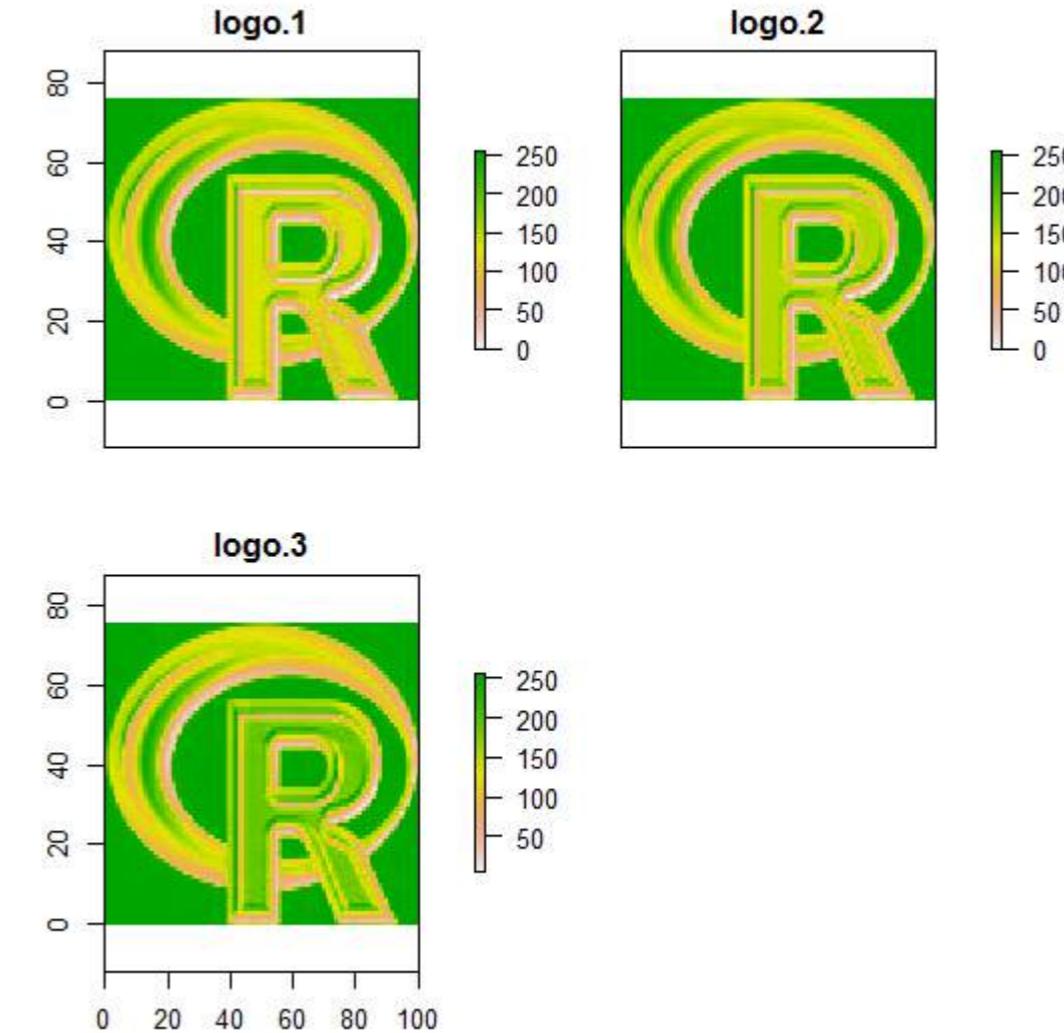
# Chapter 95: I/O for raster images

See also Raster and Image Analysis and Input and Output

## Section 95.1: Load a multilayer raster

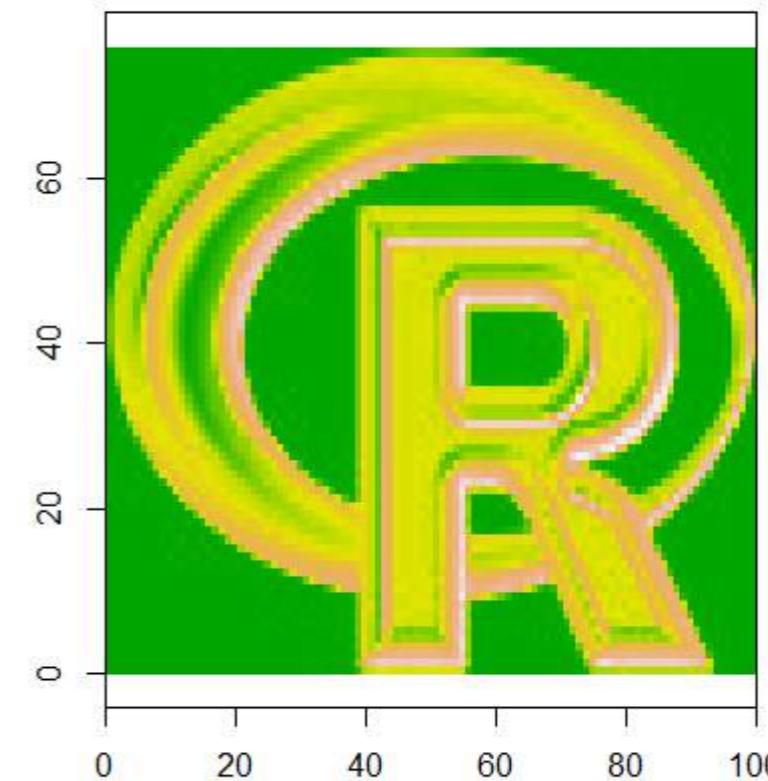
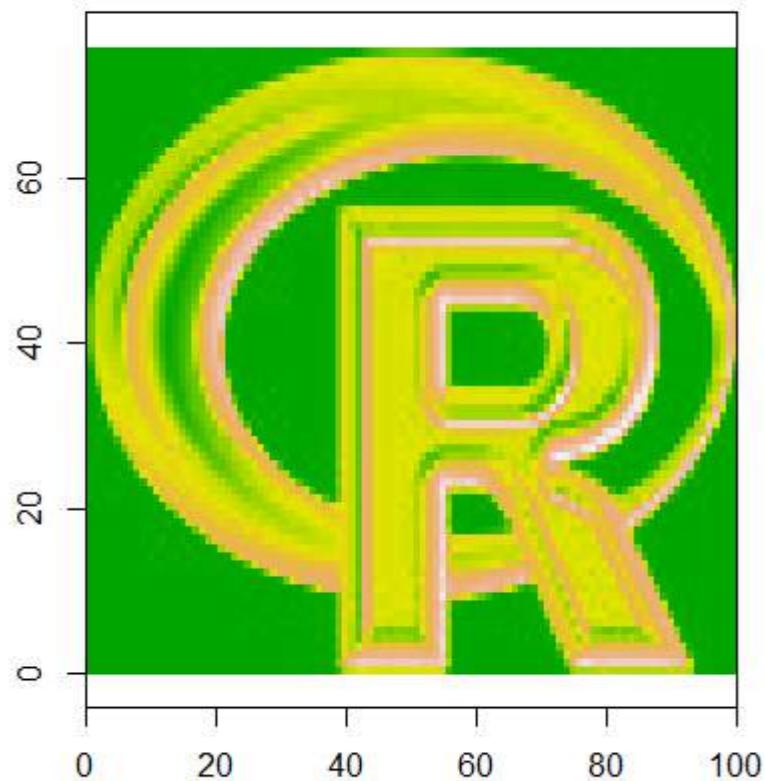
The R-Logo is a multilayer raster file (red, green, blue)

```
library(raster)
r <- stack("C:/Program Files/R/R-3.2.3/doc/html/logo.jpg")
plot(r)
```



The individual layers of the RasterStack object can be addressed by [[].

```
plot(r[[1]])
```



# 第96章：R二进制格式的输入输出

## 第96.1节：Rds 和 RData (Rda) 文件

.rds 和 .Rdata (也称为 .rda) 文件可用于以 R 原生格式存储 R 对象。与非原生存储方式（例如 write.table）相比，这种保存方式有多种优势：

- 将数据恢复到 R 中更快
- 它保留了数据中编码的 R 特定信息（例如，属性、变量类型等）。

saveRDS/readRDS 仅处理单个 R 对象。然而，它们比多对象存储方式更灵活，因为恢复对象的名称不必与存储时的对象名称相同。

例如，使用 .rds 文件保存 iris 数据集，我们可以这样写：

```
saveRDS(object = iris, file = "my_data_frame.rds")
```

要加载数据回 R 中：

```
iris2 <- readRDS(file = "my_data_frame.rds")
```

要保存多个对象，我们可以使用 save() 并输出为 .Rdata 文件。

例如，保存两个数据框：iris 和 cars

```
save(iris, cars, file = "myIrisAndCarsData.Rdata")
```

加载：

```
load("myIrisAndCarsData.Rdata")
```

## 第96.2节：环境

函数 save 和 load 允许我们指定对象将被存放的环境：

```
save(iris, cars, file = "myIrisAndCarsData.Rdata", envir = foo <- new.env())
load("myIrisAndCarsData.Rdata", envir = foo)
foo$cars

save(iris, cars, file = "myIrisAndCarsData.Rdata", envir = foo <- new.env())
load("myIrisAndCarsData.Rdata", envir = foo)
foo$cars
```

# Chapter 96: I/O for R's binary format

## Section 96.1: Rds and RData (Rda) files

.rds and .Rdata (also known as .rda) files can be used to store R objects in a format native to R. There are multiple advantages of saving this way when contrasted with non-native storage approaches, e.g. [write.table](#):

- It is faster to restore the data to R
- It keeps R specific information encoded in the data (e.g., attributes, variable types, etc).

saveRDS/readRDS only handle a single R object. However, they are more flexible than the multi-object storage approach in that the object name of the restored object need not be the same as the object name when the object was stored.

Using an .rds file, for example, saving the [iris](#) dataset we would use:

```
saveRDS(object = iris, file = "my_data_frame.rds")
```

To load it data back in:

```
iris2 <- readRDS(file = "my_data_frame.rds")
```

To save a multiple objects we can use [save\(\)](#) and output as .Rdata.

Example, to save 2 dataframes: iris and cars

```
save(iris, cars, file = "myIrisAndCarsData.Rdata")
```

To load:

```
load("myIrisAndCarsData.Rdata")
```

## Section 96.2: Environments

The functions [save](#) and [load](#) allow us to specify the environment where the object will be hosted:

```
save(iris, cars, file = "myIrisAndCarsData.Rdata", envir = foo <- new.env())
load("myIrisAndCarsData.Rdata", envir = foo)
foo$cars

save(iris, cars, file = "myIrisAndCarsData.Rdata", envir = foo <- new.env())
load("myIrisAndCarsData.Rdata", envir = foo)
foo$cars
```

# 第97章：回收利用

## 第97.1节：子集中的回收利用

回收利用可以巧妙地简化代码。

### 子集选择

如果我们想保留向量中的每第三个元素，可以这样做：

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

这里逻辑表达式被扩展到了向量的长度。

我们也可以使用回收机制进行比较：

```
my_vec <- c("foo", "bar", "soap", "mix")
my_vec == "bar"
[1] FALSE TRUE FALSE FALSE
```

这里“bar”被回收使用。

# Chapter 97: Recycling

## Section 97.1: Recycling use in subsetting

Recycling can be used in a clever way to simplify code.

### Subsetting

If we want to keep every third element of a vector we can do the following:

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

Here the logical expression was expanded to the length of the vector.

We can also perform comparisons using recycling:

```
my_vec <- c("foo", "bar", "soap", "mix")
my_vec == "bar"
[1] FALSE TRUE FALSE FALSE
```

Here “bar” gets recycled.

# 第98章：表达式：解析 + 计算

## 第98.1节：执行字符串格式的代码

在此示例中，我们想要执行存储为字符串格式的代码。

```
字符串
str <- "1+1"

字符串不是表达式。
is.expression(str)
[1] FALSE

eval(str)
[1] "1+1"

parse 将字符串转换为表达式
parsed.str <- parse(text="1+1")

is.expression(parsed.str)
[1] TRUE

eval(parsed.str)
[1] 2
```

# Chapter 98: Expression: parse + eval

## Section 98.1: Execute code in string format

In this example, we want to execute code which is stored in a string format.

```
the string
str <- "1+1"

A string is not an expression.
is.expression(str)
[1] FALSE

eval(str)
[1] "1+1"

parse convert string into expressions
parsed.str <- parse(text="1+1")

is.expression(parsed.str)
[1] TRUE

eval(parsed.str)
[1] 2
```

# 第99章：R中的正则表达式语法

本文档介绍了R中使用的正则表达式基础。有关R正则表达式语法的更多信息，请参见?regex。有关正则表达式操作符的完整列表，请参见此ICU正则表达式指南。

## 第99.1节：使用`grep`在字符串向量中查找字符串

```
通用语法：
grep(<pattern>, <character vector>)
```

```
mystring <- c('数字5',
 '数字8',
 '1是最孤独的数字',
 '公司，3是',
 'Git SSH标签是git@github.com',
 '我的个人网站是www.personal.org',
 '路径/到/我的/文件')
```

```
grep('5', mystring)
[1] 1
grep('@', mystring)
[1] 5
grep('数字', mystring)
[1] 1 2 3
```

x|y 表示查找 "x" 或 "y"

```
grep('5|8', mystring)
[1] 1 2
grep('com|org', mystring)
[1] 5 6
```

. 是正则表达式中的特殊字符。它表示“匹配任意字符”

```
grep('The number .', mystring)
[1] 1 2
```

匹配点号时要小心！

```
tricky <- c('www.personal.org', '我的朋友是个赛博格')
grep('.org', tricky)
[1] 1 2
```

要匹配字面字符，必须用反斜杠 (\) 转义字符串。然而，R 在创建字符串时会尝试识别转义字符，所以实际上你需要对反斜杠本身进行转义（即需要对正则表达式字符进行双重转义）。

```
grep('\.org', tricky)
错误：'.' 是字符字符串中未识别的转义，起始于 "\."
grep('\\.org', tricky)
[1] 1
```

如果你想匹配多个字符中的一个，可以将这些字符放在方括号内 ([])

```
grep('[13]', mystring)
```

# Chapter 99: Regular Expression Syntax in R

This document introduces the basics of regular expressions as used in R. For more information about R's regular expression syntax, see [?regex](#). For a comprehensive list of regular expression operators, see [this ICU guide on regular expressions](#).

## Section 99.1: Use `grep` to find a string in a character vector

```
General syntax:
grep(<pattern>, <character vector>)
```

```
mystring <- c('The number 5',
 'The number 8',
 '1 is the loneliest number',
 'Company, 3 is',
 'Git SSH tag is git@github.com',
 'My personal site is www.personal.org',
 'path/to/my/file')
```

```
grep('5', mystring)
[1] 1
grep('@', mystring)
[1] 5
grep('number', mystring)
[1] 1 2 3
```

x|y means look for "x" or "y"

```
grep('5|8', mystring)
[1] 1 2
grep('com|org', mystring)
[1] 5 6
```

. is a special character in Regex. It means "match any character"

```
grep('The number .', mystring)
[1] 1 2
```

Be careful when trying to match dots!

```
tricky <- c('www.personal.org', 'My friend is a cyborg')
grep('.org', tricky)
[1] 1 2
```

To match a literal character, you have to escape the string with a backslash (\). However, R tries to look for escape characters when creating strings, so you actually need to escape the backslash itself (i.e. you need to *double escape* regular expression characters.)

```
grep('\\.org', tricky)
Error: '\.' is an unrecognized escape in character string starting "'\.'"
grep('\\\\.org', tricky)
[1] 1
```

If you want to match one of several characters, you can wrap those characters in brackets ([ ])

```
grep('[13]', mystring)
```

```
[1] 3 4
grep('[@/]', mystring)
[1] 5 7
```

指明字符序列可能很有用。例如，[0-4] 将匹配 0、1、2、3 或 4，[A-Z] 将匹配任何大写字母，[A-z] 将匹配任何大写或小写字母，[A-z0-9] 将匹配任何字母或数字（即所有字母数字字符）

```
grep('[0-4]', mystring)
[1] 3 4
grep('[A-Z]', mystring)
[1] 1 2 4 5 6
```

R 还有几个可以在方括号中使用的快捷类。例如，[:lower:] 是 a-z 的简写，[:upper:] 是 A-Z 的简写，[:alpha:] 是 A-z，[:digit:] 是 0-9，[:alnum:] 是 A-z0-9。注意，这些完整表达式必须放在方括号内；例如，要匹配单个数字，可以使用 [[:digit:]]（注意双重方括号）。另一个例子，[@[:digit:]/] 将匹配字符 @、/ 或 0-9。

```
grep('[[:digit:]]', mystring)
[1] 1 2 3 4
grep('@[:digit:]/', mystring)
[1] 1 2 3 4 5 7
```

方括号也可以用来通过脱字符 (^) 来否定匹配。例如，[^5] 将匹配除

```
grep('The number [^5]', mystring)
[1] 2
```

```
[1] 3 4
grep('[@/]', mystring)
[1] 5 7
```

It may be useful to indicate character sequences. E.g. [0-4] will match 0, 1, 2, 3, or 4, [A-Z] will match any uppercase letter, [A-z] will match any uppercase or lowercase letter, and [A-z0-9] will match any letter or number (i.e. all alphanumeric characters)

```
grep('[0-4]', mystring)
[1] 3 4
grep('[A-Z]', mystring)
[1] 1 2 4 5 6
```

R also has several shortcut classes that can be used in brackets. For instance, [:lower:] is short for a-z, [:upper:] is short for A-Z, [:alpha:] is A-z, [:digit:] is 0-9, and [:alnum:] is A-z0-9. Note that these *whole expressions* must be used inside brackets; for instance, to match a single digit, you can use [[:digit:]] (note the double brackets). As another example, [@[:digit:]/] will match the characters @, / or 0-9.

```
grep('[[:digit:]]', mystring)
[1] 1 2 3 4
grep('@[:digit:]/', mystring)
[1] 1 2 3 4 5 7
```

Brackets can also be used to negate a match with a carat (^). For instance, [^5] will match any character other than "5".

```
grep('The number [^5]', mystring)
[1] 2
```

# 第100章：正则表达式 (regex)

正则表达式（也称为“regex”或“regexp”）定义了可以与字符串匹配的模式。输入  
?regex 查看官方R文档，更多细节请参见Regex文档。最重要的“陷阱”是在SO的regex/topics中不会学到的，即大多数R 正则函数在pattern参数中需要使用成对的反斜杠进行转义。

## 第100.1节：Perl和POSIX正则表达式的区别

R中实现了两种略有不同的正则表达式引擎。默认的是称为  
POSIX兼容的；R中的所有正则函数也都配备了选项，可以开启后一种类型：perl = TRUE。

### 前瞻/后顾

perl = TRUE 启用正则表达式中的前瞻和后顾功能。

- "(?<=A)B" 匹配字母 B 的出现，前提是它前面紧跟着 A，即 "ABACADABRA" 会被匹配，但 "abacadabra" 和 "aBacadabra" 不会被匹配。

## 第100.2节：验证“YYYYMMDD”格式的日期

通常的做法是使用日期作为文件名前缀，格式如下：YYYYMMDD，例如：  
20170101\_results.csv。可以使用以下正则表达式验证这种字符串格式的日期：

```
\d{4}(0[1-9]|1[012])(0[1-9]|12)[0-9]|3[01])
```

上述表达式考虑了年份范围：0000-9999，月份范围：01-12，日期范围：01-31。

例如：

```
> grep1("\d{4}(0[1-9]|1[012])(0[1-9]|12)[0-9]|3[01])", "20170101")
[1] TRUE
> grep1("\d{4}(0[1-9]|1[012])(0[1-9]|12)[0-9]|3[01])", "20171206")
[1] TRUE
> grep1("\d{4}(0[1-9]|1[012])(0[1-9]|12)[0-9]|3[01])", "29991231")
[1] TRUE
```

注意：它验证的是日期的语法，但可能出现语法正确但日期错误的情况，例如：20170229（2017年不是闰年）。

```
> grep1("\d{4}(0[1-9]|1[012])(0[1-9]|12)[0-9]|3[01])", "20170229")
[1] TRUE
```

如果你想验证一个日期，可以通过这个用户自定义函数来实现：

```
is.Date <- function(x) {return(!is.na(as.Date(as.character(x), format = '%Y%m%d')))}
```

然后

```
> is.Date(c("20170229", "20170101", 20170101))
[1] FALSE TRUE TRUE
```

# Chapter 100: Regular Expressions (regex)

Regular expressions (also called "regex" or "regexp") define patterns that can be matched against a string. Type  
?regex for the official R documentation and see the Regex Docs for more details. The most important 'gotcha' that  
will not be learned in the SO regex/topics is that most R-regex functions need the use of paired backslashes to  
escape in a pattern parameter.

## Section 100.1: Differences between Perl and POSIX regex

There are two ever-so-slightly different engines of regular expressions implemented in R. The default is called  
POSIX-consistent; all regex functions in R are also equipped with an option to turn on the latter type: perl = TRUE.

### Look-ahead/look-behind

perl = TRUE enables look-ahead and look-behind in regular expressions.

- "(?<=A)B" matches an appearance of the letter B *only if* it's preceded by A, i.e. "ABACADABRA" would be  
matched, but "abacadabra" and "aBacadabra" would not.

## Section 100.2: Validate a date in a "YYYYMMDD" format

It is a common practice to name files using the date as prefix in the following format: YYYYMMDD, for example:  
20170101\_results.csv. A date in such string format can be verified using the following regular expression:

```
\d{4}(0[1-9]|1[012])(0[1-9]|12)[0-9]|3[01])
```

The above expression considers dates from year: 0000-9999, months between: 01-12 and days 01-31.

For example:

```
> grep1("\d{4}(0[1-9]|1[012])(0[1-9]|12)[0-9]|3[01])", "20170101")
[1] TRUE
> grep1("\d{4}(0[1-9]|1[012])(0[1-9]|12)[0-9]|3[01])", "20171206")
[1] TRUE
> grep1("\d{4}(0[1-9]|1[012])(0[1-9]|12)[0-9]|3[01])", "29991231")
[1] TRUE
```

**Note:** It validates the date syntax, but we can have a wrong date with a valid syntax, for example: 20170229 (2017 it  
is not a leap year).

```
> grep1("\d{4}(0[1-9]|1[012])(0[1-9]|12)[0-9]|3[01])", "20170229")
[1] TRUE
```

If you want to validate a date, it can be done via this user defined function:

```
is.Date <- function(x) {return(!is.na(as.Date(as.character(x), format = '%Y%m%d')))}
```

Then

```
> is.Date(c("20170229", "20170101", 20170101))
[1] FALSE TRUE TRUE
```

## 第100.3节：R正则表达式模式中的转义字符

由于R和正则表达式都使用转义字符"\\"，构建正确的grep、sub、gsub或任何接受模式参数的函数的模式时，通常需要成对的反斜杠。如果你构建一个包含三项的字符向量，其中一项有换行符，另一项有制表符，还有一项没有，且希望将换行符或制表符替换成4个空格，那么构造时需要一个反斜杠，但匹配时需要成对的反斜杠：

```
x <- c("ab", "cd", "e f")x # 它是如何存储的
[1] "ab" "cd" "e f"
cat(x) # 用cat显示时的样子
#a
#b c d e f

gsub(patt="\n|\t", repl=" ", x)
#[1] "a b" "c d" "e f"
```

请注意，pattern参数（如果它是第一个参数且只需部分拼写，则为可选）是唯一需要对转义字符进行双写或配对的参数。replacement参数不需要对需要转义的字符进行双写。如果你想将所有换行符和4个空格替换为制表符，代码将是：

```
gsub("\n| ", "", x)#[1] "ab
" "cd" "ef"
```

## 第100.4节：验证美国州的邮政缩写

下面的正则表达式包括50个州以及联邦/领地（参见[www.50states.com](http://www.50states.com)）：

```
regex <-
"(A[LKSZR])|(C[AOT])|(D[EC])|(F[ML])|(G[AU])|(HI)|(I[DLNA])|(K[SY])|(LA)|(M[EHDAINSOT])|(N[EVHJMYCD])|(MP)|(O[HKR])|(P[WAR])|(RI)|(S[CD])|(T[NX])|(UT)|(V[TIA])|(W[AVIY])"
```

例如：

```
> test <- c("AL", "AZ", "AR", "AJ", "AS", "DC", "FM", "GU", "PW", "FL", "AJ", "AP")
> grep1(us.states.pattern, test)
[1] TRUE TRUE TRUE FALSE TRUE TRUE TRUE TRUE FALSE FALSE
>
```

注意：

如果您只想验证50个州，则建议使用R数据集：state.abb来自state，例如：

```
> data(state)
> test %in% state.abb
[1] TRUE TRUE TRUE FALSE FALSE FALSE FALSE TRUE FALSE FALSE
```

我们仅对50个州的缩写获得TRUE：AL, AZ, AR, FL。

## 第100.5节：验证美国电话号码

以下正则表达式：

## Section 100.3: Escaping characters in R regex patterns

Since both R and regex share the escape character, "\\", building correct patterns for **grep**, **sub**, **gsub** or any other function that accepts a pattern argument will often need pairing of backslashes. If you build a three item character vector in which one items has a linefeed, another a tab character and one neither, and the desire is to turn either the linefeed or the tab into 4-spaces then a single backslash is need for the construction, but paired backslashes for matching:

```
x <- c("a\nb", "c\tb", "e f")
x # how it's stored
[1] "a\nb" "c\tb" "e f"
cat(x) # how it will be seen with cat
#a
#b c d e f

gsub(patt="\n|\t", repl=" ", x)
#[1] "a b" "c d" "e f"
```

Note that the pattern argument (which is optional if it appears first and only needs partial spelling) is the only argument to require this doubling or pairing. The replacement argument does not require the doubling of characters needing to be escaped. If you wanted all the linefeeds and 4-space occurrences replaces with tabs it would be:

```
gsub("\n| ", "\t", x)
#[1] "a\tb" "c\tb" "e\tf"
```

## Section 100.4: Validate US States postal abbreviations

The following regex includes 50 states and also Commonwealth/Territory (see [www.50states.com](http://www.50states.com)):

```
regex <-
"(A[LKSZR])|(C[AOT])|(D[EC])|(F[ML])|(G[AU])|(HI)|(I[DLNA])|(K[SY])|(LA)|(M[EHDAINSOT])|(N[EVHJMYCD])|(MP)|(O[HKR])|(P[WAR])|(RI)|(S[CD])|(T[NX])|(UT)|(V[TIA])|(W[AVIY])"
```

For example:

```
> test <- c("AL", "AZ", "AR", "AJ", "AS", "DC", "FM", "GU", "PW", "FL", "AJ", "AP")
> grep1(us.states.pattern, test)
[1] TRUE TRUE TRUE FALSE TRUE TRUE TRUE TRUE FALSE FALSE
>
```

**Note:**

If you want to verify only the 50 States, then we recommend to use the R-dataset: **state.abb** from state, for example:

```
> data(state)
> test %in% state.abb
[1] TRUE TRUE TRUE FALSE FALSE FALSE FALSE TRUE FALSE FALSE
```

We get TRUE only for 50-States abbreviations: AL, AZ, AR, FL.

## Section 100.5: Validate US phone numbers

The following regular expression:

```
us.phones.regex <- "^\\s*(\\|+|\\s*1(-?|\\s+))*[0-9]{3}\\s*-?\\s*[0-9]{3}\\s*-?\\s*[0-9]{4}$"
```

验证格式为：+1-xxx-xxx-xxxx的电话号码，包括每组数字开头/结尾的可选空白，但中间不允许空白，例如：+1-x  
xx-xxx-xx xx无效。分隔符-可以用空格替代：xxx xxx xxx，或者无分隔符：xxxxxxxxxx。前缀+1是可选的。

让我们检查一下：

```
us.phones.regex <- "^\\s*(\\|+|\\s*1(-?|\\s+))*[0-9]{3}\\s*-?\\s*[0-9]{3}\\s*-?\\s*[0-9]{4}$"
```

```
phones.OK <- c("305-123-4567", "305 123 4567", "+1-786-123-4567",
 "+1 786 123 4567", 7861234567, 786 - 123 4567, + 1 786 - 123 4567)
```

```
phones.NOK <- c("124-456-78901", "124-456-789", "124-456-78 90",
 "124-45 6-7890", "12 4-456-7890")
```

有效案例：

```
> grep1(us.phones.regex, phones.OK)
[1] TRUE TRUE TRUE TRUE TRUE TRUE
>
```

无效情况：

```
> grep1(us.phones.regex, phones.NOK)
[1] FALSE FALSE FALSE FALSE FALSE
>
```

注意：

- \\s 匹配任何空格、制表符或换行符

```
us.phones.regex <- "^\s*(\|+\s*1(-?|\s+))*[0-9]{3}\s*-?\s*[0-9]{3}\s*-?\s*[0-9]{4}$"
```

Validates a phone number in the form of: +1-xxx-xxx-xxxx, including optional leading/trailing blanks at the beginning/end of each group of numbers, but not in the middle, for example: +1-xxx-xxx-xx xx is not valid. The - delimiter can be replaced by blanks: xxx xxx xxx or without delimiter: xxxxxxxxxxxx. The +1 prefix is optional.

Let's check it:

```
us.phones.regex <- "^\s*(\|+\s*1(-?|\s+))*[0-9]{3}\s*-?\s*[0-9]{3}\s*-?\s*[0-9]{4}$"
```

```
phones.OK <- c("305-123-4567", "305 123 4567", "+1-786-123-4567",
 "+1 786 123 4567", "7861234567", "786 - 123 4567", "+ 1 786 - 123 4567")
```

```
phones.NOK <- c("124-456-78901", "124-456-789", "124-456-78 90",
 "124-45 6-7890", "12 4-456-7890")
```

Valid cases:

```
> grep1(us.phones.regex, phones.OK)
[1] TRUE TRUE TRUE TRUE TRUE TRUE
>
```

Invalid cases:

```
> grep1(us.phones.regex, phones.NOK)
[1] FALSE FALSE FALSE FALSE FALSE
>
```

Note:

- \\s Matches any space, tab or newline character

# 第101章：组合学

## 第101.1节：枚举指定长度的组合

### 不放回抽样

使用combn时，每个向量出现在一列中：

```
combn(LETTERS, 3)

仅显示前10个。
[, 1] [, 2] [, 3] [, 4] [, 5] [, 6] [, 7] [, 8] [, 9] [, 10]
[1,] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
[2,] "B" "B" "B" "B" "B" "B" "B" "B" "B" "B"
[3,] "C" "D" "E" "F" "G" "H" "I" "J" "K" "L"
```

### 放回抽样

使用expand.grid时，每个向量出现在一行中：

```
expand.grid(LETTERS, LETTERS, LETTERS)

或者
do.call(expand.grid, rep(list(LETTERS), 3))
```

```
仅显示前10个。
Var1 Var2 Var3
1 A A A
2 B A A
3 C A A
4 D A A
5 E A A
6 F A A
7 G A A
8 H A A
9 I A A
10 J A A
```

对于成对的特殊情况，可以使用outer，将每个向量放入一个单元格中：

```
FUN 这里用作对每个生成的对执行的函数。
在这种情况下是字符串连接。
outer(LETTERS, LETTERS, FUN=paste0)

仅显示前10行和前10列
[, 1] [, 2] [, 3] [, 4] [, 5] [, 6] [, 7] [, 8] [, 9] [, 10]
[1,] "AA" "AB" "AC" "AD" "AE" "AF" "AG" "AH" "AI" "AJ"
[2,] "BA" "BB" "BC" "BD" "BE" "BF" "BG" "BH" "BI" "BJ"
[3,] "CA" "CB" "CC" "CD" "CE" "CF" "CG" "CH" "CI" "CJ"
[4,] "DA" "DB" "DC" "DD" "DE" "DF" "DG" "DH" "DI" "DJ"
[5,] "EA" "EB" "EC" "ED" "EE" "EG" "EH" "EI" "EJ"
[6,] "FA" "FB" "FC" "FD" "FE" "FF" "FG" "FH" "FI" "FJ"
[7,] "GA" "GB" "GC" "GD" "GE" "GF" "GG" "GH" "GI" "GJ"
[8,] "HA" "HB" "HC" "HD" "HE" "HF" "HG" "HH" "HI" "HJ"
[9,] "IA" "IB" "IC" "ID" "IE" "IF" "IG" "IH" "II" "IJ"
[10,] "JA" "JB" "JC" "JD" "JE" "JF" "JG" "JH" "JI" "JJ"
```

# Chapter 101: Combinatorics

## Section 101.1: Enumerating combinations of a specified length

### Without replacement

With `combn`, each vector appears in a column:

```
combn(LETTERS, 3)

Showing only first 10.
[, 1] [, 2] [, 3] [, 4] [, 5] [, 6] [, 7] [, 8] [, 9] [, 10]
[1,] "A" "A" "A" "A" "A" "A" "A" "A" "A" "A"
[2,] "B" "B" "B" "B" "B" "B" "B" "B" "B" "B"
[3,] "C" "D" "E" "F" "G" "H" "I" "J" "K" "L"
```

### With replacement

With `expand.grid`, each vector appears in a row:

```
expand.grid(LETTERS, LETTERS, LETTERS)

or
do.call(expand.grid, rep(list(LETTERS), 3))
```

```
Showing only first 10.
Var1 Var2 Var3
1 A A A
2 B A A
3 C A A
4 D A A
5 E A A
6 F A A
7 G A A
8 H A A
9 I A A
10 J A A
```

For the special case of pairs, `outer` can be used, putting each vector into a cell:

```
FUN here is used as a function executed on each resulting pair.
in this case it's string concatenation.
outer(LETTERS, LETTERS, FUN=paste0)

Showing only first 10 rows and columns
[, 1] [, 2] [, 3] [, 4] [, 5] [, 6] [, 7] [, 8] [, 9] [, 10]
[1,] "AA" "AB" "AC" "AD" "AE" "AF" "AG" "AH" "AI" "AJ"
[2,] "BA" "BB" "BC" "BD" "BE" "BF" "BG" "BH" "BI" "BJ"
[3,] "CA" "CB" "CC" "CD" "CE" "CF" "CG" "CH" "CI" "CJ"
[4,] "DA" "DB" "DC" "DD" "DE" "DF" "DG" "DH" "DI" "DJ"
[5,] "EA" "EB" "EC" "ED" "EE" "EG" "EH" "EI" "EJ"
[6,] "FA" "FB" "FC" "FD" "FE" "FF" "FG" "FH" "FI" "FJ"
[7,] "GA" "GB" "GC" "GD" "GE" "GF" "GG" "GH" "GI" "GJ"
[8,] "HA" "HB" "HC" "HD" "HE" "HF" "HG" "HH" "HI" "HJ"
[9,] "IA" "IB" "IC" "ID" "IE" "IF" "IG" "IH" "II" "IJ"
[10,] "JA" "JB" "JC" "JD" "JE" "JF" "JG" "JH" "JI" "JJ"
```

## 第101.2节：计算指定长度的组合数

不放回抽样

```
choose(length(LETTERS), 5)
[1] 65780
```

放回抽样

```
length(letters)^5
[1] 11881376
```

## Section 101.2: Counting combinations of a specified length

Without replacement

```
choose(length(LETTERS), 5)
[1] 65780
```

With replacement

```
length(letters)^5
[1] 11881376
```

# 第102章：在R中求解常微分方程 (ODE)

## 参数详情

y (命名的)数值向量：ODE系统的初始（状态）值  
times 需要输出的时间序列；times的第一个值必须是初始时间  
func 计算ODE系统中导数值的函数名称  
parms(命名的)数值向量：传递给func的参数  
method使用的积分器，默认：lsoda

## 第102.1节：洛伦兹模型

洛伦兹模型描述了三个状态变量X、Y和Z的动力。模型方程为：

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

初始条件为：

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

a、b和c是三个参数，满足



$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

```
library(deSolve)

定义R函数

Lorenz <- function (t, y, parms) {
 with(as.list(c(y, parms)), {
 dX <- a * X + Y * Z
 dY <- b * (Y - Z)
 dZ <- -X * Y + c * Y - Z

 return(list(c(dX, dY, dZ)))
 })
}

定义参数和变量
```

# Chapter 102: Solving ODEs in R

## Parameter Details

y (named) numeric vector: the initial (state) values for the ODE system  
times time sequence for which output is wanted; the first value of times must be the initial time  
func name of the function that computes the values of the derivatives in the ODE system  
parms (named) numeric vector: parameters passed to func  
method the integrator to use, by default: lsoda

## Section 102.1: The Lorenz model

The Lorenz model describes the dynamics of three state variables, X, Y and Z. The model equations are:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

The initial conditions are:

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

and a, b and c are three parameters with



$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

$$\sigma(t) = \frac{e^t}{e^t + 1} = \frac{1}{1 + e^{-t}}$$

```
library(deSolve)
```

```

Define R-function

Lorenz <- function (t, y, parms) {
 with(as.list(c(y, parms)), {
 dX <- a * X + Y * Z
 dY <- b * (Y - Z)
 dZ <- -X * Y + c * Y - Z

 return(list(c(dX, dY, dZ)))
 })
}

Define parameters and variables
```

```

parms <- c(a = -8/3, b = -10, c = 28)
yini <- c(X = 1, Y = 1, Z = 1)
times <- seq(from = 0, to = 100, by = 0.01)

求解常微分方程 (ODEs)

```

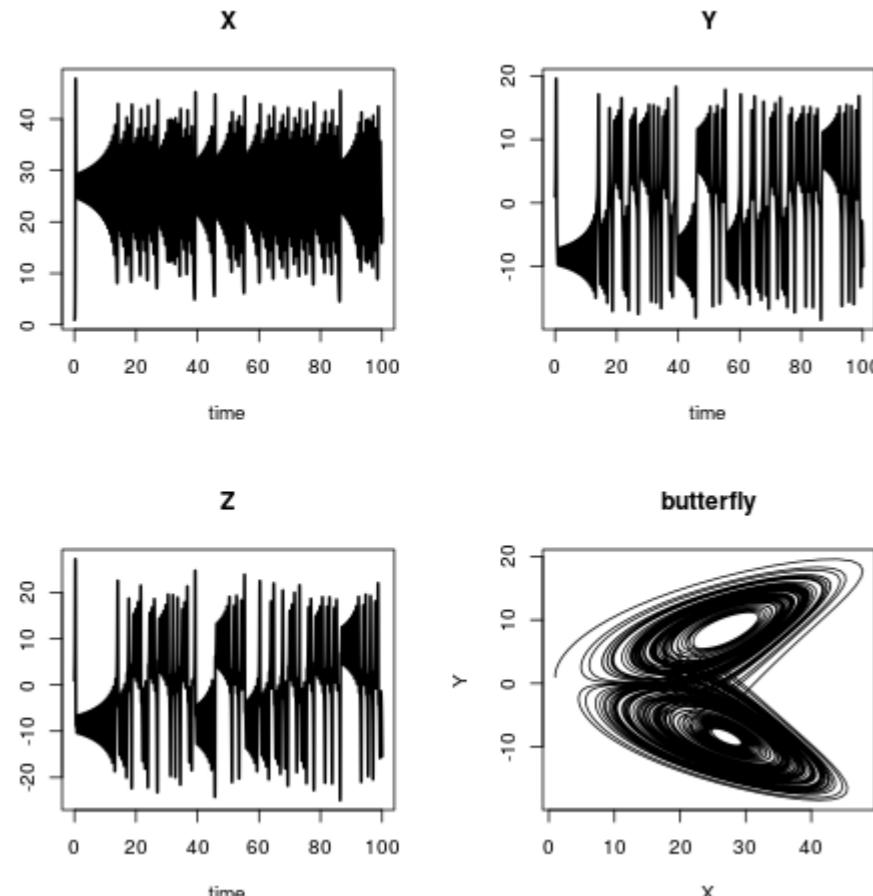
```
out <- ode(y = yini, times = times, func = Lorenz, parms = parms)
```

```

绘制结果

```

```
plot(out, lwd = 2)
plot(out[, "X"], out[, "Y"],
type = "l", xlab = "X",
ylab = "Y", main = "蝴蝶形状")
```



## 第102.2节：洛特卡-沃尔泰拉模型：猎物与捕食者

```

library(deSolve)

定义R函数

```

```

parms <- c(a = -8/3, b = -10, c = 28)
yini <- c(X = 1, Y = 1, Z = 1)
times <- seq(from = 0, to = 100, by = 0.01)

Solve the ODEs

```

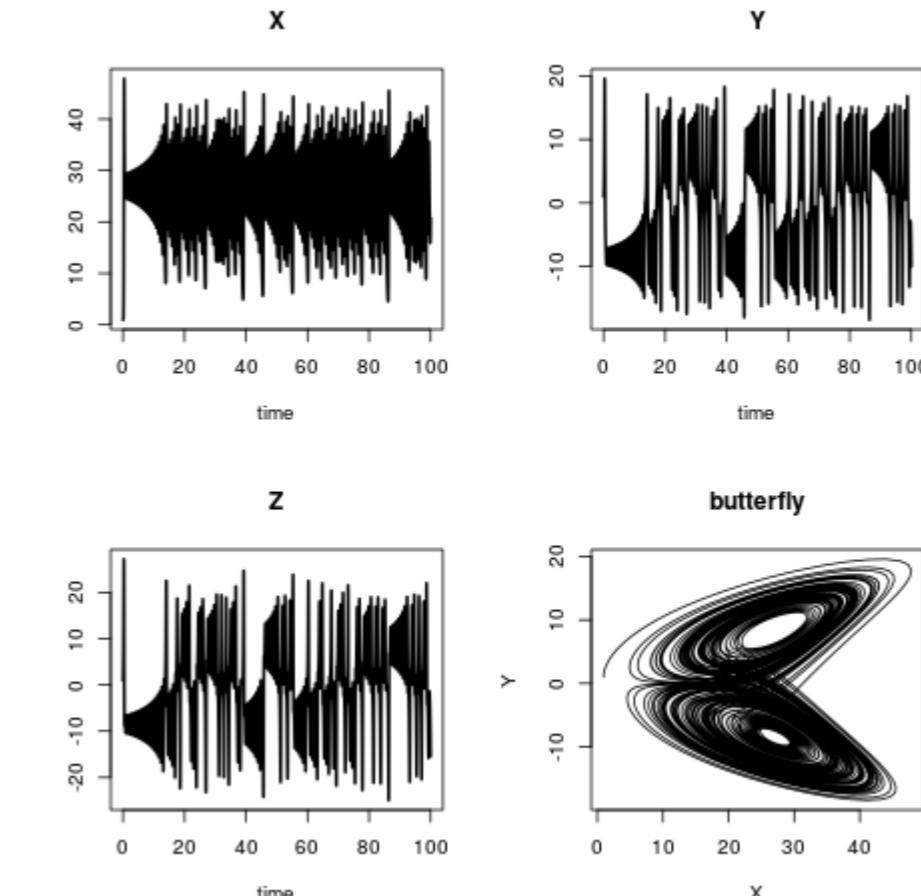
```
out <- ode(y = yini, times = times, func = Lorenz, parms = parms)
```

```

Plot the results

```

```
plot(out, lwd = 2)
plot(out[, "X"], out[, "Y"],
type = "l", xlab = "X",
ylab = "Y", main = "butterfly")
```



## Section 102.2: Lotka-Volterra or: Prey vs. predator

```

library(deSolve)

Define R-function

```

```

LV <- function(t, y, parms) {
with(as.list(c(y, parms)), {

dP <- rG * P * (1 - P/K) - rI * P * C
dC <- rI * P * C * AE - rM * C

return(list(c(dP, dC), sum = C+P))
})
}

定义参数和变量

parms <- c(rI = 0.2, rG = 1.0, rM = 0.2, AE = 0.5, K = 10)
yini <- c(P = 1, C = 2)
times <- seq(from = 0, to = 200, by = 1)

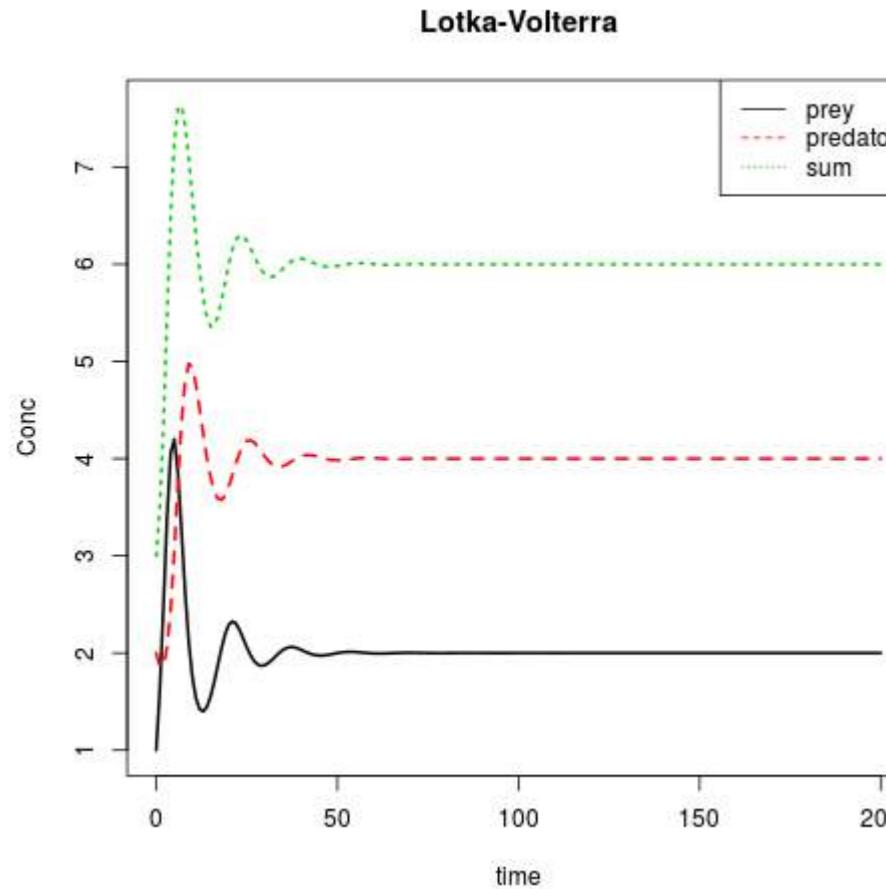
求解常微分方程 (ODEs)

out <- ode(y = yini, times = times, func = LV, parms = parms)

绘制结果

matplot(out[,1], out[,2:4], type = "l", xlab = "time", ylab = "Conc",
main = "Lotka-Volterra", lwd = 2)
legend("topright", c("prey", "predator", "sum"), col = 1:3, lty = 1:3)

```



```

LV <- function(t, y, parms) {
with(as.list(c(y, parms)), {

dP <- rG * P * (1 - P/K) - rI * P * C
dC <- rI * P * C * AE - rM * C

return(list(c(dP, dC), sum = C+P))
})
}

Define parameters and variables

parms <- c(rI = 0.2, rG = 1.0, rM = 0.2, AE = 0.5, K = 10)
yini <- c(P = 1, C = 2)
times <- seq(from = 0, to = 200, by = 1)

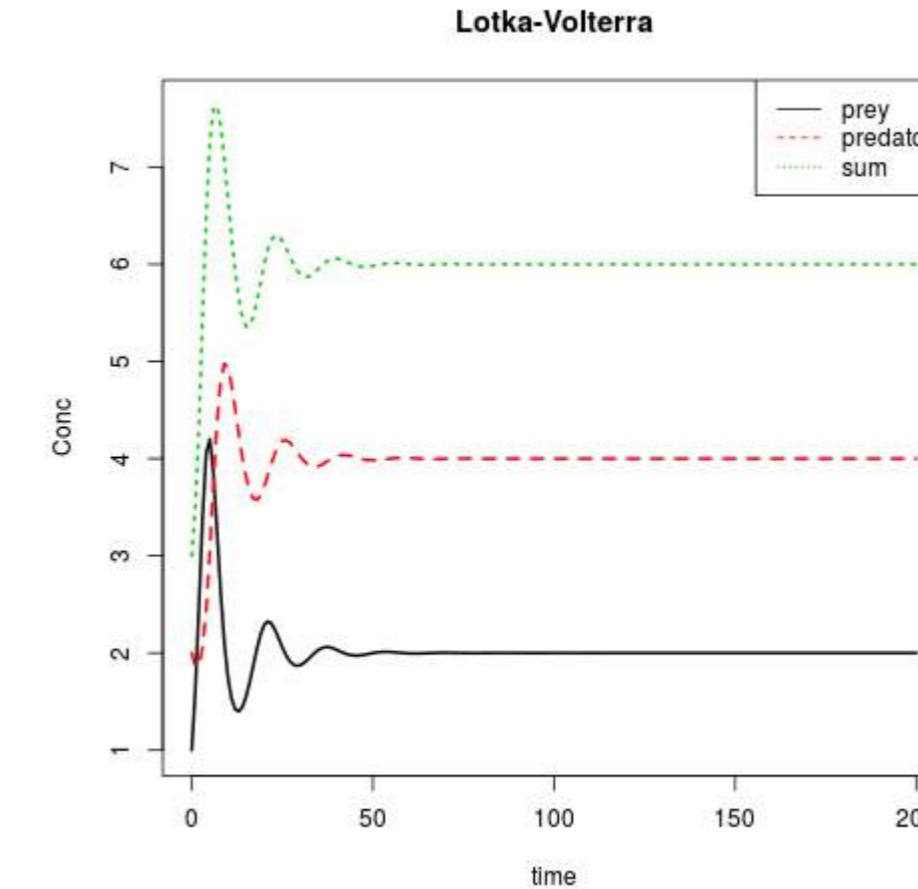
Solve the ODEs

out <- ode(y = yini, times = times, func = LV, parms = parms)

Plot the results

matplot(out[,1], out[,2:4], type = "l", xlab = "time", ylab = "Conc",
main = "Lotka-Volterra", lwd = 2)
legend("topright", c("prey", "predator", "sum"), col = 1:3, lty = 1:3)

```



## 第102.3节：编译语言中的常微分方程（ODE）——在R中的定义

```
library(deSolve)

定义参数和变量

eps <- 0.01;
M <- 10
k <- M * eps^2/2
L <- 1
L0 <- 0.5
r <- 0.1
w <- 10
g <- 1

parameter <- c(eps = eps, M = M, k = k, L = L, L0 = L0, r = r, w = w, g = g)

yini <- c(xl = 0, yl = L0, xr = L, yr = L0,
ul = -L0/L, vl = 0,
ur = -L0/L, vr = 0,
lam1 = 0, lam2 = 0)

times <- seq(from = 0, to = 3, by = 0.01)

定义R函数

caraxis_R <- function(t, y, parms) {
with(as.list(c(y, parms)), {

yb <- r * sin(w * t)
xb <- sqrt(L * L - yb * yb)
Ll <- sqrt(xl^2 + yl^2)
Lr <- sqrt((xr - xb)^2 + (yr - yb)^2)

dxl <- ul; dyl <- vl; dxr <- ur; dyr <- vr

dul <- (L0-Ll) * xl/Ll + 2 * lam2 * (xl-xr) + lam1*xb
dvl <- (L0-Ll) * yl/Ll + 2 * lam2 * (yl-yr) + lam1*yb - k * g

dur <- (L0-Lr) * (xr-xb)/Lr - 2 * lam2 * (xl-xr)
dvr <- (L0-Lr) * (yr-yb)/Lr - 2 * lam2 * (yl-yr) - k * g

c1 <- xb * xl + yb * yl
c2 <- (xl - xr)^2 + (yl - yr)^2 - L * L

return(list(c(dxl, dyl, dxr, dyr, dul, dvl, dur, dvr, c1, c2)))
})
```

## Section 102.3: ODEs in compiled languages - definition in R

```
library(deSolve)

Define parameters and variables

eps <- 0.01;
M <- 10
k <- M * eps^2/2
L <- 1
L0 <- 0.5
r <- 0.1
w <- 10
g <- 1

parameter <- c(eps = eps, M = M, k = k, L = L, L0 = L0, r = r, w = w, g = g)

yini <- c(xl = 0, yl = L0, xr = L, yr = L0,
ul = -L0/L, vl = 0,
ur = -L0/L, vr = 0,
lam1 = 0, lam2 = 0)

times <- seq(from = 0, to = 3, by = 0.01)

Define R-function

caraxis_R <- function(t, y, parms) {
with(as.list(c(y, parms)), {

yb <- r * sin(w * t)
xb <- sqrt(L * L - yb * yb)
Ll <- sqrt(xl^2 + yl^2)
Lr <- sqrt((xr - xb)^2 + (yr - yb)^2)

dxl <- ul; dyl <- vl; dxr <- ur; dyr <- vr

dul <- (L0-Ll) * xl/Ll + 2 * lam2 * (xl-xr) + lam1*xb
dvl <- (L0-Ll) * yl/Ll + 2 * lam2 * (yl-yr) + lam1*yb - k * g

dur <- (L0-Lr) * (xr-xb)/Lr - 2 * lam2 * (xl-xr)
dvr <- (L0-Lr) * (yr-yb)/Lr - 2 * lam2 * (yl-yr) - k * g

c1 <- xb * xl + yb * yl
c2 <- (xl - xr)^2 + (yl - yr)^2 - L * L

return(list(c(dxl, dyl, dxr, dyr, dul, dvl, dur, dvr, c1, c2)))
})
```

## 第102.4节：编译语言中的常微分方程（ODE）——C语言中的定义

```
sink("caraxis_C.c")
cat("
/* 参数和状态变量的合适名称 */

#include <R.h>
```

## Section 102.4: ODEs in compiled languages - definition in C

```
sink("caraxis_C.c")
cat("
/* suitable names for parameters and state variables */

#include <R.h>
```

```

#include <math.h>
static double parms[8];

#define eps parms[0]
#define m parms[1]
#define k parms[2]
#define L parms[3]
#define L0 parms[4]
#define r parms[5]
#define w parms[6]
#define g parms[7]

/*
-----初始化参数公共块-----
*/
void init_C(void (* daeparms)(int *, double *)) {
 int N = 8;
 daeparms(&N, parms);
}
/* 隔间 */

#define xl y[0]
#define yl y[1]
#define xr y[2]
#define yr y[3]
#define lam1 y[8]
#define lam2 y[9]

/*
-----残差函数-----
*/
void caraxis_C (int *neq, double *t, double *y, double *ydot,
 double *yout, int* ip)
{
 double yb, xb, Lr, Ll;

 yb = r * sin(w * *t) ;
 xb = sqrt(L * L - yb * yb);
 Ll = sqrt(xl * xl + yl * yl) ;
 Lr = sqrt((xr-xb)*(xr-xb) + (yr-yb)*(yr-yb));

 ydot[0] = y[4];
 ydot[1] = y[5];
 ydot[2] = y[6];
 ydot[3] = y[7];

 ydot[4] = (L0-Ll) * xl/Ll + lam1*xb + 2*lam2*(xl-xr) ;
 ydot[5] = (L0-Ll) * yl/Ll + lam1*yb + 2*lam2*(yl-yr) - k*g;
 ydot[6] = (L0-Lr) * (xr-xb)/Lr - 2*lam2*(xl-xr) ;
 ydot[7] = (L0-Lr) * (yr-yb)/Lr - 2*lam2*(yl-yr) - k*g ;

 ydot[8] = xb * xl + yb * yl;
 ydot[9] = (xl-xr) * (xl-xr) + (yl-yr) * (yl-yr) - L*L;

}
", fill = TRUE)
sink()
system("R CMD SHLIB caraxis_C")
dyn.load(paste("caraxis_C", .Platform$dynlib.ext, sep = ""))
dllname_C <- dyn.load(paste("caraxis_C", .Platform$dynlib.ext, sep = ""))[[1]]

```

```

#include <math.h>
static double parms[8];

#define eps parms[0]
#define m parms[1]
#define k parms[2]
#define L parms[3]
#define L0 parms[4]
#define r parms[5]
#define w parms[6]
#define g parms[7]

/*
-----initialising the parameter common block-----
*/
void init_C(void (* daeparms)(int *, double *)) {
 int N = 8;
 daeparms(&N, parms);
}
/* Compartments */

#define xl y[0]
#define yl y[1]
#define xr y[2]
#define yr y[3]
#define lam1 y[8]
#define lam2 y[9]

/*
-----the residual function-----
*/
void caraxis_C (int *neq, double *t, double *y, double *ydot,
 double *yout, int* ip)
{
 double yb, xb, Lr, Ll;

 yb = r * sin(w * *t) ;
 xb = sqrt(L * L - yb * yb);
 Ll = sqrt(xl * xl + yl * yl) ;
 Lr = sqrt((xr-xb)*(xr-xb) + (yr-yb)*(yr-yb));

 ydot[0] = y[4];
 ydot[1] = y[5];
 ydot[2] = y[6];
 ydot[3] = y[7];

 ydot[4] = (L0-Ll) * xl/Ll + lam1*xb + 2*lam2*(xl-xr) ;
 ydot[5] = (L0-Ll) * yl/Ll + lam1*yb + 2*lam2*(yl-yr) - k*g;
 ydot[6] = (L0-Lr) * (xr-xb)/Lr - 2*lam2*(xl-xr) ;
 ydot[7] = (L0-Lr) * (yr-yb)/Lr - 2*lam2*(yl-yr) - k*g ;

 ydot[8] = xb * xl + yb * yl;
 ydot[9] = (xl-xr) * (xl-xr) + (yl-yr) * (yl-yr) - L*L;

}
", fill = TRUE)
sink()
system("R CMD SHLIB caraxis_C")
dyn.load(paste("caraxis_C", .Platform$dynlib.ext, sep = ""))
dllname_C <- dyn.load(paste("caraxis_C", .Platform$dynlib.ext, sep = ""))[[1]]

```

## 第102.5节：编译语言中的常微分方程（ODE）——Fortran中的定义

```
sink("caraxis_fortran.f")
cat(
c-----
c 参数公共块的初始化器
c-----
子程序 init_fortran(daeparms)

external daeparms
整数, 参数 :: N = 8
双精度 parms(N)
公共 /myparms/parms

调用 daeparms(N, parms)
返回
结束

c-----
c 变化率
c-----
子程序 caraxis_fortran(neq, t, y, ydot, out, ip)
隐式无
整数 neq, IP(*)
双精度 t, y(neq), ydot(neq), out(*)
双精度 eps, M, k, L, L0, r, w, g
公共 /myparms/ eps, M, k, L, L0, r, w, g

双精度 xl, yl, xr, yr, ul, vl, ur, vr, lam1, lam2
双精度 yb, xb, Ll, Lr, dxl, dyl, dxr, dyr
双精度 dul, dvl, dur, dvr, c1, c2

c 展开状态变量
xl = y(1)
yl = y(2)
xr = y(3)
yr = y(4)
ul = y(5)
vl = y(6)
ur = y(7)
vr = y(8)
lam1 = y(9)
lam2 = y(10)

yb = r * sin(w * t)
xb = sqrt(L * L - yb * yb)
Ll = sqrt(xl**2 + yl**2)
Lr = sqrt((xr - xb)**2 + (yr - yb)**2)

dxl = ul
dyl = vl
dxr = ur
dyr = vr

dul = (L0-Ll) * xl/Ll + 2 * lam2 * (xl-xr) + lam1*xb
dvl = (L0-Ll) * yl/Ll + 2 * lam2 * (yl-yr) + lam1*yb - k*g
dur = (L0-Lr) * (xr-xb)/Lr - 2 * lam2 * (xl-xr)
dvr = (L0-Lr) * (yr-yb)/Lr - 2 * lam2 * (yl-yr) - k*g

c1 = xb * xl + yb * yl
c2 = (xl - xr)**2 + (yl - yr)**2 - L * L
```

## Section 102.5: ODEs in compiled languages - definition in fortran

```
sink("caraxis_fortran.f")
cat(
c-----
c Initialiser for parameter common block
c-----
subroutine init_fortran(daeparms)

external daeparms
integer, parameter :: N = 8
double precision parms(N)
common /myparms/parms

call daeparms(N, parms)
return
end

c-----
c rate of change
c-----
subroutine caraxis_fortran(neq, t, y, ydot, out, ip)
implicit none
integer neq, IP(*)
double precision t, y(neq), ydot(neq), out(*)
double precision eps, M, k, L, L0, r, w, g
common /myparms/ eps, M, k, L, L0, r, w, g

double precision xl, yl, xr, yr, ul, vl, ur, vr, lam1, lam2
double precision yb, xb, Ll, Lr, dxl, dyl, dxr, dyr
double precision dul, dvl, dur, dvr, c1, c2

c expand state variables
xl = y(1)
yl = y(2)
xr = y(3)
yr = y(4)
ul = y(5)
vl = y(6)
ur = y(7)
vr = y(8)
lam1 = y(9)
lam2 = y(10)

yb = r * sin(w * t)
xb = sqrt(L * L - yb * yb)
Ll = sqrt(xl**2 + yl**2)
Lr = sqrt((xr - xb)**2 + (yr - yb)**2)

dxl = ul
dyl = vl
dxr = ur
dyr = vr

dul = (L0-Ll) * xl/Ll + 2 * lam2 * (xl-xr) + lam1*xb
dvl = (L0-Ll) * yl/Ll + 2 * lam2 * (yl-yr) + lam1*yb - k*g
dur = (L0-Lr) * (xr-xb)/Lr - 2 * lam2 * (xl-xr)
dvr = (L0-Lr) * (yr-yb)/Lr - 2 * lam2 * (yl-yr) - k*g

c1 = xb * xl + yb * yl
c2 = (xl - xr)**2 + (yl - yr)**2 - L * L
```

```

c 函数值在 ydot
ydot(1) = dxl
ydot(2) = dyl
ydot(3) = dxr
ydot(4) = dyr
ydot(5) = dul
ydot(6) = dvl
ydot(7) = dur
ydot(8) = dvr
ydot(9) = c1
ydot(10) = c2
return
结束
", fill = TRUE)

sink()
system("R CMD SHLIB caraxis_fortran.f")
dyn.load(paste("caraxis_fortran", .Platform$dynlib.ext, sep = ""))
dllname_fortran <- dyn.load(paste("caraxis_fortran", .Platform$dynlib.ext, sep = ""))[[1]]

```

## 第102.6节：编译语言中的常微分方程（ODE）——基准测试

当你编译并加载之前三个示例中的代码（编译语言中的ODE——R中的定义，编译语言中的ODE——C中的定义，以及编译语言中的ODE——Fortran中的定义）时，你可以运行一个基准测试。

```

library(microbenchmark)

R <- function(){
out <- ode(y = yini, times = times, func = caraxis_R,
 parms = parameter)
}

C <- function(){
out <- ode(y = yini, times = times, func = "caraxis_C",
 initfunc = "init_C", parms = parameter,
 dllname = dllname_C)
}

fortran <- function(){
out <- ode(y = yini, times = times, func = "caraxis_fortran",
 initfunc = "init_fortran", parms = parameter,
 dllname = dllname_fortran)
}

```

检查结果是否相等：

```

all.equal(tail(R()), tail(fortran()))
all.equal(R()[,2], fortran()[,2])
all.equal(R()[,2], C()[,2])

```

进行基准测试（注：在您的机器上，时间当然会不同）：

```

bench <- microbenchmark::microbenchmark(
 R(),
 fortran(),
 C(),
 times = 1000
)

```

```

c function values in ydot
ydot(1) = dxl
ydot(2) = dyl
ydot(3) = dxr
ydot(4) = dyr
ydot(5) = dul
ydot(6) = dvl
ydot(7) = dur
ydot(8) = dvr
ydot(9) = c1
ydot(10) = c2
return
end
", fill = TRUE)

sink()
system("R CMD SHLIB caraxis_fortran.f")
dyn.load(paste("caraxis_fortran", .Platform$dynlib.ext, sep = ""))
dllname_fortran <- dyn.load(paste("caraxis_fortran", .Platform$dynlib.ext, sep = ""))[[1]]

```

## Section 102.6: ODEs in compiled languages - a benchmark test

When you compiled and loaded the code in the three examples before (ODEs in compiled languages - definition in R, ODEs in compiled languages - definition in C and ODEs in compiled languages - definition in fortran) you are able to run a benchmark test.

```

library(microbenchmark)

R <- function(){
 out <- ode(y = yini, times = times, func = caraxis_R,
 parms = parameter)
}

C <- function(){
 out <- ode(y = yini, times = times, func = "caraxis_C",
 initfunc = "init_C", parms = parameter,
 dllname = dllname_C)
}

fortran <- function(){
 out <- ode(y = yini, times = times, func = "caraxis_fortran",
 initfunc = "init_fortran", parms = parameter,
 dllname = dllname_fortran)
}

```

Check if results are equal:

```

all.equal(tail(R()), tail(fortran()))
all.equal(R()[,2], fortran()[,2])
all.equal(R()[,2], C()[,2])

```

Make a benchmark (Note: On your machine the times are, of course, different):

```

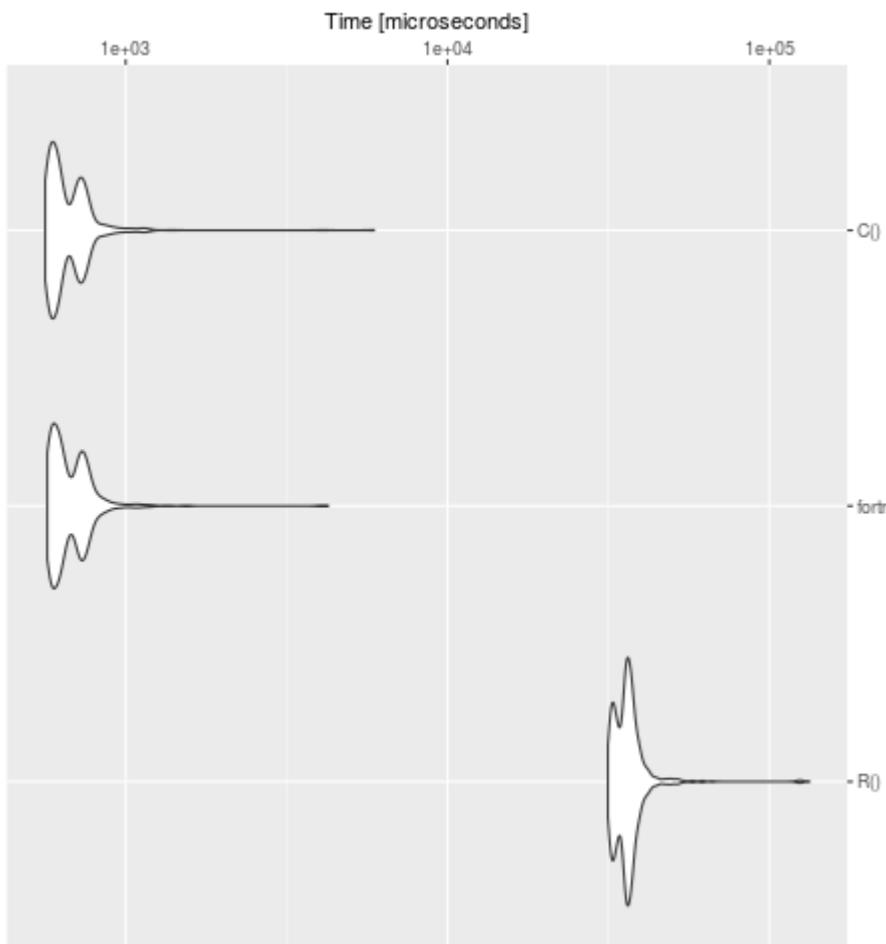
bench <- microbenchmark::microbenchmark(
 R(),
 fortran(),
 C(),
 times = 1000
)

```

)

`summary(bench)`

expr	min	lq	mean	median	uq	max	neval	cld
R()	31508.928	33651.541	36747.8733	36062.2475	37546.8025	132996.564	1000	b
fortran()	570.674	596.700	686.1084	637.4605	730.1775	4256.555	1000	a
C()	562.163	590.377	673.6124	625.0700	723.8460	5914.347	1000	a

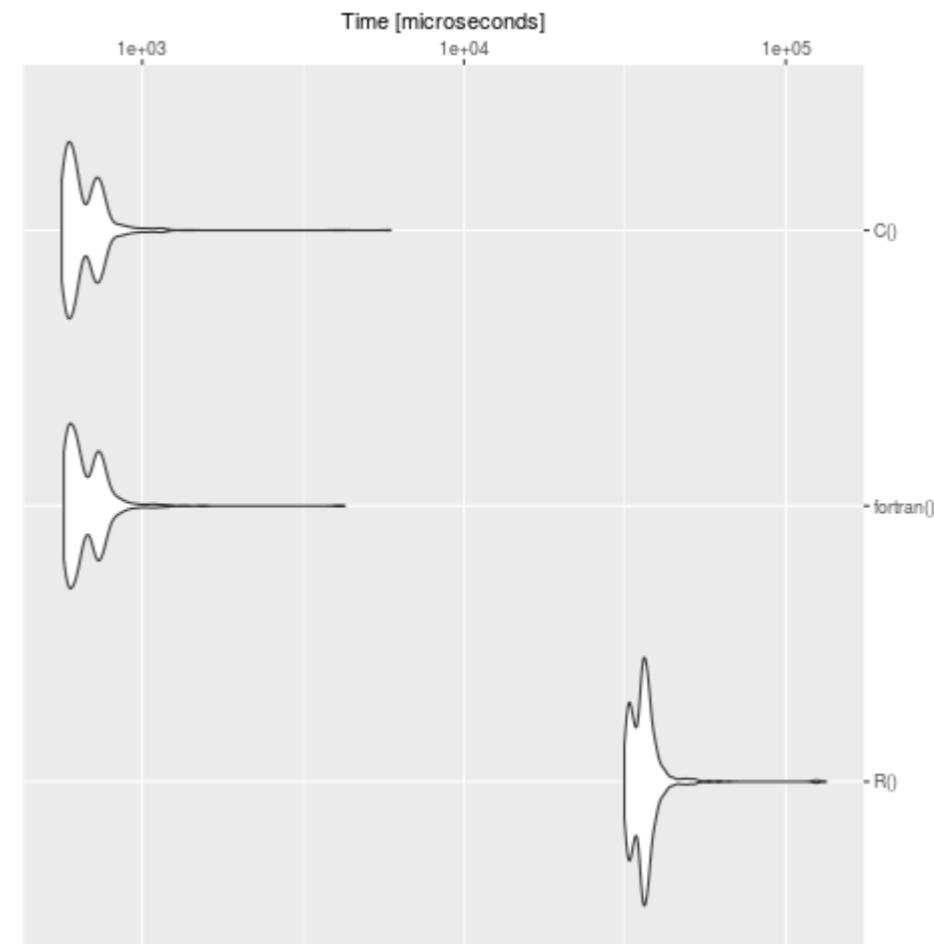


我们可以清楚地看到，R 相较于 C 和 Fortran 的定义来说运行较慢。对于大型模型，值得将问题转换为编译语言。包cOde是将 R 中的常微分方程（ODE）转换为 C 的一种可能方案。

)

`summary(bench)`

expr	min	lq	mean	median	uq	max	neval	cld
R()	31508.928	33651.541	36747.8733	36062.2475	37546.8025	132996.564	1000	b
fortran()	570.674	596.700	686.1084	637.4605	730.1775	4256.555	1000	a
C()	562.163	590.377	673.6124	625.0700	723.8460	5914.347	1000	a



We see clearly, that R is slow in contrast to the definition in C and fortran. For big models it's worth to translate the problem in a compiled language. The package c0de is one possibility to translate ODEs from R to C.

# 第103章：R中的特征选择 -- 去除多余特征

## 第103.1节：去除方差为零或接近零的特征

方差接近零的特征是很好的删除候选。

你可以手动检测低于自定义阈值的数值方差：

```
data("GermanCredit")
variances<-apply(GermanCredit, 2, var)
variances[which(variances<=0.0025)]
```

或者，你可以使用 caret 包来查找接近零的方差。这里的一个优点是它并不是通过数值计算方差来定义接近零方差，而是作为稀有性的函数来定义：

"nearZeroVar 诊断具有唯一值（即零方差预测变量）或同时满足以下两个特征的预测变量：它们相对于样本数量具有非常少的唯一值，并且最常见值的频率与第二常见值的频率之比很大..."

```
library(caret)
names(GermanCredit)[nearZeroVar(GermanCredit)]
```

## 第103.2节：移除缺失值较多的特征

如果某个特征大部分缺少数据，那么它是一个很好的移除候选项：

```
library(VIM)
data(sleep)
colMeans(is.na(sleep))

BodyWgt BrainWgt NonD Dream Sleep Span Gest
0.00000000 0.00000000 0.22580645 0.19354839 0.06451613 0.06451613 0.06451613
Pred Exp Danger
0.00000000 0.00000000 0.00000000
```

在这种情况下，我们可能想要移除NonD和Dream，这两个特征各自大约有20%的缺失值（你的阈值可能不同）

## 第103.3节：移除高度相关的特征

高度相关的特征可能会增加模型的方差，移除相关对中的一个可能有助于减少方差。有很多方法可以检测相关性。这里有一种：

```
library(purrr) # 为了使用 keep()

toCorrelate<-mtcars %>% keep(is.numeric)

计算相关矩阵
```

# Chapter 103: Feature Selection in R -- Removing Extraneous Features

## Section 103.1: Removing features with zero or near-zero variance

A feature that has near zero variance is a good candidate for removal.

You can manually detect numerical variance below your own threshold:

```
data("GermanCredit")
variances<-apply(GermanCredit, 2, var)
variances[which(variances<=0.0025)]
```

Or, you can use the caret package to find near zero variance. An advantage here is that it defines near zero variance not in the numerical calculation of variance, but rather as a function of rarity:

"nearZeroVar diagnoses predictors that have one unique value (i.e. are zero variance predictors) or predictors that have both of the following characteristics: they have very few unique values relative to the number of samples and the ratio of the frequency of the most common value to the frequency of the second most common value is large..."

```
library(caret)
names(GermanCredit)[nearZeroVar(GermanCredit)]
```

## Section 103.2: Removing features with high numbers of NA

If a feature is largely lacking data, it is a good candidate for removal:

```
library(VIM)
data(sleep)
colMeans(is.na(sleep))

BodyWgt BrainWgt NonD Dream Sleep Span Gest
0.00000000 0.00000000 0.22580645 0.19354839 0.06451613 0.06451613 0.06451613
Pred Exp Danger
0.00000000 0.00000000 0.00000000
```

In this case, we may want to remove NonD and Dream, which each have around 20% missing values (your cutoff may vary)

## Section 103.3: Removing closely correlated features

Closely correlated features may add variance to your model, and removing one of a correlated pair might help reduce that. There are lots of ways to detect correlation. Here's one:

```
library(purrr) # in order to use keep()

select correlatable vars
toCorrelate<-mtcars %>% keep(is.numeric)

calculate correlation matrix
```

```
correlationMatrix <- cor(toCorrelate)

pick only one out of each highly correlated pair's mirror image
correlationMatrix[upper.tri(correlationMatrix)]<-0

and I don't remove the highly-correlated-with-itself group
diag(correlationMatrix)<-0

find features that are highly correlated with another feature at the +- 0.85 level
apply(correlationMatrix,2, function(x) any(abs(x)>=0.85))

 mpg cyl disp hp drat wt qsec vs am gear carb
TRUE TRUE TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

我想看看 MPG 与哪些变量相关性这么强，然后决定保留哪些，舍弃哪些。cyl 和 disp 也是同样的处理。或者，我可能需要合并一些高度相关的特征。

I'll want to look at what MPG is correlated to so strongly, and decide what to keep and what to toss. Same for cyl and disp. Alternatively, I might need to combine some strongly correlated features.

# 第104章：RMD中的参考文献

YAML 头部中的参数	详细信息
目录	目录
章节编号	自动为章节编号
参考文献	参考文献文件路径
csl	样式文件路径

## 第104.1节：指定参考文献和引用作者

RMD 文件最重要的部分是 YAML 头部。对于撰写学术论文，我建议使用 PDF 输出、编号章节和目录 (toc)。

```

标题: "在 R 中撰写学术论文"
作者: "作者"
日期: "日期"
输出:
pdf_document:
编号章节: 是
目录: 是
参考文献: bibliography.bib

```

在这个例子中，我们的文件 `bibliography.bib` 内容如下：

```
@ARTICLE{Meyer2000,
作者="伯恩德·迈耶",
标题="基于约束的图示推理框架",
期刊="应用人工智能",
卷号= "14",
期号 = "4",
页码= "327--344",
年份=2000
}
```

要引用.bib文件中提到的作者，请写@和bibkey，例如Meyer2000。

```
引言
`@Meyer2000` 结果 in @Meyer2000.

`@Meyer2000 [p. 328]` 结果 in @Meyer2000 [p. 328]

`[@Meyer2000]` 结果 in [@Meyer2000]

`[-@Meyer2000]` 结果 in [-@Meyer2000]

总结

参考文献
```

通过 RStudio (Ctrl+Shift+K) 或通过控制台 `rmarkdown::render("<path-to-your-RMD-file>")` 渲染 RMD 文件，会得到以下输出：

# Chapter 104: Bibliography in RMD

Parameter in YAML header	Detail
toc	table of contents
number_sections	numbering the sections automatically
bibliography	path to the bibliography file
csl	path to the style file

## Section 104.1: Specifying a bibliography and cite authors

The most important part of your RMD file is the YAML header. For writing an academic paper, I suggest to use PDF output, numbered sections and a table of content (toc).

```

title: "Writing an academic paper in R"
author: "Author"
date: "Date"
output:
pdf_document:
number_sections: yes
toc: yes
bibliography: bibliography.bib

```

In this example, our file `bibliography.bib` looks like this:

```
@ARTICLE{Meyer2000,
AUTHOR="Bernd Meyer",
TITLE="A constraint-based framework for diagrammatic reasoning",
JOURNAL="Applied Artificial Intelligence",
VOLUME= "14",
ISSUE = "4",
PAGES= "327--344",
YEAR=2000
}
```

To cite an author mentioned in your .bib file write @ and the bibkey, e.g. Meyer2000.

```
Introduction
`@Meyer2000` results in @Meyer2000.

`@Meyer2000 [p. 328]` results in @Meyer2000 [p. 328]

`[@Meyer2000]` results in [@Meyer2000]

`[-@Meyer2000]` results in [-@Meyer2000]

Summary

References
```

Rendering the RMD file via RStudio (Ctrl+Shift+K) or via console `rmarkdown::render("<path-to-your-RMD-file>")` results in the following output:

## Writing an academic paper in R

*Author*

*Date*

### Contents

1 Introduction	1
2 Summary	1
References	1

### 1 Introduction

@Meyer2000 results in Meyer (2000).  
@Meyer2000 [p. 328] results in Meyer (2000, 328)  
[@Meyer2000] results in (Meyer 2000)  
[-@Meyer2000] results in (2000)

### 2 Summary

### References

Meyer, Bernd. 2000. "A Constraint-Based Framework for Diagrammatic Reasoning." *Applied Artificial Intelligence* 14 (4): 327–44.

1

## 第 104.2 节：内联引用

如果没有 \*.bib 文件，可以在文档的 YAML 元数据中使用 references 字段。该字段应包含一个 YAML 编码的引用数组，例如：

## Writing an academic paper in R

*Author*

*Date*

### Contents

1 Introduction	1
2 Summary	1
References	1

### 1 Introduction

@Meyer2000 results in Meyer (2000).  
@Meyer2000 [p. 328] results in Meyer (2000, 328)  
[@Meyer2000] results in (Meyer 2000)  
[-@Meyer2000] results in (2000)

### 2 Summary

### References

Meyer, Bernd. 2000. "A Constraint-Based Framework for Diagrammatic Reasoning." *Applied Artificial Intelligence* 14 (4): 327–44.

1

## Section 104.2: Inline references

If you have no \*.bib file, you can use a references field in the document's YAML metadata. This should include an array of YAML-encoded references, for example:

```

标题: "在 R 中撰写学术论文"

作者: "作者"

日期: "日期"

输出:

pdf_document:

编号章节: 是

目录: 是

references:

- id: Meyer2000

标题: 基于约束的图示推理框架

作者:

- 姓: 迈耶

名: 伯恩德

卷号: 14

期号: 4

出版社: 应用人工智能

页码: 327-344

类型: 期刊文章

发行:

年份: 2000

```

# 引言

`@Meyer2000` 结果为 @Meyer2000。

`@Meyer2000 [p. 328]` 结果为 @Meyer2000 [p. 328]

`[@Meyer2000]` 结果为 [@Meyer2000]

`[-@Meyer2000]` 结果为 [-@Meyer2000]

# 总结

# 参考文献

渲染此文件的结果与示例“指定参考文献”中的输出相同。

## 第104.3节：引用样式

默认情况下，pandoc 将使用芝加哥作者-日期格式进行引用和参考文献。要使用其他样式，您需要在csl元数据字段中指定一个CSL 1.0样式文件。以下介绍了一种常用的引用样式——爱思唯尔样式（可从 <https://github.com/citation-style-language/styles> 下载）。样式文件必须存储在与RMD文件相同的目录中，或者必须提交该文件的绝对路径。

要使用非默认样式，请使用以下代码：

```

标题: "在 R 中撰写学术论文"

作者: "作者"

日期: "日期"

输出:

pdf_document:

编号章节: 是

目录: 是

bibliography: bibliography.bib

csl: elsevier-harvard.csl

```

```

title: "Writing an academic paper in R"

author: "Author"

date: "Date"

output:

pdf_document:

number_sections: yes

toc: yes

references:

- id: Meyer2000

title: A Constraint-Based Framework for Diagrammatic Reasoning

author:

- family: Meyer

given: Bernd

volume: 14

issue: 4

publisher: Applied Artificial Intelligence

page: 327-344

type: article-journal

issued:

year: 2000

```

# Introduction

`@Meyer2000` results in @Meyer2000.

`@Meyer2000 [p. 328]` results in @Meyer2000 [p. 328]

`[@Meyer2000]` results in [@Meyer2000]

`[-@Meyer2000]` results in [-@Meyer2000]

# Summary

# References

Rendering this file results in the same output as in example "Specifying a bibliography".

## Section 104.3: Citation styles

By default, pandoc will use a Chicago author-date format for citations and references. To use another style, you will need to specify a CSL 1.0 style file in the csl metadata field. In the following a often used citation style, the elsevier style, is presented (download at <https://github.com/citation-style-language/styles> ). The style-file has to be stored in the same directory as the RMD file OR the absolute path to the file has to be submitted.

To use another style then the default one, the following code is used:

```

title: "Writing an academic paper in R"

author: "Author"

date: "Date"

output:

pdf_document:

number_sections: yes

toc: yes

bibliography: bibliography.bib

csl: elsevier-harvard.csl

```

# 引言

`@Meyer2000` 结果为 @Meyer2000。

`@Meyer2000 [p. 328]` 结果为 @Meyer2000 [p. 328]

`[@Meyer2000]` 结果为 [@Meyer2000]

`[-@Meyer2000]` 结果为 [-@Meyer2000]

# 总结

# 参考文献

# Introduction

`@Meyer2000` results in @Meyer2000.

`@Meyer2000 [p. 328]` results in @Meyer2000 [p. 328]

`[@Meyer2000]` results in [@Meyer2000]

`[-@Meyer2000]` results in [-@Meyer2000]

# Summary

# Reference

# Writing an academic paper in R

*Author*

*Date*

## Contents

1 Introduction	1
2 Summary	1
Reference	1

### 1 Introduction

@Meyer2000 results in Meyer (2000).  
@Meyer2000 [p. 328] results in Meyer (2000, p. 328)  
[@Meyer2000] results in (Meyer, 2000)  
[-@Meyer2000] results in (2000)

### 2 Summary

### Reference

Meyer, B., 2000. A constraint-based framework for diagrammatic reasoning. Applied Artificial Intelligence 14, 327–344.

1

# Writing an academic paper in R

*Author*

*Date*

## Contents

1 Introduction	1
2 Summary	1
Reference	1

### 1 Introduction

@Meyer2000 results in Meyer (2000).  
@Meyer2000 [p. 328] results in Meyer (2000, p. 328)  
[@Meyer2000] results in (Meyer, 2000)  
[-@Meyer2000] results in (2000)

### 2 Summary

### Reference

Meyer, B., 2000. A constraint-based framework for diagrammatic reasoning. Applied Artificial Intelligence 14, 327–344.

1

请注意与示例“指定参考文献并引用作者”的输出差异

Notice the differences to the output of example "Specifying a bibliography and cite authors"

# 第105章：在R中编写函数

## 第105.1节：匿名函数

匿名函数，顾名思义，是没有被赋予名称的函数。当函数是更大操作的一部分，但本身占用空间不大时，这种函数非常有用。匿名函数的一个常见用例是在Base函数的\*apply系列中使用。

计算data.frame中每列的均方根：

```
df <- data.frame(first=5:9, second=(0:4)^2, third=-1:3)
apply(df, 2, function(x) { sqrt(sum(x^2)) })
first second third
15.968719 18.814888 3.872983
```

为矩阵中每一行创建一个从最小值到最大值，步长为1的序列。

```
x <- sample(1:6, 12, replace=TRUE)
mat <- matrix(x, nrow=3)

apply(mat, 1, function(x) { seq(min(x), max(x)) })
```

匿名函数也可以独立存在：

```
(function() { 1 })()
[1] 1
```

等同于

```
f <- function() { 1 }
f()
[1] 1
```

## 第105.2节：RStudio代码片段

这只是一个为经常使用自定义函数的人准备的小技巧。

在RStudio IDE中输入“fun”并按TAB键。

```
1
2 fun
3 fun {snippet} ${1:name} <- function (${2:variables}) {
4 function {base} ${3:code}
5 }
6 functionBody {methods}
7 functionBody<- {methods}
8 }
```

结果将是一个新函数的骨架。

```
name <- function(variables) {
}
```

用户可以轻松定义自己的代码片段模板，例如下面这个

# Chapter 105: Writing functions in R

## Section 105.1: Anonymous functions

An anonymous function is, as the name implies, not assigned a name. This can be useful when the function is a part of a larger operation, but in itself does not take much place. One frequent use-case for anonymous functions is within the `*apply` family of Base functions.

Calculate the root mean square for each column in a `data.frame`:

```
df <- data.frame(first=5:9, second=(0:4)^2, third=-1:3)
apply(df, 2, function(x) { sqrt(sum(x^2)) })
first second third
15.968719 18.814888 3.872983
```

Create a sequence of step-length one from the smallest to the largest value for each row in a matrix.

```
x <- sample(1:6, 12, replace=TRUE)
mat <- matrix(x, nrow=3)

apply(mat, 1, function(x) { seq(min(x), max(x)) })
```

An anonymous function can also stand on its own:

```
(function() { 1 })()
[1] 1
```

is equivalent to

```
f <- function() { 1 }
f()
[1] 1
```

## Section 105.2: RStudio code snippets

This is just a small hack for those who use self-defined functions often.

Type "fun" RStudio IDE and hit TAB.

```
1
2 fun
3 fun {snippet} ${1:name} <- function (${2:variables}) {
4 function {base} ${3:code}
5 }
6 functionBody {methods}
7 functionBody<- {methods}
8 }
```

The result will be a skeleton of a new function.

```
name <- function(variables) {
}
```

One can easily define their own snippet template, i.e. like the one below

```
name <- function(df, x, y) {
 require(tidyverse)
 out <-
 return(out)
}
```

该选项位于“全局选项”->“代码”菜单中的“编辑代码片段”。

## 第105.3节：命名函数

R 充满了函数，毕竟它是一种[函数式编程语言](#)，但有时你需要的精确函数并不包含在基础资源中。你可以安装包含该函数的包，但也许你的需求非常具体，没有现成的函数能满足？那么你只能选择自己编写函数。

函数可以非常简单，甚至几乎毫无意义。它甚至不需要接受参数：

```
one <- function() { 1 }
one()
[1] 1

two <- function() { 1 + 1 }
two()
[1] 2
```

花括号 {} 中的内容就是函数的主体。只要你能将所有内容写在一行，花括号并非严格必要，但它们有助于保持代码的整洁。

函数可以非常简单，但又高度具体。这个函数接受一个向量（本例中为vec）作为输入，输出的是该向量中每个元素减去向量长度（本例中为6）后的结果。

```
vec <- 4:9
subtract.length <- function(x) { x - length(x) }
subtract.length(vec)
[1] -2 -1 0 1 2 3
```

注意，length()本身是一个预先提供的（即Base）函数。当然，你可以在自定义函数中调用之前自定义的函数，也可以在多行代码中赋值变量和执行其他操作：

```
vec2 <- (4:7)/2

msdf <- function(x, multiplier=4) {
 mult <- x * multiplier
 subl <- subtract.length(x)
 data.frame(mult, subl)
}

msdf(vec2, 5)
 mult subl
1 10.0 -2.0
2 12.5 -1.5
3 15.0 -1.0
4 17.5 -0.5
```

multiplier=4 确保当调用时未给出参数 multiplier 的值时，4 是该参数的默认值

```
name <- function(df, x, y) {
 require(tidyverse)
 out <-
 return(out)
}
```

The option is Edit Snippets in the Global Options -> Code menu.

## Section 105.3: Named functions

R is full of functions, it is after all a [functional programming language](#), but sometimes the precise function you need isn't provided in the Base resources. You could conceivably install a package containing the function, but maybe your requirements are just so specific that no pre-made function fits the bill? Then you're left with the option of making your own.

A function can be very simple, to the point of being pretty much pointless. It doesn't even need to take an argument:

```
one <- function() { 1 }
one()
[1] 1

two <- function() { 1 + 1 }
two()
[1] 2
```

What's between the curly braces {} is the function proper. As long as you can fit everything on a single line they aren't strictly needed, but can be useful to keep things organized.

A function can be very simple, yet highly specific. This function takes as input a vector (vec in this example) and outputs the same vector with the vector's length (6 in this case) subtracted from each of the vector's elements.

```
vec <- 4:9
subtract.length <- function(x) { x - length(x) }
subtract.length(vec)
[1] -2 -1 0 1 2 3
```

Notice that `length()` is in itself a pre-supplied (i.e. *Base*) function. You can of course use a previously self-made function within another self-made function, as well as assign variables and perform other operations while spanning several lines:

```
vec2 <- (4:7)/2

msdf <- function(x, multiplier=4) {
 mult <- x * multiplier
 subl <- subtract.length(x)
 data.frame(mult, subl)
}

msdf(vec2, 5)
 mult subl
1 10.0 -2.0
2 12.5 -1.5
3 15.0 -1.0
4 17.5 -0.5
```

`multiplier=4` makes sure that 4 is the default value of the argument `multiplier`, if no value is given when calling

将使用函数4。

以上都是命名的函数的例子，之所以称为命名函数，是因为它们被赋予了名称（one、two、subtract.length等）。

the function 4 is what will be used.

The above are all examples of *named* functions, so called simply because they have been given names (one, two, subtract.length etc.)

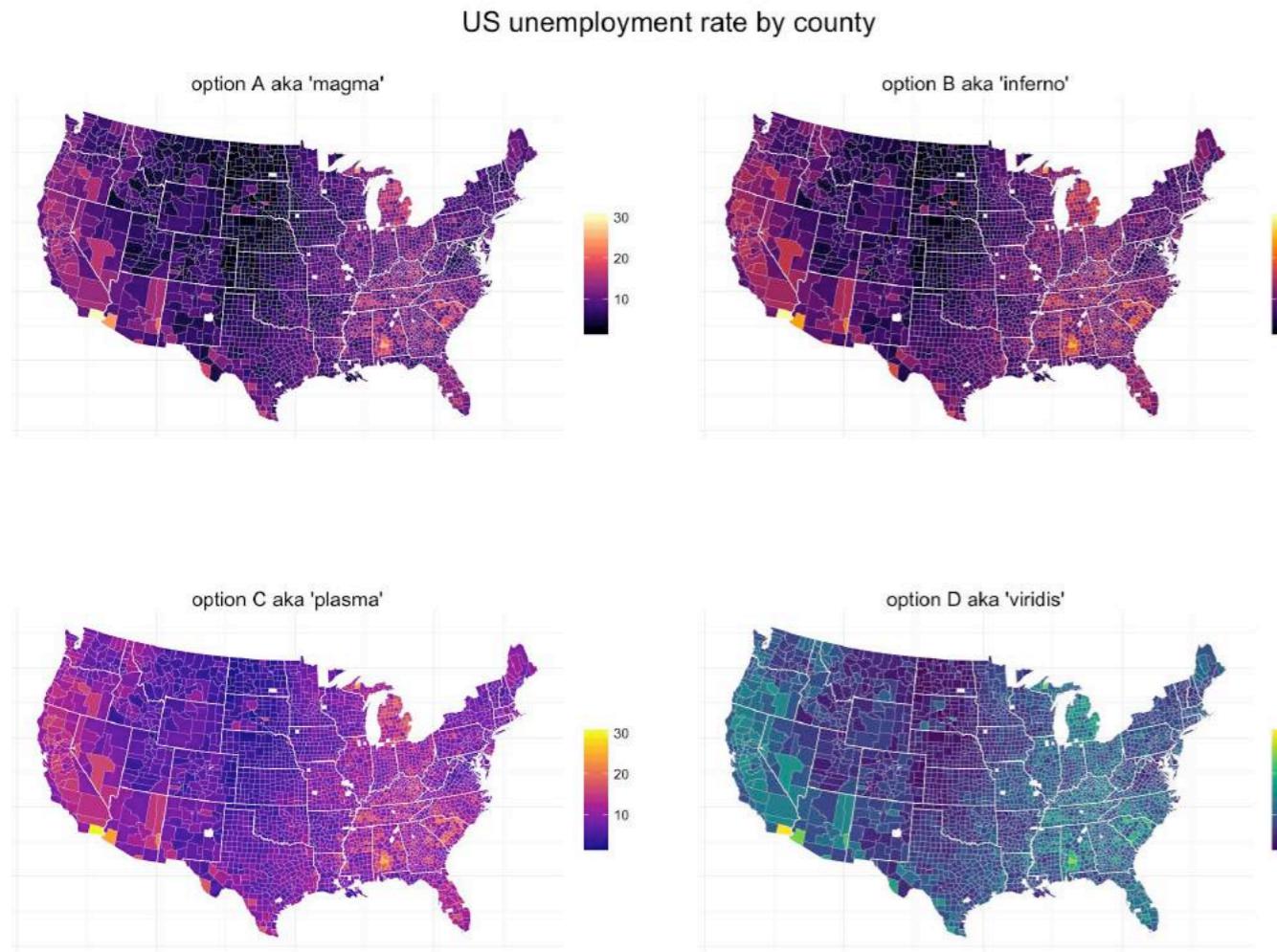
# 第106章：图形的配色方案

## 第106.1节：viridis - 适合打印和色盲友好的调色板

Viridis（以*chromis viridis*鱼命名）是Python库matplotlib最近开发的配色方案

（链接中的视频介绍了该配色方案的开发过程及其主要优点）。它已无缝移植到R语言中。

配色方案有4种变体：magma、plasma、inferno和viridis（默认）。它们通过option参数选择，分别编码为A、B、C和D。要了解这4种配色方案的效果，请查看地图：



(图片来源)

该包可以从CRAN或github安装。

vignette文档对于viridis包非常出色。

viridis配色方案的一个优点是与ggplot2的集成。包内定义了两个ggplot2专用函数：scale\_color\_viridis()和scale\_fill\_viridis()。见下面的示例：

```
library(viridis)
library(ggplot2)

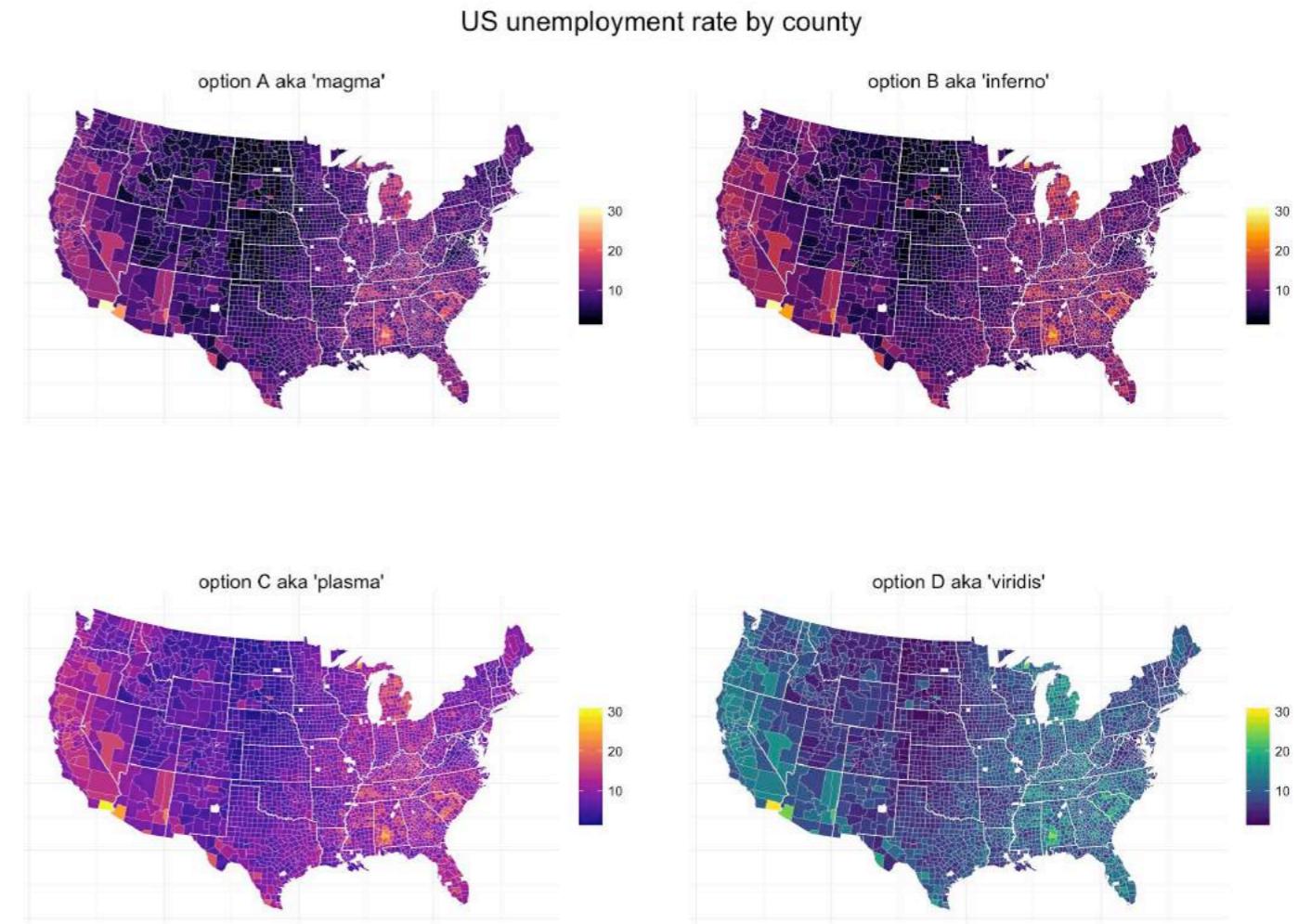
gg1 <- ggplot(mtcars) +
```

# Chapter 106: Color schemes for graphics

## Section 106.1: viridis - print and colorblind friendly palettes

Viridis (named after the *chromis viridis* fish) is a recently developed color scheme for the Python library `matplotlib` (the video presentation by the link explains how the color scheme was developed and what are its main advantages). It is seamlessly ported to R.

There are 4 variants of color schemes: `magma`, `plasma`, `inferno`, and `viridis` (default). They are chosen with the `option` parameter and are coded as A, B, C, and D, correspondingly. To have an impression of the 4 color schemes, look at the maps:



(image souce)

The package can be installed from [CRAN](#) or [github](#).

The [vignette](#) for `viridis` package is just brilliant.

Nice feature of the `viridis` color scheme is integration with `ggplot2`. Within the package two `ggplot2`-specific functions are defined: `scale_color_viridis()` and `scale_fill_viridis()`. See the example below:

```
library(viridis)
library(ggplot2)

gg1 <- ggplot(mtcars) +
```

```

geom_point(aes(x = mpg, y = hp, color = disp), size = 3)+

 scale_color_viridis(option = "B")

theme_minimal()

 theme(legend.position = c(.8,.8))

gg2 <- ggplot(mtcars)+

 geom_violin(aes(x = factor(cyl), y = hp, fill = factor(cyl)))+

 scale_fill_viridis(discrete = T)

theme_minimal()

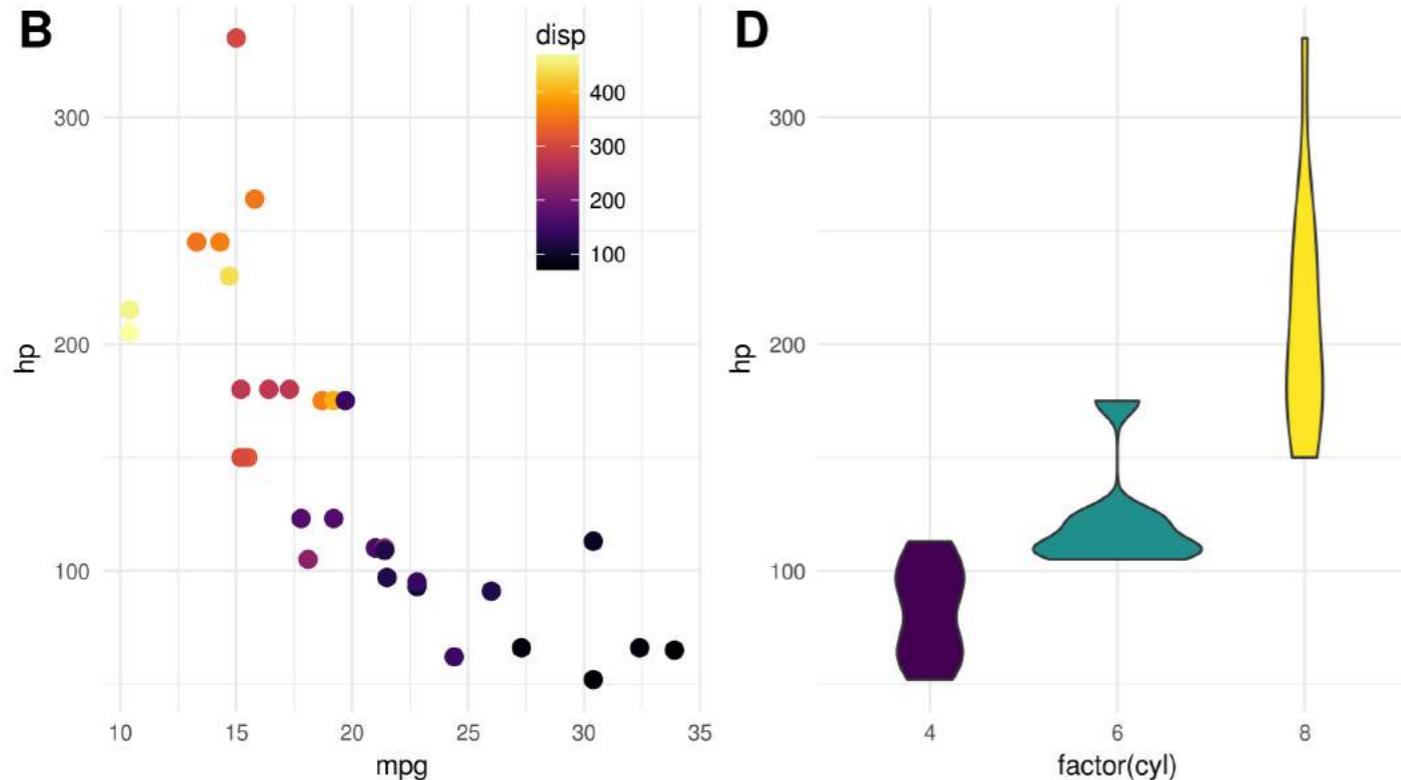
 theme(legend.position = 'none')

library(cowplot)

output <- plot_grid(gg1,gg2, labels = c('B','D'),label_size = 20)

print(output)

```



## 第106.2节：一个方便的函数用于快速查看颜色向量

经常需要快速查看所选的调色板。一种优雅的解决方案是以下自定义函数：

```

color_glimpse <- function(colors_string){

 n <- length(colors_string)

 hist(1:n,breaks=0:n,col=colors_string)

}

```

### 使用示例

```
color_glimpse(blues9)
```

```

geom_point(aes(x = mpg, y = hp, color = disp), size = 3)+

 scale_color_viridis(option = "B")

theme_minimal()

 theme(legend.position = c(.8,.8))

gg2 <- ggplot(mtcars)+

 geom_violin(aes(x = factor(cyl), y = hp, fill = factor(cyl)))+

 scale_fill_viridis(discrete = T)

theme_minimal()

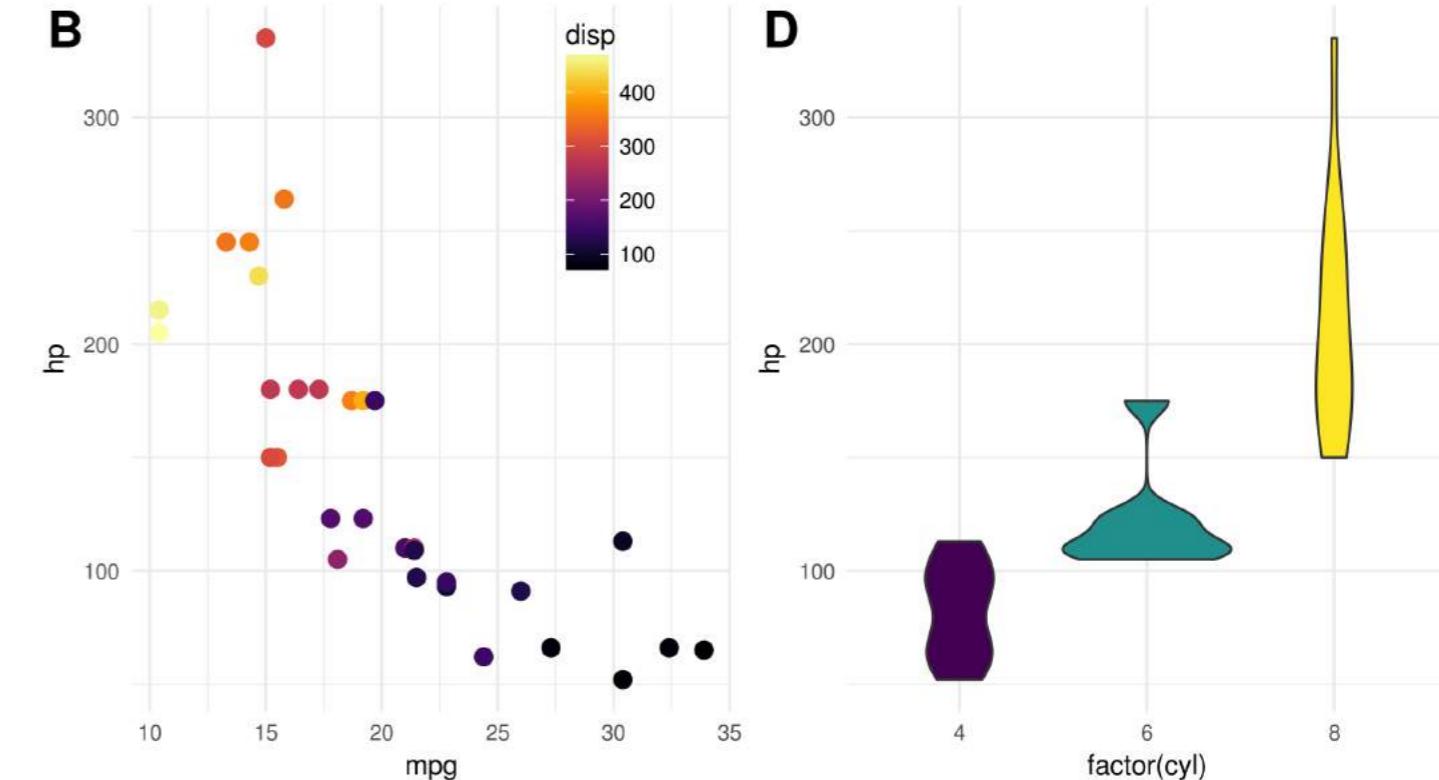
 theme(legend.position = 'none')

library(cowplot)

output <- plot_grid(gg1,gg2, labels = c('B','D'),label_size = 20)

print(output)

```



## Section 106.2: A handy function to glimpse a vector of colors

Quite often there is a need to glimpse the chosen color palette. One elegant solution is the following self defined function:

```

color_glimpse <- function(colors_string){

 n <- length(colors_string)

 hist(1:n,breaks=0:n,col=colors_string)

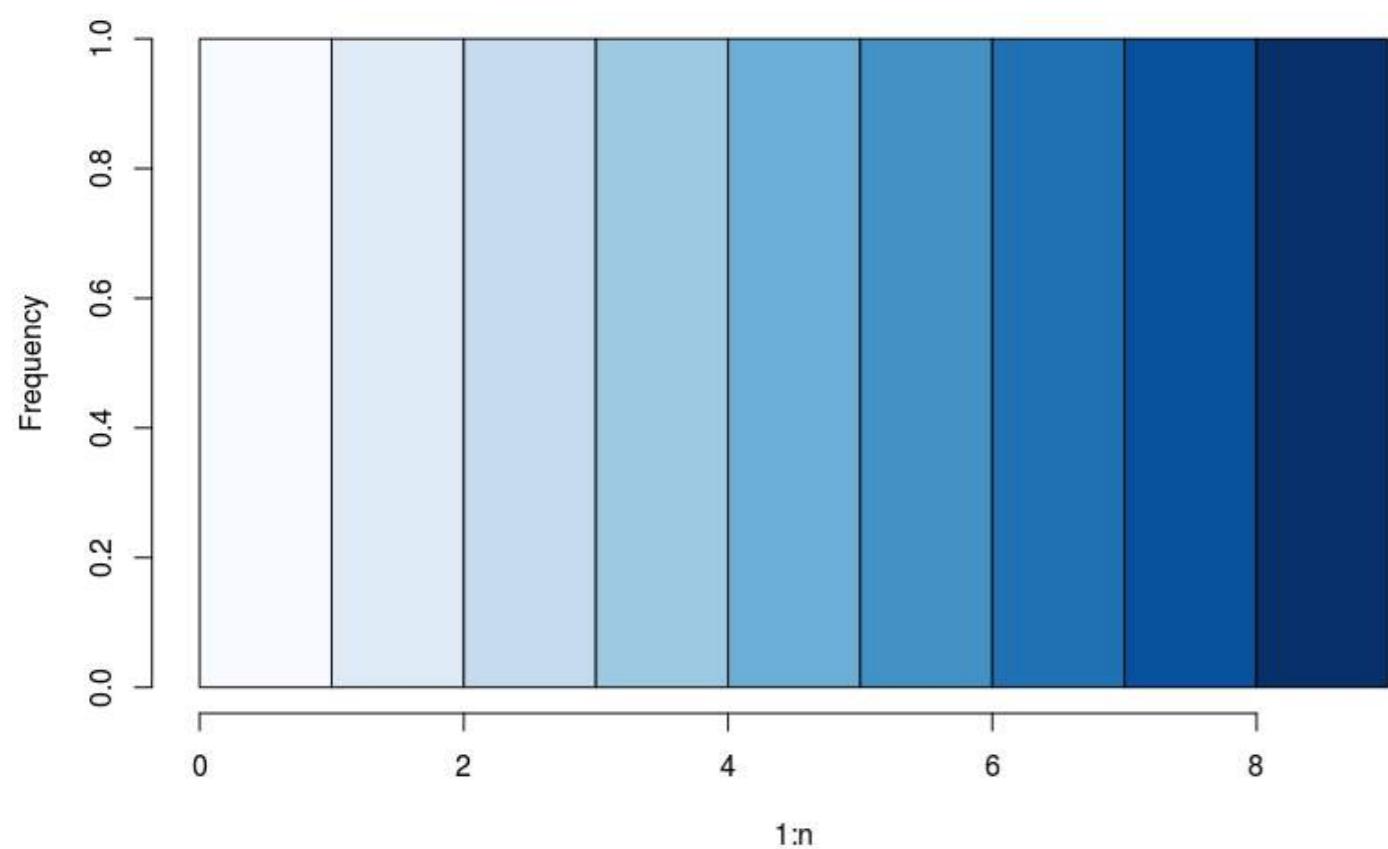
}

```

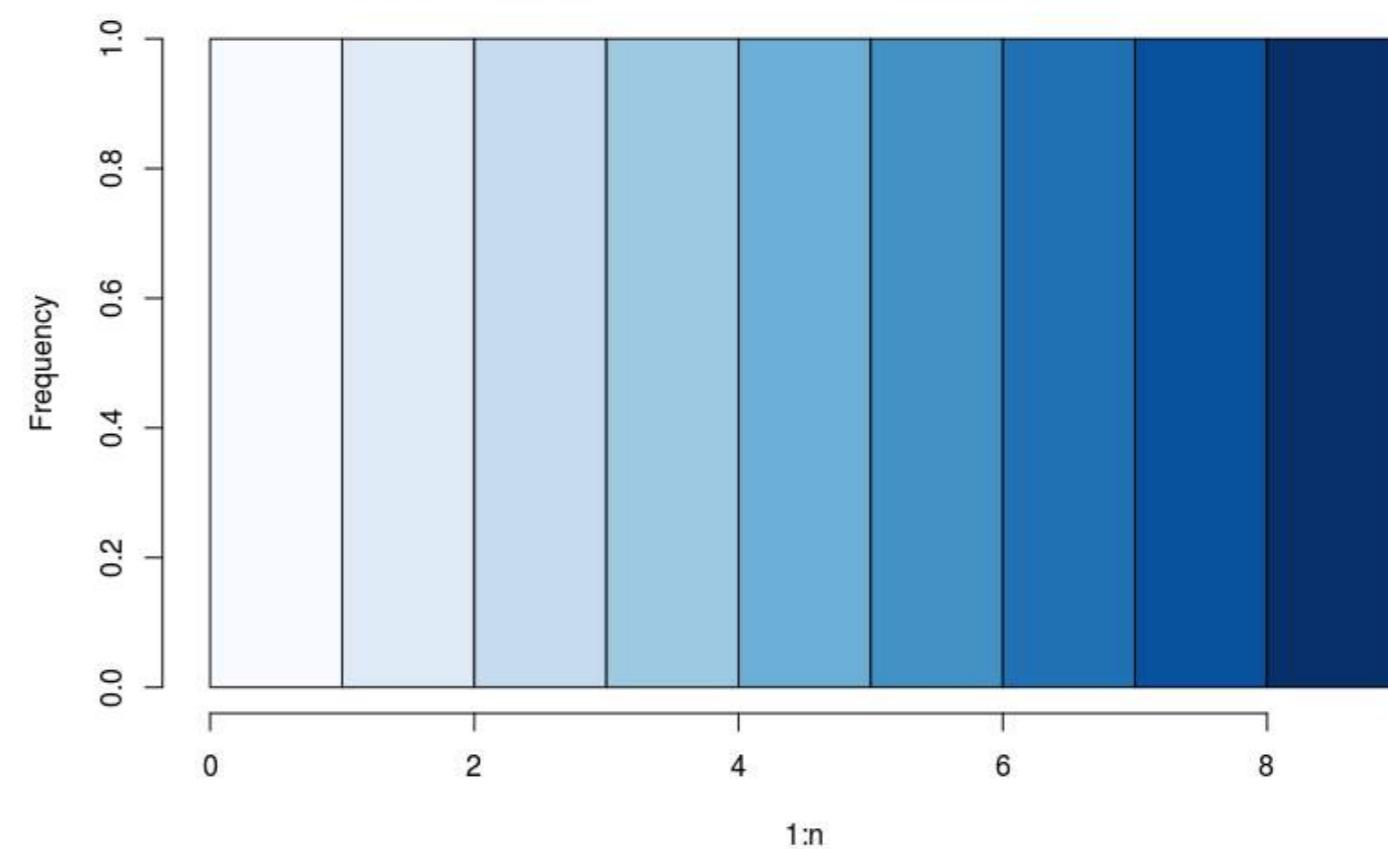
### An example of use

```
color_glimpse(blues9)
```

**Histogram of 1:n**



**Histogram of 1:n**

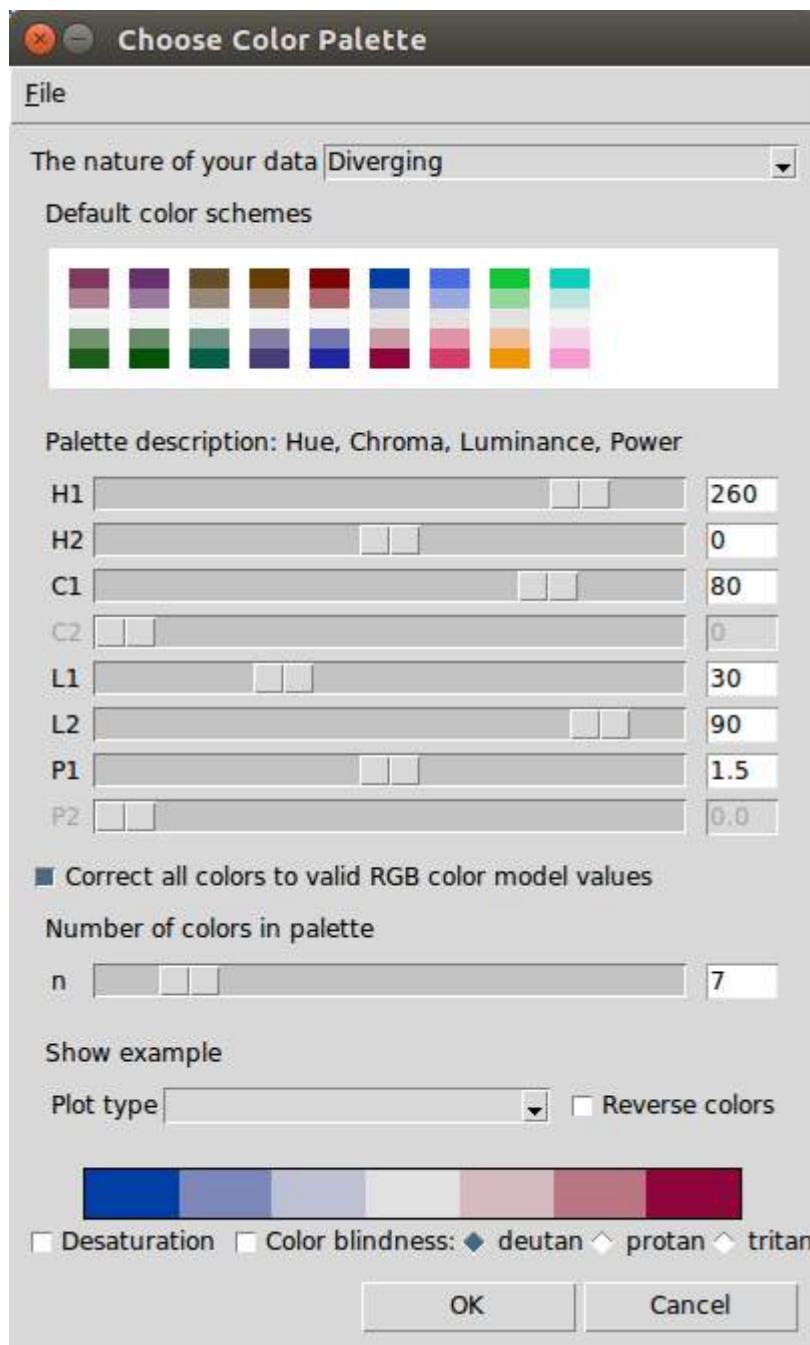


### 第106.3节：colorspace - 用于颜色的点击&拖动界面

包colorspace提供了用于选择调色板的图形用户界面。在调用choose\_palette()函数时，会弹出以下窗口：

### Section 106.3: colorspace - click&drag interface for colors

The package `colorspace` provides GUI for selecting a palette. On the call of `choose_palette()` function the following window pops-up:



选择调色板后，只需点击OK，别忘了将输出存储到变量中，例如pal。

```
pal <- choose_palette()
```

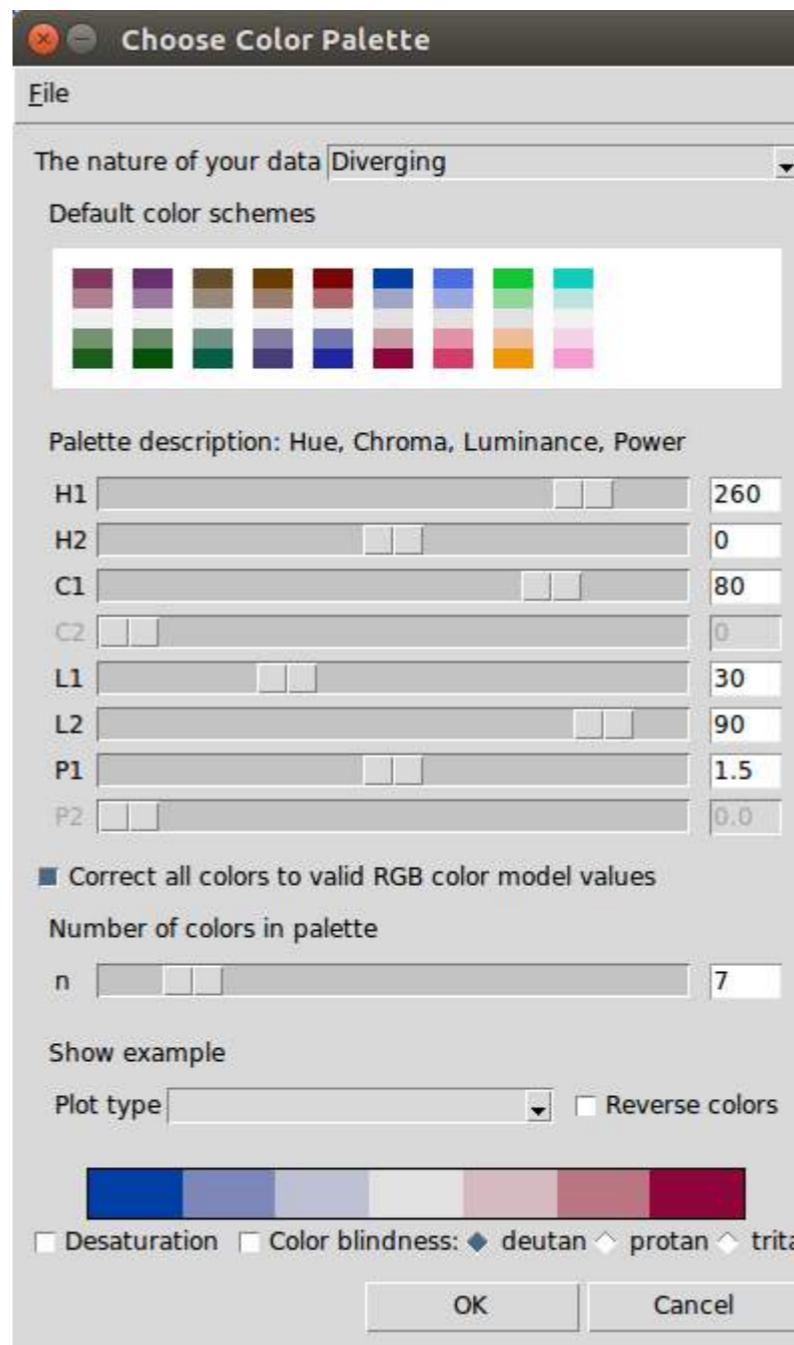
输出是一个函数，输入为 n (数字)，根据所选调色板生成长度为 n 的颜色向量。

```
pal(10)
[1] "#023FA5" "#6371AF" "#959CC3" "#BEC1D4" "#DBDCE0" "#E0DBDC" "#D6BCC0" "#C6909A" "#AE5A6D"
 "#8E063B"
```

## 第106.4节：色盲友好调色板

尽管色盲者能够识别广泛的颜色，但可能很难区分某些颜色。

RColorBrewer 提供对色盲友好的调色板：



When the palette is chosen, just hit OK and do not forget to store the output in a variable, e.g. pal.

```
pal <- choose_palette()
```

The output is a function that takes n (number) as input and produces a color vector of length n according to the selected palette.

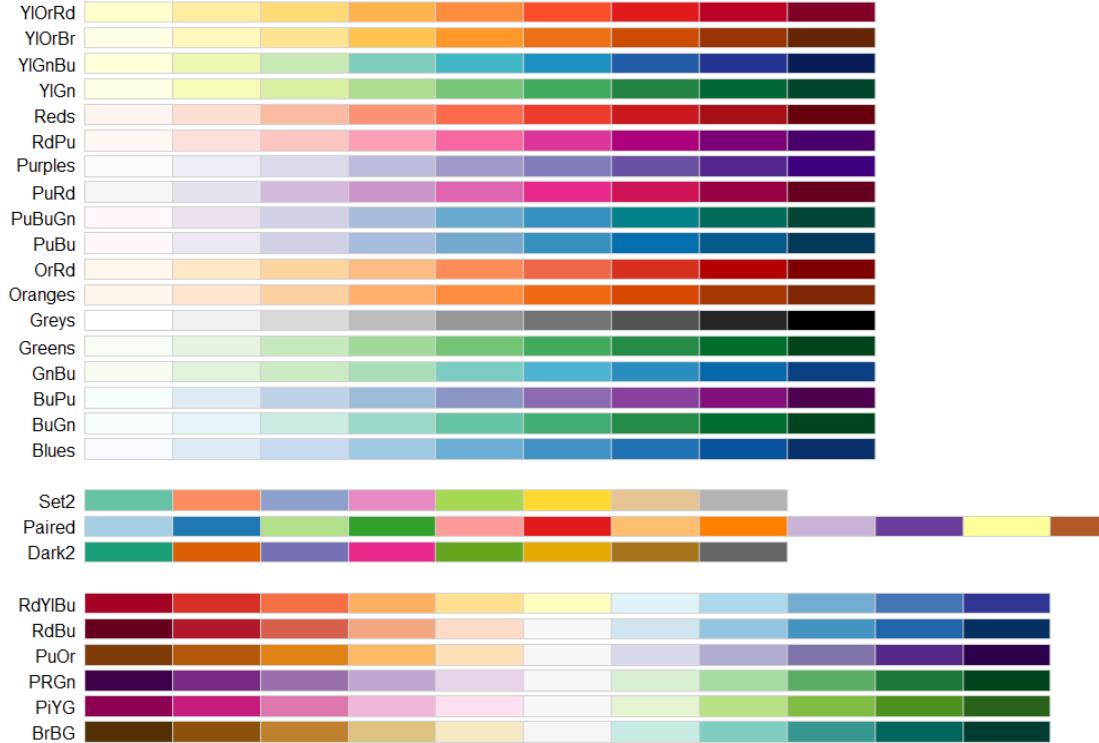
```
pal(10)
[1] "#023FA5" "#6371AF" "#959CC3" "#BEC1D4" "#DBDCE0" "#E0DBDC" "#D6BCC0" "#C6909A" "#AE5A6D"
 "#8E063B"
```

## Section 106.4: Colorblind-friendly palettes

Even though colorblind people can recognize a wide range of colors, it might be hard to differentiate between certain colors.

RColorBrewer provides colorblind-friendly palettes:

```
library(RColorBrewer)
display.brewer.all(colorblindFriendly = T)
```



东京大学的Color Universal Design提出了以下调色板：

```
#使用灰色的调色板
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
 "#CC79A7")

#使用黑色的调色板
cbbPalette <- c("#000000", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
 "#CC79A7")
```

## 第106.5节：RColorBrewer

ColorBrewer项目是一个非常流行的工具，用于选择和谐匹配的调色板。 RColorBrewer是该项目的R语言移植版本，也提供对色盲友好的调色板。

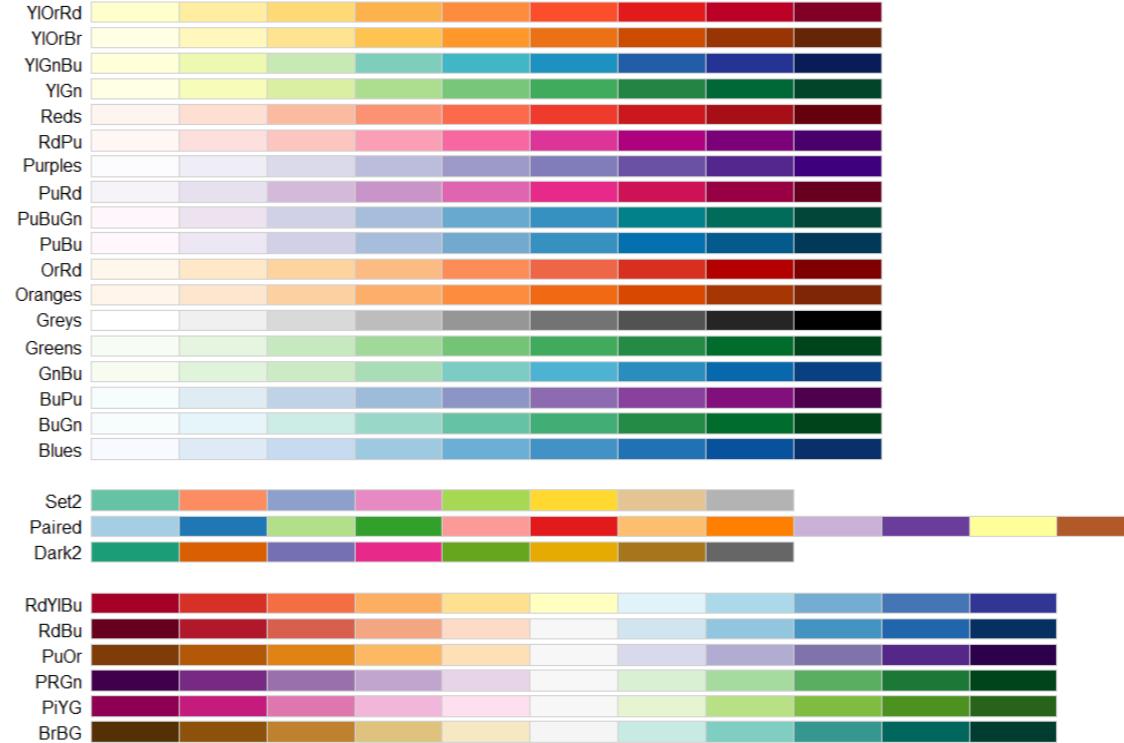
### 使用示例

```
colors_vec <- brewer.pal(5, name = 'BrBG')
print(colors_vec)
[1] "#A6611A" "#DFC27D" "#F5F5F5" "#80CDC1" "#018571"
```

RColorBrewer 为 ggplot2 创建配色选项：scale\_color\_brewer 和 scale\_fill\_brewer。

```
library(ggplot2)
```

```
library(RColorBrewer)
display.brewer.all(colorblindFriendly = T)
```



The [Color Universal Design](#) from the University of Tokyo proposes the following palettes:

```
#palette using grey
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
 "#CC79A7")

#palette using black
cbbPalette <- c("#000000", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00",
 "#CC79A7")
```

## Section 106.5: RColorBrewer

[ColorBrewer](#) project is a very popular tool to select harmoniously matching color palettes. RColorBrewer is a port of the project for R and provides also colorblind-friendly palettes.

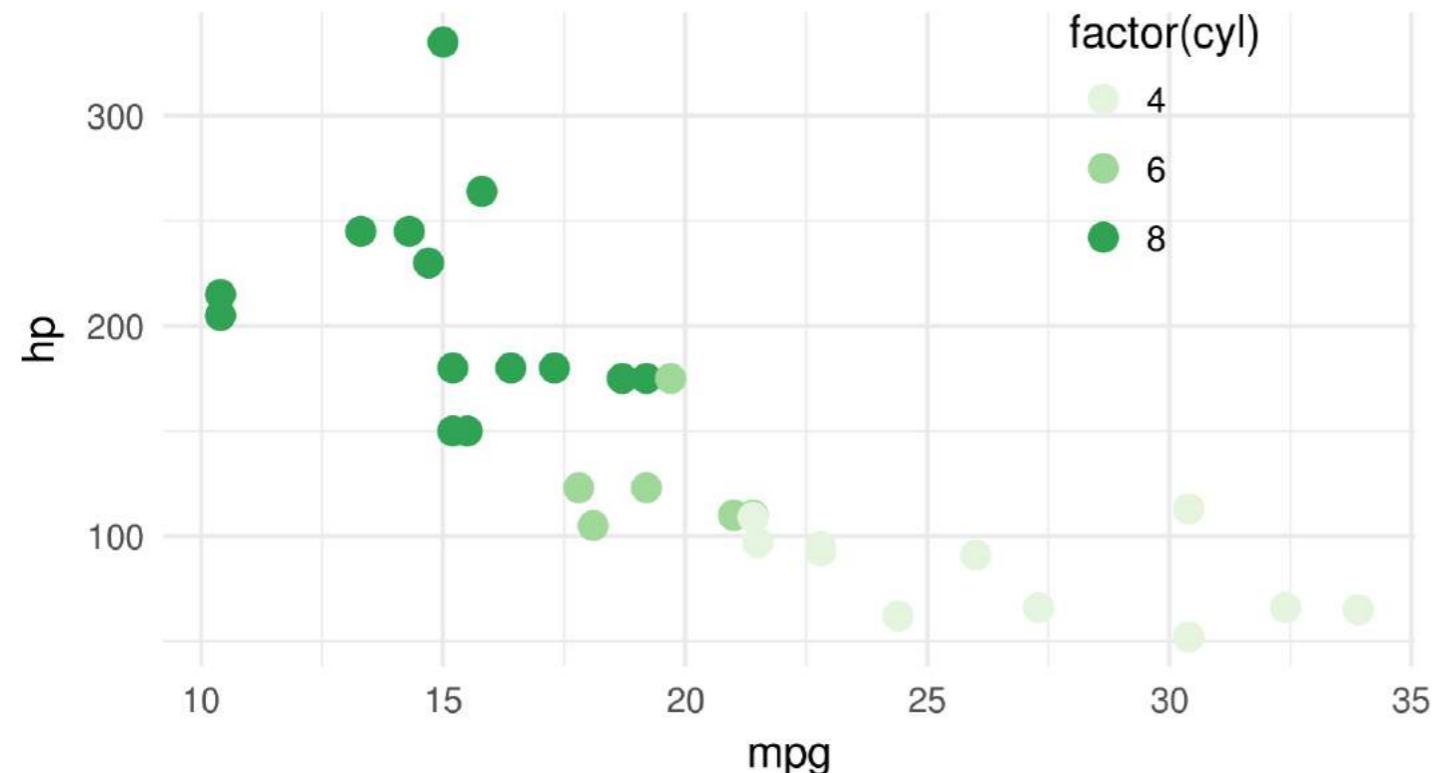
### An example of use

```
colors_vec <- brewer.pal(5, name = 'BrBG')
print(colors_vec)
[1] "#A6611A" "#DFC27D" "#F5F5F5" "#80CDC1" "#018571"
```

RColorBrewer creates coloring options for ggplot2: scale\_color\_brewer and scale\_fill\_brewer.

```
library(ggplot2)
```

```
ggplot(mtcars)+
 geom_point(aes(x = mpg, y = hp, color = factor(cyl)), size = 3)+
 scale_color_brewer(palette = 'Greens')+
 theme_minimal()
 theme(legend.position = c(.8, .8))
```



## 第106.6节：基本的R颜色函数

函数 `colors()` 列出所有R识别的颜色名称。有一个很好的PDF，可以实际看到 \_\_\_\_\_ 这些颜色。

`colorRampPalette` 创建一个函数，该函数插值一组给定颜色以创建新的调色板。该输出函数以 `n` (数字) 为输入，生成一个长度为 `n` 的颜色向量，插值初始颜色。

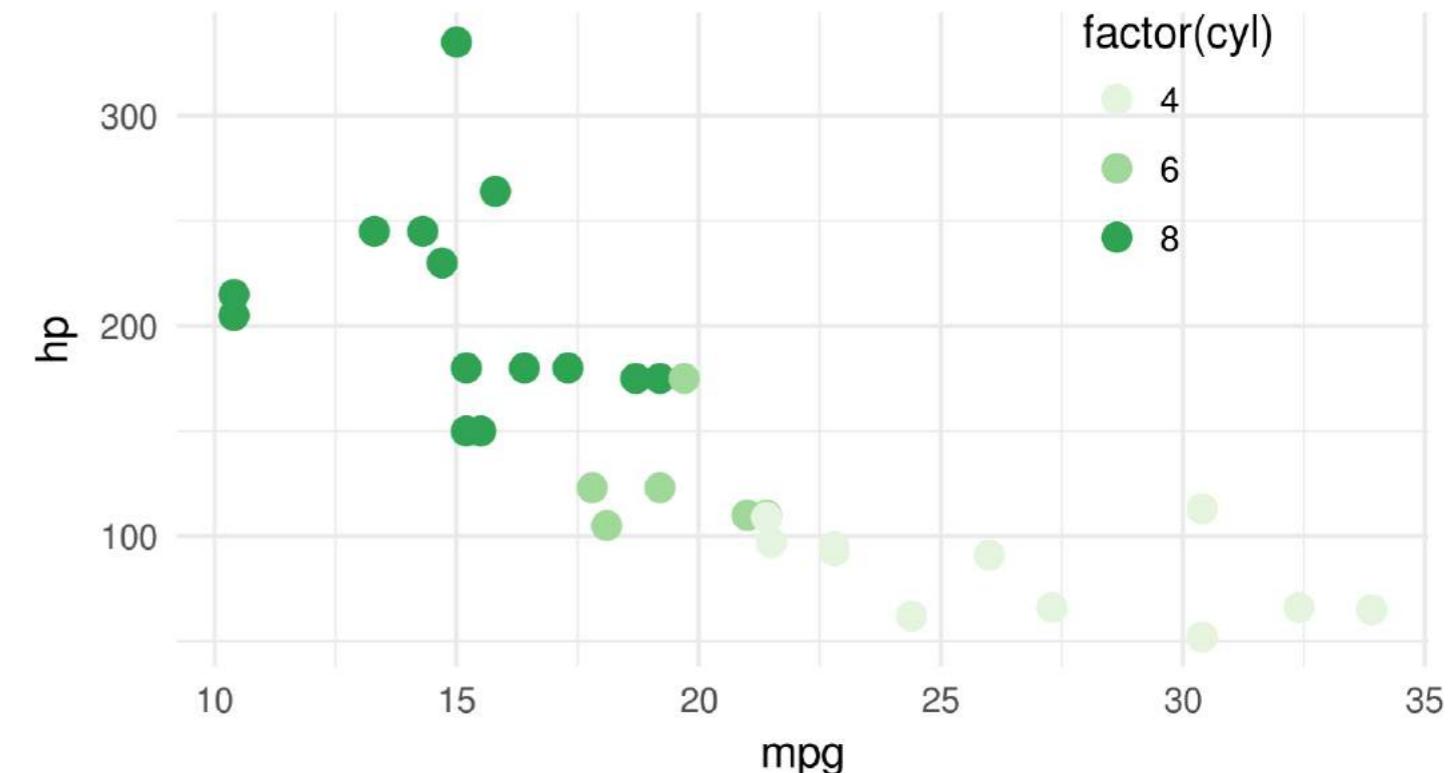
```
pal <- colorRampPalette(c('white', 'red'))
pal(5)
[1] "#FFFFFF" "#FFBFBF" "#FF7F7F" "#FF3F3F" "#FF0000"
```

任何特定颜色都可以通过 `rgb()` 函数生成：

```
rgb(0,1,0)
```

生成绿色颜色。

```
ggplot(mtcars)+
 geom_point(aes(x = mpg, y = hp, color = factor(cyl)), size = 3)+
 scale_color_brewer(palette = 'Greens')+
 theme_minimal()
 theme(legend.position = c(.8, .8))
```



## Section 106.6: basic R color functions

Function `colors()` lists all the color names that are recognized by R. There is [a nice PDF](#) where one can actually see those colors.

`colorRampPalette` creates a function that interpolate a set of given colors to create new color palettes. This output function takes `n` (number) as input and produces a color vector of length `n` interpolating the initial colors.

```
pal <- colorRampPalette(c('white', 'red'))
pal(5)
[1] "#FFFFFF" "#FFBFBF" "#FF7F7F" "#FF3F3F" "#FF0000"
```

Any specific color may be produced with an `rgb()` function:

```
rgb(0,1,0)
```

produces green color.

# 第107章：使用

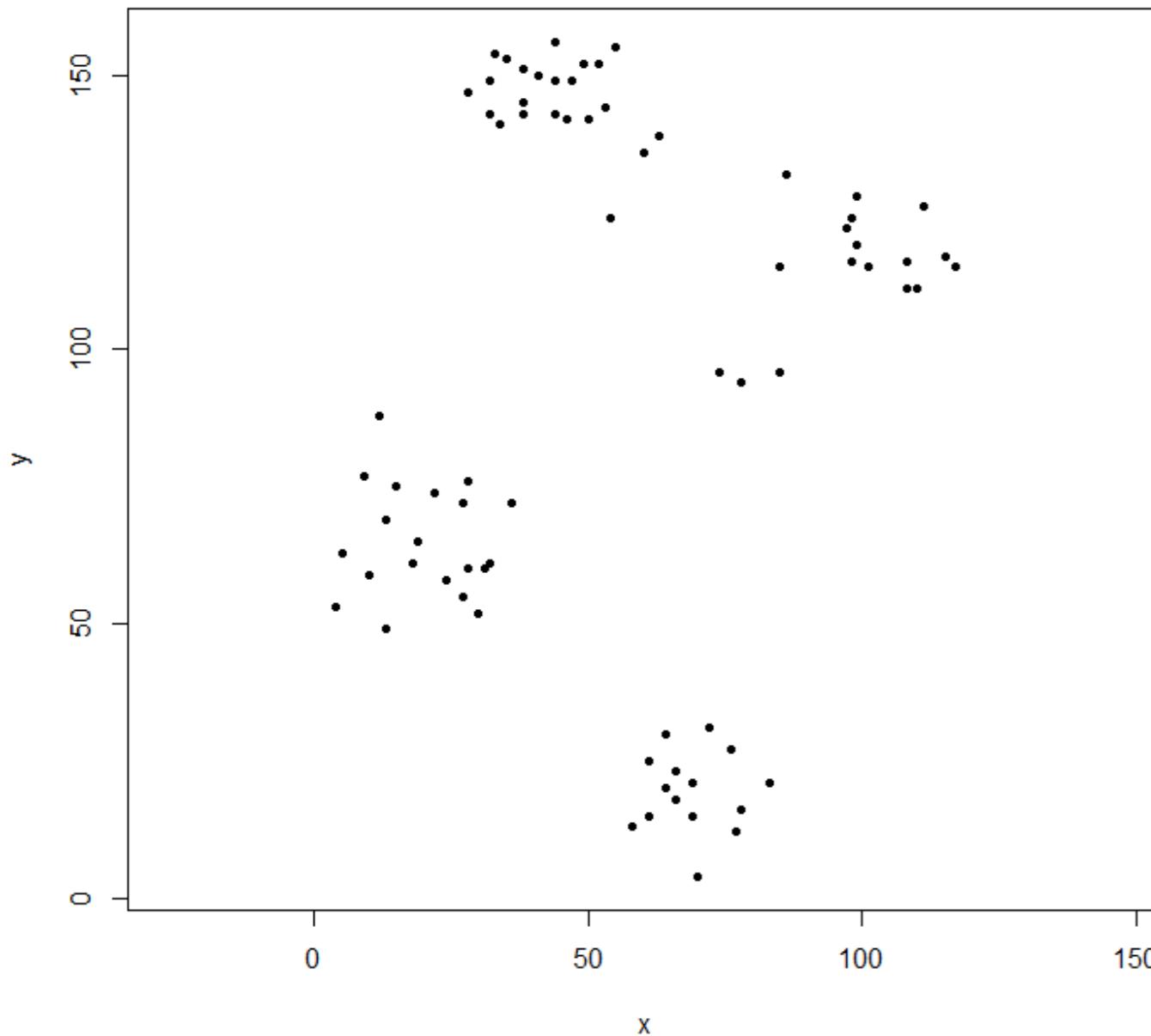
[hclust进行层次聚类](#)

stats包提供了 `hclust`函数来执行层次聚类。

## 第107.1节：示例1 - `hclust`的基本用法，树状图的显示，聚类的绘制

cluster库包含ruspini数据——一个用于说明聚类分析的标准数据集。

```
library(cluster) ## 获取ruspini数据
plot(ruspini, asp=1, pch=20) ## 查看数据
```



`hclust`期望输入的是距离矩阵，而非原始数据。我们使用默认参数计算树状图，且

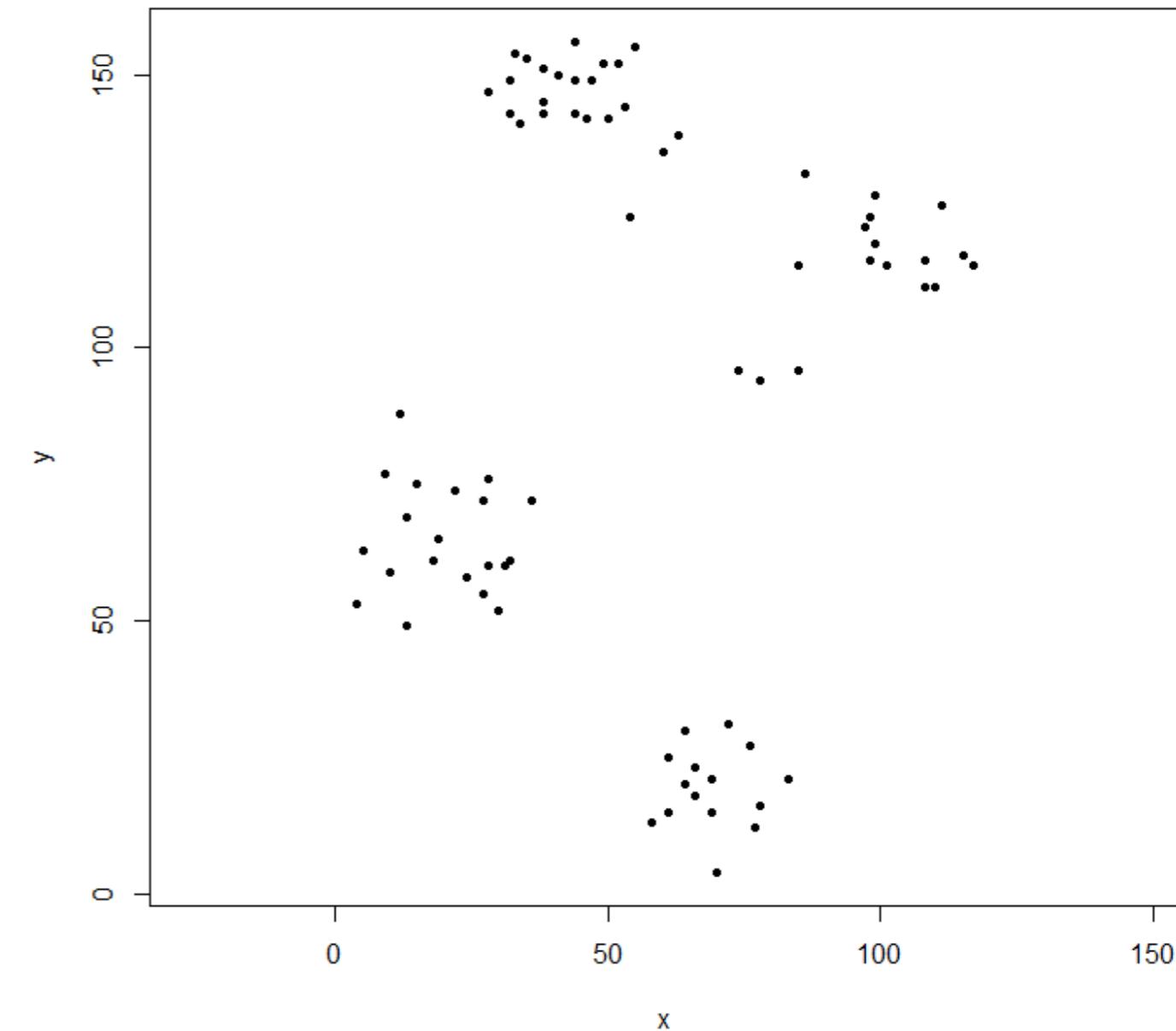
# Chapter 107: Hierarchical clustering with `hclust`

The stats package provides the `hclust` function to perform hierarchical clustering.

## Section 107.1: Example 1 - Basic use of `hclust`, display of dendrogram, plot clusters

The cluster library contains the ruspini data - a standard set of data for illustrating cluster analysis.

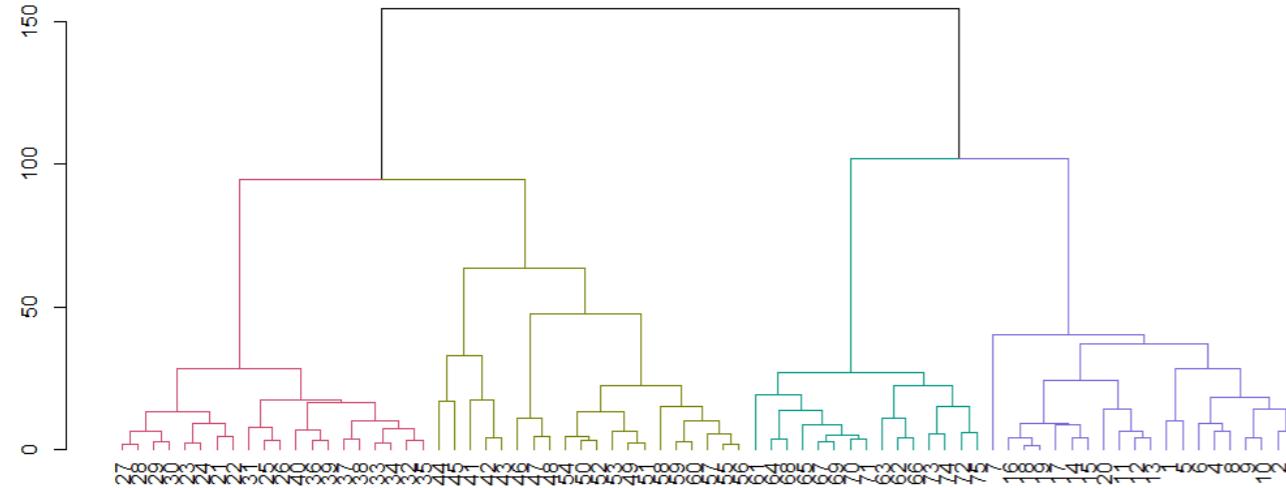
```
library(cluster) ## to get the ruspini data
plot(ruspini, asp=1, pch=20) ## take a look at the data
```



`hclust` expects a distance matrix, not the original data. We compute the tree using the default parameters and

显示它。hang 参数将树的所有叶子沿基线排列。

```
ruspini_hc_defaults <- hclust(dist(ruspini))
dend <- as.dendrogram(ruspini_hc_defaults)
if(!require(dendextend)) install.packages("dendextend"); library(dendextend)
dend <- color_branches(dend, k = 4)
plot(dend)
```

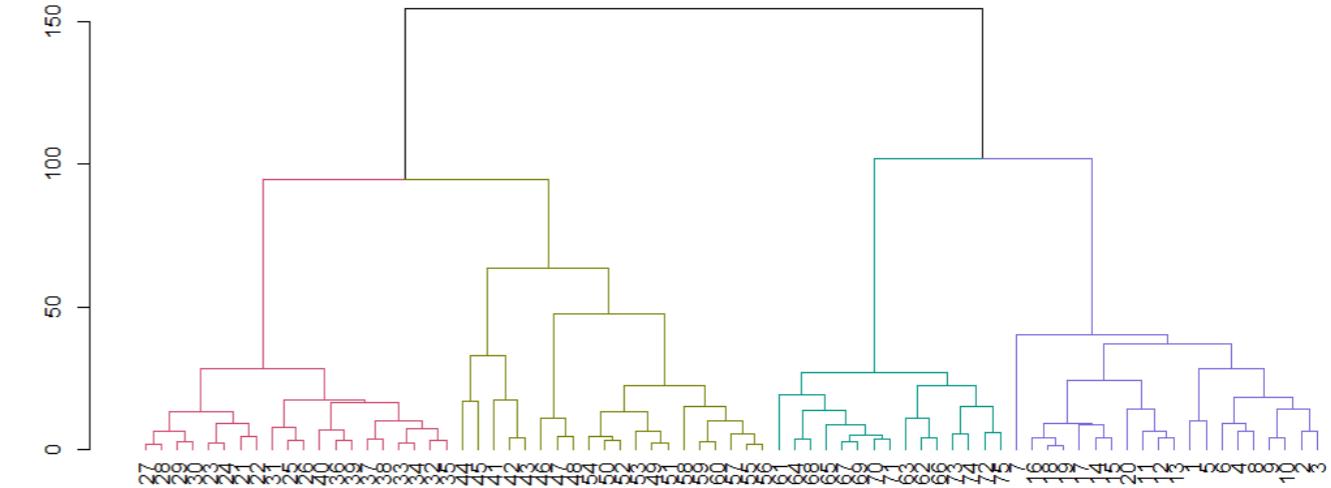


将树切割成四个簇，并重新绘制数据，按簇对点进行着色。k 是所需的簇数。

```
rhc_def_4 = cutree(ruspini_hc_defaults,k=4)
plot(ruspini, pch=20, asp=1, col=rhc_def_4)
```

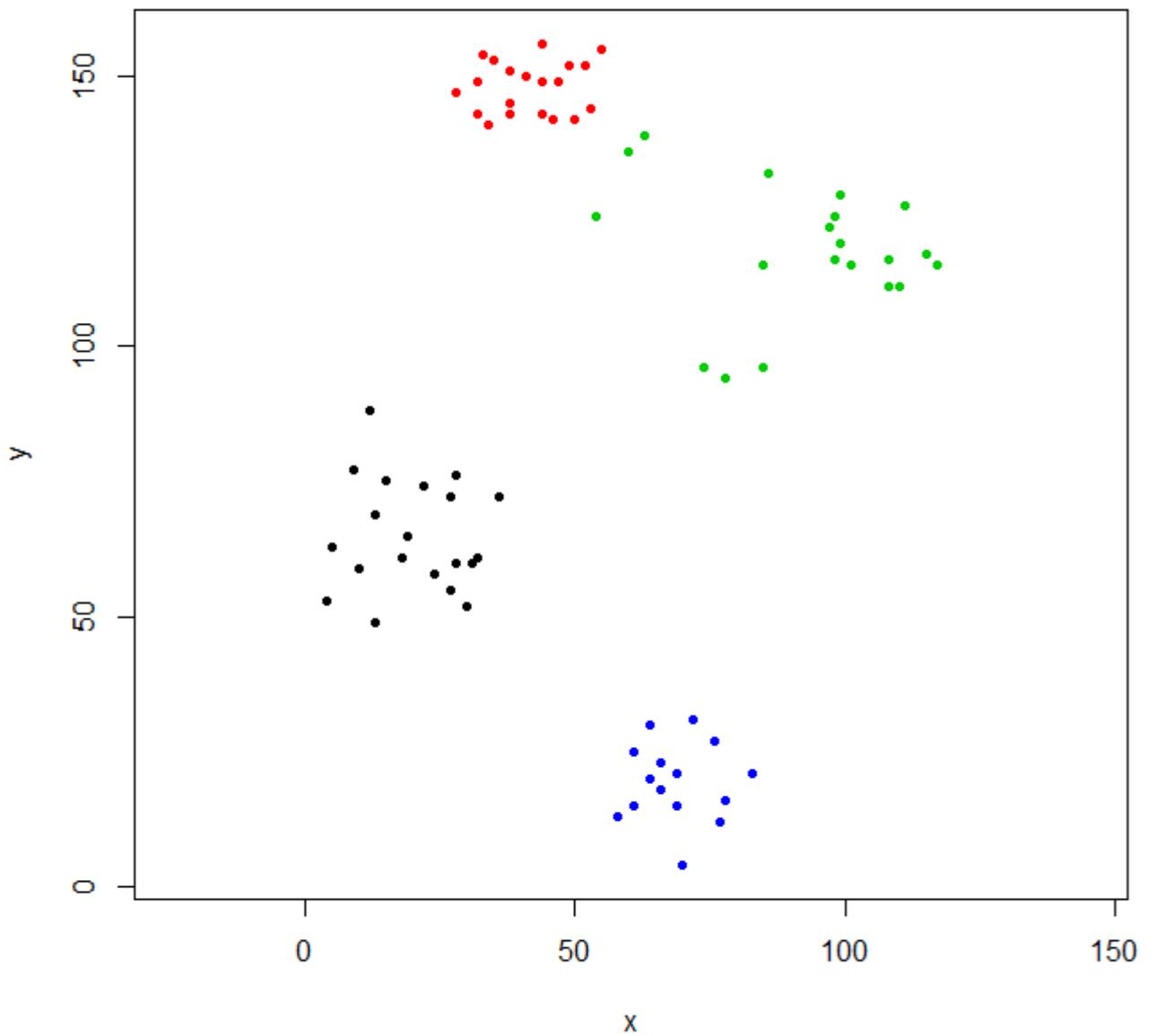
display it. The hang parameter lines up all of the leaves of the tree along the baseline.

```
ruspini_hc_defaults <- hclust(dist(ruspini))
dend <- as.dendrogram(ruspini_hc_defaults)
if(!require(dendextend)) install.packages("dendextend"); library(dendextend)
dend <- color_branches(dend, k = 4)
plot(dend)
```



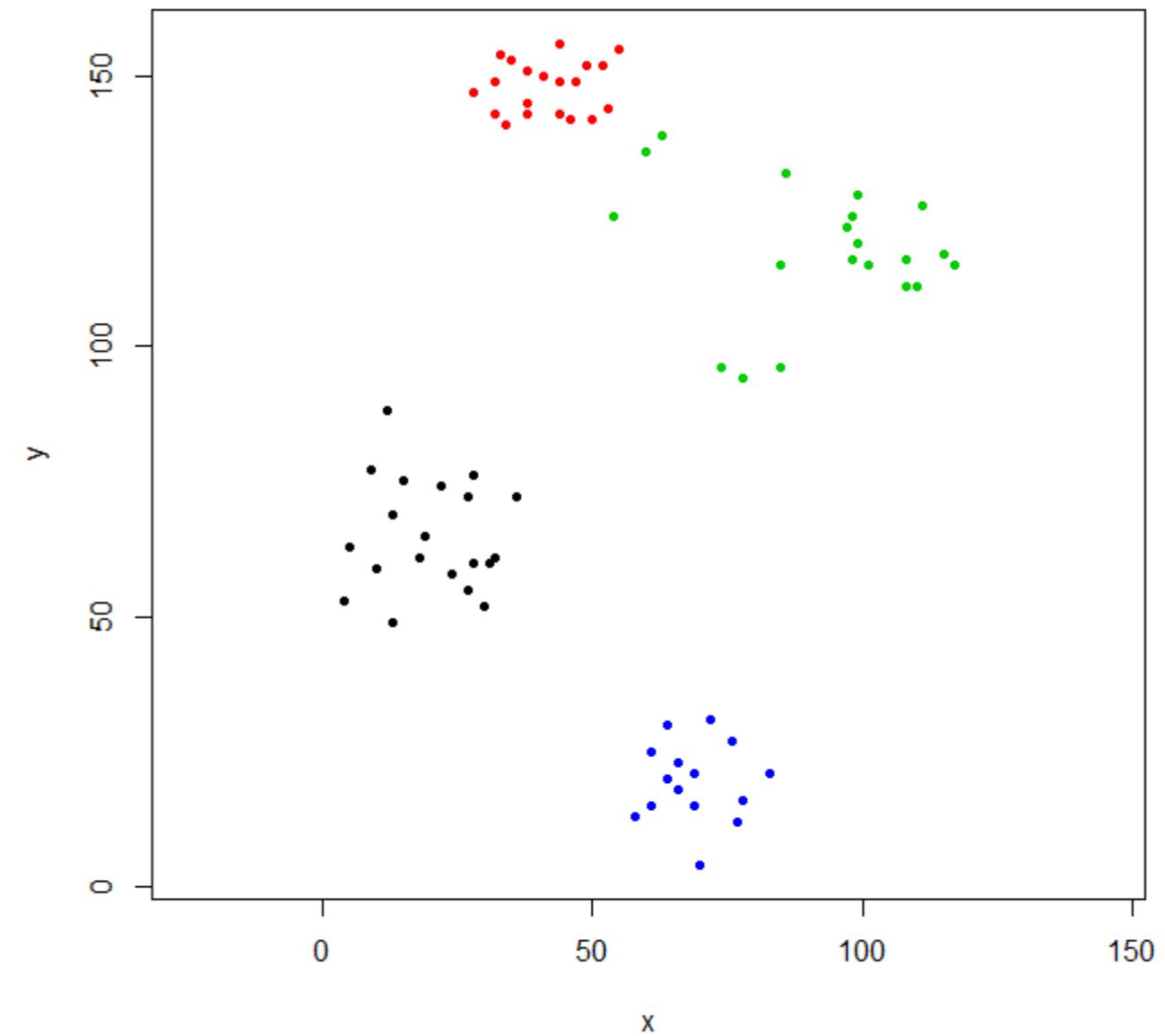
Cut the tree to give four clusters and replot the data coloring the points by cluster. k is the desired number of clusters.

```
rhc_def_4 = cutree(ruspini_hc_defaults,k=4)
plot(ruspini, pch=20, asp=1, col=rhc_def_4)
```



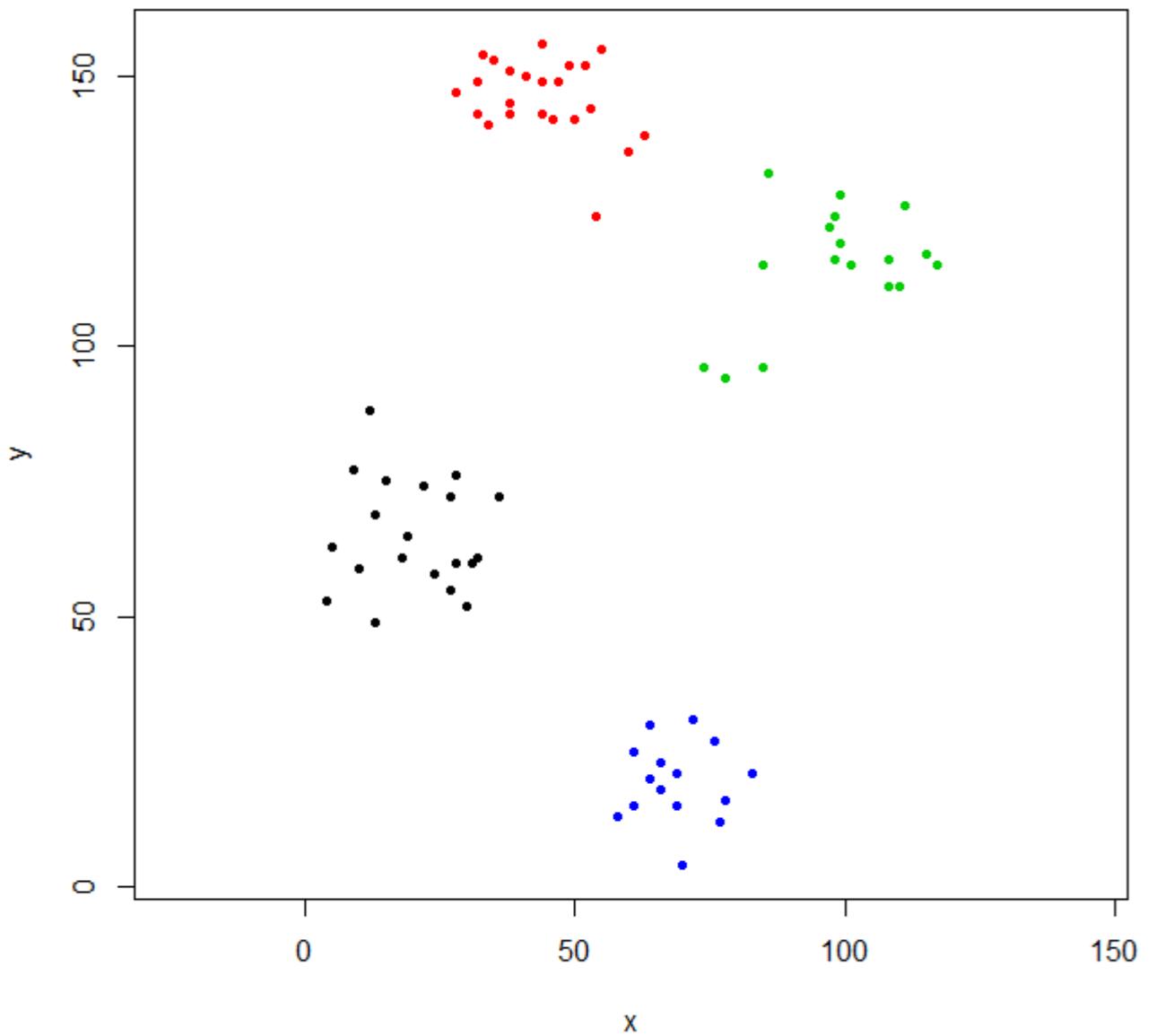
这种聚类有点奇怪。我们可以先对数据进行缩放，从而获得更好的聚类效果。

```
scaled_ruspini_hc_defaults = hclust(dist(scale(ruspini)))
srhc_def_4 = cutree(scaled_ruspini_hc_defaults, 4)
plot(ruspini, pch=20, asp=1, col=srhc_def_4)
```



This clustering is a little odd. We can get a better clustering by scaling the data first.

```
scaled_ruspini_hc_defaults = hclust(dist(scale(ruspini)))
srhc_def_4 = cutree(scaled_ruspini_hc_defaults, 4)
plot(ruspini, pch=20, asp=1, col=srhc_def_4)
```



比较簇的默认不相似度量是“complete”。你可以通过 method 参数指定不同的度量方法。

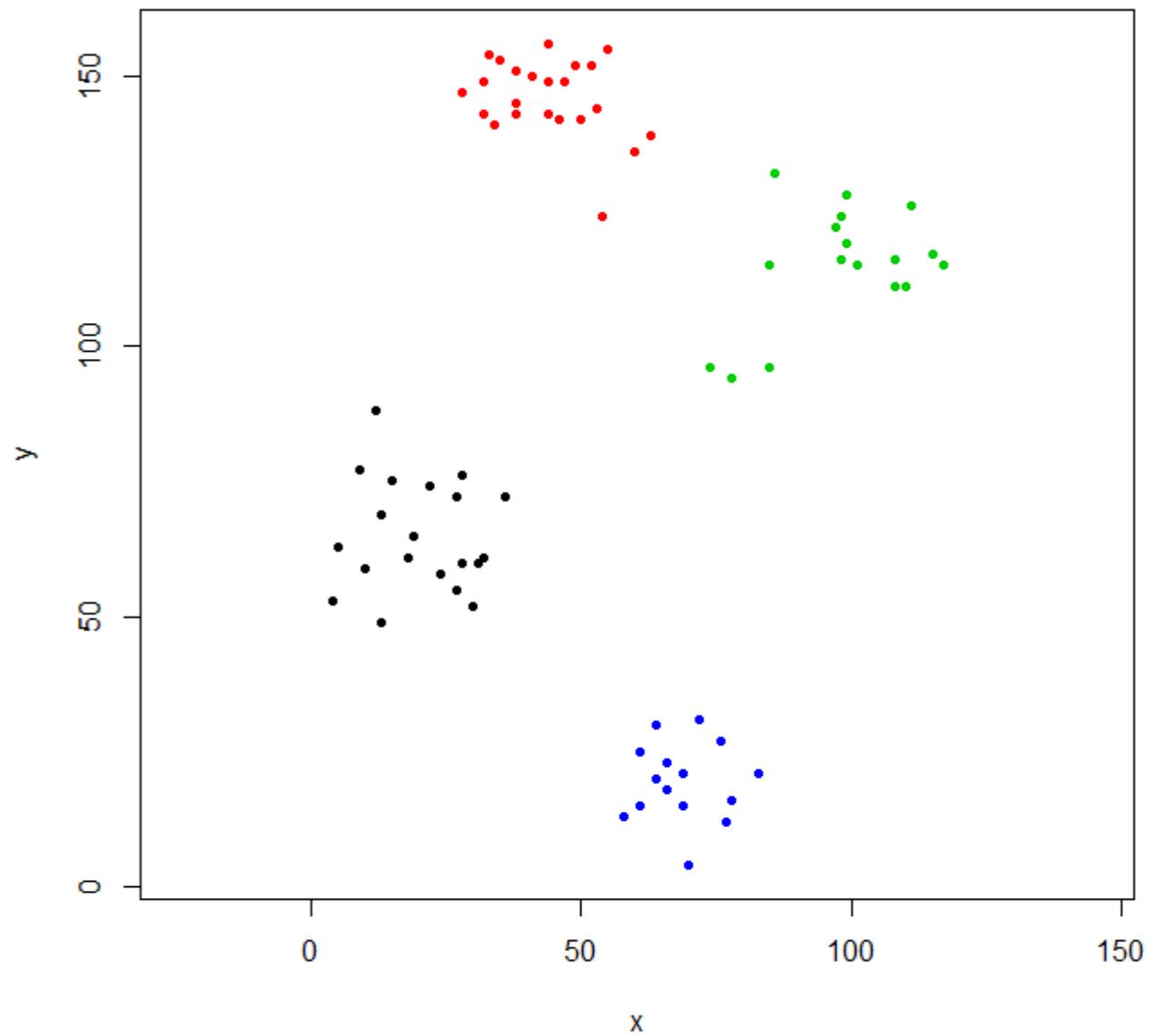
```
ruspini_hc_single = hclust(dist(ruspini), method="single")
```

## 第107.2节：示例2 - 层次聚类和异常值

在层次聚类中，异常值通常表现为单点簇。

生成三个高斯分布以说明异常值的影响。

```
set.seed(656)
x = c(rnorm(150, 0, 1), rnorm(150, 9, 1), rnorm(150, 4.5, 1))
y = c(rnorm(150, 0, 1), rnorm(150, 0, 1), rnorm(150, 5, 1))
XYdf = data.frame(x,y)
plot(XYdf, pch=20)
```



The default dissimilarity measure for comparing clusters is "complete". You can specify a different measure with the method parameter.

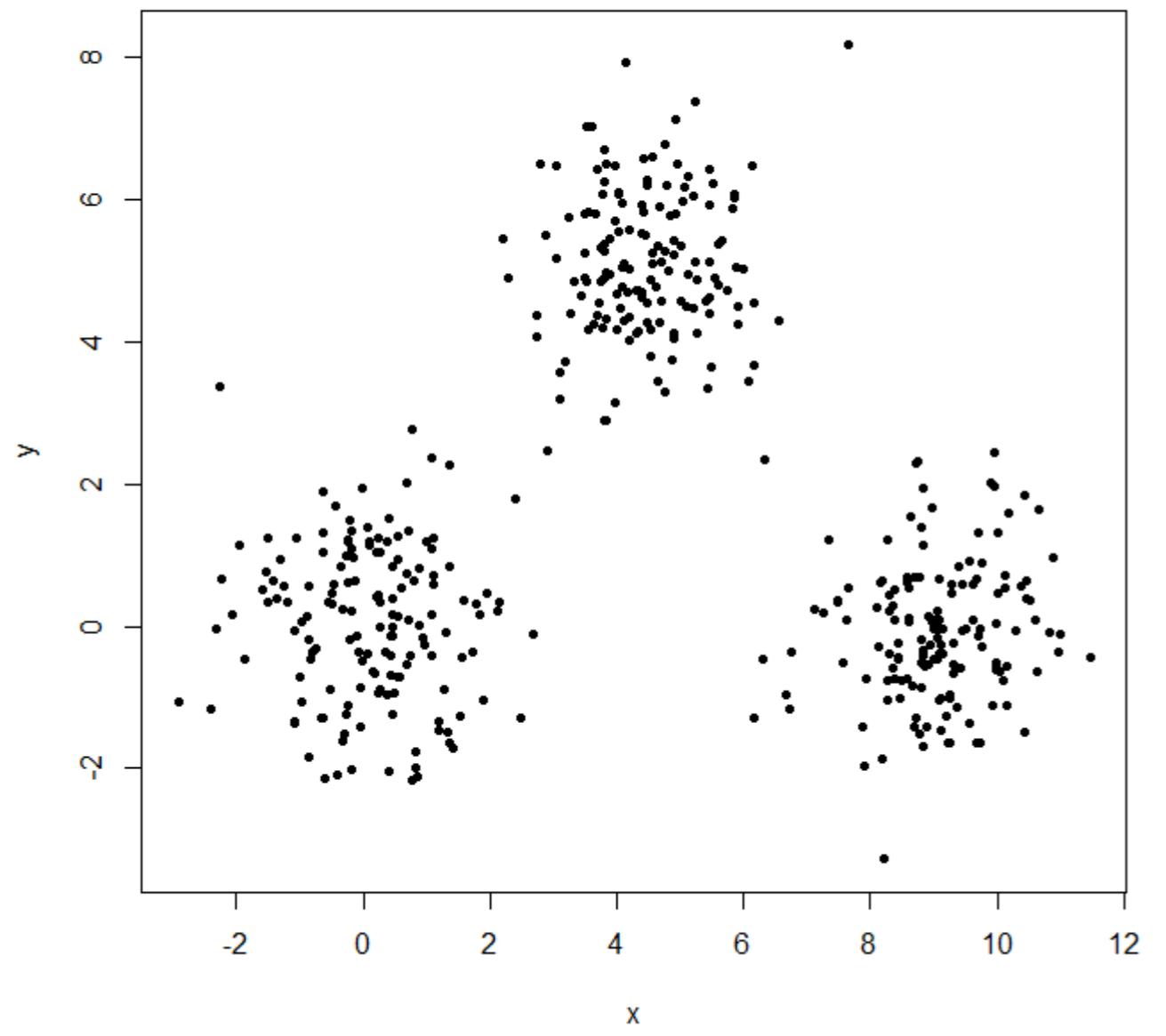
```
ruspini_hc_single = hclust(dist(ruspini), method="single")
```

## Section 107.2: Example 2 - hclust and outliers

With hierarchical clustering, outliers often show up as one-point clusters.

Generate three Gaussian distributions to illustrate the effect of outliers.

```
set.seed(656)
x = c(rnorm(150, 0, 1), rnorm(150, 9, 1), rnorm(150, 4.5, 1))
y = c(rnorm(150, 0, 1), rnorm(150, 0, 1), rnorm(150, 5, 1))
XYdf = data.frame(x,y)
plot(XYdf, pch=20)
```

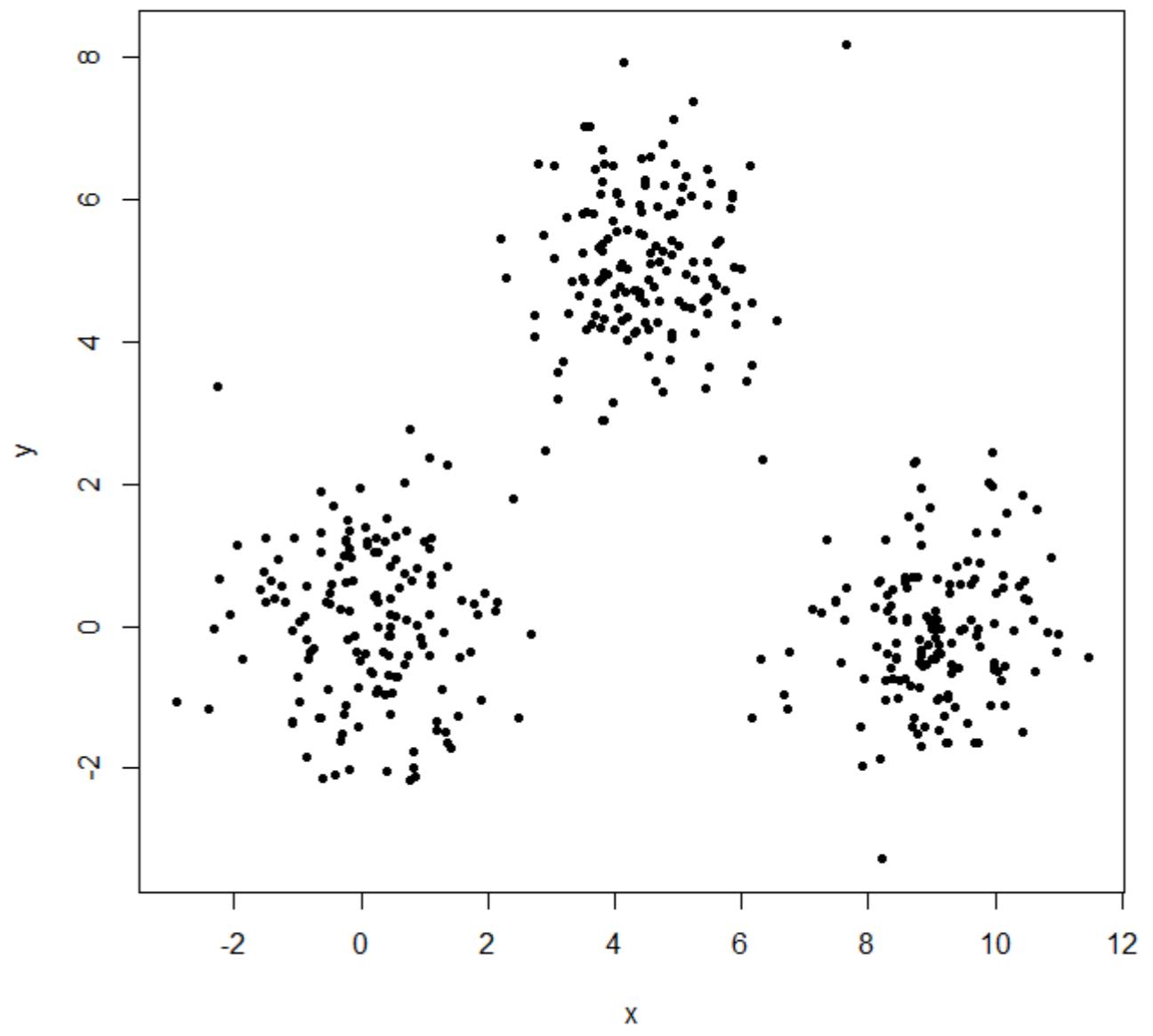


构建聚类结构，将其分成三个簇。

```
XY_sing = hclust(dist(XYdf), method="single")
XYs3 = cutree(XY_sing, k=3)
table(XYs3)
XYs3
 1 2 3
448 1 1
```

`hclust` 发现了两个异常值，并将其他所有数据归为一个大簇。要获得“真实”的簇，您可能需要将 `k` 设置得更高。

```
XYs6 = cutree(XY_sing, k=6)
table(XYs6)
XYs6
 1 2 3 4 5 6
148 150 1 149 1 1
plot(XYdf, pch=20, col=XYs6)
```

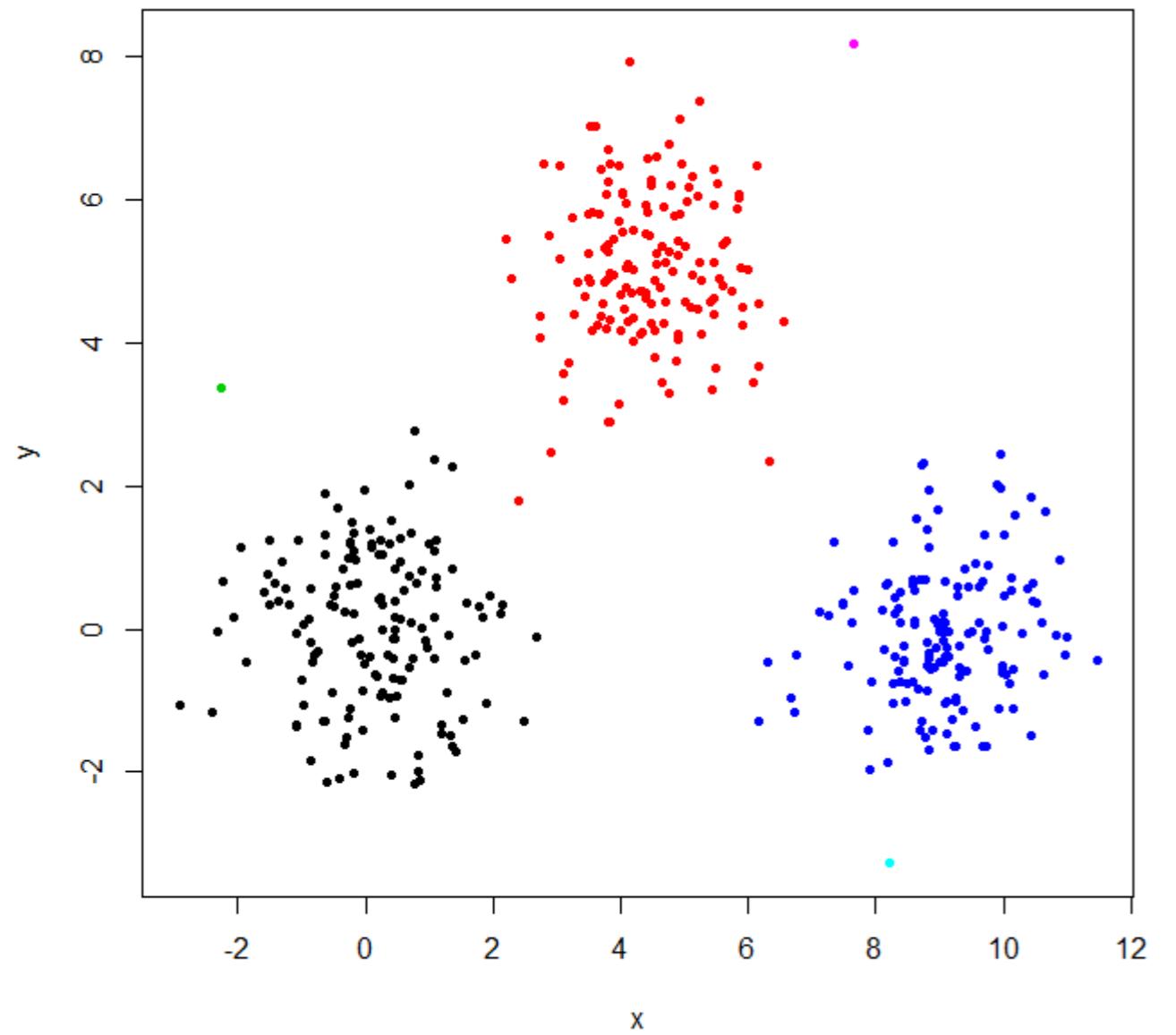


Build the cluster structure, split it into three cluster.

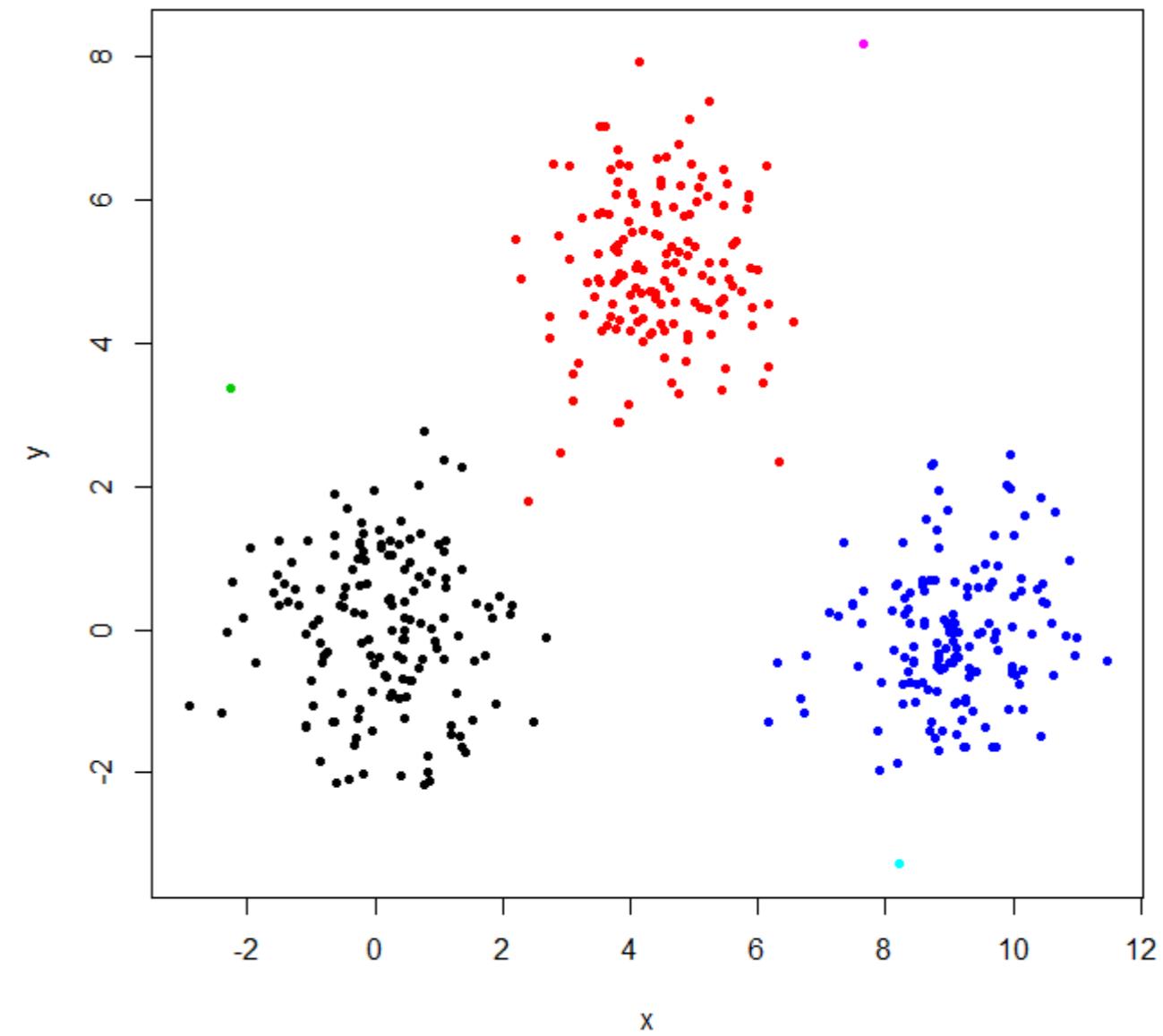
```
XY_sing = hclust(dist(XYdf), method="single")
XYs3 = cutree(XY_sing, k=3)
table(XYs3)
XYs3
 1 2 3
448 1 1
```

`hclust` found two outliers and put everything else into one big cluster. To get the "real" clusters, you may need to set `k` higher.

```
XYs6 = cutree(XY_sing, k=6)
table(XYs6)
XYs6
 1 2 3 4 5 6
148 150 1 149 1 1
plot(XYdf, pch=20, col=XYs6)
```



这个[StackOverflow](#) 帖子提供了一些关于如何选择簇数的指导，但要注意层次聚类中的这种行为。



This [StackOverflow post](#) has some guidance on how to pick the number of clusters, but be aware of this behavior in hierarchical clustering.

# 第108章：随机森林算法

随机森林是一种用于分类或回归的集成方法，可以减少数据过拟合的可能性。该方法的详细信息可以在维基百科关于随机森林的文章中找到。R语言的主要实现是在randomForest包中，但也有其他实现。请参见CRAN机器学习视图。

## 第108.1节：基本示例 - 分类和回归

```
用于分类和回归示例
library(randomForest)
library(car) ## 用于Soils数据
data(Soils)

#####
随机森林分类示例
set.seed(656) ## 为了结果可复现
S_RF_Class = randomForest(Gp ~ ., data=Soils[,c(4,6:14)])
Gp_RF = predict(S_RF_Class, Soils[,6:14])
length(which(Gp_RF != Soils$Gp)) ## 无错误

朴素贝叶斯作为对比
library(e1071)
S_NB = naiveBayes(Soils[,6:14], Soils[,4])
Gp_NB = predict(S_NB, Soils[,6:14], type="class")
length(which(Gp_NB != Soils$Gp)) ## 6个错误
```

这个例子在训练数据上进行了测试，但说明随机森林（RF）可以构建非常好的模型。

```
#####
随机森林回归示例
set.seed(656) ## 为了可重复性
S_RF_Reg = randomForest(pH ~ ., data=Soils[,6:14])
pH_RF = predict(S_RF_Reg, Soils[,6:14])

比较随机森林和线性模型的预测值与实际值
S_LM = lm(pH ~ ., data=Soils[,6:14])
pH_LM = predict(S_LM, Soils[,6:14])
par(mfrow=c(1,2))
plot(Soils$pH, pH_RF, pch=20, ylab="预测值", main="随机森林")
abline(0,1)
plot(Soils$pH, pH_LM, pch=20, ylab="预测值", main="线性模型")
abline(0,1)
```

# Chapter 108: Random Forest Algorithm

RandomForest is an ensemble method for classification or regression that reduces the chance of overfitting the data. Details of the method can be found in the [Wikipedia article on Random Forests](#). The main implementation for R is in the randomForest package, but there are other implementations. See the [CRAN view on Machine Learning](#).

## Section 108.1: Basic examples - Classification and Regression

```
Used for both Classification and Regression examples
library(randomForest)
library(car) ## For the Soils data
data(Soils)

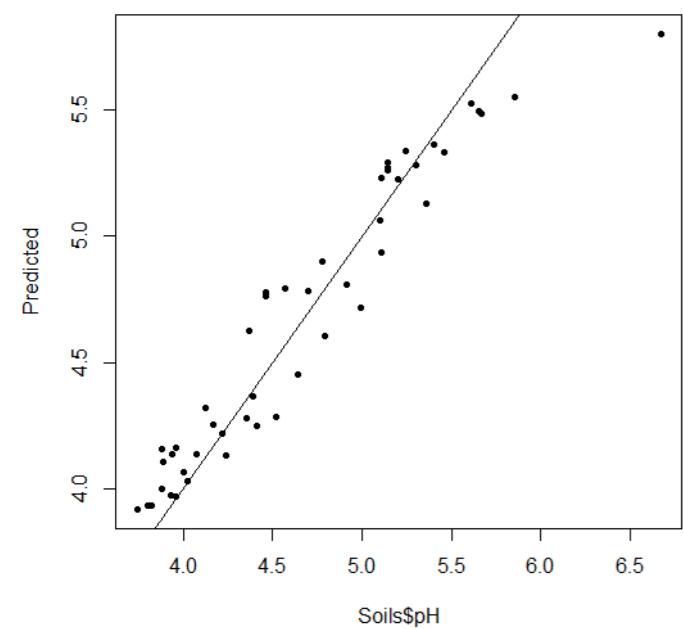
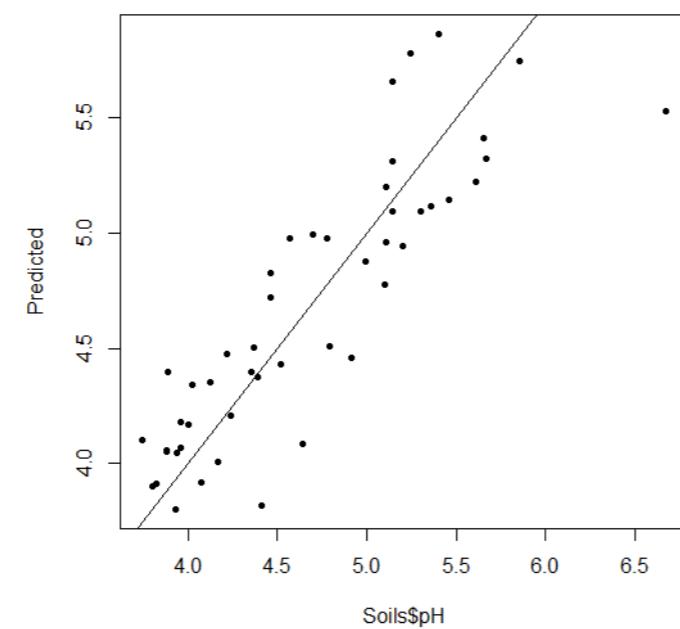
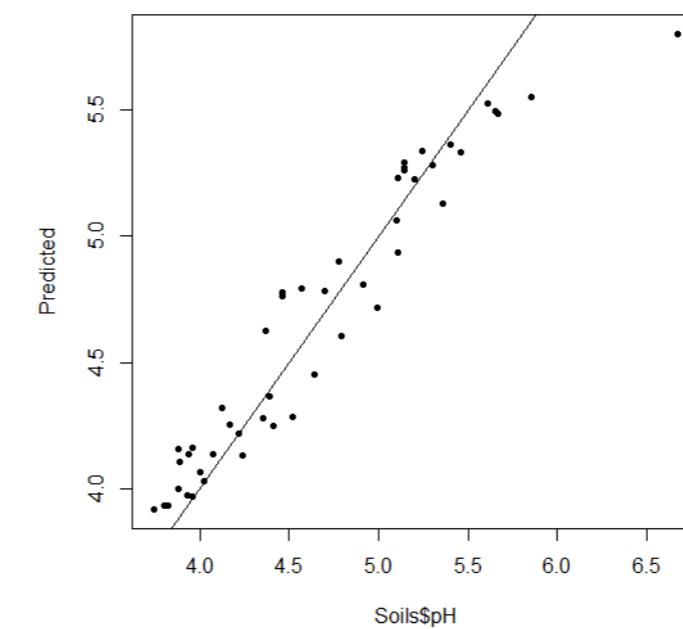
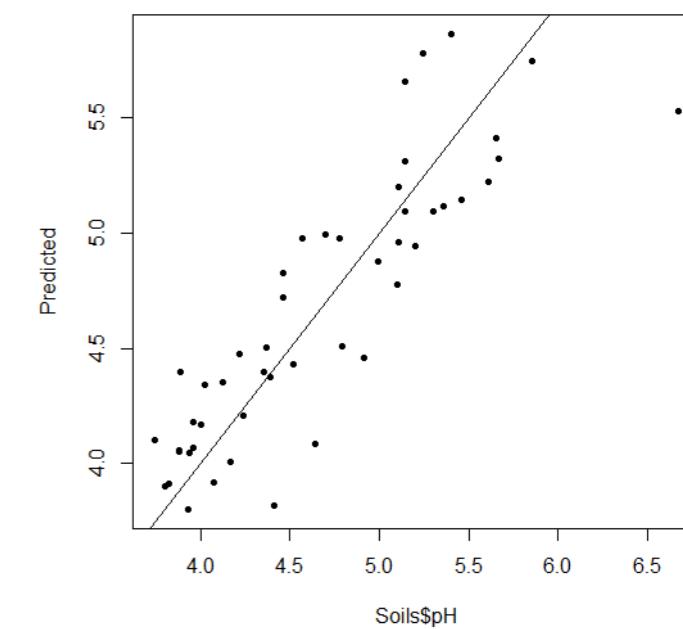
#####
RF Classification Example
set.seed(656) ## for reproducibility
S_RF_Class = randomForest(Gp ~ ., data=Soils[,c(4,6:14)])
Gp_RF = predict(S_RF_Class, Soils[,6:14])
length(which(Gp_RF != Soils$Gp)) ## No Errors

Naive Bayes for comparison
library(e1071)
S_NB = naiveBayes(Soils[,6:14], Soils[,4])
Gp_NB = predict(S_NB, Soils[,6:14], type="class")
length(which(Gp_NB != Soils$Gp)) ## 6 Errors
```

This example tested on the training data, but illustrates that RF can make very good models.

```
#####
RF Regression Example
set.seed(656) ## for reproducibility
S_RF_Reg = randomForest(pH ~ ., data=Soils[,6:14])
pH_RF = predict(S_RF_Reg, Soils[,6:14])

Compare Predictions with Actual values for RF and Linear Model
S_LM = lm(pH ~ ., data=Soils[,6:14])
pH_LM = predict(S_LM, Soils[,6:14])
par(mfrow=c(1,2))
plot(Soils$pH, pH_RF, pch=20, ylab="Predicted", main="Random Forest")
abline(0,1)
plot(Soils$pH, pH_LM, pch=20, ylab="Predicted", main="Linear Model")
abline(0,1)
```

**Random Forest****Linear Model****Random Forest****Linear Model**

# 第109章：RESTful R服务

[OpenCPU](#) 使用标准的 R 包来开发、发布和部署网页应用程序。

## 第109.1节：openCPU 应用程序

官方网站包含很好的应用示例：<https://www.opencpu.org/apps.html>

以下代码用于启动一个 R 会话：

```
library(openCPU)
openCPU$start(port = 5936)
```

执行此代码后，您可以使用 URL 访问 R 会话的函数。结果可以是 XML、html、JSON 或其他定义的格式。

例如，可以通过 cURL 调用访问上述 R 会话：

```
#curl 使用 HTTP POST 方法, 即 -X POST 或 -d "arg=value"
curl http://localhost:5936/ocpu/library/MASS/scripts/ch01.R -X POST
curl http://localhost:5936/ocpu/library/stats/R/rnorm -d "n=10&mean=5"
```

该调用是异步的，意味着在等待调用完成时 R 会话不会被阻塞（与 shiny 不同）。

调用结果保存在临时会话中，存储路径为 /ocpu/tmp/

以下是如何检索临时会话的示例：

```
curl https://public.openCPU.org/ocpu/library/stats/R/rnorm -d n=5
/ocpu/tmp/x009f9e7630/R/.val
/ocpu/tmp/x009f9e7630/stdout
/ocpu/tmp/x009f9e7630/source
/ocpu/tmp/x009f9e7630/console
/ocpu/tmp/x009f9e7630/info
```

x009f9e7630 是会话的名称。

指向 /ocpu/tmp/x009f9e7630/R/.val 将返回 `rnorm(5)` 的结果值，  
/ocpu/tmp/x009f9e7630/R/console 将返回 `rnorm(5)` 的控制台内容，等等。

# Chapter 109: RESTful R Services

[OpenCPU](#) uses standard R packaging to develop, ship and deploy web applications.

## Section 109.1: openCPU Apps

The official website contain good exemple of apps: <https://www.openCPU.org/apps.html>

The following code is used to serve a R session:

```
library(openCPU)
openCPU$start(port = 5936)
```

After this code is executed, you can use URLs to access the functions of the R session. The result could be XML, html, JSON or some other defined formats.

For exemple, the previous R session can be accessed by a cURL call:

```
#curl uses http post method for -X POST or -d "arg=value"
curl http://localhost:5936/ocpu/library/MASS/scripts/ch01.R -X POST
curl http://localhost:5936/ocpu/library/stats/R/rnorm -d "n=10&mean=5"
```

The call is asynchronous, meaning that the R session is not blocked while waiting for the call to finish (contrary to shiny).

The call result is kept in a temporary session stored in /ocpu/tmp/

An exemple of how to retrieve the temporary session:

```
curl https://public.openCPU.org/ocpu/library/stats/R/rnorm -d n=5
/ocpu/tmp/x009f9e7630/R/.val
/ocpu/tmp/x009f9e7630/stdout
/ocpu/tmp/x009f9e7630/source
/ocpu/tmp/x009f9e7630/console
/ocpu/tmp/x009f9e7630/info
```

x009f9e7630 is the name of the session.

Pointing to /ocpu/tmp/x009f9e7630/R/.val will return the value resulting of `rnorm(5)`,  
/ocpu/tmp/x009f9e7630/R/console will return the content of the console of `rnorm(5)`, etc..

# 第110章：机器学习

## 第110.1节：创建随机森林模型

机器学习算法的一个例子是随机森林算法 (Breiman, L. (2001). Random Forests. *Machine Learning* 45(5), 第5-32页)。该算法在R中根据Breiman最初的Fortran实现，在 `randomForest` 包中实现。

可以通过将类别变量准备为 `factor` 来在R中创建随机森林分类器对象，这在 `iris` 数据集中已经体现。因此我们可以轻松地创建一个随机森林：

```
library(randomForest)

rf <- randomForest(x = iris[, 1:4],
 y = iris$Species,
 ntree = 500,
 do.trace = 100)

调用：
randomForest(x = iris[, 1:4], y = iris$Species, ntree = 500, do.trace = 100)
随机森林类型：分类
树的数量：500
每次分裂尝试的变量数：2
#
OOB 误差率估计：4%
混淆矩阵：
山鸢尾 变色鸢尾 维吉尼亚鸢尾 分类错误率
山鸢尾 50 0 0 0.00
变色鸢尾 0 47 3 0.06
维吉尼亚鸢尾 0 3 47 0.06
```

### 参数 描述

- x 包含类别描述变量的数据框  
y 各个观测值的类别。如果该向量是`factor`类型，则创建分类模型，否则创建回归模型。  
ntree 构建的单个 CART 树的数量  
do.trace 每第  $i$ 步，返回整体及各类别的袋外误差

# Chapter 110: Machine learning

## Section 110.1: Creating a Random Forest model

One example of machine learning algorithms is the Random Forest algorithm (Breiman, L. (2001). Random Forests. *Machine Learning* 45(5), p. 5-32). This algorithm is implemented in R according to Breiman's original Fortran implementation in the `randomForest` package.

Random Forest classifier objects can be created in R by preparing the class variable as `factor`, which is already apparent in the `iris` data set. Therefore we can easily create a Random Forest by:

```
library(randomForest)

rf <- randomForest(x = iris[, 1:4],
 y = iris$Species,
 ntree = 500,
 do.trace = 100)

Call:
randomForest(x = iris[, 1:4], y = iris$Species, ntree = 500, do.trace = 100)
Type of random forest: classification
Number of trees: 500
No. of variables tried at each split: 2
#
OOB estimate of error rate: 4%
Confusion matrix:
setosa versicolor virginica class.error
setosa 50 0 0 0.00
versicolor 0 47 3 0.06
virginica 0 3 47 0.06
```

### parameters Description

- x a data frame holding the describing variables of the classes  
y the classes of the individual observations. If this vector is `factor`, a classification model is created, if not a regression model is created.  
ntree The number of individual CART trees built  
do.trace every  $i$ th step, the out-of-the-box errors overall and for each class are returned

# 第111章：使用 texreg 以论文格式导出模型

texreg 包有助于以整洁、适合发表的方式导出模型（或多个模型）。结果可以导出为 HTML 或 .doc（微软 Office Word）格式。

## 第 111.1 节：打印线性回归结果

```
模型
fit1 <- lm(mpg ~ wt, data = mtcars)
fit2 <- lm(mpg ~ wt+hp, data = mtcars)
fit3 <- lm(mpg ~ wt+hp+cyl, data = mtcars)

导出为 html
texreg::htmlreg(list(fit1,fit2,fit3),file='models.html')

导出为 doc
texreg::htmlreg(list(fit1,fit2,fit3),file='models.doc')
```

结果看起来像论文中的表格。

	Model 1	Model 2	Model 3
(Intercept)	37.29*** (1.88)	37.23*** (1.60)	38.75*** (1.79)
wt	-5.34*** (0.56)	-3.88*** (0.63)	-3.17*** (0.74)
hp		-0.03** (0.01)	-0.02 (0.01)
cyl			-0.94 (0.55)
R2	0.75	0.83	0.84
Adj. R2	0.74	0.81	0.83
Num. obs.	32	32	32
RMSE	3.05	2.59	2.51

\*\*\*p < 0.001, \*\*p < 0.01, \*p < 0.05

Statistical models

在 `texreg::htmlreg()` 函数中还有几个额外的实用参数。以下是最有用参数的一个使用示例。

```
导出为 html
texreg::htmlreg(list(fit1,fit2,fit3),file='models.html',
 single.row = T,
 custom.model.names = LETTERS[1:3],
 leading.zero = F,
 digits = 3)
```

这将产生如下表格

# Chapter 111: Using texreg to export models in a paper-ready way

The `texreg` package helps to export a model (or several models) in a neat paper-ready way. The result may be exported as HTML or .doc (MS Office Word).

## Section 111.1: Printing linear regression results

```
models
fit1 <- lm(mpg ~ wt, data = mtcars)
fit2 <- lm(mpg ~ wt+hp, data = mtcars)
fit3 <- lm(mpg ~ wt+hp+cyl, data = mtcars)

export to html
texreg::htmlreg(list(fit1,fit2,fit3),file='models.html')

export to doc
texreg::htmlreg(list(fit1,fit2,fit3),file='models.doc')
```

The result looks like a table in a paper.

	Model 1	Model 2	Model 3
(Intercept)	37.29*** (1.88)	37.23*** (1.60)	38.75*** (1.79)
wt	-5.34*** (0.56)	-3.88*** (0.63)	-3.17*** (0.74)
hp		-0.03** (0.01)	-0.02 (0.01)
cyl			-0.94 (0.55)
R2	0.75	0.83	0.84
Adj. R2	0.74	0.81	0.83
Num. obs.	32	32	32
RMSE	3.05	2.59	2.51

\*\*\*p < 0.001, \*\*p < 0.01, \*p < 0.05

Statistical models

There are several additional handy parameters in `texreg::htmlreg()` function. Here is a use case for the most helpful parameters.

```
export to html
texreg::htmlreg(list(fit1,fit2,fit3),file='models.html',
 single.row = T,
 custom.model.names = LETTERS[1:3],
 leading.zero = F,
 digits = 3)
```

Which result in a table like this

	<b>A</b>	<b>B</b>	<b>C</b>
(Intercept)	37.285 (1.878)***	37.227 (1.599)***	38.752 (1.787)***
wt	-5.344 (0.559)***	-3.878 (0.633)***	-3.167 (0.741)***
hp		-0.032 (0.009)**	-0.018 (0.012)
cyl			-0.942 (0.551)
R2	0.753	0.827	0.843
Adj. R2	0.745	0.815	0.826
Num. obs.	32	32	32
RMSE	3.046	2.593	2.512

\*\*\*p < 0.001, \*\*p < 0.01, \*p < 0.05

Statistical models

	<b>A</b>	<b>B</b>	<b>C</b>
(Intercept)	37.285 (1.878)***	37.227 (1.599)***	38.752 (1.787)***
wt	-5.344 (0.559)***	-3.878 (0.633)***	-3.167 (0.741)***
hp		-0.032 (0.009)**	-0.018 (0.012)
cyl			-0.942 (0.551)
R2	0.753	0.827	0.843
Adj. R2	0.745	0.815	0.826
Num. obs.	32	32	32
RMSE	3.046	2.593	2.512

\*\*\*p < 0.001, \*\*p < 0.01, \*p < 0.05

Statistical models

# 第112章：出版

有许多方法可以格式化R代码、表格和图形以供出版。

## 第112.1节：格式化表格

这里，“表格”是广义的（涵盖[data.frame](#)、[table](#)、

### 打印为纯文本

打印（如控制台所见）可能足以用于以等宽字体查看的纯文本文档：

注意：在制作下面的示例数据之前，请确保你处于一个可以写入的空文件夹中。运行 [getwd\(\)](#)，  
如果需要更改文件夹，请阅读 [?setwd](#)。

```
..w = options()$width
options(width = 500) # 减少文本换行
sink(file = "mytab.txt")
 摘要(mtcars)
sink()
选项(宽度 = ..w)
rm(..w)
```

### 打印分隔表格

写入CSV（或其他常见格式），然后在电子表格编辑器中打开以进行最后润色是另一种选择：

注意：在制作下面的示例数据之前，请确保你处于一个可以写入的空文件夹中。运行 [getwd\(\)](#)，  
如果需要更改文件夹，请阅读 [?setwd](#)。

```
write.csv(mtcars, file="mytab.csv")
```

### 更多资源

- knitr::kable
- stargazer
- 表格::表格环境
- texreg
- xtable

## 第112.2节：格式化整个文档

Sweave来自utils包，允许在LaTeX文档中将代码、文字、图表和表格一起格式化。

### 更多资源

- Knitr和RMarkdown

# Chapter 112: Publishing

有许多方法可以格式化R代码、表格和图形以供出版。

## Section 112.1: Formatting tables

Here, "table" is meant broadly (covering [data.frame](#), [table](#),

### Printing to plain text

Printing (as seen in the console) might suffice for a plain-text document to be viewed in monospaced font:

*Note: Before making the example data below, make sure you're in an empty folder you can write to. Run [getwd\(\)](#) and read [?setwd](#) if you need to change folders.*

```
..w = options()$width
options(width = 500) # reduce text wrapping
sink(file = "mytab.txt")
 summary(mtcars)
sink()
options(width = ..w)
rm(..w)
```

### Printing delimited tables

Writing to CSV (or another common format) and then opening in a spreadsheet editor to apply finishing touches is another option:

*Note: Before making the example data below, make sure you're in an empty folder you can write to. Run [getwd\(\)](#) and read [?setwd](#) if you need to change folders.*

```
write.csv(mtcars, file="mytab.csv")
```

### Further resources

- knitr::kable
- stargazer
- tables::tabular
- texreg
- xtable

## Section 112.2: Formatting entire documents

[Sweave](#) from the [utils](#) package allows for formatting code, prose, graphs and tables together in a LaTeX document.

### Further Resources

- Knitr and RMarkdown

# 第113章：使用S4类实现状态机模式

有限状态机的概念通常在面向对象编程（OOP）语言中实现，例如使用Java语言，基于GOF中定义的状态模式（参考书籍：“设计模式”）。

R提供了多种机制来模拟面向对象范式，我们这里应用S4对象系统来实现该模式。

## 第113.1节：使用状态机解析行

我们将应用状态机模式，利用R的S4类特性解析具有特定模式的行。

### 问题陈述

我们需要解析一个文件，其中每行提供一个人的信息，使用分隔符（";"），但有些信息是可选的，且不是以空字段表示，而是缺失。每行可能包含以下信息：姓名；[地址；]电话。地址信息是可选的，有时有，有时没有，例如：

格雷戈里·布朗；25 NE 25街；+1-786-987-6543  
大卫·史密斯；786-123-4567  
艾伦·佩雷斯；25 SE 50街；+1-786-987-5553

第二行未提供地址信息。因此分隔符的数量可能不同，如本例中一行有一个分隔符，其他行有两个分隔符。由于分隔符数量可能变化，解决此问题的一种方法是根据字段的模式识别该字段是否存在。在这种情况下，我们可以使用正则表达式来识别这些模式。例如：

- 姓名: "`^([A-Z]?\s+)* [A-Z]+(\s+[A-Z]{1,2}\s+)?*[A-Z]+(-|\s+)[A-Z]+)*$`"。例如：  
拉斐尔·里尔、大卫·R·史密斯、埃内斯托·佩雷斯·冈萨雷斯、O'康纳·布朗、路易斯·佩雷斯·梅纳，等等。
- 地址: "`^\s[0-9]{1,4}(\s+[A-Z]{1,2}[0-9]{1,2}[A-Z]{1,2}|[A-Z]\s[0-9]+)$`"。例如：11020乐琼路，87 SW 27街。为简化起见，这里未包含邮政编码、城市、州，但可以包含在此字段中或添加额外字段。
- 电话: "`^\s*(\s+1(-|\s+))*[0-9]{3}(-|\s+)[0-9]{3}(-|\s+)[0-9]{4}$`"。例如：  
305-123-4567, 305 123 4567, +1-786-123-4567。

### 注意事项:

- 我考虑的是美国地址和电话的最常见模式，可以很容易地扩展以考虑更一般的情况。
- 在R语言中，符号" "对字符变量有特殊含义，因此需要对其进行转义。
- 为了简化定义正则表达式的过程，一个很好的建议是使用以下网页：[regex101.com](http://regex101.com)，这样你可以通过给定的示例进行操作，直到获得所有可能组合的预期结果。

这个想法是基于先前定义的模式来识别每一行字段。状态模式定义了以下实体（类），它们协作以控制特定行为（状态模式是一种行为模式）：

# Chapter 113: Implement State Machine Pattern using S4 Class

[Finite States Machine](#) concepts are usually implemented under Object Oriented Programming (OOP) languages, for example using [Java language, based on the State pattern](#) defined in GOF (refers to the book: "Design Patterns").

R provides several mechanisms to simulate the OO paradigm, let's apply [S4 Object System](#) for implementing this pattern.

## Section 113.1: Parsing Lines using State Machine

Let's apply the [State Machine pattern](#) for parsing lines with the specific pattern using S4 Class feature from R.

### PROBLEM ENUNCIATION

We need to parse a file where each line provides information about a person, using a delimiter (";"), but some information provided is optional, and instead of providing an empty field, it is missing. On each line we can have the following information: Name ; [Address ; ] Phone. Where the address information is optional, sometimes we have it and sometimes don't, for example:

GREGORY BROWN; 25 NE 25TH; +1-786-987-6543  
DAVID SMITH; 786-123-4567  
ALAN PEREZ; 25 SE 50TH; +1-786-987-5553

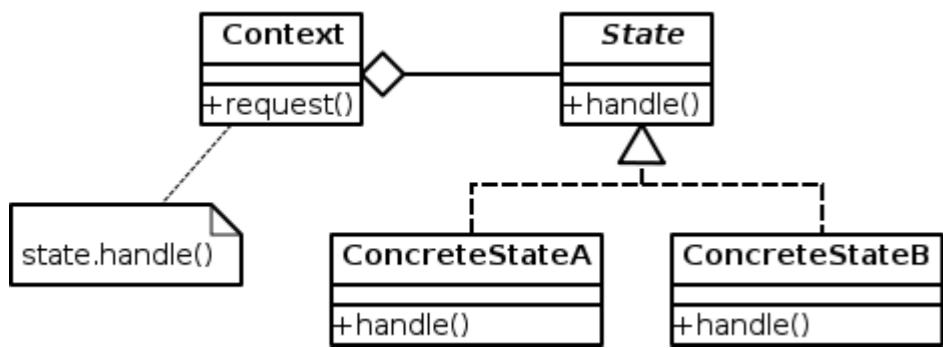
The second line does not provide address information. Therefore the number of delimiters may be different like in this case with one delimiter and for the other lines two delimiters. Because the number of delimiters may vary, one way to attack this problem is to recognize the presence or not of a given field based on its pattern. In such case we can use a [regular expression](#) for identifying such patterns. For example:

- **Name:** "`^([A-Z]?\s+)* [A-Z]+(\s+[A-Z]{1,2}\s+)?*[A-Z]+(-|\s+)[A-Z]+)*$`". For example:  
RAFAEL REAL, DAVID R. SMITH, ERNESTO PEREZ GONZALEZ, O'CONNOR BROWN, LUIS PEREZ-MENA, etc.
- **Address:** "`^\s[0-9]{1,4}(\s+[A-Z]{1,2}[0-9]{1,2}[A-Z]{1,2}|[A-Z]\s[0-9]+)$`". For example: 11020 LE JEUNE ROAD, 87 SW 27TH. For the sake of simplicity we don't include here the zipcode, city, state, but I can be included in this field or adding additional fields.
- **Phone:** "`^\s*(\s+1(-|\s+))*[0-9]{3}(-|\s+)[0-9]{3}(-|\s+)[0-9]{4}$`". For example:  
305-123-4567, 305 123 4567, +1-786-123-4567.

### Notes:

- I am considering the most common pattern of US addresses and phones, it can be easily extended to consider more general situations.
- In R the sign "\ " has special meaning for character variables, therefore we need to escape it.
- In order to simplify the process of defining regular expressions a good recommendation is to use the following web page: [regex101.com](http://regex101.com), so you can play with it, with a given example, until you get the expected result for all possible combinations.

The idea is to identify each line field based on previously defined patterns. The State pattern defines the following entities (classes) that collaborate to control the specific behavior (The State Pattern is a behavior pattern):



让我们结合问题的上下文来描述每个元素：

- 上下文：存储解析过程的上下文信息，即当前状态，并处理整个状态机过程。对于每个状态，会执行一个动作（handle()），但上下文根据状态，将其委托给特定状态定义的动作方法（State类中的handle()）。它定义了客户端感兴趣的接口。我们的Context类可以这样定义：
  - 属性：state
  - 方法：handle(), ...
- 状态：表示状态机中任意状态的抽象类。它定义了一个接口，用于封装与上下文特定状态相关的行为。它可以这样定义：
  - 属性：name, pattern
  - 方法：doAction(), isState（使用pattern属性验证输入参数是否属于该状态模式），...
- 具体状态（状态子类）：State类的每个子类实现与Context的某个状态相关的行为。我们的子类有：InitState、NameState、AddressState、PhoneState。这些类仅使用特定状态的逻辑实现通用方法，不需要额外的属性。

注意：执行动作的方法命名是个人偏好问题，可以是handle()、doAction()或goNext()。方法名doAction()可以在两个类（State或Context）中相同，我们倾向于在Context类中命名为handle()，以避免定义两个输入参数相同但类不同的通用方法时产生混淆。

## 人员类

使用S4语法，我们可以这样定义一个人员类：

```

setClass(Class = "Person",
slots = c(name = "character", address = "character", phone = "character")
)

```

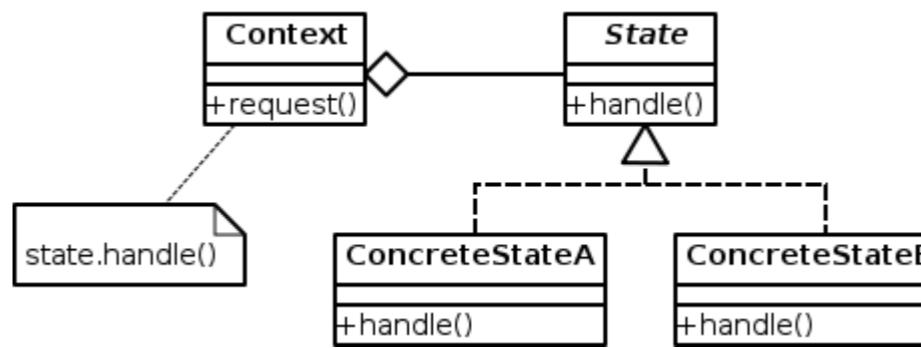
建议初始化类属性。setClass文档建议使用一个名为"initialize"的通用方法，而不是使用已弃用的属性，如：prototype、representation。

```

setMethod("initialize", "Person",
definition = function(.Object, name = NA_character_,
address = NA_character_, phone = NA_character_) {
 .Object@name <- name
 .Object@address <- address
 .Object@phone <- phone
 .Object
}
)

```

因为initialize方法已经是methods包的标准通用方法，我们需要遵守



Let's describe each element considering the context of our problem:

- Context: Stores the context information of the parsing process, i.e. the current state and handles the entire State Machine Process. For each state, an action is executed (handle()), but the context delegates it, based on the state, on the action method defined for a particular state (handle() from State class). It defines the interface of interest to clients. Our Context class can be defined like this:
  - Attributes: state
  - Methods: handle(), ...
- State: The abstract class that represents any state of the State Machine. It defines an interface for encapsulating the behavior associated with a particular state of the context. It can be defined like this:
  - Attributes: name, pattern
  - Methods: doAction(), isState (using pattern attribute verify whether the input argument belong to this state pattern or not), ...
- Concrete States (state sub-classes): Each subclass of the class State that implements a behavior associated with a state of the Context. Our sub-classes are: InitState, NameState, AddressState, PhoneState. Such classes just implements the generic method using the specific logic for such states. No additional attributes are required.

**Note:** It is a matter of preference how to name the method that carries out the action, handle(), doAction() or goNext(). The method name doAction() can be the same for both classes (State or Context) we preferred to name as handle() in the Context class for avoiding a confusion when defining two generic methods with the same input arguments, but different class.

## PERSON CLASS

Using the S4 syntax we can define a Person class like this:

```

setClass(Class = "Person",
slots = c(name = "character", address = "character", phone = "character")
)

```

It is a good recommendation to initialize the class attributes. The **setClass** documentation suggests using a generic method labeled as "**initialize**", instead of using deprecated attributes such as: **prototype**, **representation**.

```

setMethod("initialize", "Person",
definition = function(.Object, name = NA_character_,
address = NA_character_, phone = NA_character_) {
 .Object@name <- name
 .Object@address <- address
 .Object@phone <- phone
 .Object
}
)

```

Because the initialize method is already a standard generic method of package **methods**, we need to respect the

原始参数定义。我们可以在 R 提示符下输入以验证：

```
> initialize
```

它返回整个函数定义，你可以在顶部看到函数是如何定义的：

```
function (.Object, ...) {...}
```

因此，当我们使用setMethod时，需要严格遵循相同的语法 (.Object)。

另一个已有的泛型方法是show，它相当于Java中的toString()方法，为类domain提供一个特定实现是个好主意：

```
setMethod("show", signature = "Person",
definition = function(object) {
info <- sprintf("%s@[name='%s', address='%s', phone='%s']",
 class(object), object@name, object@address, object@phone)
cat(info)
invisible(NULL)
})
```

注意：我们使用的约定与Java默认的toString()实现相同。

假设我们想将解析的信息（一个Person对象列表）保存到数据集中，那么我们首先应该能够将对象列表转换成R可以转换的东西（例如，将对象强制转换为列表）。我们可以定义以下附加方法（更多细节请参见post）

```
setGeneric(name = "as.list", signature = c('x'),
def = function(x) standardGeneric("as.list"))

建议来源于此处：
#
http://stackoverflow.com/questions/30386009/how-to-extend-as-list-in-a-canonical-way-to-s4-objects
setMethod("as.list", signature = "Person",
definition = function(x) {
 mapply(function(y) {
 # 如果槽位是用户自定义对象，则递归应用 as.list
 # 因此，as.list 会递归应用
 if (inherits(slot(x,y), "Person")) {
 as.list(slot(x,y))
 } else {
 # 否则直接返回槽位
 slot(x,y)
 }
 },
 slotNames(class(x)),
 SIMPLIFY=FALSE
)}
```

R 不提供面向对象的简化语法，因为该语言最初设计是为了为统计学家提供有价值的函数。因此，每个用户方法需要两个部分：1) 定义部分（通过setGeneric）和2) 实现部分（通过setMethod）。如上例所示。

## 状态类

按照 S4 语法，定义抽象的状态类。

original argument definition. We can verify it typing on R prompt:

```
> initialize
```

It returns the entire function definition, you can see at the top who the function is defined like:

```
function (.Object, ...) {...}
```

Therefore when we use **setMethod** we need to follow *exactly* the same syntax (.Object).

Another existing generic method is **show**, it is equivalent **toString()** method from Java and it is a good idea to have a specific implementation for class domain:

```
setMethod("show", signature = "Person",
definition = function(object) {
 info <- sprintf("%s@[name='%s', address='%s', phone='%s']",
 class(object), object@name, object@address, object@phone)
 cat(info)
 invisible(NULL)
})
```

**Note:** We use the same convention as in the default **toString()** Java implementation.

Let's say we want to save the parsed information (a list of Person objects) into a dataset, then we should be able first to convert a list of objects to into something the R can transform (for example coerce the object as a list). We can define the following additional method (for more detail about this see the [post](#))

```
setGeneric(name = "as.list", signature = c('x'),
def = function(x) standardGeneric("as.list"))

Suggestion taken from here:
#
http://stackoverflow.com/questions/30386009/how-to-extend-as-list-in-a-canonical-way-to-s4-objects
setMethod("as.list", signature = "Person",
definition = function(x) {
 mapply(function(y) {
 #apply as.list if the slot is again an user-defined object
 #therefore, as.list gets applied recursively
 if (inherits(slot(x,y), "Person")) {
 as.list(slot(x,y))
 } else {
 #otherwise just return the slot
 slot(x,y)
 }
 },
 slotNames(class(x)),
 SIMPLIFY=FALSE
})
```

R does not provide a sugar syntax for OO because the language was initially conceived to provide valuable functions for Statisticians. Therefore each user method requires two parts: 1) the Definition part (via **setGeneric**) and 2) the implementation part (via **setMethod**). Like in the above example.

## STATE CLASS

Following S4 syntax, let's define the abstract State class.

```

setClass(Class = "State", slots = c(name = "character", pattern = "character"))

setMethod("initialize", "State",
definition = function(.Object, name = NA_character_, pattern = NA_character_) {
 .Object@name <- name
 .Object@pattern <- pattern
 .Object
})

setMethod("show", signature = "State",
 definition = function(object) {
info <- sprintf("%s@[name='%s', pattern='%s']", class(object),
 object@name, object@pattern)
 cat(info)
 invisible(NULL)
}

setGeneric(name = "isState", signature = c('obj', 'input'),
 def = function(obj, input) standardGeneric("isState"))

setGeneric(name = "doAction", signature = c('obj', 'input', 'context'),
 def = function(obj, input, context) standardGeneric("doAction"))

```

每个状态的子类都会关联一个name和pattern，同时还会有一种方法来判断给定的输入是否属于该状态 (isState() 方法)，并且实现该状态对应的操作 (doAction() 方法)。

为了理解该过程，让我们根据接收到的输入为每个状态定义转移矩阵：

输入/当前状态初始化	姓名地址 电话
姓名	姓名
地址	地址
电话	电话
结束	结束

注意：单元格 [行, 列]=[i, j] 表示当前状态 j 在接收到输入

这意味着在状态“姓名”下，它可以接收两种输入：地址或电话号码。另一种表示转移表的方法是使用以下UML状态机图：

```

setClass(Class = "State", slots = c(name = "character", pattern = "character"))

setMethod("initialize", "State",
definition = function(.Object, name = NA_character_, pattern = NA_character_) {
 .Object@name <- name
 .Object@pattern <- pattern
 .Object
})

setMethod("show", signature = "State",
 definition = function(object) {
info <- sprintf("%s@[name='%s', pattern='%s']", class(object),
 object@name, object@pattern)
 cat(info)
 invisible(NULL)
}

setGeneric(name = "isState", signature = c('obj', 'input'),
 def = function(obj, input) standardGeneric("isState"))

setGeneric(name = "doAction", signature = c('obj', 'input', 'context'),
 def = function(obj, input, context) standardGeneric("doAction"))

```

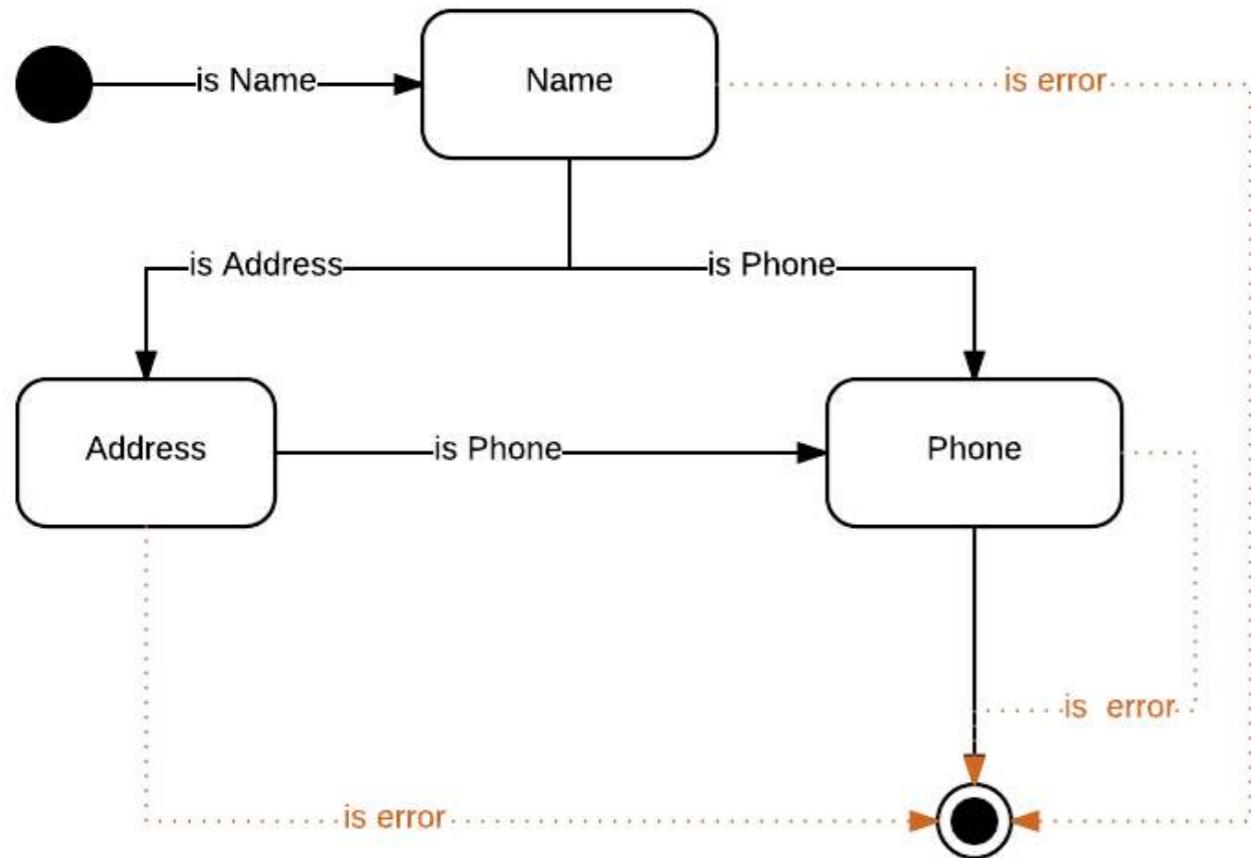
Every sub-class from State will have associated a name and pattern, but also a way to identify whether a given input belongs to this state or not (isState() method), and also implement the corresponding actions for this state (doAction() method).

In order to understand the process, let's define the transition matrix for each state based on the input received:

Input/Current State	Init	Name	Address	Phone
Name		Name		
Address			Address	
Phone			Phone	Phone
End				End

**Note:** The cell [row, col]=[i, j] represents the destination state for the current state j, when it receives the input i.

It means that under the state Name it can receive two inputs: an address or a phone number. Another way to represents the transaction table is using the following [UML State Machine](#) diagram:



**is error:** when the input argument has an invalid pattern

让我们将每个特定状态实现为类State的子状态

### 状态子类

初始化状态：

初始状态将通过以下类实现：

```

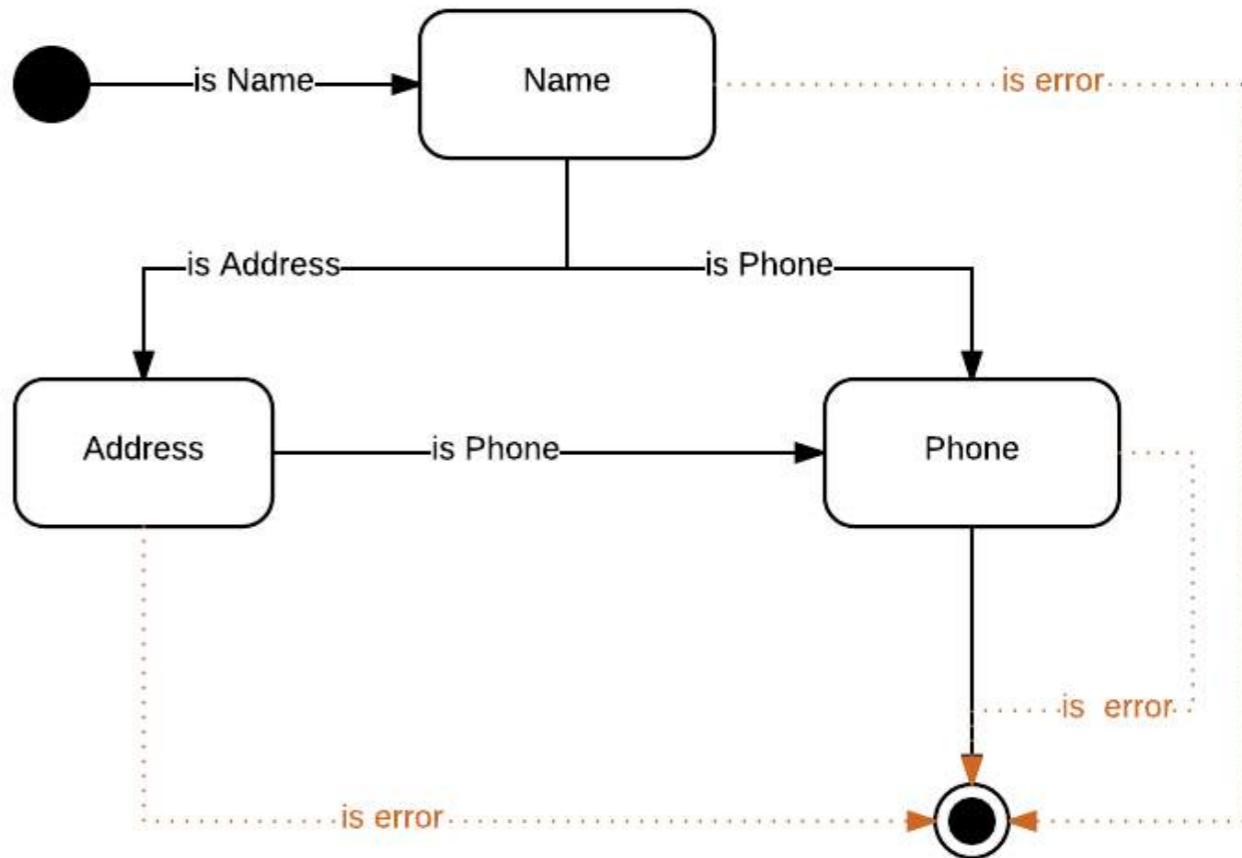
setClass("InitState", 包含 = "State")

setMethod("initialize", "InitState",
定义 = function(.Object, name = "init", pattern = NA_character_) {
 .Object@name <- name
 .Object@pattern <- pattern
 .Object
}

setMethod("show", 签名 = "InitState",
 定义 = function(object) {
 调用下一个方法()
 }
)

```

在R中，表示一个类是另一个类的子类是使用属性contains并指明父类的类名。



**is error:** when the input argument has an invalid pattern

Let's implement each particular state as a sub-state of the class State

### STATE SUB-CLASSES

**Init State:**

The initial state will be implemented via the following class:

```

setClass("InitState", contains = "State")

setMethod("initialize", "InitState",
 definition = function(.Object, name = "init", pattern = NA_character_) {
 .Object@name <- name
 .Object@pattern <- pattern
 .Object
 }
)

setMethod("show", signature = "InitState",
 definition = function(object) {
 callNextMethod()
 }
)

```

In R to indicate a class is a sub-class of other class is using the attribute contains and indicating the class name of the parent class.

因为子类只是实现通用方法，没有添加额外的属性，所以**show**方法只是调用上层类的等效方法（通过方法：**callNextMethod()**）

初始状态没有关联的模式，它只是表示过程的开始，然后我们用一个NA值初始化类。

现在让我们实现State类中的通用方法：

```
setMethod(f = "isState", signature = "InitState",
 definition = function(obj, input) {
 nameState <- new("NameState")
 result <- isState(nameState, input)
 return(result)
 }
)
```

对于这个特定状态（没有pattern），其想法只是初始化解析过程，期望第一个字段是一个name，否则将报错。

```
setMethod(f = "doAction", signature = "InitState",
 definition = function(obj, input, context) {
 nameState <- new("NameState")
 if (isState(nameState, input)) {
 person <- context@person
 person@name <- trimws(input)
 context@person <- person
 context@state <- nameState
 } else {
 msg <- sprintf("输入参数：'%s' 无法识别", input)
 stop(msg)
 }
 return(context)
 }
)
```

**doAction** 方法提供了状态转换并使用提取的信息更新上下文。这里我们通过 @-操作符访问上下文信息。相反，我们也可以定义 **get/set** 方法来封装这个过程（正如面向对象最佳实践中所要求的封装），但这会为每个 **get-set** 添加四个额外的方法，而对本例的目的没有实际价值。

在所有 **doAction** 实现中，建议在输入参数未被正确识别时添加保护措施。

## 名称 状态

这是该类定义的定义：

```
setClass ("NameState", contains = "State")

setMethod("initialize","NameState",
definition = function(.Object, name = "name",
pattern = "^([A-Z]? \s+)* [A-Z] + (\s+ [A-Z]{1,2} \s+)? ,? +)* [A-Z] + ((-| \s+)[A-Z]+)*$") {
 .Object@pattern <- pattern
 .Object@name <- name
 .Object
}
)
```

Because the sub-classes just implement the generic methods, without adding additional attributes, then the **show** method, just call the equivalent method from the upper class (via method: **callNextMethod()**)

The initial state does not have associated a pattern, it just represents the beginning of the process, then we initialize the class with an NA value.

Now lets to implement the generic methods from the State class:

```
setMethod(f = "isState", signature = "InitState",
 definition = function(obj, input) {
 nameState <- new("NameState")
 result <- isState(nameState, input)
 return(result)
 }
)
```

For this particular state (without pattern), the idea it just initializes the parsing process expecting the first field will be a name, otherwise it will be an error.

```
setMethod(f = "doAction", signature = "InitState",
 definition = function(obj, input, context) {
 nameState <- new("NameState")
 if (isState(nameState, input)) {
 person <- context@person
 person@name <- trimws(input)
 context@person <- person
 context@state <- nameState
 } else {
 msg <- sprintf("The input argument: '%s' cannot be identified", input)
 stop(msg)
 }
 return(context)
 }
)
```

The **doAction** method provides the transition and updates the context with the information extracted. Here we are accessing to context information via the @-operator. Instead, we can define **get/set** methods, to encapsulate this process (as it is mandated in OO best practices: encapsulation), but that would add four more methods per **get-set** without adding value for the purpose of this example.

It is a good recommendation in all **doAction** implementation, to add a safeguard when the input argument is not properly identified.

## Name State

Here is the definition of this class definition:

```
setClass ("NameState", contains = "State")

setMethod("initialize","NameState",
definition=function(.Object, name="name",
pattern = "^([A-Z]? \s+)* [A-Z] + (\s+ [A-Z]{1,2} \s+)? ,? +)* [A-Z] + ((-| \s+)[A-Z]+)*$") {
 .Object@pattern <- pattern
 .Object@name <- name
 .Object
}
)
```

```

setMethod("show", signature = "NameState",
definition = function(object) {
 调用下一个方法()
}
)

```

我们使用函数grep来验证输入是否属于给定的模式。

```

setMethod(f="isState", signature="NameState",
definition=function(obj, input) {
result <- grep(obj@pattern, input, perl=TRUE)
 return(result)
}
)

```

现在我们定义针对给定状态要执行的操作：

```

setMethod(f = "doAction", signature = "NameState",
definition=function(obj, input, context) {
addressState <- new("AddressState")
 phoneState <- new("PhoneState")
 person <- context@person
 if (isState(addressState, input)) {
 person@address <- trimws(input)
 context@person <- person
 context@state <- addressState
 } else if (isState(phoneState, input)) {
 person@phone <- trimws(input)
 }
 context@person <- person
 context@state <- phoneState
} else {
 msg <- sprintf("输入参数：'%s' 无法识别", input)
 stop(msg)
}
return(context)
}
)

```

这里我们考虑两种可能的转换：一种是地址状态，另一种是电话状态。在所有情况下，我们都会更新上下文信息：

- 人员信息：地址或电话与输入参数。
- 进程的状态

识别状态的方法是调用特定状态的isState()方法。我们创建默认的特定状态 (addressState, phoneState)，然后进行特定的验证。

其他子类（每个状态一个）的实现逻辑非常相似。

## 地址状态

```

setClass("AddressState", contains = "State")

setMethod("initialize", "AddressState",
definition = function(.Object, name="address",
 pattern = "^\s[0-9]{1,4}(\s+[A-Z]{1,2}[0-9]{1,2}[A-Z]{1,2})[A-Z\s0-9]+$")
 .Object@pattern <- pattern
.name <- name
.Object
)

```

```

setMethod("show", signature = "NameState",
definition = function(object) {
 callNextMethod()
}
)

```

We use the function `grep` for verifying the input belongs to a given pattern.

```

setMethod(f="isState", signature="NameState",
definition=function(obj, input) {
 result <- grep(obj@pattern, input, perl=TRUE)
 return(result)
}
)

```

Now we define the action to carry out for a given state:

```

setMethod(f = "doAction", signature = "NameState",
definition=function(obj, input, context) {
 addressState <- new("AddressState")
 phoneState <- new("PhoneState")
 person <- context@person
 if (isState(addressState, input)) {
 person@address <- trimws(input)
 context@person <- person
 context@state <- addressState
 } else if (isState(phoneState, input)) {
 person@phone <- trimws(input)
 context@person <- person
 context@state <- phoneState
 } else {
 msg <- sprintf("The input argument: '%s' cannot be identified", input)
 stop(msg)
 }
 return(context)
}
)

```

Here we consider two possible transitions: one for Address state and the other one for Phone state. In all cases we update the context information:

- The `person` information: address or phone with the input argument.
- The state of the process

The way to identify the state is to invoke the method: `isState()` for a particular state. We create a default specific states (addressState, phoneState) and then ask for a particular validation.

The logic for the other sub-classes (one per state) implementation is very similar.

## Address State

```

setClass("AddressState", contains = "State")

setMethod("initialize", "AddressState",
definition = function(.Object, name="address",
 pattern = "^\s[0-9]{1,4}(\s+[A-Z]{1,2}[0-9]{1,2}[A-Z]{1,2})[A-Z\s0-9]+$")
 .Object@pattern <- pattern
.name <- name
.Object
)

```

```

 }

setMethod("show", signature = "AddressState",
 definition = function(object) {
 callNextMethod()
 }
)

setMethod(f="isState", signature="AddressState",
 definition=function(obj, input) {
result <- grepl(obj@pattern, input, perl=TRUE)
 return(result)
}
)

setMethod(f = "doAction", "AddressState",
 definition=function(obj, input, context) {
 phoneState <- new("PhoneState")
 if (isState(phoneState, input)) {
 person <- context@person
 person@phone <- trimws(input)
 context@person <- person
 context@state <- phoneState
 } else {
 msg <- sprintf("输入参数 :'%s' 无法识别", input)
 stop(msg)
 }
 return(context)
}
)

```

## 电话状态

```

setClass("PhoneState", 包含 = "State")

setMethod("initialize", "PhoneState",
 定义 = 函数(.Object, 名称 = "phone",
 模式 = "^\s*(\+|-|\s+)*[0-9]{3}(-|\s+)[0-9]{3}(-|\s+)[0-9]{4}$") {
 .Object@pattern <- 模式
 .Object@name <- name
 .Object
 }
)

设置方法("show", 签名 = "PhoneState",
 定义 = 函数(对象) {
 callNextMethod()
 }
)

设置方法(f = "isState", 签名 = "PhoneState",
 定义 = 函数(obj, 输入) {
结果 <- grepl(obj@pattern, 输入, perl = TRUE)
 return(result)
}
)

```

这里是我们将人员信息添加到context的persons列表中的地方。

设置方法(f = "doAction", "PhoneState",

```

 }

setMethod("show", signature = "AddressState",
 definition = function(object) {
 callNextMethod()
 }
)

setMethod(f="isState", signature="AddressState",
 definition=function(obj, input) {
result <- grepl(obj@pattern, input, perl=TRUE)
 return(result)
}
)

setMethod(f = "doAction", "AddressState",
 definition=function(obj, input, context) {
 phoneState <- new("PhoneState")
 if (isState(phoneState, input)) {
 person <- context@person
 person@phone <- trimws(input)
 context@person <- person
 context@state <- phoneState
 } else {
 msg <- sprintf("The input argument: '%s' cannot be identified", input)
 stop(msg)
 }
 return(context)
}
)

```

## Phone State

```

setClass("PhoneState", contains = "State")

setMethod("initialize", "PhoneState",
 definition = function(.Object, name = "phone",
 pattern = "^\s*(\+|-|\s+)*[0-9]{3}(-|\s+)[0-9]{3}(-|\s+)[0-9]{4}$") {
 .Object@pattern <- pattern
 .Object@name <- name
 .Object
 }
)

setMethod("show", signature = "PhoneState",
 definition = function(object) {
 callNextMethod()
 }
)

setMethod(f = "isState", signature = "PhoneState",
 definition = function(obj, input) {
 result <- grepl(obj@pattern, input, perl = TRUE)
 return(result)
}
)

```

Here is where we add the person information into the list of persons of the context.

setMethod(f = "doAction", "PhoneState",

```

定义 = 函数(obj, 输入, context) {
 context <- addPerson(context, context@person)
 context@state <- new("InitState")
 返回(context)
}
)

```

## 上下文类

现在让我们来解释Context类的实现。我们可以考虑以下属性来定义它：

```

setClass(Class = "Context",
slots = c(state = "State", persons = "list", person = "Person")
)

```

其中

- state：流程的当前状态
- person：当前人员，表示我们已经从当前行解析出的信息。
- persons：已处理的解析人员列表。

**注意：**可选地，我们可以添加一个name来通过名称识别上下文，以防我们正在使用多个解析器类型。

```

setMethod(f="initialize", signature="Context",
 definition = function(.Object) {
 .Object@state <- new("InitState")
 .Object@persons <- list()
 .Object@person <- new("Person")
 return(.Object)
 }
)

setMethod("show", signature = "Context",
 definition = function(object) {
 cat("一个类为 ", class(object), "", sep = "") info <- sprintf("[state=%
's', persons='%s', person='%s']", object@state,toString(object@persons), object@person)
 cat(info)
 invisible(NULL)
 }
)

setGeneric(name = "handle", signature = c('obj', 'input', 'context'),
 def = function(obj, input, context) standardGeneric("handle"))

setGeneric(name = "addPerson", signature = c('obj', 'person'),
 def = function(obj, person) standardGeneric("addPerson"))

setGeneric(name = "parseLine", signature = c('obj', 's'),
 def = function(obj, s) standardGeneric("parseLine"))

setGeneric(name = "parseLines", signature = c('obj', 's'),
 def = function(obj, s) standardGeneric("parseLines"))

setGeneric(name = "as.df",
 signature = c('obj'),
 def = function(obj) standardGeneric("as.df"))

```

通过这些泛型方法，我们控制了解析过程的整体行为：

```

definition = function(obj, input, context) {
 context <- addPerson(context, context@person)
 context@state <- new("InitState")
 return(context)
}
)

```

## CONTEXT CLASS

Now the lets to explain the Context class implementation. We can define it considering the following attributes:

```

setClass(Class = "Context",
slots = c(state = "State", persons = "list", person = "Person")
)

```

Where

- state: The current state of the process
- person: The current person, it represents the information we have already parsed from the current line.
- persons: The list of parsed persons processed.

**Note:** Optionally, we can add a name to identify the context by name in case we are working with more than one parser type.

```

setMethod(f="initialize", signature="Context",
 definition = function(.Object) {
 .Object@state <- new("InitState")
 .Object@persons <- list()
 .Object@person <- new("Person")
 return(.Object)
 }
)

setMethod("show", signature = "Context",
 definition = function(object) {
 cat("An object of class ", class(object), "\n", sep = "")
 info <- sprintf("[state='%s', persons='%s', person='%s']", object@state,
 toString(object@persons), object@person)
 cat(info)
 invisible(NULL)
 }
)

setGeneric(name = "handle", signature = c('obj', 'input', 'context'),
 def = function(obj, input, context) standardGeneric("handle"))

setGeneric(name = "addPerson", signature = c('obj', 'person'),
 def = function(obj, person) standardGeneric("addPerson"))

setGeneric(name = "parseLine", signature = c('obj', 's'),
 def = function(obj, s) standardGeneric("parseLine"))

setGeneric(name = "parseLines", signature = c('obj', 's'),
 def = function(obj, s) standardGeneric("parseLines"))

setGeneric(name = "as.df",
 signature = c('obj'),
 def = function(obj) standardGeneric("as.df"))

```

With such generic methods, we control the entire behavior of the parsing process:

- handle(): 将调用当前state的特定 doAction()方法。
- addPerson: 一旦达到结束状态，我们需要将一个person添加到已解析的persons列表中。
- parseLine(): 解析单行
- parseLines(): 解析多行 (行数组)
- as.df(): 将persons列表中的信息提取到数据框对象中。

现在让我们继续对应的实现：

handle() 方法，委托给当前 context 的 doAction() 方法的 state :

```
setMethod(f = "handle", signature = "Context",
 definition = function(obj, input) {
 obj <- doAction(obj@state, input, obj)
 return(obj)
 }
)

setMethod(f = "addPerson", signature = "Context",
 definition = function(obj, person) {
 obj@persons <- c(obj@persons, person)
 return(obj)
 }
)
```

首先，我们使用分隔符通过 R 函数 strsplit() 将原始行拆分成数组，以识别每个元素，然后对每个元素作为给定状态的输入值进行迭代。 handle() 方法再次返回带有更新信息 (state、person、persons 属性) 的 context。

```
setMethod(f = "parseLine", signature = "Context",
 definition = function(obj, s) {
 elements <- strsplit(s, ";")[[1]]
 # 添加一个空字段以考虑结束状态。
 elements <- c(elements, "")
 n <- length(elements)
 input <- NULL
 for (i in 1:n)) {
 input <- elements[i]
 obj <- handle(obj, input)
 }
 return(obj@person)
 }
)
```

因为 R 会复制输入参数，我们需要返回上下文 (obj) :

```
setMethod(f = "parseLines", signature = "Context",
 definition = function(obj, s) {
 n <- length(s)
 listOfPersons <- list()
 for (i in 1:n)) {
 ipersons <- parseLine(obj, s[i])
 listOfPersons[[i]] <- ipersons
 }
 obj@persons <- listOfPersons
 return(obj)
 }
)
```

- handle(): Will invoke the particular doAction() method of the current state.
- addPerson: Once we reach the end state, we need to add a **person** to the list of persons we have parsed.
- parseLine(): Parse a single line
- parseLines(): Parse multiple lines (an array of lines)
- as.df(): Extract the information from persons list into a data frame object.

Let's go on now with the corresponding implementations:

handle() method, delegates on doAction() method from the current state of the context:

```
setMethod(f = "handle", signature = "Context",
 definition = function(obj, input) {
 obj <- doAction(obj@state, input, obj)
 return(obj)
 }
)

setMethod(f = "addPerson", signature = "Context",
 definition = function(obj, person) {
 obj@persons <- c(obj@persons, person)
 return(obj)
 }
)
```

First, we split the original line in an array using the delimiter to identify each element via the R-function **strsplit()**, then iterate for each element as an input value for a given state. The handle() method returns again the context with the updated information (state, **person**, persons attribute).

```
setMethod(f = "parseLine", signature = "Context",
 definition = function(obj, s) {
 elements <- strsplit(s, ";")[[1]]
 # Adding an empty field for considering the end state.
 elements <- c(elements, "")
 n <- length(elements)
 input <- NULL
 for (i in 1:n)) {
 input <- elements[i]
 obj <- handle(obj, input)
 }
 return(obj@person)
 }
)
```

Because R makes a copy of the input argument, we need to return the context (obj):

```
setMethod(f = "parseLines", signature = "Context",
 definition = function(obj, s) {
 n <- length(s)
 listOfPersons <- list()
 for (i in 1:n)) {
 ipersons <- parseLine(obj, s[i])
 listOfPersons[[i]] <- ipersons
 }
 obj@persons <- listOfPersons
 return(obj)
 }
)
```

属性persons是S4类Person实例的列表。由于R无法识别如何处理用户定义类的实例，因此无法将其强制转换为任何标准类型。解决方案是使用之前定义的as.list方法将Person转换为列表。然后，我们可以通过lapply()函数将此函数应用于列表persons的每个元素。接着，在下一次调用lapply()函数时，应用data.frame函数将persons.list的每个元素转换为数据框。最后，调用rbind()函数将每个转换后的元素作为数据框的新行添加进来（更多细节请参见该帖子）

```
建议来源于该帖子：
http://stackoverflow.com/questions/4227223/r-list-to-data-frame
setMethod(f = "as.df", signature = "Context",
 definition = function(obj) {
 persons <- obj@persons
 persons.list <- lapply(persons, as.list)
 persons.ds <- do.call(rbind, lapply(persons.list, data.frame, stringsAsFactors = FALSE))
 return(persons.ds)
 })
```

## 全部整合

最后，让我们测试整个解决方案。定义要解析的行，其中第二行缺少地址信息。

```
s <- c(
 "格雷戈里·布朗；东北25街25号；+1-786-987-6543",
 "大卫·史密斯；786-123-4567",
 "艾伦·佩雷斯；东南25街50号；+1-786-987-5553")
```

现在我们初始化上下文，并解析这些行：

```
context <- new("Context")
context <- parseLines(context, s)
```

最后获取相应的数据集并打印：

```
df <- as.df(context)
> df
姓名 地址 电话
1 格雷戈里·布朗 25 东北25街 +1-786-987-6543
2 大卫·史密斯 <NA> 786-123-4567
3 艾伦·佩雷斯 25 东南25街 +1-786-987-5553
```

现在测试show方法：

```
> show(context@persons[[1]])
Person@[name='格雷戈里·布朗', address='25 NE 25TH', phone='+1-786-987-6543']
```

对于某些子状态：

```
> show(new("PhoneState"))
PhoneState@[name='phone', pattern='^\\s*(\\+1(-|\\s+))*[0-9]{3}(-|\\s+)[0-9]{3}(-|\\s+)[0-9]{4}$']
```

最后，测试as.list()方法：

The attribute persons is a list of instance of S4 Person class. This something cannot be coerced to any standard type because R does not know how to treat an instance of a user defined class. The solution is to convert a Person into a list, using the `as.list` method previously defined. Then we can apply this function to each element of the list persons, via the `lapply()` function. Then in the next invocation to `lapply()` function, now applies the `data.frame` function for converting each element of the `persons.list` into a data frame. Finally, the `rbind()` function is called for adding each element converted as a new row of the data frame generated (for more detail about this see this [post](#))

```
Sugestion taken from this post:
http://stackoverflow.com/questions/4227223/r-list-to-data-frame
setMethod(f = "as.df", signature = "Context",
 definition = function(obj) {
 persons <- obj@persons
 persons.list <- lapply(persons, as.list)
 persons.ds <- do.call(rbind, lapply(persons.list, data.frame, stringsAsFactors = FALSE))
 return(persons.ds)
 })
```

## PUTTING ALL TOGETHER

Finally, lets to test the entire solution. Define the lines to parse where for the second line the address information is missing.

```
s <- c(
 "GREGORY BROWN; 25 NE 25TH; +1-786-987-6543",
 "DAVID SMITH;786-123-4567",
 "ALAN PEREZ; 25 SE 50TH; +1-786-987-5553")
```

Now we initialize the context, and parse the lines:

```
context <- new("Context")
context <- parseLines(context, s)
```

Finally obtain the corresponding dataset and print it:

```
df <- as.df(context)
> df
 name address phone
1 GREGORY BROWN 25 NE 25TH +1-786-987-6543
2 DAVID SMITH <NA> 786-123-4567
3 ALAN PEREZ 25 SE 50TH +1-786-987-5553
```

Let's test now the `show` methods:

```
> show(context@persons[[1]])
Person@[name='GREGORY BROWN', address='25 NE 25TH', phone='+1-786-987-6543']
```

And for some sub-state:

```
> show(new("PhoneState"))
PhoneState@[name='phone', pattern='^\\s*(\\+1(-|\\s+))*[0-9]{3}(-|\\s+)[0-9]{3}(-|\\s+)[0-9]{4}$']
```

Finally, test the `as.list()` method:

```
> as.list(context@persons[[1]])
$name
[1] "格雷戈里·布朗"

$address
[1] "25 NE 25TH"

$phone
[1] "+1-786-987-6543"

>
```

## 结论

本示例展示了如何实现状态模式，使用R中可用的面向对象范式机制之一。然而，R的面向对象解决方案不够用户友好，并且与其他面向对象编程语言差异很大。你需要转换思维方式，因为语法完全不同，更像是函数式编程范式。例如，Java/C#中是object.setID("A1")，而在R中必须这样调用方法：setID(object, "A1")。因此，你总是需要将对象作为输入参数来提供函数的上下文。同样，没有特殊的this类属性，也没有用于访问给定类的方法或属性的"." 符号。由于通过属性值（如"Person"、"isState"等）来引用类或方法，这种方式更容易出错。

如上所述，S4类解决方案相比传统的Java/C#语言完成简单任务需要更多代码行数。无论如何，状态模式是解决此类问题的良好且通用的方案。它通过将逻辑委托给特定状态简化了流程。我们不再使用一个庞大的if-else块来控制所有情况，而是在每个State子类实现中使用较小的if-else块来实现每个状态下要执行的操作。

附件：你可以[在这里](#)下载完整脚本。

欢迎任何建议。

```
> as.list(context@persons[[1]])
$name
[1] "GREGORY BROWN"

$address
[1] "25 NE 25TH"

$phone
[1] "+1-786-987-6543"

>
```

## CONCLUSION

This example shows how to implement the State pattern, using one of the available mechanisms from R for using the OO paradigm. Nevertheless, the R OO solution is not user-friendly and differs so much from other OOP languages. You need to switch your mindset because the syntax is completely different, it reminds more the functional programming paradigm. For example instead of: `object.setID("A1")` as in Java/C#, for R you have to invoke the method in this way: `setID(object, "A1")`. Therefore you always have to include the object as an input argument to provide the context of the function. On the same way, there is no special `this` class attribute and either a `"."` notation for accessing methods or attributes of the given class. It is more error prompt because to refer a class or methods is done via attribute value (`"Person"`, `"isState"`, etc.).

Said the above, S4 class solution, requires much more lines of codes than a traditional Java/C# languages for doing simple tasks. Anyway, the State Pattern is a good and generic solution for such kind of problems. It simplifies the process delegating the logic into a particular state. Instead of having a big `if-else` block for controlling all situations, we have smaller `if-else` blocks inside on each State sub-class implementation for implementing the action to carry out in each state.

**Attachment:** [Here](#) you can download the entire script.

Any suggestion is welcome.

# 第114章：使用tidyr重塑数据

tidyr有两个用于重塑数据的工具：gather（宽格式转长格式）和spread（长格式转宽格式）。

有关其他选项，请参见重塑数据。

## 第114.1节：使用spread()将长格式重塑为宽格式

```
library(tidyr)

示例数据
set.seed(123)
df <- data.frame(
 name = rep(c("firstName", "secondName"), each=4),
 numbers = rep(1:4, 2),
 value = rnorm(8)
)
df

name numbers value
1 firstName 1 -0.56047565
2 firstName 2 -0.23017749
3 firstName 3 1.55870831
4 firstName 4 0.07050839
5 secondName 1 0.12928774
6 secondName 2 1.71506499
7 secondName 3 0.46091621
8 secondName 4 -1.26506123
```

我们可以将“numbers”列“展开”为单独的列：

```
spread(data = df,
 key = numbers,
 value = value)
name 1 2 3 4
1 firstName -0.5604756 -0.2301775 1.5587083 0.07050839
2 secondName 0.1292877 1.7150650 0.4609162 -1.26506123
```

或者将“name”列展开为单独的列：

```
spread(data = df,
 key = name,
 value = value)
numbers firstName secondName
1 1 -0.56047565 0.1292877
2 2 -0.23017749 1.7150650
3 3 1.55870831 0.4609162
4 4 0.07050839 -1.2650612
```

## 第114.2节：使用gather() 将宽格式转换为长格式

```
library(tidyr)

示例数据
df <- read.table(text = " numbers firstName secondName
1 1 1.5862639 0.4087477
2 2 0.1499581 0.9963923")
```

# Chapter 114: Reshape using tidyr

tidyr has two tools for reshaping data: gather (wide to long) and spread (long to wide).

See Reshaping data for other options.

## Section 114.1: Reshape from long to wide format with spread()

```
library(tidyr)

example data
set.seed(123)
df <- data.frame(
 name = rep(c("firstName", "secondName"), each=4),
 numbers = rep(1:4, 2),
 value = rnorm(8)
)
df

name numbers value
1 firstName 1 -0.56047565
2 firstName 2 -0.23017749
3 firstName 3 1.55870831
4 firstName 4 0.07050839
5 secondName 1 0.12928774
6 secondName 2 1.71506499
7 secondName 3 0.46091621
8 secondName 4 -1.26506123
```

We can "spread" the 'numbers' column, into separate columns:

```
spread(data = df,
 key = numbers,
 value = value)
name 1 2 3 4
1 firstName -0.5604756 -0.2301775 1.5587083 0.07050839
2 secondName 0.1292877 1.7150650 0.4609162 -1.26506123
```

Or spread the 'name' column into separate columns:

```
spread(data = df,
 key = name,
 value = value)
numbers firstName secondName
1 1 -0.56047565 0.1292877
2 2 -0.23017749 1.7150650
3 3 1.55870831 0.4609162
4 4 0.07050839 -1.2650612
```

## Section 114.2: Reshape from wide to long format with gather()

```
library(tidyr)

example data
df <- read.table(text = " numbers firstName secondName
1 1 1.5862639 0.4087477
2 2 0.1499581 0.9963923")
```

```
3 3 0.4117353 0.3740009
4 4 -0.4926862 0.4437916", header = T)
df
numbers firstName secondName
1 1 1.5862639 0.4087477
2 2 0.1499581 0.9963923
3 3 0.4117353 0.3740009
4 4 -0.4926862 0.4437916
```

我们可以使用 'numbers' 作为键列将这些列合并：

```
gather(data = df,
 key = numbers,
 value = myValue)
numbers numbers myValue
1 1 firstName 1.5862639
2 2 firstName 0.1499581
3 3 firstName 0.4117353
4 4 firstName -0.4926862
5 1 secondName 0.4087477
6 2 secondName 0.9963923
7 3 secondName 0.3740009
8 4 secondName 0.4437916
```

```
3 3 0.4117353 0.3740009
4 4 -0.4926862 0.4437916", header = T)
df
numbers firstName secondName
1 1 1.5862639 0.4087477
2 2 0.1499581 0.9963923
3 3 0.4117353 0.3740009
4 4 -0.4926862 0.4437916
```

We can gather the columns together using 'numbers' as the key column:

```
gather(data = df,
 key = numbers,
 value = myValue)
numbers numbers myValue
1 1 firstName 1.5862639
2 2 firstName 0.1499581
3 3 firstName 0.4117353
4 4 firstName -0.4926862
5 1 secondName 0.4087477
6 2 secondName 0.9963923
7 3 secondName 0.3740009
8 4 secondName 0.4437916
```

# 第115章：通过替换修改字符串

`sub` 和 `gsub` 用于使用模式编辑字符串。有关相关函数的更多信息，请参见模式匹配与替换；有关如何构建模式，请参见正则表达式。

## 第115.1节：使用捕获组重新排列字符串

如果你想改变字符串的顺序，可以在pattern中使用括号将字符串的部分内容分组。这些分组可以在replacement参数中通过连续的数字来引用。

下面的示例展示了如何将形式为“姓氏, 名字”的名字向量重新排序为“名字 姓氏”的向量。

```
library(randomNames)
set.seed(1)

strings <- randomNames(5)
strings
[1] "Sigg, Zachary" "Holt, Jake" "Ortega, Sandra" "De La Torre, Nichole"
[5] "Perkins, Donovan"

sub("^(.+)\s(.+)$", "\2 \1", strings)
[1] "Zachary Sigg" "Jake Holt" "Sandra Ortega" "Nichole De La Torre"
[5] "Perkins, Donovan"
```

如果你只需要姓氏，可以只引用第一个括号内的内容。

```
sub("^(.+)\s(.+)", "\1", strings)
[1] "西格" "霍尔特" "奥尔特加" "德拉托雷" "珀金斯"
```

## 第115.2节：消除重复的连续元素

假设我们想要从字符串中消除重复的子序列元素（可能不止一个）。例如：

2,14,14,14,19

并将其转换为：

2,14,19

使用`gsub`，我们可以实现：

```
gsub("(\\d+)(,\\1)+", "\\1", "2,14,14,14,19")
[1] "2,14,19"
```

它同样适用于多个不同的重复，例如：

```
> gsub("(\\d+)(,\\1)+", "\\1", "2,14,14,14,19,19,20,21")
[1] "2,14,19,20,21"
```

# Chapter 115: Modifying strings by substitution

`sub` 和 `gsub` 用于编辑字符串使用模式。请参见模式匹配和替换了解更多相关信息；有关如何构建模式，请参见正则表达式。

## Section 115.1: Rearrange character strings using capture groups

If you want to change the order of a character strings you can use parentheses in the pattern to group parts of the string together. These groups can in the replacement argument be addressed using consecutive numbers.

The following example shows how you can reorder a vector of names of the form "surname, forename" into a vector of the form "forename surname".

```
library(randomNames)
set.seed(1)

strings <- randomNames(5)
strings
[1] "Sigg, Zachary" "Holt, Jake" "Ortega, Sandra" "De La Torre, Nichole"
[5] "Perkins, Donovan"

sub("^(.+),\\s(.+)$", "\\2 \\1", strings)
[1] "Zachary Sigg" "Jake Holt" "Sandra Ortega" "Nichole De La Torre"
[5] "Perkins, Donovan"
```

If you only need the surname you could just address the first pairs of parentheses.

```
sub("^(.+),\\s(.+)", "\\1", strings)
[1] "Sigg" "Holt" "Ortega" "De La Torre" "Perkins"
```

## Section 115.2: Eliminate duplicated consecutive elements

Let's say we want to eliminate duplicated subsequence element from a string (it can be more than one). For example:

2,14,14,14,19

and convert it into:

2,14,19

Using `gsub`, we can achieve it:

```
gsub("(\\d+)(,\\1)+", "\\1", "2,14,14,14,19")
[1] "2,14,19"
```

It works also for more than one different repetition, for example:

```
> gsub("(\\d+)(,\\1)+", "\\1", "2,14,14,14,19,20,21")
[1] "2,14,19,20,21"
```

让我们解释一下正则表达式：

1. `(\\d+)` : 一个由()界定的第1组，匹配任意数字（至少一个）。请记住这里需要使用双反斜杠`\\\``，因为对于字符变量，反斜杠表示特殊的转义字符，用于字面字符串定界符`\\"` 或 \\'`。`\\d\`` 等价于`: [0-9]`。
2. `,` : 一个标点符号`,`，(我们可以包含空格或任何其他分隔符)
3. `\\1` : 与第1组相同的字符串，即：重复的数字。如果不匹配，则模式不匹配。  
不匹配。

让我们尝试一个类似的情况：去除连续重复的单词：

```
one, two, two, three, four, four, five, six
```

然后，只需将`\\d`替换为`\\w`，其中`\\w`匹配任何单词字符，包括：任意字母、数字或下划线。它等价于`[a-zA-Z0-9_]`：

```
> gsub("(\\w+)(,\\w+)", "\\1", "one, two, two, three, four, four, five, six")
[1] "one, two, three, four, five, six"
>
```

那么，上述模式包括了重复数字的特殊情况。

Let's explain the regular expression:

1. `(\\d+)`: A group 1 delimited by () and finds any digit (at least one). Remember we need to use the double backslash (\\) here because for a character variable a backslash represents special escape character for literal string delimiters (\ " or \'). `\\d\`` is equivalent to: [0-9].
2. `,`: A punctuation sign: , (we can include spaces or any other delimiter)
3. `\\1`: An identical string to the group 1, i.e.: the repeated number. If that doesn't happen, then the pattern doesn't match.

Let's try a similar situation: eliminate consecutive repeated words:

```
one, two, two, three, four, four, five, six
```

Then, just replace `\\d` by `\\w`, where `\\w` matches any word character, including: any letter, digit or underscore. It is equivalent to `[a-zA-Z0-9_]`:

```
> gsub("(\\w+)(,\\w+)", "\\1", "one, two, two, three, four, four, five, six")
[1] "one, two, three, four, five, six"
>
```

Then, the above pattern includes as a particular case duplicated digits case.

# 第116章：非标准求值与标准求值

R中的dplyr和许多现代库在交互式编程中使用非标准求值（NSE），而在编程中使用标准求值（SE）<sup>1</sup>。

例如，summarise()函数使用非标准求值，但依赖于使用标准求值的summarise\_()函数。

lazyeval库使将标准求值函数转换为NSE函数变得简单。

## 第116.1节：使用标准dplyr动词的示例

非标准求值函数应在交互式编程中使用。然而，在开发新包中的新函数时，最好使用标准求值版本。

加载dplyr和lazyeval：

```
library(dplyr)
library(lazyeval)
```

### 过滤

非标准求值版本

```
filter(mtcars, cyl == 8)
filter(mtcars, cyl < 6)
filter(mtcars, cyl < 6 & vs == 1)
```

SE 版本（用于在新包中编写函数时使用）

```
filter_(mtcars, .dots = list(~ cyl == 8))
filter_(mtcars, .dots = list(~ cyl < 6))
filter_(mtcars, .dots = list(~ cyl < 6, ~ vs == 1))
```

### 汇总

非标准求值版本

```
summarise(mtcars, mean(disp))
summarise(mtcars, mean_disp = mean(disp))
```

SE 版本

```
summarise_(mtcars, .dots = lazeval::interp(~ mean(x), x = quote(disp)))
summarise_(mtcars, .dots = setNames(list(lazeval::interp(~ mean(x), x = quote(disp))), "mean_disp"))
summarise_(mtcars, .dots = list("mean_disp" = lazeval::interp(~ mean(x), x = quote(disp))))
```

### 变换

非标准求值版本

```
mutate(mtcars, displ_l = disp / 61.0237)
```

# Chapter 116: Non-standard evaluation and standard evaluation

Dplyr and many modern libraries in R use non-standard evaluation (NSE) for interactive programming and standard evaluation (SE) for programming<sup>1</sup>.

For instance, the summarise() function use non-standard evaluation but relies on the summarise\_() which uses standard evaluation.

The lazeval library makes it easy to turn standard evaluation function into NSE functions.

## Section 116.1: Examples with standard dplyr verbs

NSE functions should be used in interactive programming. However, when developing new functions in a new package, it's better to use SE version.

Load dplyr and lazeval :

```
library(dplyr)
library(lazyeval)
```

### Filtering

NSE version

```
filter(mtcars, cyl == 8)
filter(mtcars, cyl < 6)
filter(mtcars, cyl < 6 & vs == 1)
```

SE version (to be use when programming functions in a new package)

```
filter_(mtcars, .dots = list(~ cyl == 8))
filter_(mtcars, .dots = list(~ cyl < 6))
filter_(mtcars, .dots = list(~ cyl < 6, ~ vs == 1))
```

### Summarise

NSE version

```
summarise(mtcars, mean(disp))
summarise(mtcars, mean_disp = mean(disp))
```

SE version

```
summarise_(mtcars, .dots = lazeval::interp(~ mean(x), x = quote(disp)))
summarise_(mtcars, .dots = setNames(list(lazeval::interp(~ mean(x), x = quote(disp))), "mean_disp"))
summarise_(mtcars, .dots = list("mean_disp" = lazeval::interp(~ mean(x), x = quote(disp))))
```

### Mutate

NSE version

```
mutate(mtcars, displ_l = disp / 61.0237)
```

```
mutate_(
 .data = mtcars,
 .dots = list(
 "displ_l" = lazyeval::interp(
 ~ x / 61.0237, x = quote(disp)
)
)
)
```

```
mutate_(
 .data = mtcars,
 .dots = list(
 "displ_l" = lazyeval::interp(
 ~ x / 61.0237, x = quote(disp)
)
)
)
```

# 第117章：随机化

R语言常用于统计分析。因此，它包含了一套强大的随机化选项。有关从概率分布中抽样的具体信息，请参阅分布函数的文档。

## 第117.1节：随机抽样和排列

`sample` 命令可用于模拟经典概率问题，如有放回和无放回的抽取，或生成随机排列。

请注意，在本示例中始终使用 `set.seed` 以确保示例代码可复现。然而，`sample` 即使不显式调用 `set.seed` 也能正常工作。

### 随机排列

在最简单的形式中，`sample` 会创建一个整数向量的随机排列。这可以通过以下方式实现：

```
set.seed(1251)
sample(x = 10)

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

当没有给出其他参数时，`sample` 会返回从1到x的向量的随机排列。这在尝试随机化数据框中行的顺序时非常有用。这是创建试验随机化表或选择用于分析的随机子集行时的常见任务。

用于试验的随机化表，或选择用于分析的随机子集行时，这是一个常见任务。

```
library(datasets)
set.seed(1171)
iris_rand <- iris[sample(x = 1:nrow(iris)),]

> head(iris)
 Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1 5.1 3.5 1.4 0.2 setosa
2 4.9 3.0 1.4 0.2 setosa
3 4.7 3.2 1.3 0.2 setosa
4 4.6 3.1 1.5 0.2 山鸢尾
5 5.0 3.6 1.4 0.2 山鸢尾
6 5.4 3.9 1.7 0.4 山鸢尾

> head(iris_rand)
 萼片长度 萼片宽度 花瓣长度 花瓣宽度 物种
145 6.7 3.3 5.7 2.5 维吉尼卡
5 5.0 3.6 1.4 0.2 山鸢尾
85 5.4 3.0 4.5 1.5 变色鸢尾
137 6.3 3.4 5.6 2.4 维吉尼卡
128 6.1 3.0 4.9 1.8 维吉尼卡
105 6.5 3.0 5.8 2.2 维吉尼卡
```

### 无放回抽样

使用`sample`，我们也可以模拟有放回和无放回的抽样。要进行无放回抽样（默认），必须向`sample`提供一个抽样集合和抽样次数。抽样集合以向量形式给出。

```
set.seed(7043)
sample(x = LETTERS, size = 7)
```

# Chapter 117: Randomization

The R language is commonly used for statistical analysis. As such, it contains a robust set of options for randomization. For specific information on sampling from probability distributions, see the documentation for distribution functions.

## Section 117.1: Random draws and permutations

The `sample` command can be used to simulate classic probability problems like drawing from an urn with and without replacement, or creating random permutations.

Note that throughout this example, `set.seed` is used to ensure that the example code is reproducible. However, `sample` will work without explicitly calling `set.seed`.

### Random permutation

In the simplest form, `sample` creates a random permutation of a vector of integers. This can be accomplished with:

```
set.seed(1251)
sample(x = 10)

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

When given no other arguments, `sample` returns a random permutation of the vector from 1 to x. This can be useful when trying to randomize the order of the rows in a data frame. This is a common task when creating randomization tables for trials, or when selecting a random subset of rows for analysis.

```
library(datasets)
set.seed(1171)
iris_rand <- iris[sample(x = 1:nrow(iris)),]

> head(iris)
 Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1 5.1 3.5 1.4 0.2 setosa
2 4.9 3.0 1.4 0.2 setosa
3 4.7 3.2 1.3 0.2 setosa
4 4.6 3.1 1.5 0.2 setosa
5 5.0 3.6 1.4 0.2 setosa
6 5.4 3.9 1.7 0.4 山鸢尾

> head(iris_rand)
 Sepal.Length Sepal.Width Petal.Length Petal.Width Species
145 6.7 3.3 5.7 2.5 virginica
5 5.0 3.6 1.4 0.2 setosa
85 5.4 3.0 4.5 1.5 versicolor
137 6.3 3.4 5.6 2.4 virginica
128 6.1 3.0 4.9 1.8 virginica
105 6.5 3.0 5.8 2.2 virginica
```

### Draws without Replacement

Using `sample`, we can also simulate drawing from a set with and without replacement. To sample without replacement (the default), you must provide `sample` with a set to be drawn from and the number of draws. The set to be drawn from is given as a vector.

```
set.seed(7043)
sample(x = LETTERS, size = 7)
```

```
[1] "S" "P" "J" "F" "Z" "G" "R"
```

请注意，如果 `size` 的参数与 `x` 的参数长度相同，则您正在创建一个随机排列。还要注意，在不放回抽样时，不能指定大于 `x` 长度的大小。

```
set.seed(7305)
sample(x = letters, size = 26)
```

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

```
sample(x = letters, size = 30)
Error in sample.int(length(x), size, replace, prob) :
当'replace = FALSE'时，样本大小不能超过总体大小
```

这就引出了有放回抽样。

## 有放回抽样

要从一个集合中进行有放回的随机抽取，可以使用 `replace` 参数来调用 `sample`。默认情况下，`replace` 是 `FALSE`。将其设置为 `TRUE` 意味着被抽取的集合中的每个元素在最终结果中可能出现多次。

```
set.seed(5062)
sample(x = c("A", "B", "C", "D"), size = 8, replace = TRUE)
```

```
[1] "D" "C" "D" "B" "A" "A" "A" "A"
```

## 更改抽取概率

默认情况下，当你使用 `sample` 时，它假设每个元素被选中的概率相同。可以将其视为一个基本的“抽签”问题。下面的代码相当于从一个罐子中抽取彩色弹珠 20 次，记录颜色，然后将弹珠放回罐子。罐子里有一个红色、一个蓝色和一个绿色弹珠，这意味着抽中每种颜色的概率都是  $1/3$ 。

```
set.seed(6472)
sample(x = c("Red", "Blue", "Green"),
 size = 20,
 replace = TRUE)
```

假设我们想执行相同任务，但罐子里有 2 个红色弹珠、1 个蓝色弹珠和 1 个绿色弹珠。一种方法是更改传递给 `x` 的参数，添加一个额外的 `Red`。然而，更好的选择是使用 `sample` 的 `prob` 参数。

`prob` 参数接受一个向量，表示抽取每个元素的概率。在上面的例子中，抽中红色弹珠的概率是  $1/2$ ，而抽中蓝色或绿色弹珠的概率是  $1/4$ 。

```
set.seed(28432)
sample(x = c("Red", "Blue", "Green"),
 size = 20,
 replace = TRUE,
 prob = c(0.5, 0.25, 0.25))
```

反直觉的是，传递给 `prob` 的参数不需要加起来等于 1。R 会自动将给定的参数转换为总和为 1 的概率。例如，考虑我们上面提到的 2 个红色、1 个蓝色和 1 个绿色的例子。

```
[1] "S" "P" "J" "F" "Z" "G" "R"
```

Note that if the argument to `size` is the same as the length of the argument to `x`, you are creating a random permutation. Also note that you cannot specify a size greater than the length of `x` when doing sampling without replacement.

```
set.seed(7305)
sample(x = letters, size = 26)
```

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

```
sample(x = letters, size = 30)
Error in sample.int(length(x), size, replace, prob) :
cannot take a sample larger than the population when 'replace = FALSE'
```

This brings us to drawing with replacement.

## Draws with Replacement

To make random draws from a set with replacement, you use the `replace` argument to `sample`. By default, `replace` is `FALSE`. Setting it to `TRUE` means that each element of the set being drawn from may appear more than once in the final result.

```
set.seed(5062)
sample(x = c("A", "B", "C", "D"), size = 8, replace = TRUE)
```

```
[1] "D" "C" "D" "B" "A" "A" "A" "A"
```

## Changing Draw Probabilities

By default, when you use `sample`, it assumes that the probability of picking each element is the same. Consider it as a basic “urn” problem. The code below is equivalent to drawing a colored marble out of an urn 20 times, writing down the color, and then putting the marble back in the urn. The urn contains one red, one blue, and one green marble, meaning that the probability of drawing each color is  $1/3$ .

```
set.seed(6472)
sample(x = c("Red", "Blue", "Green"),
 size = 20,
 replace = TRUE)
```

Suppose that, instead, we wanted to perform the same task, but our urn contains 2 red marbles, 1 blue marble, and 1 green marble. One option would be to change the argument we send to `x` to add an additional `Red`. However, a better choice is to use the `prob` argument to `sample`.

The `prob` argument accepts a vector with the probability of drawing each element. In our example above, the probability of drawing a red marble would be  $1/2$ , while the probability of drawing a blue or a green marble would be  $1/4$ .

```
set.seed(28432)
sample(x = c("Red", "Blue", "Green"),
 size = 20,
 replace = TRUE,
 prob = c(0.5, 0.25, 0.25))
```

Counter-intuitively, the argument given to `prob` does not need to sum to 1. R will always transform the given arguments into probabilities that total to 1. For instance, consider our above example of 2 Red, 1 Blue, and 1 Green.

你可以使用这些数字实现与我们之前代码相同的结果：

```
set.seed(28432)
frac_prob_example <- sample(x = c("Red", "Blue", "Green"),
 size = 200,
 replace = TRUE,
prob = c(0.50, 0.25, 0.25))

set.seed(28432)
numeric_prob_example <- sample(x = c("Red", "Blue", "Green"),
 size = 200,
 replace = TRUE,
prob = c(2, 1, 1))

> identical(frac_prob_example, numeric_prob_example)
[1] TRUE
```

主要限制是你不能将所有概率都设为零，且任何概率都不能小于零。

当replace设置为FALSE时，也可以使用prob。在这种情况下，每次抽取一个元素后，剩余元素的prob值的比例决定下一次抽取的概率。在这种情况下，你必须有足够的非零概率以达到你抽样的 size。例如：

```
set.seed(21741)
sample(x = c("Red", "Blue", "Green"),
 size = 2,
 replace = FALSE,
prob = c(0.8, 0.19, 0.01))
```

在这个例子中，红色在第一次抽取中被抽中（作为第一个元素）。红色被抽中的概率是80%，蓝色被抽中的概率是19%，绿色被抽中的概率是1%。

在下一次抽取中，红色不再在罐子里。剩余物品的概率总和是20%（蓝色19%，绿色1%）。在这次抽取中，抽中蓝色的概率是95%（19/20），抽中绿色的概率是5%（1/20）。

## 第117.2节：设置随机种子

set.seed函数用于为所有随机化函数设置随机种子。如果你使用R进行随机化，并希望能够复现结果，应首先使用set.seed。

```
set.seed(1643)
samp1 <- sample(x = 1:5, size = 200, replace = TRUE)

set.seed(1643)
samp2 <- sample(x = 1:5, size = 200, replace = TRUE)

> identical(x = samp1, y = samp2)
[1] TRUE
```

请注意，使用并行处理时需要对随机种子进行特殊处理，相关内容在其他地方有详细说明。

You can achieve the same results as our previous code using those numbers:

```
set.seed(28432)
frac_prob_example <- sample(x = c("Red", "Blue", "Green"),
 size = 200,
 replace = TRUE,
prob = c(0.50, 0.25, 0.25))

set.seed(28432)
numeric_prob_example <- sample(x = c("Red", "Blue", "Green"),
 size = 200,
 replace = TRUE,
prob = c(2, 1, 1))

> identical(frac_prob_example, numeric_prob_example)
[1] TRUE
```

The major restriction is that you cannot set all the probabilities to be zero, and none of them can be less than zero.

You can also utilize prob when replace is set to FALSE. In that situation, after each element is drawn, the proportions of the prob values for the remaining elements give the probability for the next draw. In this situation, you must have enough non-zero probabilities to reach the size of the sample you are drawing. For example:

```
set.seed(21741)
sample(x = c("Red", "Blue", "Green"),
 size = 2,
 replace = FALSE,
prob = c(0.8, 0.19, 0.01))
```

In this example, Red is drawn in the first draw (as the first element). There was an 80% chance of Red being drawn, a 19% chance of Blue being drawn, and a 1% chance of Green being drawn.

For the next draw, Red is no longer in the urn. The total of the probabilities among the remaining items is 20% (19% for Blue and 1% for Green). For that draw, there is a 95% chance the item will be Blue (19/20) and a 5% chance it will be Green (1/20).

## Section 117.2: Setting the seed

The **set.seed** function is used to set the random seed for all randomization functions. If you are using R to create a randomization that you want to be able to reproduce, you should use **set.seed** first.

```
set.seed(1643)
samp1 <- sample(x = 1:5, size = 200, replace = TRUE)

set.seed(1643)
samp2 <- sample(x = 1:5, size = 200, replace = TRUE)

> identical(x = samp1, y = samp2)
[1] TRUE
```

Note that parallel processing requires special treatment of the random seed, described more elsewhere.

# 第118章：R语言中的面向对象编程

本说明文档介绍了R语言中的四种对象系统及其高级相似点和差异。  
关于每个系统的更详细内容可以在其各自的专题页面找到。

这四个系统是：S3、S4、引用类（Reference Classes）和S6。

## 第118.1节：S3

S3对象系统是R中一个非常简单的面向对象系统。

每个对象都有一个S3类。可以用函数class来获取。

```
> class(3)
[1] "numeric"
```

也可以用函数class来设置：

```
> bicycle <- 2
> 类(自行车) <- '车辆'
> 类(自行车)
[1] "vehicle"
```

也可以用函数attr来设置：

```
> velocipede <- 2
> attr(velocipede, 'class') <- 'vehicle'
> class(velocipede)
[1] "vehicle"
```

一个对象可以有多个类：

```
> class(x = bicycle) <- c('人力车辆', class(x = bicycle))
> class(x = bicycle)
[1] "人力车辆" "vehicle"
```

使用泛型函数时，R 会使用类中第一个有可用泛型的元素。

例如：

```
> summary.vehicle <- function(object, ...) {
+ message('这是一个车辆')
+ }
> summary(object = my_bike)
这是 一辆 交通工具
```

但是如果我们现在定义一个 summary.bicycle：

```
> summary.bicycle <- function(object, ...) {
+ message('这是一个自行车')
+ }
> summary(object = my_bike)
这是 一辆 自行车
```

# Chapter 118: Object-Oriented Programming in R

This documentation page describes the four object systems in R and their high-level similarities and differences.  
Greater detail on each individual system can be found on its own topic page.

The four systems are: S3, S4, Reference Classes, and S6.

## Section 118.1: S3

The S3 object system is a very simple OO system in R.

Every object has an S3 class. It can be get (got?) with the function `class`.

```
> class(3)
[1] "numeric"
```

It can also be set with the function `class`:

```
> bicycle <- 2
> class(bicycle) <- 'vehicle'
> class(bicycle)
[1] "vehicle"
```

It can also be set with the function `attr`:

```
> velocipede <- 2
> attr(velocipede, 'class') <- 'vehicle'
> class(velocipede)
[1] "vehicle"
```

An object can have many classes:

```
> class(x = bicycle) <- c('human-powered vehicle', class(x = bicycle))
> class(x = bicycle)
[1] "human-powered vehicle" "vehicle"
```

When using a generic function, R uses the first element of the class that has an available generic.

For example:

```
> summary.vehicle <- function(object, ...) {
+ message('this is a vehicle')
+ }
> summary(object = my_bike)
this is a vehicle
```

But if we now define a `summary.bicycle`:

```
> summary.bicycle <- function(object, ...) {
+ message('this is a bicycle')
+ }
> summary(object = my_bike)
this is a bicycle
```

# 第119章：强制转换

在R语言中，当对象的类型在计算过程中被隐式改变或通过使用显式强制转换函数（如as.numeric、as.data.frame等）改变时，就会发生强制转换。

## 第119.1节：隐式强制转换

在R语言中，数据类型经常会发生隐式强制转换，以便数据能够容纳所有的值。例如，

```
x = 1:3
x
[1] 1 2 3
typeof(x)
#[1] "integer"

x[2] = "hi"x

#[1] "1" "hi" "3"
typeof(x)
#[1] "character"
```

注意，起初，`x` 的类型是 `integer`。但当我们赋值 `x[2] = "hi"` 时，`x` 的所有元素都被强制转换为 `character`，因为 R 中的向量只能包含单一类型的数据。

# Chapter 119: Coercion

Coercion happens in R when the type of objects are changed during computation either implicitly or by using functions for explicit coercion (such as as.numeric, as.data.frame, etc.).

## Section 119.1: Implicit Coercion

Coercion happens with data types in R, often implicitly, so that the data can accommodate all the values. For example,

```
x = 1:3
x
[1] 1 2 3
typeof(x)
#[1] "integer"

x[2] = "hi"
x
#[1] "1" "hi" "3"
typeof(x)
#[1] "character"
```

Notice that at first, `x` is of type `integer`. But when we assigned `x[2] = "hi"`, all the elements of `x` were coerced into `character` as vectors in R can only hold data of single type.

# 第120章：通过编写独立的 R 脚本来标准化分析

如果你想定期对许多独立的数据文件应用 R 分析，或者向他人提供可重复的分析方法，执行型 R 脚本是一种用户友好的方式。你的用户无需调用 R 并通过 `source()` 或函数调用在 R 内执行脚本，而是可以像调用程序一样直接调用该脚本。

## 第120.1节：独立 R 程序的基本结构及调用方法

### 第一个独立的 R 脚本

独立的 R 脚本不是由程序 R (Windows 下为 R.exe) 执行，而是由一个名为 `Rscript` (`Rscript.exe`) 的程序执行，该程序默认包含在你的 R 安装中。

为了提示这一点，独立的R脚本以一行特殊的代码开头，称为**Shebang行**，其内容如下：  
`#!/usr/bin/env Rscript`。在Windows系统下，还需要额外的措施，后面会详细说明。

下面这个简单的独立R脚本将接收到的数字保存为名为“hist.png”的直方图文件：

```
#!/usr/bin/env Rscript# 用

户提示信息 (= 换行)
cat("请输入数字，数字之间用空格分隔：")# 将用户输入
作为一个字符串读取 (n=1 -> 只读取一行) input <- readLines(file('stdin'), n
=1)
在每个空格处拆分字符串 (\s == 任意空白字符)
input <- strsplit(input, "\s")[[1]]
将得到的字符串向量转换为数字
input <- as.numeric(input)

打开输出图片文件
png("hist.png",width=400, height=300)
绘制直方图
hist(input)
关闭输出文件
dev.off()
```

您可以看到一个独立R脚本的几个关键元素。第一行是Shebang行。紧接着，`cat("....")` 用于向用户打印信息。每当您想指定“控制台上的用户输入”作为数据来源时，请使用`file("stdin")`。它可以替代多个数据读取函数中的文件名（如 `scan`、`read.table`、`read.csv`等）。在用户输入从字符串转换为数字后，绘图开始。在那里，可以看到，打算写入文件的绘图命令必须被两个命令包围。

在这种情况下，这些是`png()`和`dev.off()`。第一个函数取决于所需的输出文件格式（其他常见选择包括`jpeg()`和`pdf()`）。第二个函数`dev.off()`始终是必需的。它将绘图写入文件并结束绘图过程。

### 准备一个独立的R脚本

#### Linux/Mac

独立脚本文件必须首先设置为可执行。可以通过右键点击文件，在弹出菜单中打开“属性”，然后在“权限”标签页中勾选“可执行”复选框来实现。或者，

# Chapter 120: Standardize analyses by writing standalone R scripts

If you want to routinely apply an R analysis to a lot of separate data files, or provide a repeatable analysis method to other people, an executable R script is a user-friendly way to do so. Instead of you or your user having to call R and execute your script inside R via `source( . )` or a function call, your user may simply call the script itself as if it was a program.

## Section 120.1: The basic structure of standalone R program and how to call it

### The first standalone R script

Standalone R scripts are not executed by the program R (R.exe under Windows), but by a program called `Rscript` (`Rscript.exe`), which is included in your R installation by default.

To hint at this fact, standalone R scripts start with a special line called **Shebang line**, which holds the following content: `#!/usr/bin/env Rscript`. Under Windows, an additional measure is needed, which is detailed later.

The following simple standalone R script saves a histogram under the file name "hist.png" from numbers it receives as input:

```
#!/usr/bin/env Rscript

User message (\n = end the line)
cat("Input numbers, separated by space:\n")
Read user input as one string (n=1 -> Read only one line)
input <- readLines(file('stdin'), n=1)
Split the string at each space (\s == any space)
input <- strsplit(input, "\s")[[1]]
convert the obtained vector of strings to numbers
input <- as.numeric(input)

Open the output picture file
png("hist.png",width=400, height=300)
Draw the histogram
hist(input)
Close the output file
dev.off()
```

You can see several key elements of a standalone R script. In the first line, you see the Shebang line. Followed by that, `cat("....\n")` is used to print a message to the user. Use `file("stdin")` whenever you want to specify "User input on console" as a data origin. This can be used instead of a file name in several data reading functions (`scan`, `read.table`, `read.csv`,...). After the user input is converted from strings to numbers, the plotting begins. There, it can be seen, that plotting commands which are meant to be written to a file must be enclosed in two commands. These are in this case `png()` and `dev.off()`. The first function depends on the desired output file format (other common choices being `jpeg()` and `pdf()`). The second function, `dev.off()` is always required. It writes the plot to the file and ends the plotting process.

### Preparing a standalone R script

#### Linux/Mac

The standalone script's file must first be made executable. This can happen by right-clicking the file, opening "Properties" in the opening menu and checking the "Executable" checkbox in the "Permissions" tab. Alternatively,

```
chmod +x PATH/TO/SCRIPT/SCRIPTNAME.R
```

。

## Windows

对于每个独立脚本，必须编写一个包含以下内容的批处理文件：

```
"C:\Program Files\RXXXXXXX\bin\Rscript.exe" "%~dp0\XXXXXX.R" %*
```

批处理文件是普通文本文件，但扩展名为\*.bat而非\*.txt。使用文本编辑器（如notepad，而非Word）创建，并在保存对话框中将文件名用引号括起来 ("FILENAME.bat")。要编辑现有批处理文件，右键点击它并选择“编辑”。

你必须在所有写有 XXX... 的地方适配上面显示的代码：

- 插入你的 R 安装所在的正确文件夹
- 插入你的脚本的正确名称，并将其放置在与此批处理文件相同的目录中。

代码元素说明：第一部分 "C:\...\Rscript.exe" 告诉 Windows 在哪里找到 Rscript.exe 程序。第二部分 "%~dp0\XXX.R" 告诉 Rscript 执行你编写的 R 脚本，该脚本位于与批处理文件相同的文件夹中 (%~dp0 代表批处理文件所在文件夹)。最后，%\* 将你给批处理文件的任何命令行参数传递给 R 脚本。

如果你双击批处理文件，R 脚本将被执行。如果你将文件拖到批处理文件上，相应的文件名将作为命令行参数传递给 R 脚本。

## 第120.2节：使用 littler 执行 R 脚本

`littler` (发音为 little r) ([cran](#)) 除了其他功能外，还提供了两种通过 littler 的 `r` 命令从命令行运行 R 脚本的方式 (适用于 Linux 或 MacOS)。

### 安装 littler

从 R 终端：

```
install.packages("littler")
```

`r` 的路径会在终端打印出来，例如

```
你可以将安装在 '/home/*USER*/R/x86_64-pc-linux-gnu-library/3.4/littler/bin/r' 中的 'r' 二进制文件链接到 '/usr/local/bin'，以便使用 'r' 进行脚本编写。
```

要能够从系统命令行调用 `r`，需要一个符号链接：

```
ln -s /home/*USER*/R/x86_64-pc-linux-gnu-library/3.4/littler/bin/r /usr/local/bin/r
```

使用 `apt-get` (Debian, Ubuntu)：

```
sudo apt-get install littler
```

### 使用 littler 运行标准 .r 脚本

使用来自 littler 的 `r`，可以在不修改脚本的情况下执行独立的 R 脚本。示例脚本：

the command

```
chmod +x PATH/TO/SCRIPT/SCRIPTNAME.R
```

can be called in a Terminal.

## Windows

For each standalone script, a batch file must be written with the following contents:

```
"C:\Program Files\R-XXXXXX\bin\Rscript.exe" "%~dp0\XXXXXX.R" %*
```

A batch file is a normal text file, but which has a \*.bat extension except a \*.txt extension. Create it using a text editor like notepad (not Word) or similar and put the file name into quotation marks "[FILENAME.bat](#)") in the save dialog. To edit an existing batch file, right-click on it and select "Edit".

You have to adapt the code shown above everywhere XXX... is written:

- Insert the correct folder where your R installation resides
- Insert the correct name of your script and place it into the same directory as this batch file.

Explanation of the elements in the code: The first part "C:\...\Rscript.exe" tells Windows where to find the Rscript.exe program. The second part "%~dp0\XXX.R" tells Rscript to execute the R script you've written which resides in the same folder as the batch file (%~dp0 stands for the batch file folder). Finally, %\* forwards any command line arguments you give to the batch file to the R script.

If you double-click on the batch file, the R script is executed. If you drag files on the batch file, the corresponding file names are given to the R script as command line arguments.

## Section 120.2: Using littler to execute R scripts

`littler` (pronounced little r) ([cran](#)) provides, besides other features, two possibilities to run R scripts from the command line with littler's `r` command (when one works with Linux or MacOS).

### Installing littler

From R:

```
install.packages("littler")
```

The path of `r` is printed in the terminal, like

```
You could link to the 'r' binary installed in
'/home/*USER*/R/x86_64-pc-linux-gnu-library/3.4/littler/bin/r'
from '/usr/local/bin' in order to use 'r' for scripting.
```

To be able to call `r` from the system's command line, a symlink is needed:

```
ln -s /home/*USER*/R/x86_64-pc-linux-gnu-library/3.4/littler/bin/r /usr/local/bin/r
```

Using `apt-get` (Debian, Ubuntu):

```
sudo apt-get install littler
```

### Using littler with standard .r scripts

With `r` from littler it is possible to execute standalone R scripts without any changes to the script. Example script:

```
用户消息 (= 换行)
cat("请输入数字，数字之间用空格分隔：")# 将用户输入
作为一个字符串读取 (n=1 -> 只读取一行) input <- readLines(file('stdin'), n
=1)
在每个空格处拆分字符串 (\s == 任意空白字符)
input <- strsplit(input, "\s")[[1]]
将得到的字符串向量转换为数字
input <- as.numeric(input)

打开输出图片文件
png("hist.png",width=400, height=300)
绘制直方图
hist(input)
关闭输出文件
dev.off()
```

注意脚本顶部没有 shebang。保存为例如 hist.r 后，可以直接通过系统命令调用：

```
r hist.r
```

#### 在带有 shebang 的脚本中使用 littler

也可以使用 littler 创建可执行的 R 脚本，方法是在脚本顶部使用 shebang

```
#!/usr/bin/env r
```

。相应的 R 脚本必须通过 chmod +X /path/to/script.r 命令设置为可执行，并且可以直接从系统终端调用。

```
User message (\n = end the line)
cat("Input numbers, separated by space:\n")
Read user input as one string (n=1 -> Read only one line)
input <- readLines(file('stdin'), n=1)
Split the string at each space (\s == any space)
input <- strsplit(input, "\s")[[1]]
convert the obtained vector of strings to numbers
input <- as.numeric(input)

Open the output picture file
png("hist.png",width=400, height=300)
Draw the histogram
hist(input)
Close the output file
dev.off()
```

Note that no shebang is at the top of the scripts. When saved as for example hist.r, it is directly callable from the system command:

```
r hist.r
```

#### Using littler on *shebanged* scripts

It is also possible to create executable R scripts with littler, with the use of the shebang

```
#!/usr/bin/env r
```

at the top of the script. The corresponding R script has to be made executable with chmod +X /path/to/script.r and is directly callable from the system terminal.

# 第121章：使用 R 分析推文

(可选) 每个主题都有一个重点。告诉读者他们将在这里找到什么，并让未来的贡献者知道什么内容属于此处。

## 第121.1节：下载推文

你首先需要做的是下载推文。你需要设置你的推特账户。网上有很多关于如何操作的信息。以下两个链接对我的设置很有帮助（最后检查时间为2017年5月）

特别是我发现以下两个链接很有用（最后检查时间为2017年5月）：

[链接1](#)

[链接2](#)

### R 库

您将需要以下R包

```
library("devtools")
library("twitteR")
library("ROAuth")
```

假设您已经有了密钥，您需要运行以下代码

```
api_key <- XXXXXXXXXXXXXXXXXXXXXXXX
api_secret <- XXXXXXXXXXXXXXXXXXXXXXXX
access_token <- XXXXXXXXXXXXXXXXXXXXXXXX
access_token_secret <- XXXXXXXXXXXXXXXXXXXXXXXX

setup_twitter_oauth(api_key,api_secret)
```

将XXXXXXXXXXXXXXXXXXXX替换为您的密钥（如果您已经设置了推特账户，您就知道我指的是哪些密钥）。

现在假设我们想下载关于咖啡的推文。以下代码将实现此功能

```
search.string <- "#coffee"
no.of.tweets <- 1000

c_tweets <- searchTwitter(search.string, n=no.of.tweets, lang="en")
```

您将获得1000条关于“coffee”的推文。

## 第121.2节：获取推文文本

现在我们需要访问推文的文本。我们这样做（同时需要用sapply函数清理推文中的特殊字符，这些字符目前我们不需要，比如表情符号）。

```
coffee_tweets = sapply(c_tweets, function(t) t$text)
```

# Chapter 121: Analyze tweets with R

(Optional) Every topic has a focus. Tell the readers what they will find here and let future contributors know what belongs.

## Section 121.1: Download Tweets

The first think you need to do is to download tweets. You need to Setup your tweeter account. Much Information can be found in Internet on how to do it. The following two links were useful for my Setup (last checked in May 2017)

In particular I found the following two links useful (last checked in May 2017):

[Link 1](#)

[Link 2](#)

### R Libraries

You will need the following R packages

```
library("devtools")
library("twitteR")
library("ROAuth")
```

Supposing you have your keys You have to run the following code

```
api_key <- XXXXXXXXXXXXXXXXXXXXXXXX
api_secret <- XXXXXXXXXXXXXXXXXXXXXXXX
access_token <- XXXXXXXXXXXXXXXXXXXXXXXX
access_token_secret <- XXXXXXXXXXXXXXXXXXXXXXXX
```

```
setup_twitter_oauth(api_key,api_secret)
```

Change XXXXXXXXXXXXXXXXXXXXXXXX to your keys (if you have Setup your tweeter account you know which keys I mean).

Let's now suppose we want to download tweets on coffee. The following code will do it

```
search.string <- "#coffee"
no.of.tweets <- 1000

c_tweets <- searchTwitter(search.string, n=no.of.tweets, lang="en")
```

You will get 1000 tweets on "coffee".

## Section 121.2: Get text of tweets

Now we need to access the text of the tweets. So we do it in this way (we also need to clean up the tweets from special characters that for now we don't need, like emoticons with the sapply function.)

```
coffee_tweets = sapply(c_tweets, function(t) t$text)
```

```
coffee_tweets <- sapply(coffee_tweets,function(row) iconv(row, "latin1", "ASCII", sub=""))
```

你可以用head函数查看你的推文。

```
head(coffee_tweets)
```

```
coffee_tweets <- sapply(coffee_tweets,function(row) iconv(row, "latin1", "ASCII", sub=""))
```

and you can check your tweets with the **head** function.

```
head(coffee_tweets)
```

# 第122章：自然语言处理

自然语言处理（NLP）是计算机科学领域，专注于从人类生成的文本输入中提取信息。

## 第122.1节：创建词频矩阵

解决该问题最简单的方法（也是迄今为止最常用的方法）是将句子拆分成词元。

简化来说，**单词**对使用和接收它们的人具有抽象和主观的含义，**标记** (*tokens*) 则有一个客观的解释：一个有序的字符（或字节）序列。一旦句子被拆分，标记的顺序就被忽略了。这种处理问题的方法被称为**词袋** (**bag of words**) 模型。

一个词频 (term frequency) 是一个字典，其中每个标记被分配一个权重 (weight)。在第一个例子中，我们使用R包m从语料库 (corpus, 一组文档 (documents)) 构建了一个词频矩阵。

```
require(tm)
doc1 <- "drugs hospitals doctors"
doc2 <- "smog pollution environment"
doc3 <- "doctors hospitals healthcare"
doc4 <- "pollution environment water"
corpus <- c(doc1, doc2, doc3, doc4)
tm_corpus <- Corpus(VectorSource(corpus))
```

在这个例子中，我们用包m定义的类Corpus创建了一个语料库，使用了两个函数Corpus和VectorSource，后者从字符串向量返回一个VectorSource对象。对象tm\_corpus是一个包含我们文档的列表，并附带了描述每个文档的额外（可选）元数据。

```
str(tm_corpus)
#> 长度为4的列表
#> $ 1:包含2个元素的列表
#> ..$ content:字符型 "drugs hospitals doctors"
#> ..$ meta :包含7个元素的列表
#> ..$ author :字符型(0)
#> ..$ timestamp:POSIXlt[1:1], 格式: "2017-06-03 00:31:34"
#> ..$ description :字符型(0)
#> ..$ heading : chr(0)
#> ..$ id : chr "1"
#> ..$ language : chr "en"
#> ..$ origin : chr(0)
#> ..- attr(*, "class")= chr "TextDocumentMeta"
#> ..- attr(*, "class")= chr [1:2] "PlainTextDocument" "TextDocument"
#> [truncated]
```

一旦我们有了一个Corpus，我们就可以继续对Corpus中包含的标记进行预处理，以提高最终输出（词频矩阵）的质量。为此，我们使用tm\_map函数，该函数类似于apply系列函数，通过对语料库中的每个文档应用一个函数来转换文档。

```
tm_corpus <- tm_map(tm_corpus, tolower)
tm_corpus <- tm_map(tm_corpus, removeWords, stopwords("english"))
tm_corpus <- tm_map(tm_corpus, removeNumbers)
tm_corpus <- tm_map(tm_corpus, PlainTextDocument)
tm_corpus <- tm_map(tm_corpus, stemDocument, language="english")
tm_corpus <- tm_map(tm_corpus, stripWhitespace)
tm_corpus <- tm_map(tm_corpus, PlainTextDocument)
```

完成这些转换后，我们最终使用以下命令创建词频矩阵

# Chapter 122: Natural language processing

Natural language processing (NLP) is the field of computer sciences focused on retrieving information from textual input generated by human beings.

## Section 122.1: Create a term frequency matrix

The simplest approach to the problem (and the most commonly used so far) is to split sentences into *tokens*.

Simplifying, **words** have abstract and subjective meanings to the people using and receiving them, **tokens** have an objective interpretation: an ordered sequence of characters (or bytes). Once sentences are split, the order of the token is disregarded. This approach to the problem is known as **bag of words** model.

A **term frequency** is a dictionary, in which to each token is assigned a **weight**. In the first example, we construct a term frequency matrix from a corpus **corpus** (a collection of **documents**) with the R package **tm**.

```
require(tm)
doc1 <- "drugs hospitals doctors"
doc2 <- "smog pollution environment"
doc3 <- "doctors hospitals healthcare"
doc4 <- "pollution environment water"
corpus <- c(doc1, doc2, doc3, doc4)
tm_corpus <- Corpus(VectorSource(corpus))
```

In this example, we created a corpus of class Corpus defined by the package tm with two functions Corpus and VectorSource, which returns a VectorSource object from a character vector. The object tm\_corpus is a list our documents with additional (and optional) metadata to describe each document.

```
str(tm_corpus)
#> List of 4
#> $ 1:List of 2
#> ..$ content: chr "drugs hospitals doctors"
#> ..$ meta :List of 7
#> ..$ author : chr(0)
#> ..$ timestamp: POSIXlt[1:1], format: "2017-06-03 00:31:34"
#> ..$ description : chr(0)
#> ..$ heading : chr(0)
#> ..$ id : chr "1"
#> ..$ language : chr "en"
#> ..$ origin : chr(0)
#> ..- attr(*, "class")= chr "TextDocumentMeta"
#> ..- attr(*, "class")= chr [1:2] "PlainTextDocument" "TextDocument"
#> [truncated]
```

Once we have a Corpus, we can proceed to preprocess the tokens contained in the Corpus to improve the quality of the final output (the term frequency matrix). To do this we use the tm function tm\_map, which similarly to the apply family of functions, transform the documents in the corpus by applying a function to each document.

```
tm_corpus <- tm_map(tm_corpus, tolower)
tm_corpus <- tm_map(tm_corpus, removeWords, stopwords("english"))
tm_corpus <- tm_map(tm_corpus, removeNumbers)
tm_corpus <- tm_map(tm_corpus, PlainTextDocument)
tm_corpus <- tm_map(tm_corpus, stemDocument, language="english")
tm_corpus <- tm_map(tm_corpus, stripWhitespace)
tm_corpus <- tm_map(tm_corpus, PlainTextDocument)
```

Following these transformations, we finally create the term frequency matrix with

```
tdm <- TermDocumentMatrix(tm_corpus)
```

这将生成一个

```
<<TermDocumentMatrix (terms: 8, documents: 4)>>
非稀疏/稀疏条目 : 12/20
稀疏度 : 62%
最大词长: 9
加权方式 : 词频 (tf)
```

我们可以通过将其转换为矩阵来查看

```
as.matrix(tdm)
```

文档	词汇	character(0)	character(0)	character(0)	character(0)
	doctor	1	0	1	0
	drug	1	0	0	0
	environ	0	1	0	1
	healthcar	0	0	1	0
	hospit	1	0	1	0
	pollut	0	1	0	1
	smog	0	1	0	0
	water	0	0	0	1

每一行表示每个词元的频率——正如你注意到的，这些词元已经被词干化（例如environment变为environ）——在每个文档中（4个文档，4列）。

在前面的几行中，我们用绝对频率（即词元在文档中出现的次数）对每个词元/文档对进行了加权。

```
tdm <- TermDocumentMatrix(tm_corpus)
```

which gives a

```
<<TermDocumentMatrix (terms: 8, documents: 4)>>
Non-/sparse entries: 12/20
Sparsity : 62%
Maximal term length: 9
Weighting : term frequency (tf)
```

that we can view by transforming it to a matrix

```
as.matrix(tdm)
```

Terms	Docs			
	character(0)	character(0)	character(0)	character(0)
doctor	1	0	1	0
drug	1	0	0	0
environ	0	1	0	1
healthcar	0	0	1	0
hospit	1	0	1	0
pollut	0	1	0	1
smog	0	1	0	0
water	0	0	0	1

Each row represents the frequency of each token - that as you noticed have been stemmed (e.g. environment to environ) - in each document (4 documents, 4 columns).

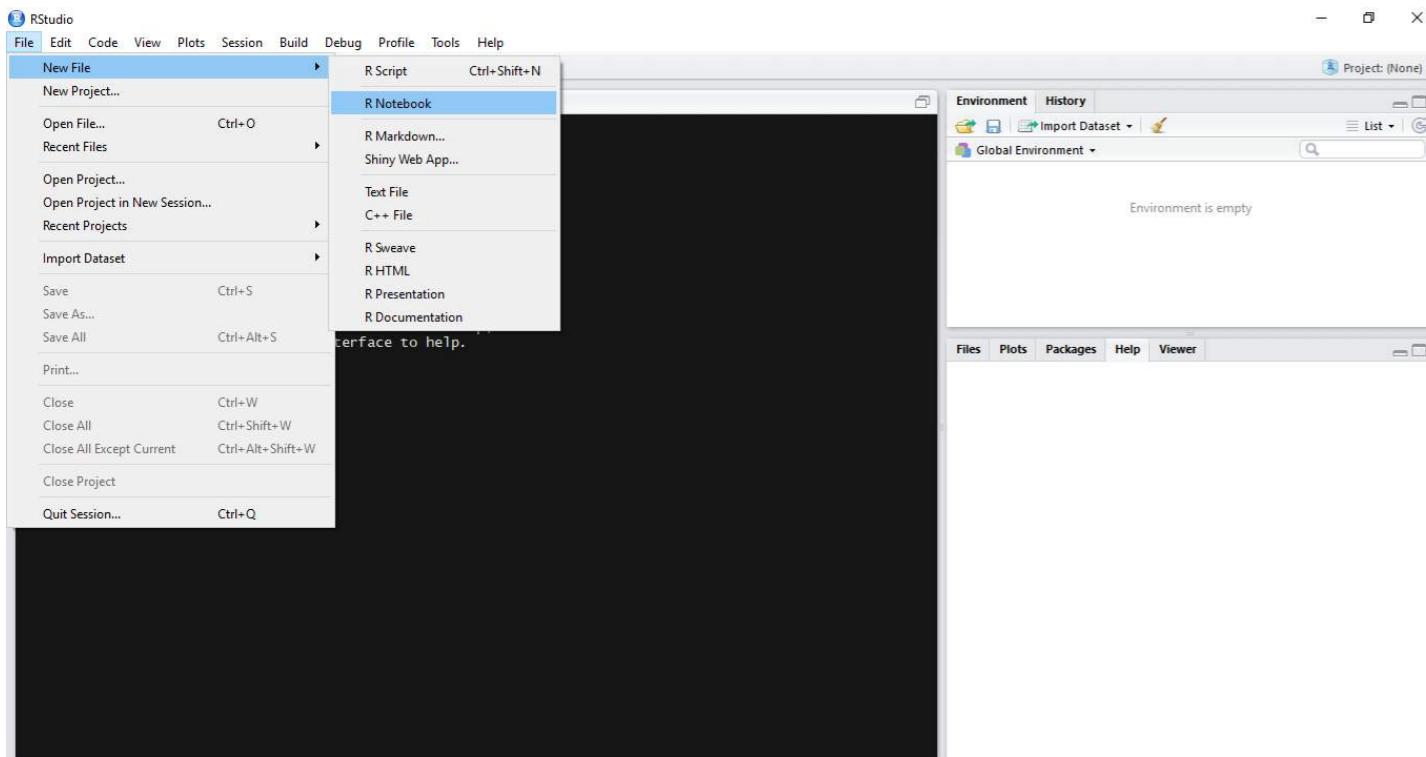
In the previous lines, we have weighted each pair token/document with the absolute frequency (i.e. the number of instances of the token that appear in the document).

# 第123章：R Markdown笔记本（来自RStudio）

R笔记本是一个R Markdown文档，包含可以独立且交互式执行的代码块，输出结果会立即显示在输入代码的下方。它们类似于R Markdown文档，区别在于结果显示在R笔记本的创建/编辑模式中，而不是渲染后的输出中。注意：R笔记本是RStudio的新功能，仅在RStudio 1.0及以上版本中可用。

## 第123.1节：创建笔记本

你可以通过菜单命令 文件 -> 新建文件 -> R笔记本 在RStudio中创建一个新的笔记本。如果你没有看到R笔记本的选项，则需要更新你的RStudio版本。有关RStudio的安装，请参阅此指南

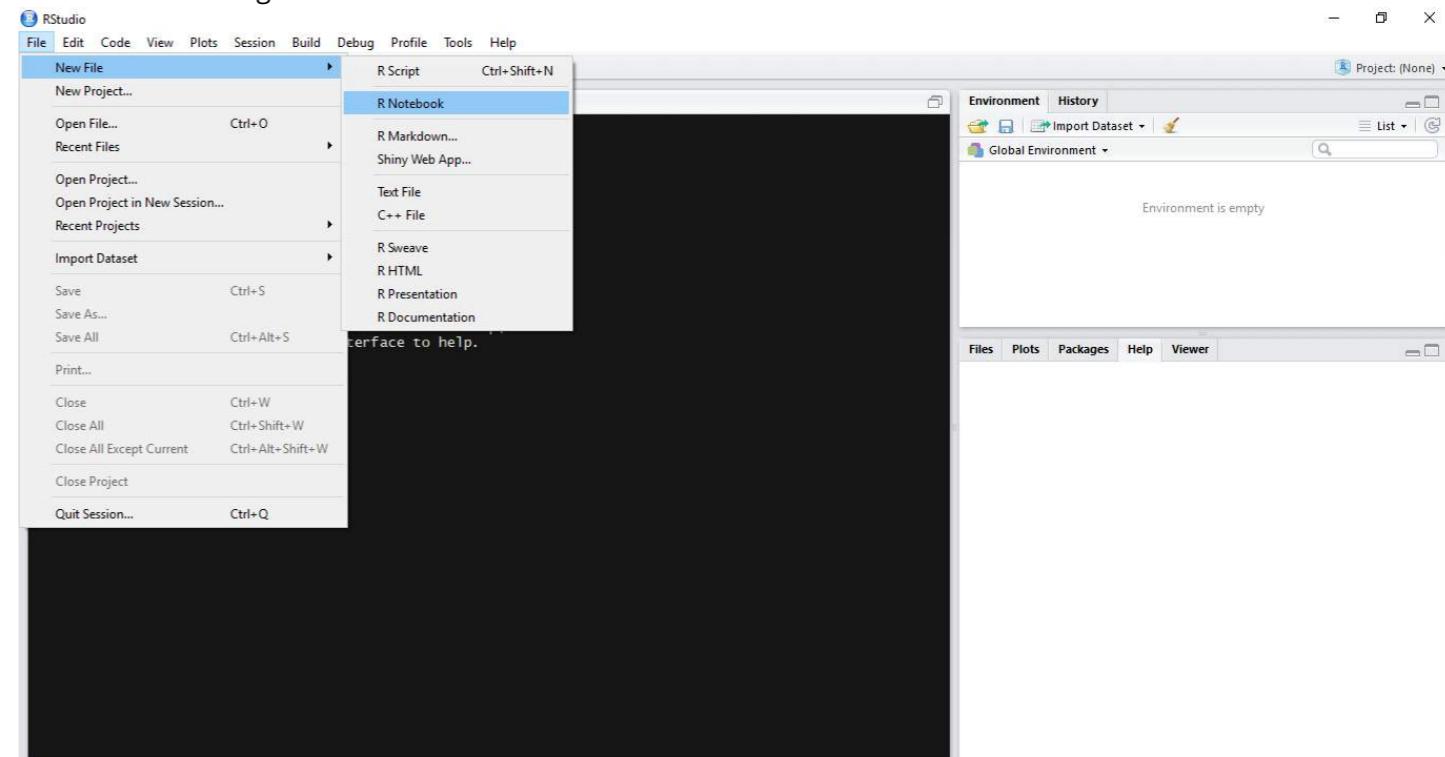


## Chapter 123: R Markdown Notebooks (from RStudio)

An R Notebook is an R Markdown document with chunks that can be executed independently and interactively, with output visible immediately beneath the input. They are similar to R Markdown documents with the exception of results being displayed in the R Notebook creation/edit mode rather than in the rendered output. **Note:** R Notebooks are new feature of RStudio and are only available in version 1.0 or higher of RStudio.

## Section 123.1: Creating a Notebook

You can create a new notebook in RStudio with the menu command File -> New File -> R Notebook. If you don't see the option for R Notebook, then you need to update your version of RStudio. For installation of RStudio follow this guide



## 第123.2节：插入代码块

代码块是可以交互式执行的代码片段。要插入新的代码块，可以点击笔记本工具栏上的插入按钮，并选择你想要的代码平台（此处为R，因为我们要编写R代码）。或者，我们也可以使用键盘快捷键插入新的代码块Ctrl + Alt + I (OS X: Cmd + Option + I)

## Section 123.2: Inserting Chunks

Chunks are pieces of code that can be executed interactively. In-order to insert a new chunk by clicking on the **insert** button present on the notebook toolbar and select your desired code platform (R in this case, since we want to write R code). Alternatively we can use keyboard shortcuts to insert a new chunk **Ctrl + Alt + I (OS X: Cmd + Option + I)**

```

1 * ----
2 title: "R Notebook"
3 output: html_notebook
4 ---
5
6 This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook.
notebook, the results appear beneath the code.
7
8 Try executing this chunk by clicking the *Run* button within the
inside it and pressing *Ctrl+Shift+Enter*.
9
10 ``{r}
11 plot(cars)
12
13 Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.
14
15 When you save the notebook, an HTML file containing the code and output will be saved alongside it
(click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).
16
17

```

17:1 | (Top Level) | R Markdown | Console

Environment History

Global Environment

Environment is empty

Files Plots Packages Help Viewer

Install Update

Name	Description	Version
User Library		
Amelia	A Program for Missing Data	1.7.4
arules	Mining Association Rules and Frequent Itemsets	1.5-2
arulesViz	Visualizing Association Rules and Frequent Itemsets	1.2-1
assertthat	Easy Pre and Post Assertions	0.2.0
backports	Reimplementations of Functions Introduced Since R-3.0.0	1.1.0
base64enc	Tools for base64 encoding	0.1-3
BH	Boost C++ Header Files	1.62.0-1
bindr	Parametrized Active Bindings	0.1
bindrcpp	An 'Rcpp' Interface to Active Bindings	0.2
bitops	Bitwise Operations	1.0-6
broom	Convert Statistical Analysis Objects into Tidy Data Frames	0.4.2
caTools	Tools: moving window statistics, GIF, ROCR, ROC.AUC, etc.	1.17.1

17:1 | (Top Level) | R Markdown | Console

## 第123.3节：执行代码块

你可以点击代码块右侧的运行当前代码块（绿色播放按钮）来运行当前代码块。或者，我们也可以使用键盘快捷键Ctrl + Shift + Enter (OS X: Cmd + Shift + Enter)

代码块中所有行的输出结果将显示在代码块下方。

### 将代码拆分成块

由于代码块的输出显示在代码块下方，当单个代码块中有多行代码产生多个输出时，通常将其拆分为多个代码块，每个代码块产生一个输出会更有帮助。

为此，选择你想拆分成新代码块的代码，然后按Ctrl + Alt + I (OS X: Cmd + Option + I)

```

1 * ----
2 title: "R Notebook"
3 output: html_notebook
4 ---
5
6 This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook.
notebook, the results appear beneath the code.
7
8 Try executing this chunk by clicking the *Run* button within the
inside it and pressing *Ctrl+Shift+Enter*.
9
10 ``{r}
11 plot(cars)
12
13 Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.
14
15 When you save the notebook, an HTML file containing the code and output will be saved alongside it
(click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).
16
17

```

17:1 | (Top Level) | R Markdown | Console

## Section 123.3: Executing Chunk Code

You can run the current chunk by clicking **Run current Chunk (green play button)** present on the right side of the chunk. Alternatively we can use keyboard shortcut **Ctrl + Shift + Enter (OS X: Cmd + Shift + Enter)**

The output from all the lines in the chunk will appear beneath the chunk.

### Splitting Code into Chunks

Since a chunk produces its output beneath the chunk, when having multiple lines of code in a single chunk that produces multiples outputs it is often helpful to split into multiple chunks such that each chunk produces one output.

To do this, select the code to you want to split into a new chunk and press **Ctrl + Alt + I (OS X: Cmd + Option + I)**

```

1 * ----
2 title: "R Notebook"
3 output: html_notebook
4 ---
5
6 This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the
7 notebook, the results appear beneath the code.
8 Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor
9 inside it and pressing *Ctrl+Shift+Enter*.
10
11 plot(cars)
12
13 Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.
14 When you save the notebook, an HTML file containing the code and output will be saved alongside it
15 (click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).
16
17

```

## 第123.4节：执行进度

当你在笔记本中执行代码时，边栏会出现一个指示器显示执行进度。已发送到R的代码行标记为深绿色；尚未发送到R的代码行标记为浅绿色。

### 执行多个代码块

通过按“运行”按钮逐个运行或重新运行文档中的所有代码块可能会很麻烦。

我们可以使用工具栏插入菜单中的运行全部来运行笔记本中所有的代码块。快捷键是Ctrl + Alt + R (OS X : Cmd + Option + R)

还有一个选项重启R并运行所有代码块命令（在编辑器工具栏的运行菜单中可用），它会在运行所有代码块之前启动一个新的R会话。

我们还有运行选中代码块以上所有代码块和运行选中代码块以下所有代码块等选项，用于运行选中代码块上方或下方的代码块。

```

1 * ----
2 title: "R Notebook"
3 output: html_notebook
4 ---
5
6 This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the
7 notebook, the results appear beneath the code.
8 Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor
9 inside it and pressing *Ctrl+Shift+Enter*.
10
11 plot(cars)
12
13 Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.
14 When you save the notebook, an HTML file containing the code and output will be saved alongside it
15 (click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).
16
17

```

## Section 123.4: Execution Progress

When you execute code in a notebook, an indicator will appear in the gutter to show you execution progress. Lines of code which have been sent to R are marked with dark green; lines which have not yet been sent to R are marked with light green.

### Executing Multiple Chunks

Running or Re-Running individual chunks by pressing Run for all the chunks present in a document can be painful. We can use **Run All** from the Insert menu in the toolbar to Run all the chunks present in the notebook. Keyboard shortcut is **Ctrl + Alt + R (OS X: Cmd + Option + R)**

There's also a option **Restart R and Run All Chunks** command (available in the Run menu on the editor toolbar), which gives you a fresh R session prior to running all the chunks.

We also have options like **Run All Chunks Above** and **Run All Chunks Below** to run chunks Above or Below from a selected chunk.

RStudio interface showing the preview output toolbar button highlighted with a red arrow.

```

14 data("iris")
15 head(iris,5)
16 ...
 Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1 5.1 3.5 1.4 0.2 setosa
2 4.9 3.0 1.4 0.2 setosa
3 4.7 3.2 1.3 0.2 setosa
4 4.6 3.1 1.5 0.2 setosa
5 5.0 3.6 1.4 0.2 setosa
 5 rows
17
18 Divide Iris data to x (contain the all features) and y (only the classes)
19 {r}
20 x <- subset(iris, select=-Species)
21 y <- iris$Species
22 ...
23 Create SVM Model and show summary
24 {r}
25 svm_model <- svm(x,y)
26 summary(svm_model)
27 ...
28 Run Prediction
29 {r}
30 pred <- predict(svm_model,x)
31 ...
32 you can time taken by using system.time
33 pred <- predict(svm_model,x)
34 ...
35 pred <- predict(svm_model,x)
36 you can time taken by using system.time
37:1 [Top Level] : Run All:

```

## 第123.5节：预览输出

在渲染笔记本的最终版本之前，我们可以预览输出。点击工具栏上的预览按钮，然后选择所需的输出格式。

您可以通过使用输出选项更改输出类型，如“pdf\_document”或“html\_notebook”。

RStudio interface showing the preview output toolbar button highlighted with a red arrow.

```

1 --- "R Notebook"
2 title: "R Notebook"
3 output: html_notebook
4 ...
5
6 This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the
notebook, the results appear beneath the code.
7
8 Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor
inside it and pressing *Ctrl+Shift+Enter*.
9
10 {r}
11 plot(cars)
12 ...
13
14 Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.
15
16 When you save the notebook, an HTML file containing the code and output will be saved alongside it
(click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).
17

```

## 第123.6节：保存与共享

当笔记本.Rmd被保存时，会在旁边生成一个.nb.html文件。该文件是一个自包含的HTML文件，其中包含了带有所有当前代码块输出的渲染版笔记本（适合在网站上显示）以及笔记本的.Rmd副本。

RStudio interface showing the preview output toolbar button highlighted with a red arrow.

```

14 data("iris")
15 head(iris,5)
16 ...
 Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1 5.1 3.5 1.4 0.2 setosa
2 4.9 3.0 1.4 0.2 setosa
3 4.7 3.2 1.3 0.2 setosa
4 4.6 3.1 1.5 0.2 setosa
5 5.0 3.6 1.4 0.2 setosa
 5 rows
17
18 Divide Iris data to x (contain the all features) and y (only the classes)
19 {r}
20 x <- subset(iris, select=-Species)
21 y <- iris$Species
22 ...
23 Create SVM Model and show summary
24 {r}
25 svm_model <- svm(x,y)
26 summary(svm_model)
27 ...
28 Run Prediction
29 {r}
30 pred <- predict(svm_model,x)
31 ...
32 you can time taken by using system.time
33 pred <- predict(svm_model,x)
34 ...
35 pred <- predict(svm_model,x)
36 you can time taken by using system.time
37:1 [Top Level] : Run All:

```

## Section 123.5: Preview Output

Before rendering the final version of a notebook we can preview the output. Click on the **Preview** button on the toolbar and select the desired output format.

You can change the type of output by using the output options as “pdf\_document” or “html\_notebook”

RStudio interface showing the preview output toolbar button highlighted with a red arrow.

```

1 --- "R Notebook"
2 title: "R Notebook"
3 output: html_notebook
4 ...
5
6 This is an [R Markdown](http://rmarkdown.rstudio.com) Notebook. When you execute code within the
notebook, the results appear beneath the code.
7
8 Try executing this chunk by clicking the *Run* button within the chunk or by placing your cursor
inside it and pressing *Ctrl+Shift+Enter*.
9
10 {r}
11 plot(cars)
12 ...
13
14 Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.
15
16 When you save the notebook, an HTML file containing the code and output will be saved alongside it
(click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).
17

```

## Section 123.6: Saving and Sharing

When a notebook .Rmd is saved, an .nb.html file is created alongside it. This file is a self-contained HTML file which contains both a rendered copy of the notebook with all current chunk outputs (suitable for display on a website) and a copy of the notebook .Rmd itself.

更多信息可见于RStudio文档

More info can be found at [RStudio docs](#)

# 第124章：数据框的聚合

聚合是R最常见的用途之一。R中有多种实现方式，下面我们将进行说明。

## 第124.1节：使用data.table进行聚合

使用data.table包进行分组的语法是`dt[i, j, by]`，可以口头读作：“取`dt`，使用`i`筛选行，然后计算`j`，按`by`分组。”在`dt`语句中，多个计算或分组应放入列表中。由于`list()`的别名是`.()`，两者可以互换使用。下面的示例中我们使用`.()`。

代码：

```
使用 data.table 进行聚合
library(data.table)

dt = data.table(group=c("组 1","组 1","组 2","组 2"), 子组 =
c("A","A","A","A","B"), 值 = c(2,2.5,1,2,1.5))
print(dt)

按一列分组求和
dt[.(值=sum(值)),组]

按一列分组求均值
dt[.(值=mean(值)),组]

按多列分组求和
dt[, .(值=sum(值)), .(组,子组)]

自定义函数，按一列分组
在此示例中，我们想要每组中所有大于2的值的和。
dt[.(value=sum(value[value>2])),group]
```

输出：

```
> # 使用 data.table 进行聚合
> library(data.table)

dt = data.table(group=c("组 1","组 1","组 2","组 2"), 子组 =
c("A","A","A","A","B"), value = c(2,2.5,1,2,1.5))
> print(dt)
组别 子组 值
1: 组 1 A 2.0
2: 组 1 A 2.5
3: 组 2 A 1.0
4: 组 2 A 2.0
5: 组 2 B 1.5
>
> # 求和，按一列分组
> dt[.(value=sum(value)),group]
 分组值
1: 组1 4.5
2: 组2 4.5
>
> # 平均值，按一列分组
> dt[.(value=mean(value)),group]
 组 值
```

# Chapter 124: Aggregating data frames

Aggregation is one of the most common uses for R. There are several ways to do so in R, which we will illustrate here.

## Section 124.1: Aggregating with data.table

Grouping with the data.table package is done using the syntax `dt[i, j, by]` Which can be read out loud as: "Take `dt`, subset rows using `i`, then calculate `j`, grouped by `by`." Within the `dt` statement, multiple calculations or groups should be put in a list. Since an alias for `list()` is `.()`, both can be used interchangeably. In the examples below we use `.()`.

CODE:

```
Aggregating with data.table
library(data.table)

dt = data.table(group=c("Group 1","Group 1","Group 2","Group 2","Group 2"),
c("A","A","A","A","B"),value = c(2,2.5,1,2,1.5))
print(dt)

sum, grouping by one column
dt[, .(value=sum(value)),group]

mean, grouping by one column
dt[, .(value=mean(value)),group]

sum, grouping by multiple columns
dt[, .(value=sum(value)),.(group,subgroup)]

custom function, grouping by one column
in this example we want the sum of all values larger than 2 per group.
dt[.,.(value=sum(value[value>2])),group]
```

OUTPUT:

```
> # Aggregating with data.table
> library(data.table)
>
> dt = data.table(group=c("Group 1","Group 1","Group 2","Group 2","Group 2"),
c("A","A","A","A","B"),value = c(2,2.5,1,2,1.5))
> print(dt)
 group subgroup value
1: Group 1 A 2.0
2: Group 1 A 2.5
3: Group 2 A 1.0
4: Group 2 A 2.0
5: Group 2 B 1.5
>
> # sum, grouping by one column
> dt[, .(value=sum(value)),group]
 group value
1: Group 1 4.5
2: Group 2 4.5
>
> # mean, grouping by one column
> dt[, .(value=mean(value)),group]
 group value
```

```

1: 组1 2.25
2: 组2 1.50
>
> # 求和, 按多列分组
> dt[,(value=sum(value)),.(group,subgroup)]
 组 子组 值
1: 组1 A 4.5
2: 组2 A 3.0
3: 组2 B 1.5
>
> # 自定义函数, 按一列分组
> # 在此示例中, 我们想要每组中所有大于2的值的总和。
> dt[,(value=sum(value[value>2])),group]
 group value
1: 组 1 2.5
2: 组 2 0.0

```

## 第124.2节：使用基础R进行聚合

为此, 我们将使用aggregate函数, 其用法如下:

**aggregate(formula,function,data)**

下面的代码展示了使用aggregate函数的多种方式。

代码：

```

df = data.frame(group=c("组 1","组 1","组 2","组 2"), subgroup =c("A","A","A","B"),value = c(2,2.5
,1,2,1.5))

按一列分组求和
aggregate(value~group, FUN=sum, data=df)

计算均值, 按一列分组
aggregate(value~group, FUN=mean, data=df)

计算总和, 按多列分组
aggregate(value~group+subgroup,FUN=sum,data=df)

自定义函数, 按一列分组
在此示例中, 我们想要计算每组中所有大于2的值的总和。
aggregate(value~group, FUN=function(x) sum(x[x>2]), data=df)

```

输出：

```

> df = data.frame(group=c("组 1","组 1","组 2","组 2"), subgroup =
c("A","A","A","B"),value = c(2,2.5,1,2,1.5))
> print(df)
 group subgroup value
1组 1 A 2.0
2组 1 A 2.5
3组 2 A 1.0
4组 2 A 2.0
5组 2 B 1.5

>
> aggregate(value~group, FUN=sum, data=df)
 group value
1 Group 1 4.5

```

```

1: Group 1 2.25
2: Group 2 1.50
>
> # sum, grouping by multiple columns
> dt[,(value=sum(value)),.(group,subgroup)]
 group subgroup value
1: Group 1 A 4.5
2: Group 2 A 3.0
3: Group 2 B 1.5
>
> # custom function, grouping by one column
> # in this example we want the sum of all values larger than 2 per group.
> dt[,(value=sum(value[value>2])),group]
 group value
1: Group 1 2.5
2: Group 2 0.0

```

## Section 124.2: Aggregating with base R

For this, we will use the function aggregate, which can be used as follows:

**aggregate(formula,function,data)**

The following code shows various ways of using the aggregate function.

CODE:

```

df = data.frame(group=c("Group 1","Group 1","Group 2","Group 2","Group 2"),
subgroup =c("A","A","A","A","B"),value = c(2,2.5,1,2,1.5))

sum, grouping by one column
aggregate(value~group, FUN=sum, data=df)

mean, grouping by one column
aggregate(value~group, FUN=mean, data=df)

sum, grouping by multiple columns
aggregate(value~group+subgroup,FUN=sum,data=df)

custom function, grouping by one column
in this example we want the sum of all values larger than 2 per group.
aggregate(value~group, FUN=function(x) sum(x[x>2]), data=df)

```

OUTPUT:

```

> df = data.frame(group=c("Group 1","Group 1","Group 2","Group 2","Group 2"),
subgroup =c("A","A","A","A","B"),value = c(2,2.5,1,2,1.5))
> print(df)
 group subgroup value
1 Group 1 A 2.0
2 Group 1 A 2.5
3 Group 2 A 1.0
4 Group 2 A 2.0
5 Group 2 B 1.5
>
> # sum, grouping by one column
> aggregate(value~group, FUN=sum, data=df)
 group value
1 Group 1 4.5

```

```

2 Group 2 4.5

> aggregate(value~group, FUN=mean, data=df)
 group value
1 Group 1 2.25
2 Group 2 1.50

>
aggregate(value~group+subgroup, FUN=sum, data=df)
 group subgroup value
1组1 A 4.5
2组2 A 3.0
3组 2 B 1.5

>
在此示例中，我们想要计算每组中所有大于2的值的总和。
> aggregate(value~group, FUN=function(x) sum(x[x>2]), data=df)
 group value
1组1 2.5
2组2 0.0

```

## 第124.3节：使用dplyr进行聚合

使用dplyr进行聚合很简单！你可以使用group\_by()和summarize()函数。下面给出一些示例。

代码：

```

使用dplyr进行聚合
library(dplyr)

df = data.frame(group=c("组 1","组 1","组 2","组 2"), subgroup =c("A","A","A","B"),value = c(2,2.5,1,2,1.5))
print(df)

按一列分组求和
df %>% group_by(group) %>% summarize(value = sum(value)) %>% as.data.frame()

计算均值，按一列分组
df %>% group_by(group) %>% summarize(value = mean(value)) %>% as.data.frame()

求和，按多列分组
df %>% group_by(group,subgroup) %>% summarize(value = sum(value)) %>% as.data.frame()

自定义函数，按一列分组
在此示例中，我们想要计算每组中所有大于2的值的总和。
df %>% group_by(group) %>% summarize(value = sum(value[value>2])) %>% as.data.frame()

```

输出：

```

> library(dplyr)
>
> df = data.frame(group=c("组 1","组 1","组 2","组 2"), subgroup =
c("A","A","A","B"),value = c(2,2.5,1,2,1.5))
> print(df)
 group subgroup value
1组 1 A 2.0
2组 1 A 2.5
3组 2 A 1.0

```

```

2 Group 2 4.5
>
mean, grouping by one column
> aggregate(value~group, FUN=mean, data=df)
 group value
1 Group 1 2.25
2 Group 2 1.50
>
sum, grouping by multiple columns
> aggregate(value~group+subgroup, FUN=sum, data=df)
 group subgroup value
1 Group 1 A 4.5
2 Group 2 A 3.0
3 Group 2 B 1.5
>
custom function, grouping by one column
> # in this example we want the sum of all values larger than 2 per group.
> aggregate(value~group, FUN=function(x) sum(x[x>2]), data=df)
 group value
1 Group 1 2.5
2 Group 2 0.0

```

## Section 124.3: Aggregating with dplyr

Aggregating with dplyr is easy! You can use the group\_by() and the summarize() functions for this. Some examples are given below.

CODE:

```

Aggregating with dplyr
library(dplyr)

df = data.frame(group=c("Group 1","Group 1","Group 2","Group 2","Group 2"), subgroup =
c("A","A","A","A","B"),value = c(2,2.5,1,2,1.5))
print(df)

sum, grouping by one column
df %>% group_by(group) %>% summarize(value = sum(value)) %>% as.data.frame()

mean, grouping by one column
df %>% group_by(group) %>% summarize(value = mean(value)) %>% as.data.frame()

sum, grouping by multiple columns
df %>% group_by(group,subgroup) %>% summarize(value = sum(value)) %>% as.data.frame()

custom function, grouping by one column
in this example we want the sum of all values larger than 2 per group.
df %>% group_by(group) %>% summarize(value = sum(value[value>2])) %>% as.data.frame()

```

OUTPUT:

```

> library(dplyr)
>
> df = data.frame(group=c("Group 1","Group 1","Group 2","Group 2","Group 2"), subgroup =
c("A","A","A","A","B"),value = c(2,2.5,1,2,1.5))
> print(df)
 group subgroup value
1 Group 1 A 2.0
2 Group 1 A 2.5
3 Group 2 A 1.0

```

```

4组 2 A 2.0
5组 2 B 1.5

>
> df %>% group_by(group) %>% summarize(value = sum(value)) %>% as.data.frame()
 group value
1 Group 1 4.5
2 Group 2 4.5

>
> df %>% group_by(group) %>% summarize(value = mean(value)) %>% as.data.frame()
 group value
1 Group 1 2.25
2 Group 2 1.50

>
> df %>% group_by(group, subgroup) %>% summarize(value = sum(value)) %>% as.data.frame()
 group subgroup value
1组1 A 4.5
2组2 A 3.0
3组 2 B 1.5

>
在此示例中，我们想要计算每组中所有大于2的值的总和。
> df %>% group_by(group) %>% summarize(value = sum(value[value>2])) %>% as.data.frame()
 group value
1组1 2.5
2组2 0.0

```

```

4 Group 2 A 2.0
5 Group 2 B 1.5
>
> # sum, grouping by one column
> df %>% group_by(group) %>% summarize(value = sum(value)) %>% as.data.frame()
 group value
1 Group 1 4.5
2 Group 2 4.5
>
> # mean, grouping by one column
> df %>% group_by(group) %>% summarize(value = mean(value)) %>% as.data.frame()
 group value
1 Group 1 2.25
2 Group 2 1.50
>
> # sum, grouping by multiple columns
> df %>% group_by(group, subgroup) %>% summarize(value = sum(value)) %>% as.data.frame()
 group subgroup value
1 Group 1 A 4.5
2 Group 2 A 3.0
3 Group 2 B 1.5
>
> # custom function, grouping by one column
> # in this example we want the sum of all values larger than 2 per group.
> df %>% group_by(group) %>% summarize(value = sum(value[value>2])) %>% as.data.frame()
 group value
1 Group 1 2.5
2 Group 2 0.0

```

# 第125章：数据采集

直接将数据导入R会话。R的一个优点是数据获取的便捷性。使用R包有多种数据传播方式。

## 第125.1节：内置数据集

R拥有大量内置数据集。通常，它们用于教学目的，以创建快速且易于复现的示例。有一个很好的网页列出了内置数据集：

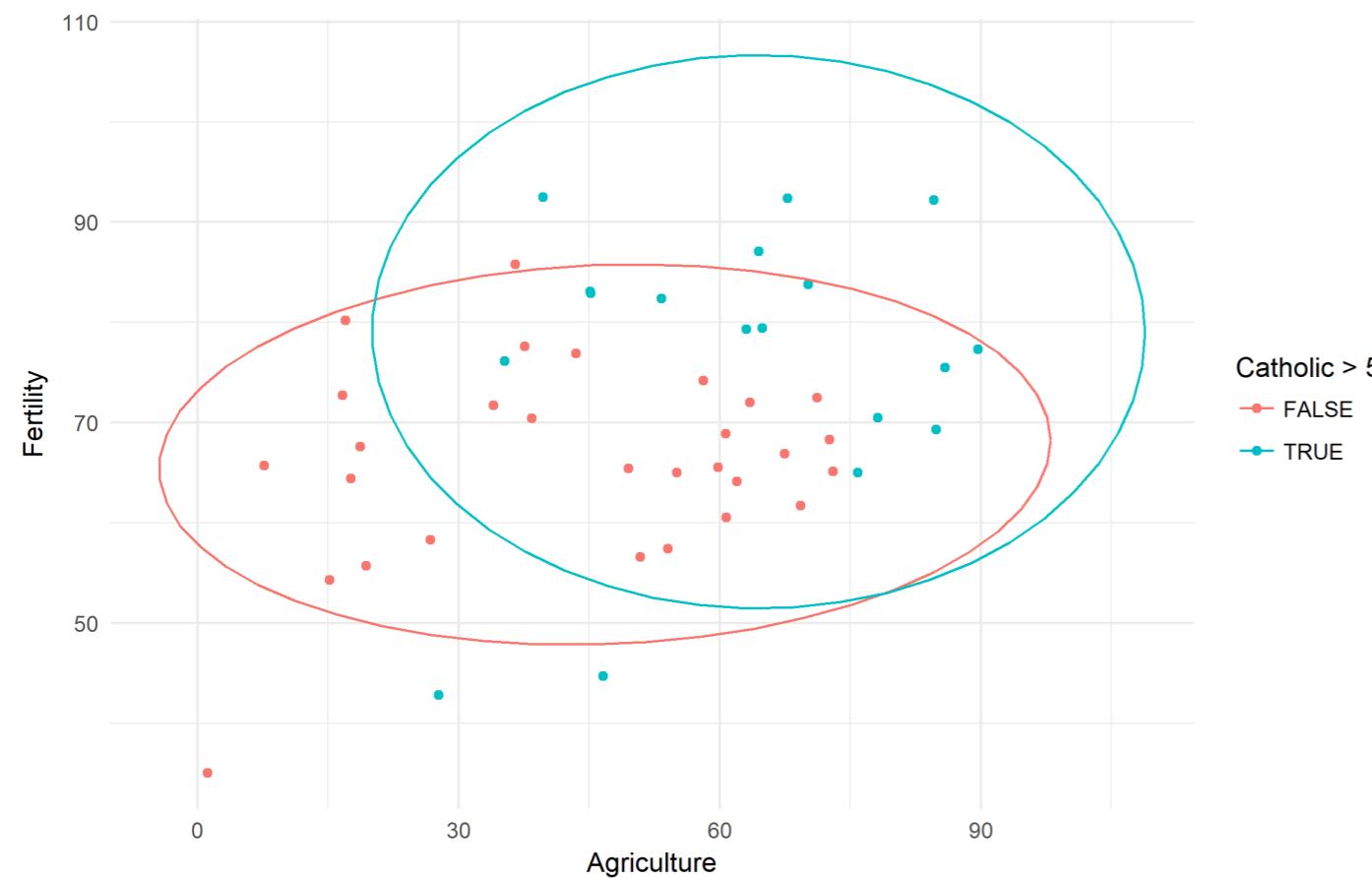
<https://vincentarelbundock.github.io/Rdatasets/datasets.html>

### 示例

瑞士生育率和社会经济指标（1888年）数据。让我们检查基于农村程度和天主教人口占比的生育率差异。

```
library(tidyverse)

swiss %>%
 ggplot(aes(x = Agriculture, y = Fertility,
 color = Catholic > 50))+
 geom_point() +
 stat_ellipse()
```



## 第125.2节：访问开放数据库的包

许多包是专门为访问某些数据库而创建的。使用它们可以节省大量时间

# Chapter 125: Data acquisition

Get data directly into an R session. One of the nice features of R is the ease of data acquisition. There are several ways data dissemination using R packages.

## Section 125.1: Built-in datasets

R has a vast collection of built-in datasets. Usually, they are used for teaching purposes to create quick and easily reproducible examples. There is a nice web-page listing the built-in datasets:

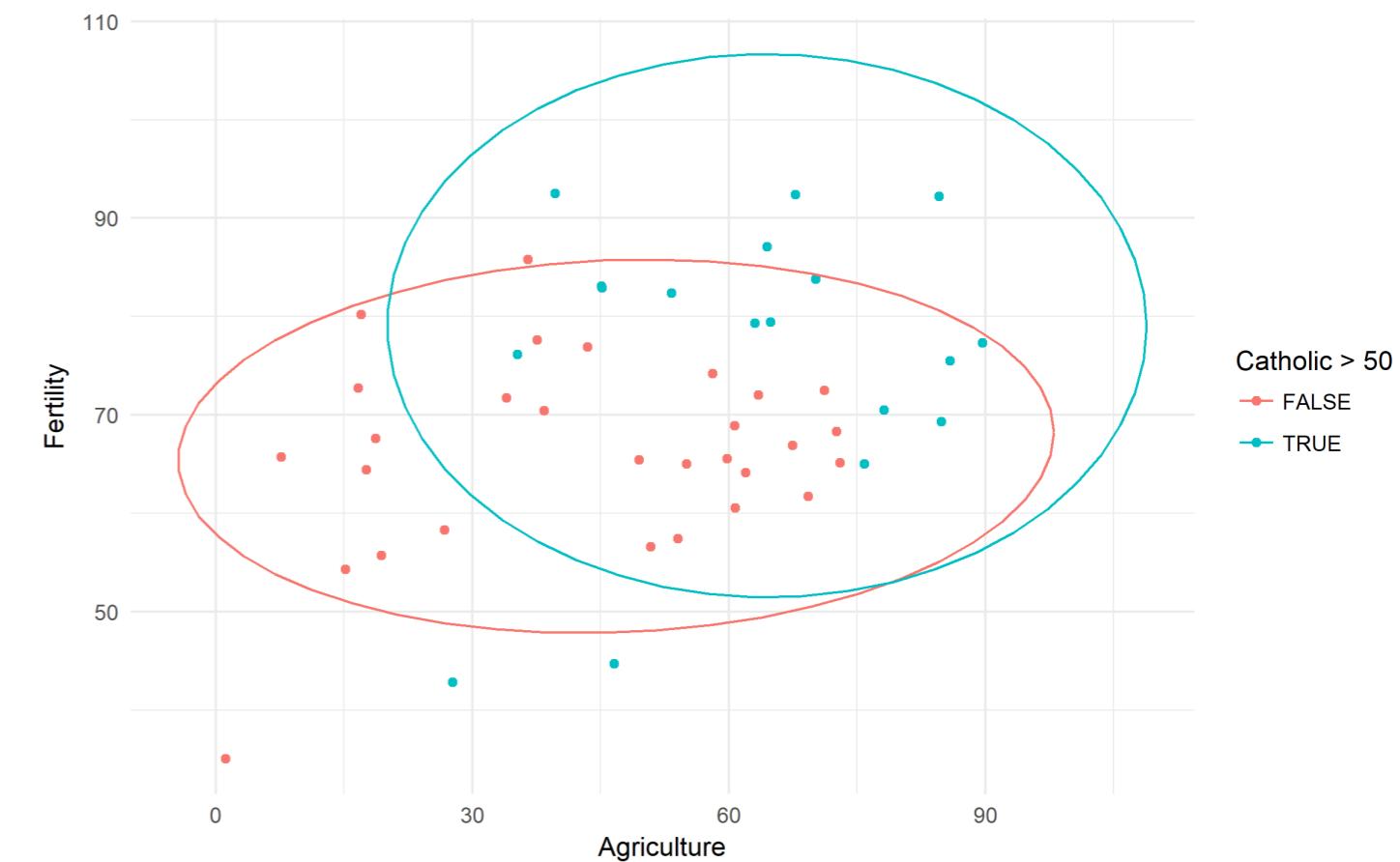
<https://vincentarelbundock.github.io/Rdatasets/datasets.html>

### Example

Swiss Fertility and Socioeconomic Indicators (1888) Data. Let's check the difference in fertility based of rurality and domination of Catholic population.

```
library(tidyverse)

swiss %>%
 ggplot(aes(x = Agriculture, y = Fertility,
 color = Catholic > 50))+
 geom_point() +
 stat_ellipse()
```



## Section 125.2: Packages to access open databases

Numerous packages are created specifically to access some databases. Using them can save a bunch of time on

**Eurostat**

尽管 eurostat 包含一个函数 search\_eurostat()，但它无法找到所有相关的数据集。因此，手动在Eurostat网站上浏览数据集代码更为方便：国家数据库，或区域数据库。如果自动下载失败，可以通过批量下载工具手动获取数据。

```
library(tidyverse)
library(lubridate)
library(forcats)
library(eurostat)
library(geofacet)
library(viridis)
library(ggthemes)
library(extrafont)

下载国家的NEET数据
neet <- get_eurostat("edat_lfse_22")

neet %>%
 filter(geo %>% paste %>% nchar == 2,
 sex == "T", age == "Y18-24") %>%
 group_by(geo) %>%
 mutate(avg = values %>% mean()) %>%
 ungroup() %>%
 ggplot(aes(x = time %>% year(),
 y = values))+
 geom_path(aes(group = 1))+
 geom_point(aes(fill = values), pch = 21)+
 scale_x_continuous(breaks = seq(2000, 2015, 5),
 labels = c("2000", "'05", "'10", "'15"))+
 scale_y_continuous(expand = c(0, 0), limits = c(0, 40))+
 scale_fill_viridis("NEET, %", option = "B")+
 facet_geo(~ geo, grid = "eu_grid1")+
 labs(x = "年份",
 y = "NEET, %",
 title = "欧洲未就业且未接受教育和培训的年轻人",
 subtitle = "数据来源：Eurostat区域数据库，2000-2016",
 caption = "ikashnitsky.github.io")+
 theme_few(base_family = "Roboto Condensed", base_size = 15)+
 theme(axis.text = element_text(size = 10),
 panel.spacing.x = unit(1, "行"),
 legend.position = c(0, 0),
 legend.justification = c(0, 0))
```

reading/formatting the data.

**Eurostat**

Even though eurostat package has a function search\_eurostat(), it does not find all the relevant datasets available. This, it's more convenient to browse the code of a dataset manually at the Eurostat website: [Countries Database](#), or [Regional Database](#). If the automated download does not work, the data can be grabbed manually at via [Bulk Download Facility](#).

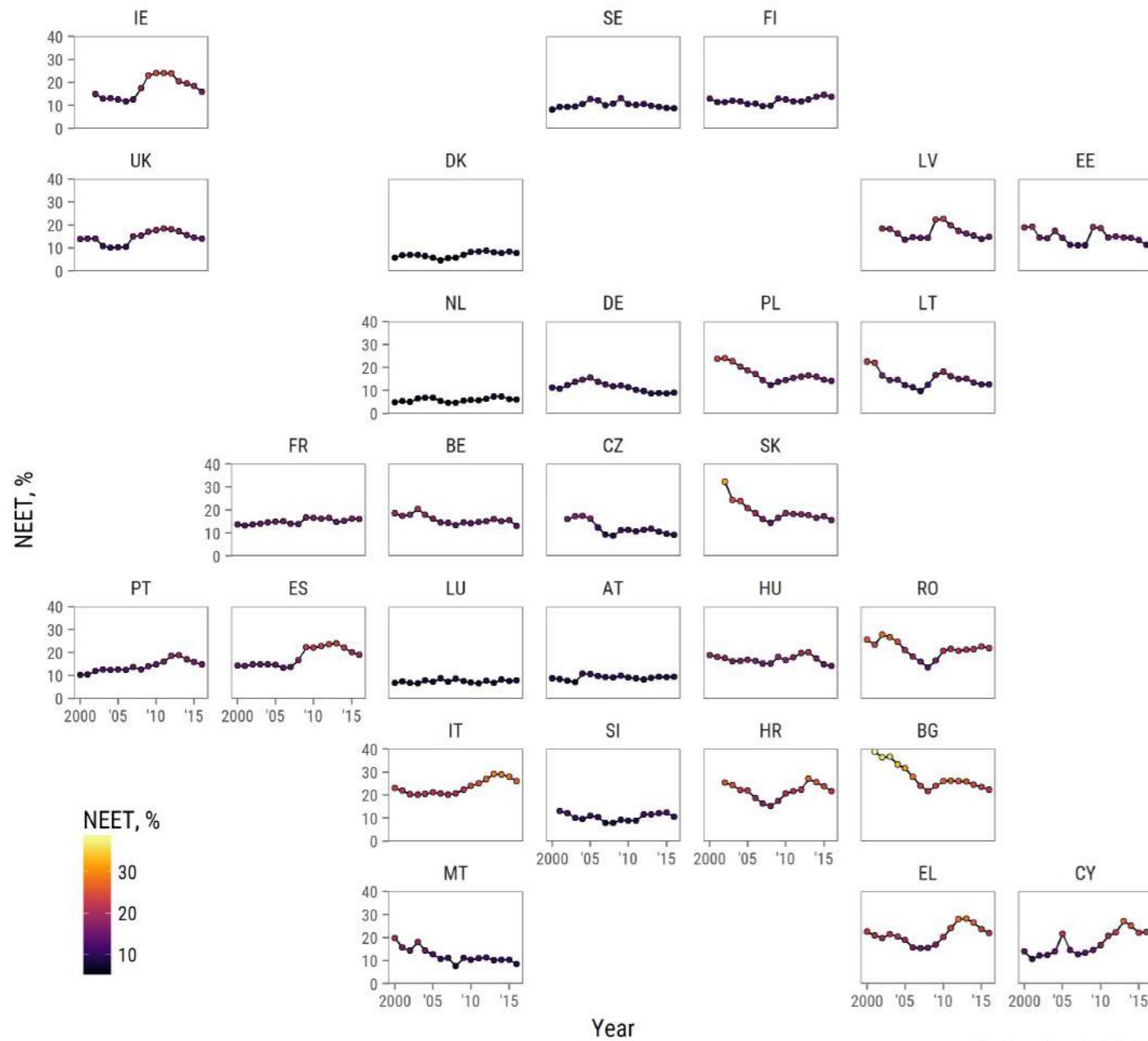
```
library(tidyverse)
library(lubridate)
library(forcats)
library(eurostat)
library(geofacet)
library(viridis)
library(ggthemes)
library(extrafont)

download NEET data for countries
neet <- get_eurostat("edat_lfse_22")

neet %>%
 filter(geo %>% paste %>% nchar == 2,
 sex == "T", age == "Y18-24") %>%
 group_by(geo) %>%
 mutate(avg = values %>% mean()) %>%
 ungroup() %>%
 ggplot(aes(x = time %>% year(),
 y = values))+
 geom_path(aes(group = 1))+
 geom_point(aes(fill = values), pch = 21)+
 scale_x_continuous(breaks = seq(2000, 2015, 5),
 labels = c("2000", "'05", "'10", "'15"))+
 scale_y_continuous(expand = c(0, 0), limits = c(0, 40))+
 scale_fill_viridis("NEET, %", option = "B")+
 facet_geo(~ geo, grid = "eu_grid1")+
 labs(x = "Year",
 y = "NEET, %",
 title = "Young people neither in employment nor in education and training in Europe",
 subtitle = "Data: Eurostat Regional Database, 2000-2016",
 caption = "ikashnitsky.github.io")+
 theme_few(base_family = "Roboto Condensed", base_size = 15)+
 theme(axis.text = element_text(size = 10),
 panel.spacing.x = unit(1, "lines"),
 legend.position = c(0, 0),
 legend.justification = c(0, 0))
```

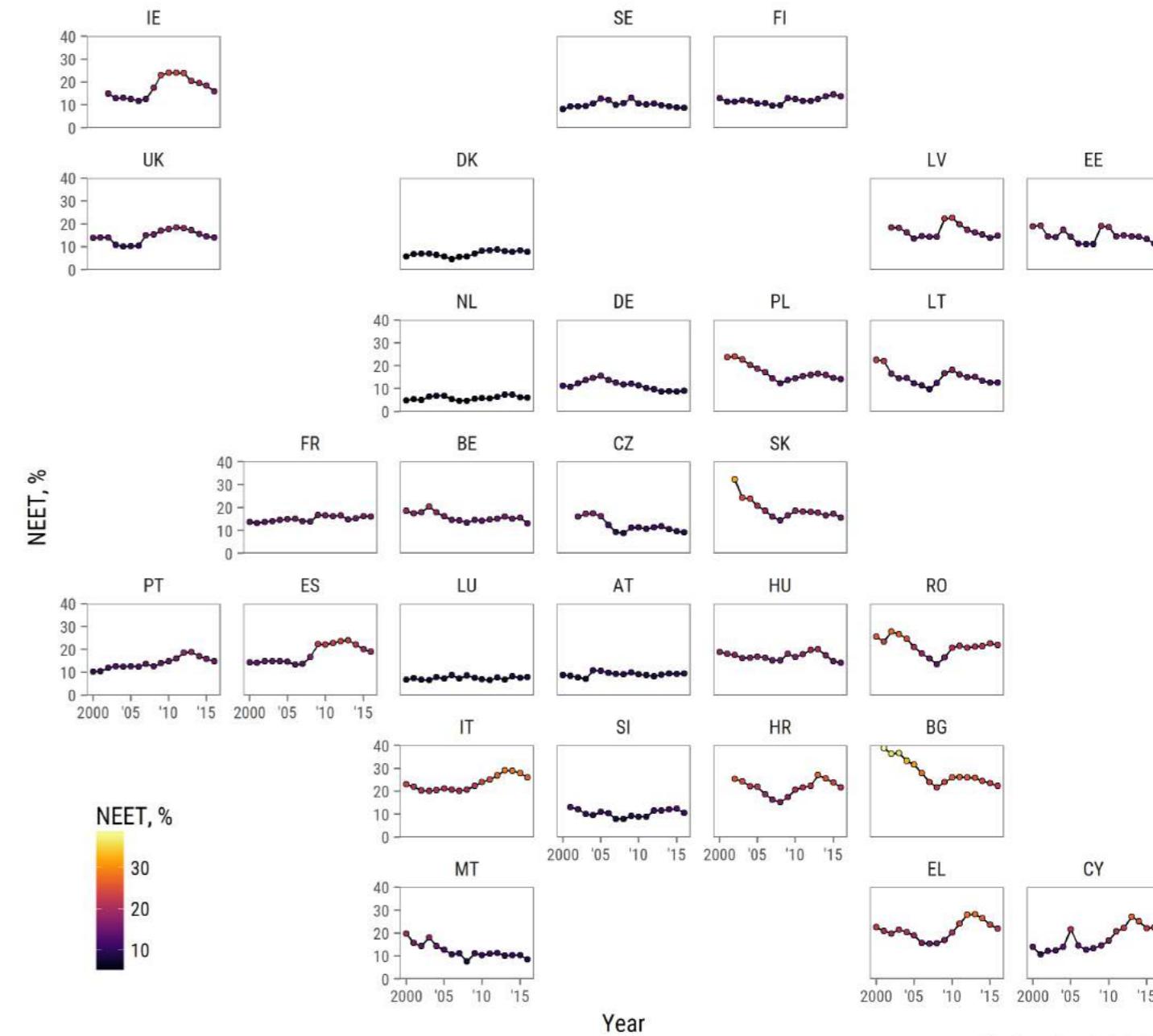
## Young people neither in employment nor in education and training in Europe

Data: Eurostat Regional Database, 2000-2016



## Young people neither in employment nor in education and training in Europe

Data: Eurostat Regional Database, 2000-2016



## 第125.3节：访问受限数据的软件包

### 人类死亡率数据库

人类死亡率数据库是马克斯·普朗克人口研究所的一个项目，收集并预处理那些拥有或多或少可靠统计数据国家的人类死亡率数据。

```
加载所需的软件包
library(tidyverse)
library(extrafont)
library(HMDHDFplus)

country <- getHMDcountries()

exposures <- list()
for (i in 1: length(country)) {
```

## Section 125.3: Packages to access restricted data

### Human Mortality Database

[Human Mortality Database](#) is a project of the [Max Planck Institute for Demographic Research](#) that gathers and pre-process human mortality data for those countries, where more or less reliable statistics is available.

```
load required packages
library(tidyverse)
library(extrafont)
library(HMDHDFplus)

country <- getHMDcountries()

exposures <- list()
for (i in 1: length(country)) {
```

```

cnt <- country[i]
exposures[[cnt]] <- readHMDweb(cnt, "Exposures_1x1", user_hmd, pass_hmd)
 # 打印进度
 paste(i,'out of',length(country))
} # 这将花费相当多的时间

```

请注意，参数user\_hmd和pass\_hmd是人类死亡率数据库网站的登录凭证。为了访问数据，用户需要在http://www.mortality.org/创建一个账户，并将自己的凭证提供给readHMDweb()函数。

```

sr_age <- list()

for (i in 1:length(exposures)) {
 di <- exposures[[i]]
 sr_agei <- di %>% select(Year, Age, Female, Male) %>%
 filter(Year %in% 2012) %>%
 select(-Year) %>%
 transmute(country = names(exposures)[i],
 age = Age, sr_age = Male / Female * 100)
 sr_age[[i]] <- sr_agei
}
sr_age <- bind_rows(sr_age)

删除可选人群
sr_age <- sr_age %>% filter(!country %in% c("FRACNP", "DEUTE", "DEUTW", "GBRCENW", "GBR_NP"))

汇总所有年龄大于90岁的数据（波动过大）
sr_age_90 <- sr_age %>% filter(age %in% 90:110) %>%
 group_by(country) %>% summarise(sr_age = mean(sr_age, na.rm = T)) %>%
 ungroup() %>% transmute(country, age=90, sr_age)

df_plot <- bind_rows(sr_age %>% filter(!age %in% 90:110), sr_age_90)

最后 - 绘图
df_plot %>%
 ggplot(aes(age, sr_age, color = country, group = country))+
 geom_hline(yintercept = 100, color = 'grey50', size = 1)+
 geom_line(size = 1)+
 scale_y_continuous(limits = c(0, 120), expand = c(0, 0), breaks = seq(0, 120, 20))+
 scale_x_continuous(limits = c(0, 90), expand = c(0, 0), breaks = seq(0, 80, 20))+
 xlab('年龄')+
 ylab('性别比，男性对每100名女性')+
 facet_wrap(~country, ncol=6)+
 theme_minimal(base_family = "Roboto Condensed", base_size = 15)+
 theme(legend.position='none',
 panel.border = element_rect(size = .5, fill = NA))

```

```

cnt <- country[i]
exposures[[cnt]] <- readHMDweb(cnt, "Exposures_1x1", user_hmd, pass_hmd)
 # let's print the progress
 paste(i,'out of',length(country))
} # this will take quite a lot of time

Please note, the arguments user_hmd and pass_hmd are the login credentials at the website of Human Mortality Database. In order to access the data, one needs to create an account at http://www.mortality.org/ and provide their own credentials to the readHMDweb() function.

sr_age <- list()

for (i in 1:length(exposures)) {
 di <- exposures[[i]]
 sr_agei <- di %>% select(Year, Age, Female, Male) %>%
 filter(Year %in% 2012) %>%
 select(-Year) %>%
 transmute(country = names(exposures)[i],
 age = Age, sr_age = Male / Female * 100)
 sr_age[[i]] <- sr_agei
}
sr_age <- bind_rows(sr_age)

remove optional populations
sr_age <- sr_age %>% filter(!country %in% c("FRACNP", "DEUTE", "DEUTW", "GBRCENW", "GBR_NP"))

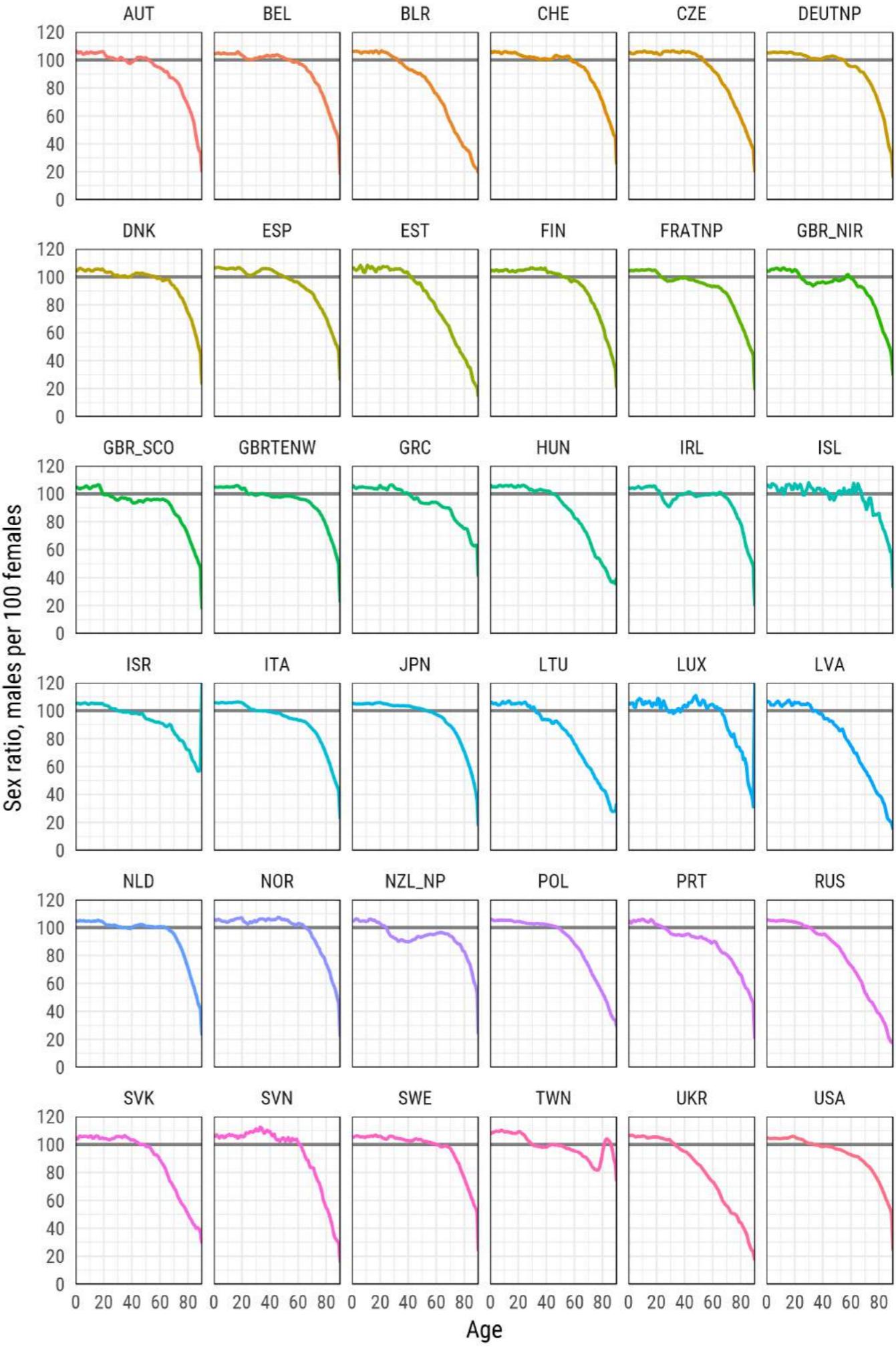
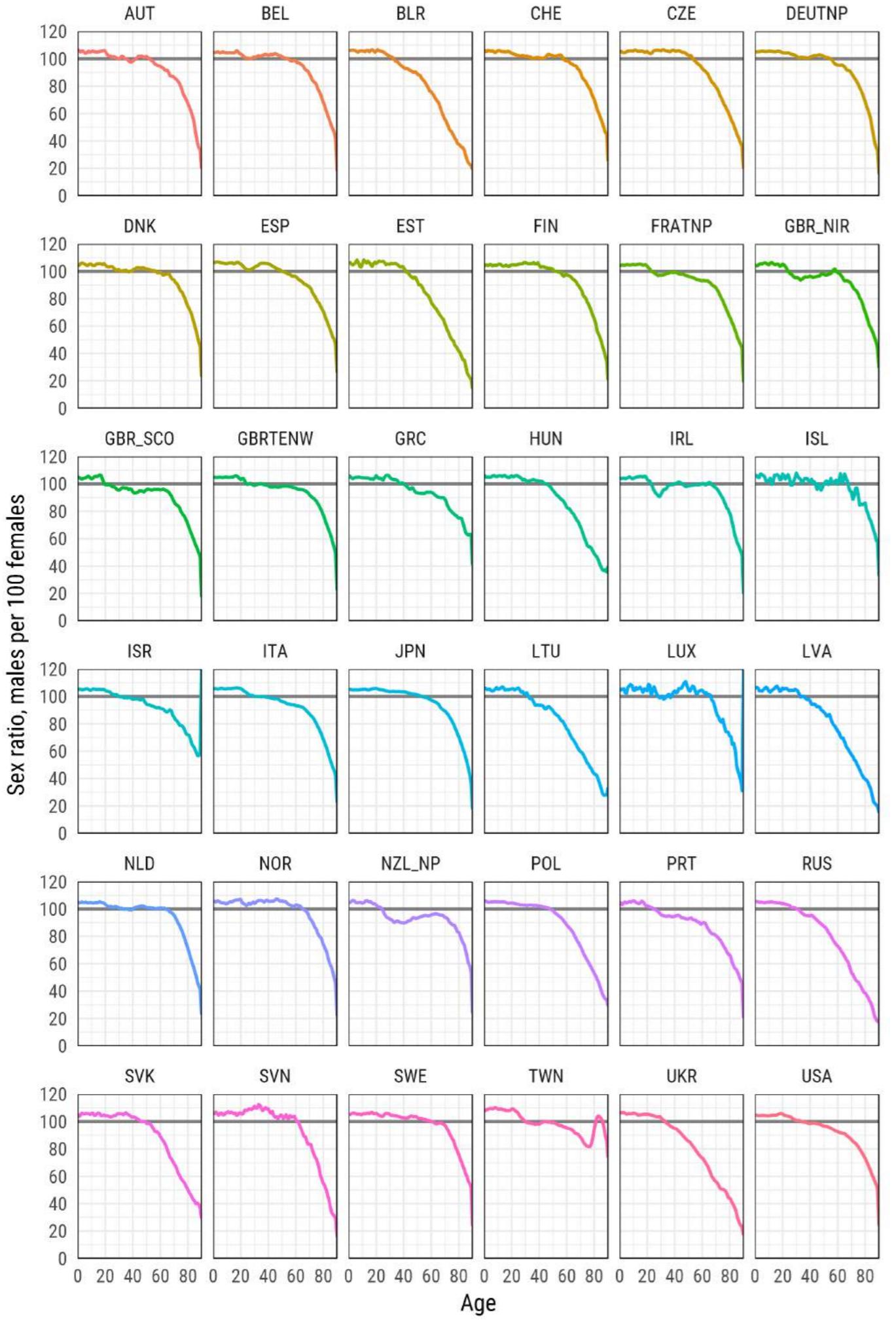
summarize all ages older than 90 (too jerky)
sr_age_90 <- sr_age %>% filter(age %in% 90:110) %>%
 group_by(country) %>% summarise(sr_age = mean(sr_age, na.rm = T)) %>%
 ungroup() %>% transmute(country, age=90, sr_age)

df_plot <- bind_rows(sr_age %>% filter(!age %in% 90:110), sr_age_90)

finaly - plot
df_plot %>%
 ggplot(aes(age, sr_age, color = country, group = country))+
 geom_hline(yintercept = 100, color = 'grey50', size = 1)+
 geom_line(size = 1)+
 scale_y_continuous(limits = c(0, 120), expand = c(0, 0), breaks = seq(0, 120, 20))+
 scale_x_continuous(limits = c(0, 90), expand = c(0, 0), breaks = seq(0, 80, 20))+
 xlab('Age')+
 ylab('Sex ratio, males per 100 females')+
 facet_wrap(~country, ncol=6)+
 theme_minimal(base_family = "Roboto Condensed", base_size = 15)+
 theme(legend.position='none',
 panel.border = element_rect(size = .5, fill = NA))

```





## 第125.4节：包内的数据集

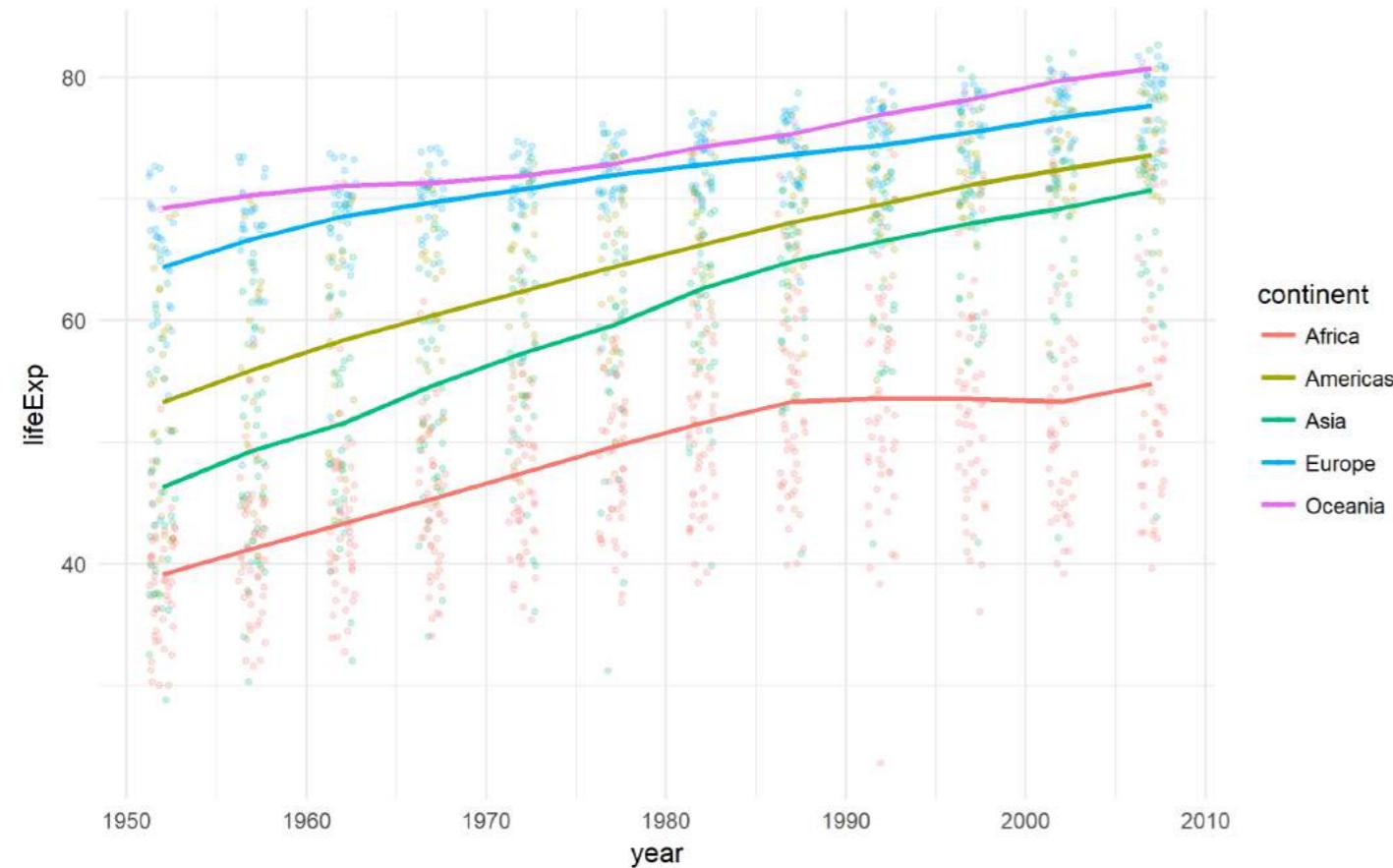
有些包包含数据，或者专门创建用于传播数据集。当加载此类包 (`library(pkg)`) 时，附带的数据集将作为R对象可用；或者需要使用`data()`函数调用。

### Gapminder

一个关于国家发展情况的优质数据集。

```
library(tidyverse)
library(gapminder)

gapminder %>%
 ggplot(aes(x = year, y = lifeExp,
 color = continent)) +
 geom_jitter(size = 1, alpha = .2, width = .75) +
 stat_summary(geom = "path", fun.y = mean, size = 1) +
 theme_minimal()
```



2015年世界人口展望 - 联合国人口司

让我们看看1950年至2015年间世界男性出生时预期寿命的趋同情况。

```
library(tidyverse)
library(forcats)
library(wpp2015)
library(ggjoy)
library(viridis)
```

## Section 125.4: Datasets within packages

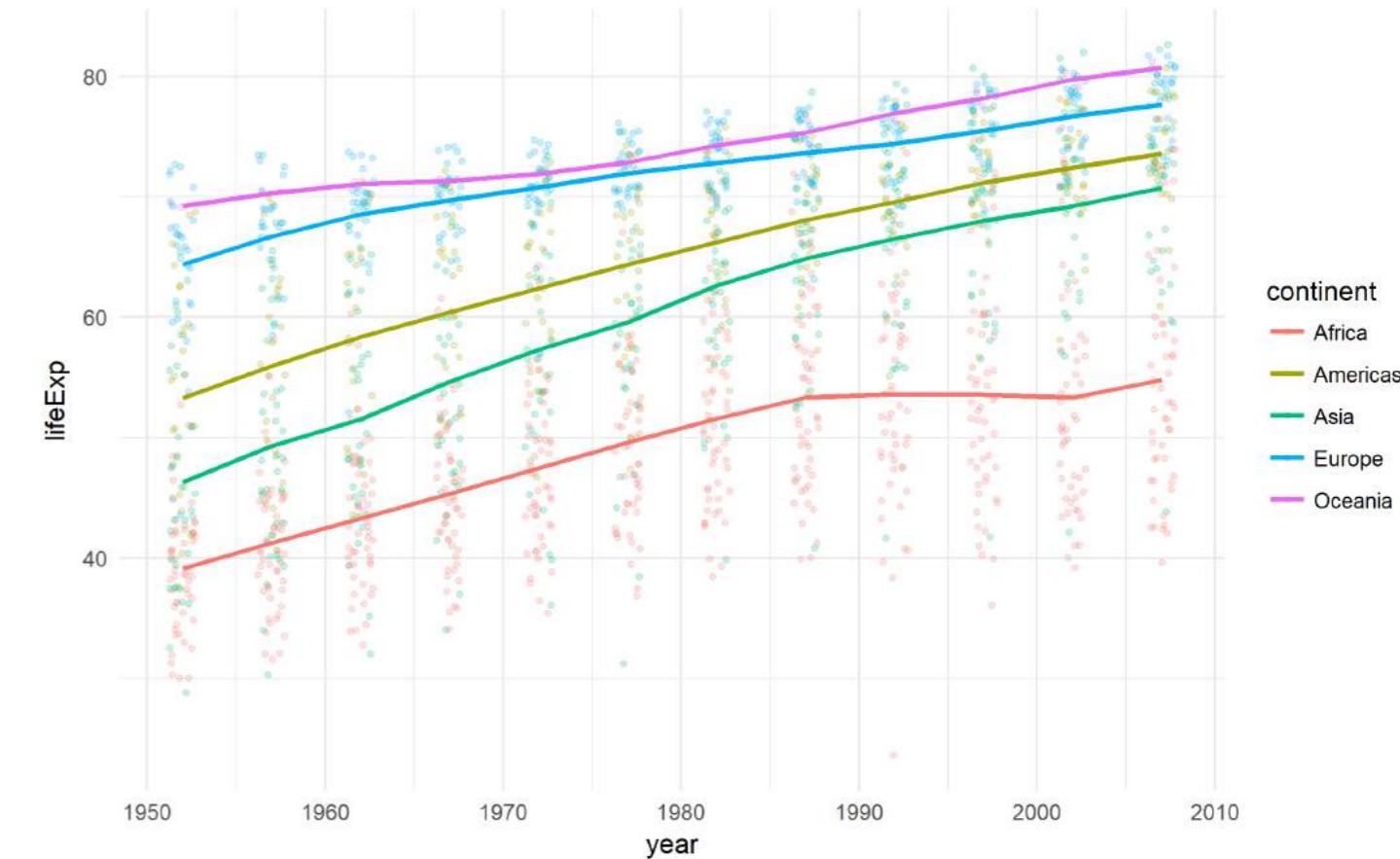
There are packages that include data or are created specifically to disseminate datasets. When such a package is loaded (`library(pkg)`), the attached datasets become available either as R objects; or they need to be called with the `data()` function.

### Gapminder

A nice dataset on the development of countries.

```
library(tidyverse)
library(gapminder)

gapminder %>%
 ggplot(aes(x = year, y = lifeExp,
 color = continent)) +
 geom_jitter(size = 1, alpha = .2, width = .75) +
 stat_summary(geom = "path", fun.y = mean, size = 1) +
 theme_minimal()
```



World Population Prospects 2015 - United Nations Population Department

Let's see how the world has converged in male life expectancy at birth over 1950-2015.

```
library(tidyverse)
library(forcats)
library(wpp2015)
library(ggjoy)
library(viridis)
```

```
library(extrafont)
```

```
data(UNlocations)
```

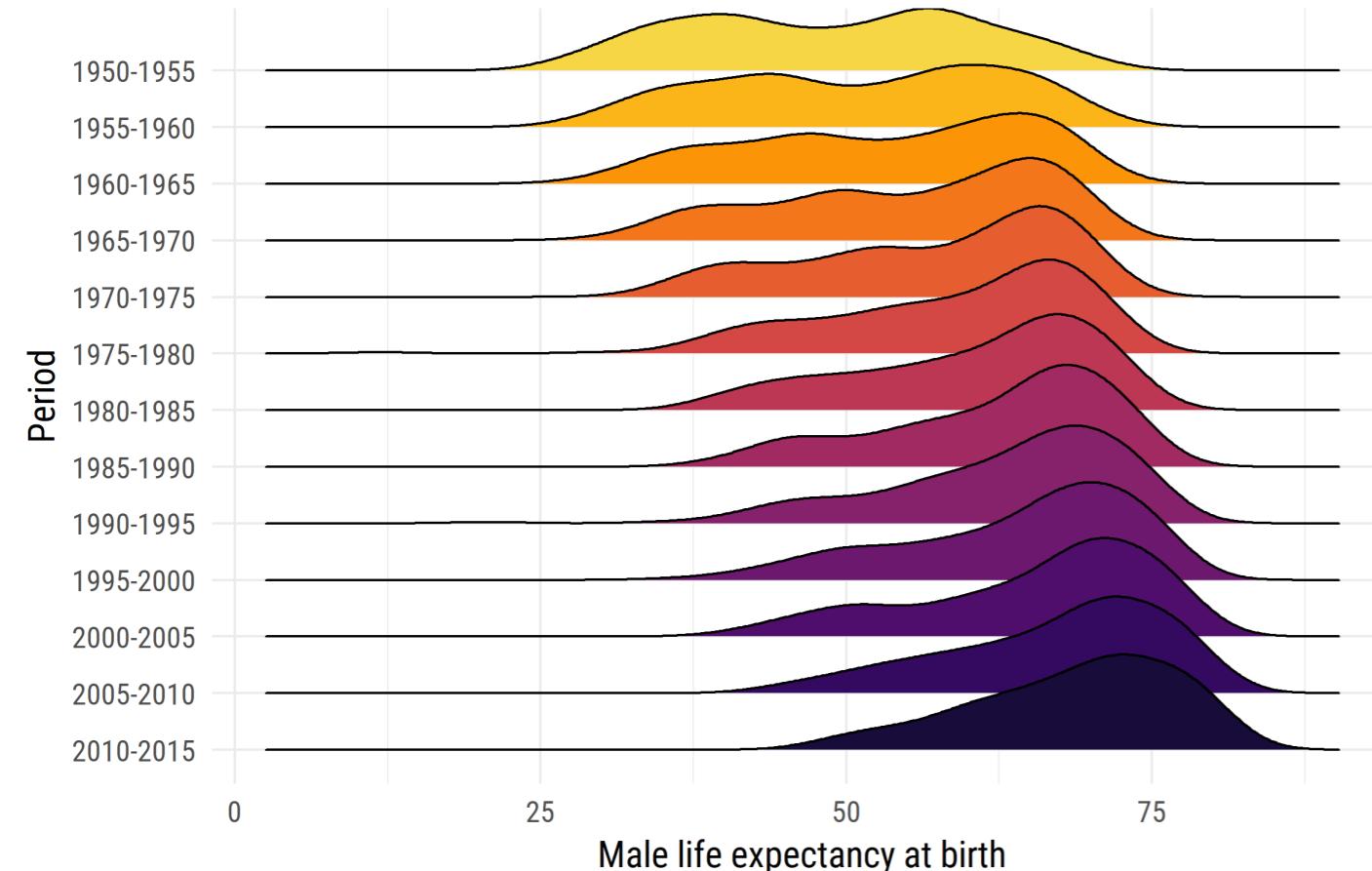
```
countries <- UNlocations %>%
 filter(location_type == 4) %>%
 transmute(name = name %>% paste()) %>%
 as_vector()
```

```
data(e0M)
```

```
e0M %>%
 filter(country %in% countries) %>%
 select(-last.observed) %>%
 gather(period, value, 3:15) %>%
 ggplot(aes(x = value, y = period %>% fct_rev()))+
 geom_joy(aes(fill = period))+
 scale_fill_viridis(discrete = T, option = "B", direction = -1,
 begin = .1, end = .9)+
 labs(x = "出生时男性预期寿命",
 y = "时期",
 title = "自1950年以来全球男性出生时预期寿命的趋同",
 subtitle = "数据来源：联合国人口司世界人口展望2015年修订版",
 caption = "ikashnitsky.github.io")+
 theme_minimal(base_family = "Roboto Condensed", base_size = 15)+
 theme(legend.position = "none")
```

## The world convergence in male life expectancy at birth since 1950

Data: UNPD World Population Prospects 2015 Revision



ikashnitsky.github.io

```
library(extrafont)
```

```
data(UNlocations)
```

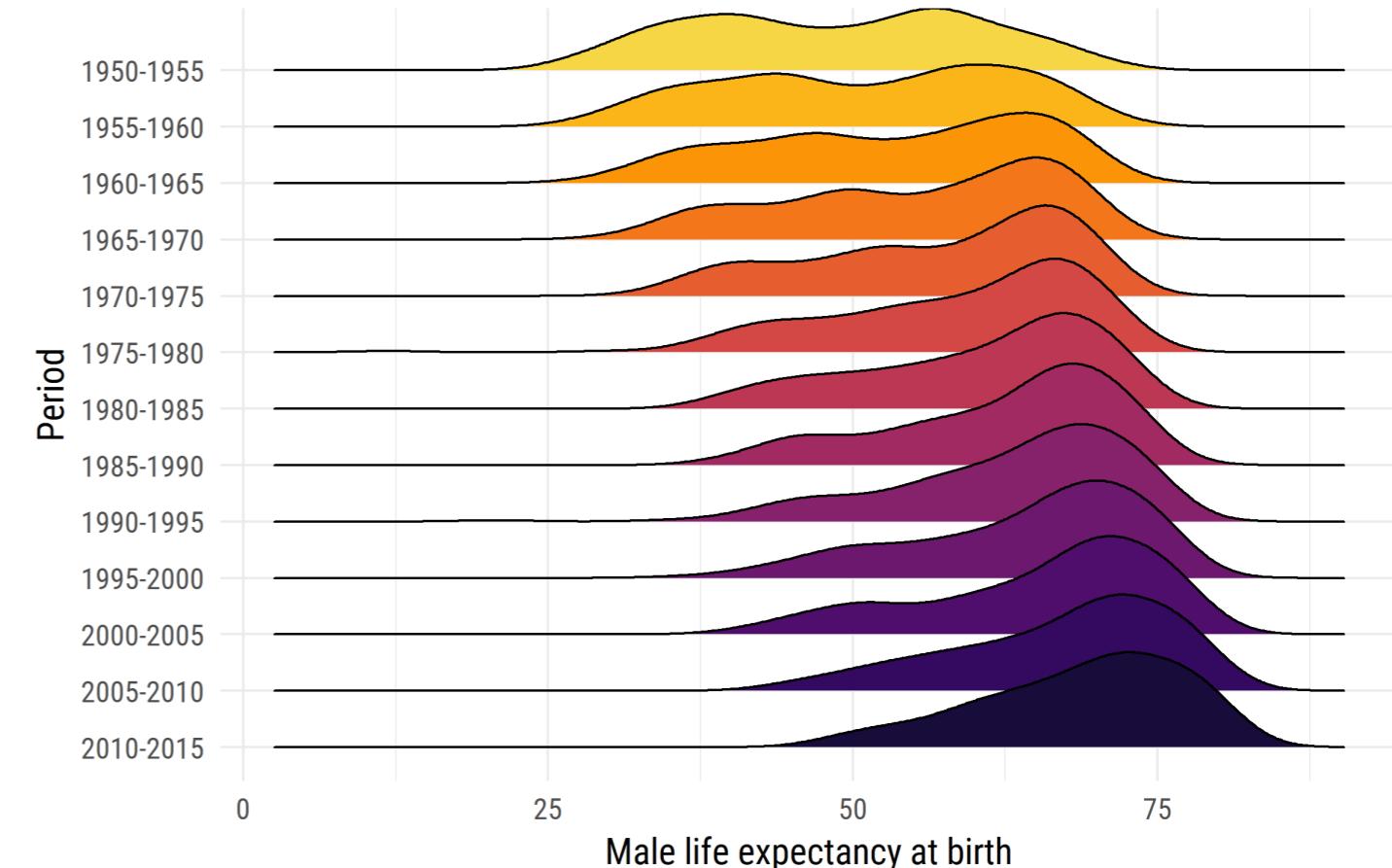
```
countries <- UNlocations %>%
 filter(location_type == 4) %>%
 transmute(name = name %>% paste()) %>%
 as_vector()
```

```
data(e0M)
```

```
e0M %>%
 filter(country %in% countries) %>%
 select(-last.observed) %>%
 gather(period, value, 3:15) %>%
 ggplot(aes(x = value, y = period %>% fct_rev()))+
 geom_joy(aes(fill = period))+
 scale_fill_viridis(discrete = T, option = "B", direction = -1,
 begin = .1, end = .9)+
 labs(x = "Male life expectancy at birth",
 y = "Period",
 title = "The world convergence in male life expectancy at birth since 1950",
 subtitle = "Data: UNPD World Population Prospects 2015 Revision",
 caption = "ikashnitsky.github.io")+
 theme_minimal(base_family = "Roboto Condensed", base_size = 15)+
 theme(legend.position = "none")
```

## The world convergence in male life expectancy at birth since 1950

Data: UNPD World Population Prospects 2015 Revision



ikashnitsky.github.io

# 第126章：通过示例回顾R语言

本主题旨在作为R语言的备忘录，无需文字说明，配以自解释的示例。

每个示例都尽可能简洁明了。

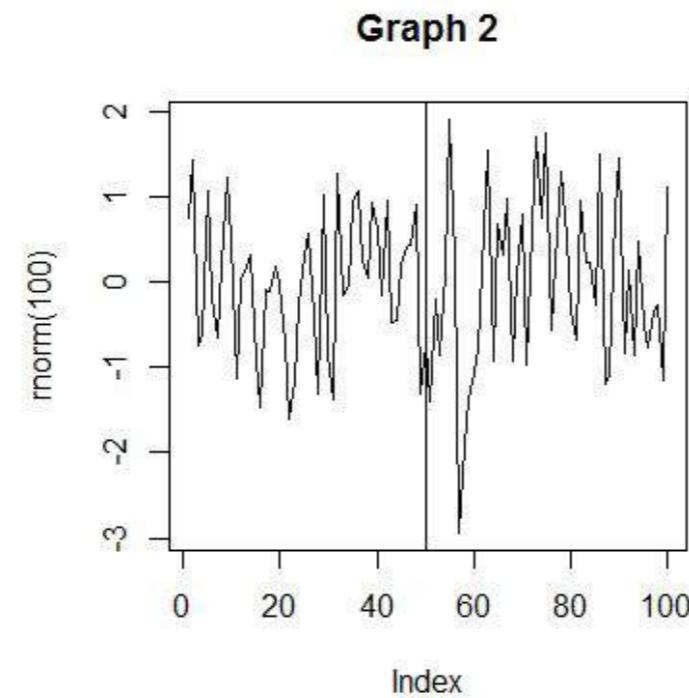
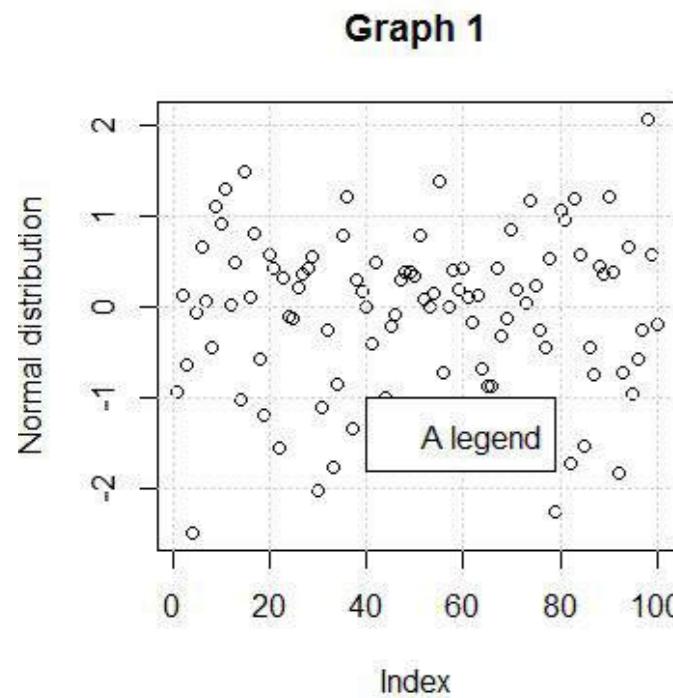
## 第126.1节：绘图（使用plot）

```
创建一个1行2列的格式
par(mfrow=c(1,2))

plot(rnorm(100), main = "图1", ylab = "正态分布"
grid()
legend(x = 40, y = -1, legend = "图例")

plot(rnorm(100), main = "图2", type = "l")
abline(v = 50)
```

结果：



## 第126.2节：常用函数

```
在向量中创建100个标准正态分布随机数
x <- rnorm(100, mean = 0, sd = 1)

计算向量长度
length(x)

计算均值
mean(x)

计算标准差
sd(x)

计算中位数
median(x)
```

# Chapter 126: R memento by examples

This topic is meant to be a memento about the R language without any text, with self-explanatory examples.

Each example is meant to be as succinct as possible.

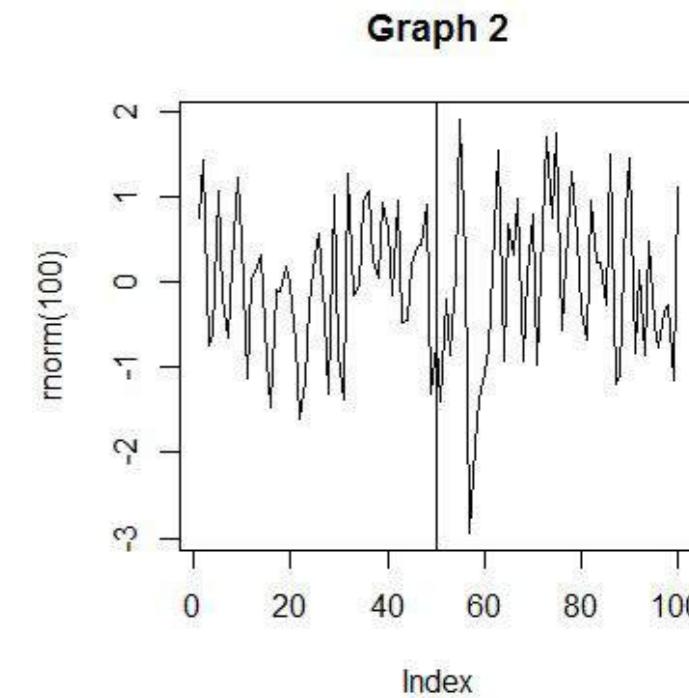
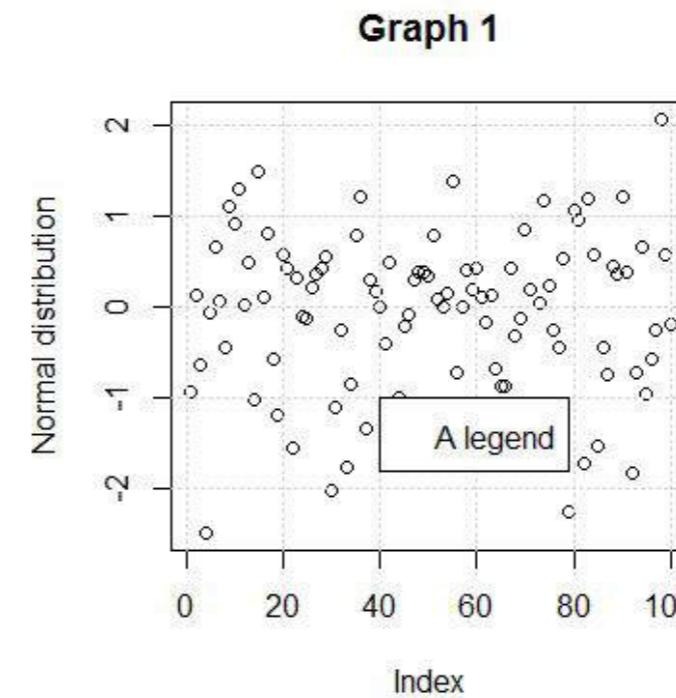
## Section 126.1: Plotting (using plot)

```
Creates a 1 row - 2 columns format
par(mfrow=c(1,2))

plot(rnorm(100), main = "Graph 1", ylab = "Normal distribution")
grid()
legend(x = 40, y = -1, legend = "A legend")

plot(rnorm(100), main = "Graph 2", type = "l")
abline(v = 50)
```

Result:



## Section 126.2: Commonly used functions

```
Create 100 standard normals in a vector
x <- rnorm(100, mean = 0, sd = 1)

Find the lenght of a vector
length(x)

Compute the mean
mean(x)

Compute the standard deviation
sd(x)

Compute the median value
median(x)
```

```
计算范围 (最小值, 最大值)
```

```
range(x)
```

```
求和可迭代对象
```

```
sum(x)
```

```
累积和 (x[1], x[1]+x[2], ...)
```

```
cumsum(x)
```

```
显示前3个元素
```

```
head(3, x)
```

```
显示最小值、第1四分位数、中位数、均值、第3四分位数、最大值
```

```
summary(x)
```

```
计算元素之间的连续差分
```

```
diff(x)
```

```
创建一个从1到10, 步长为1的序列
```

```
1:10
```

```
创建一个从1到10, 步长为0.1的序列
```

```
seq(1, 10, 0.1)
```

```
打印字符串
```

```
print("hello world")
```

```
Compute the range (min, max)
```

```
range(x)
```

```
Sum an iterable
```

```
sum(x)
```

```
Cumulative sum (x[1], x[1]+x[2], ...)
```

```
cumsum(x)
```

```
Display the first 3 elements
```

```
head(3, x)
```

```
Display min, 1st quartile, median, mean, 3rd quartile, max
```

```
summary(x)
```

```
Compute successive difference between elements
```

```
diff(x)
```

```
Create a range from 1 to 10 step 1
```

```
1:10
```

```
Create a range from 1 to 10 step 0.1
```

```
seq(1, 10, 0.1)
```

```
Print a string
```

```
print("hello world")
```

## 第126.3节：数据类型

### 向量

```
a <- c(1, 2, 3)
```

```
b <- c(4, 5, 6)
```

```
mean_ab <- (a + b) / 2
```

```
d <- c(1, 0, 1)
```

```
only_1_3 <- a[d == 1]
```

### 矩阵

```
mat <- matrix(c(1,2,3,4), nrow = 2, ncol = 2)
```

```
dimnames(mat) <- list(c(), c("a", "b", "c"))
```

```
mat[,] == mat
```

### 数据框

```
df <- data.frame(qualifiers = c("Buy", "Sell", "Sell"),
 symbols = c("AAPL", "MSFT", "GOOGL"),
 values = c(326.0, 598.3, 201.5))
```

```
df$symbols == df[[2]]
```

```
df$symbols == df[["symbols"]]
```

```
df[[2, 1]] == "AAPL"
```

### 列表

```
l <- list(a = 500, "aaa", 98.2)
```

```
length(l) == 3
```

```
class(l[1]) == "list"
```

```
class(l[[1]]) == "numeric"
```

```
class(l$a) == "numeric"
```

### 环境

```
env <- new.env()
```

```
env[["foo"]] = "bar"
```

```
env2 <- env
```

## Section 126.3: Data types

### Vectors

```
a <- c(1, 2, 3)
```

```
b <- c(4, 5, 6)
```

```
mean_ab <- (a + b) / 2
```

```
d <- c(1, 0, 1)
```

```
only_1_3 <- a[d == 1]
```

### Matrices

```
mat <- matrix(c(1,2,3,4), nrow = 2, ncol = 2)
```

```
dimnames(mat) <- list(c(), c("a", "b", "c"))
```

```
mat[,] == mat
```

### Dataframes

```
df <- data.frame(qualifiers = c("Buy", "Sell", "Sell"),
```

```
 symbols = c("AAPL", "MSFT", "GOOGL"),
```

```
 values = c(326.0, 598.3, 201.5))
```

```
df$symbols == df[[2]]
```

```
df$symbols == df[["symbols"]]
```

```
df[[2, 1]] == "AAPL"
```

### Lists

```
l <- list(a = 500, "aaa", 98.2)
```

```
length(l) == 3
```

```
class(l[1]) == "list"
```

```
class(l[[1]]) == "numeric"
```

```
class(l$a) == "numeric"
```

### Environments

```
env <- new.env()
```

```
env[["foo"]] = "bar"
```

```
env2 <- env
```

```
env2[["foo"]] = "BAR"

env[["foo"]] == "BAR"
get("foo", envir = env) == "BAR"
rm("foo", envir = env)
env[["foo"]] == NULL
```

```
env2[["foo"]] = "BAR"

env[["foo"]] == "BAR"
get("foo", envir = env) == "BAR"
rm("foo", envir = env)
env[["foo"]] == NULL
```

# 第127章：更新R版本

安装或更新您的软件将获得新功能和错误修复。更新您的R安装可以通过几种方式完成。一种简单的方法是访问R网站并下载适合您系统的最新版本。

## 第127.1节：从R网站安装

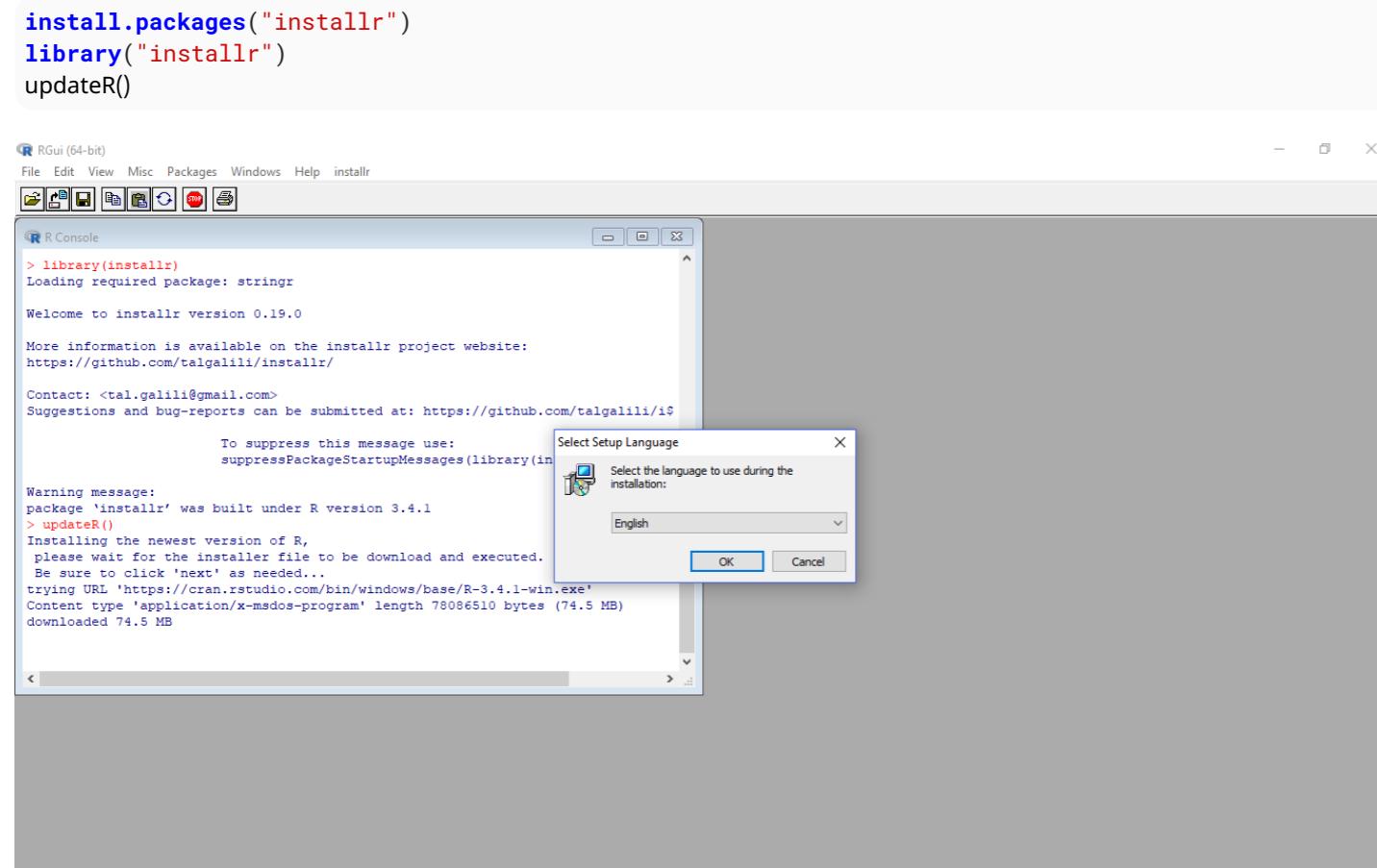
要获取最新版本，请访问<https://cran.r-project.org/>并下载适合您操作系统的文件。打开下载的文件并按照屏幕上的安装步骤操作。除非您想更改某些行为，否则所有设置均可保持默认。

## 第127.2节：使用installr包在R内部更新

您也可以通过使用一个名为installr的实用包在R内部更新R。

打开R控制台（不是RStudio，RStudio中无法使用此方法），运行以下代码安装该包并启动更新。

```
install.packages("installr")
library("installr")
updateR()
```



## 第127.3节：决定旧版本包的处理

安装完成后，点击完成按钮。

现在它会询问是否要将旧版本R中的包复制到新版本R中。一旦选择是，所有包都会被复制到新版本R中。

# Chapter 127: Updating R version

Installing or Updating your Software will give access to new features and bug fixes. Updating your R installation can be done in a couple of ways. One Simple way is go to [R website](https://cran.r-project.org/) and download the latest version for your system.

## Section 127.1: Installing from R Website

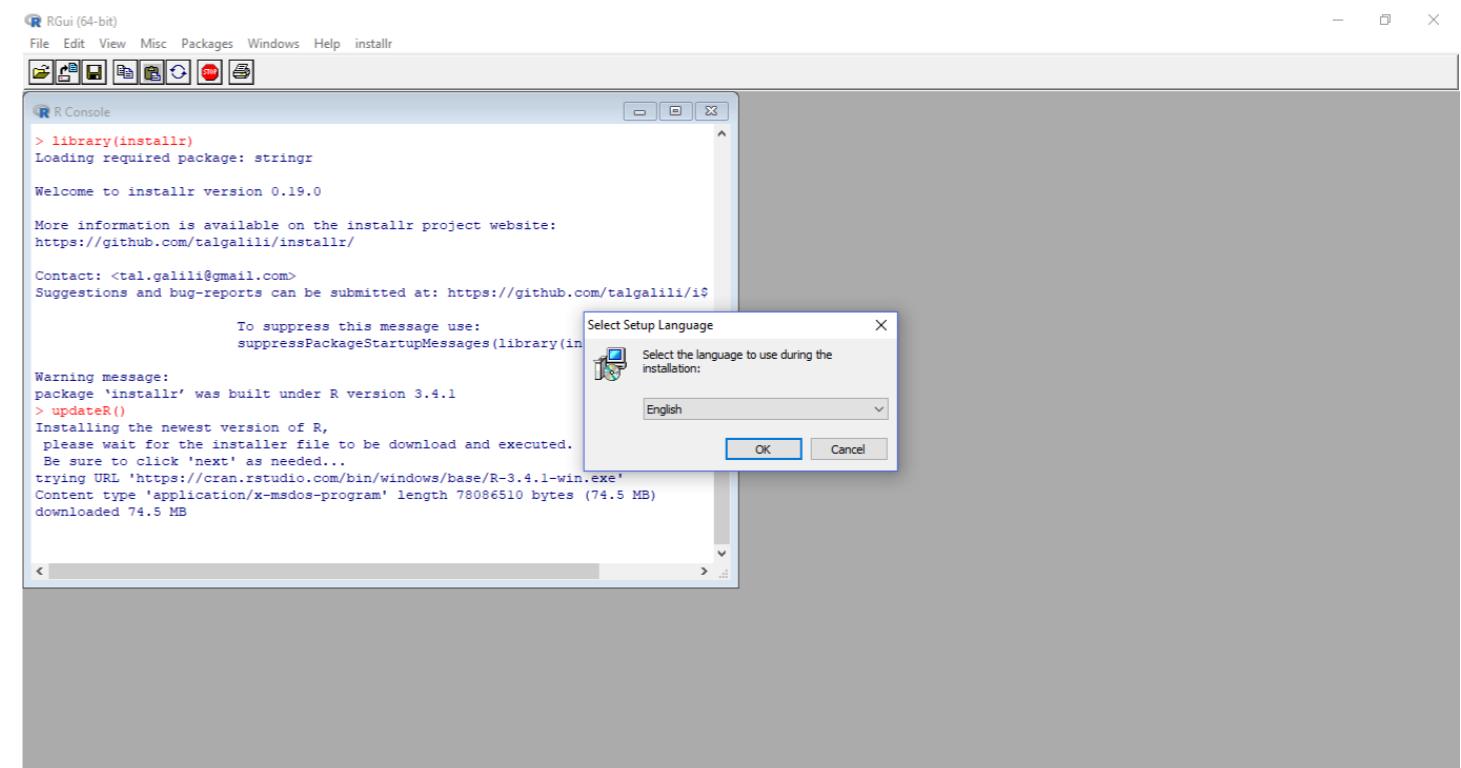
To get the latest release go to <https://cran.r-project.org/> and download the file for your operating system. Open the downloaded file and follow the on-screen installation steps. All the settings can be left on default unless you want to change a certain behaviour.

## Section 127.2: Updating from within R using installr Package

You can also update R from within R by using a handy package called **installr**.

Open R Console (NOT RStudio, this doesn't work from RStudio) and run the following code to install the package and initiate update.

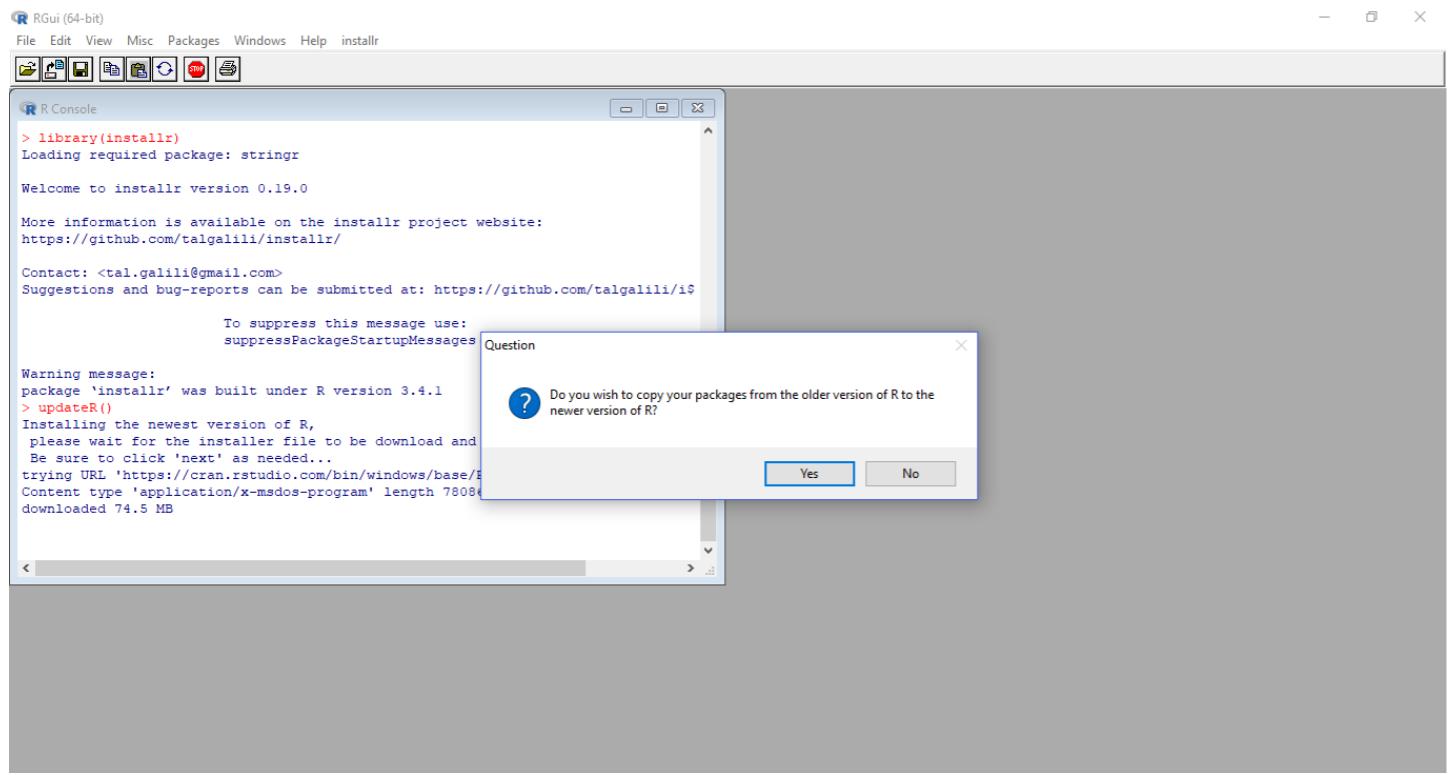
```
install.packages("installr")
library("installr")
updateR()
```



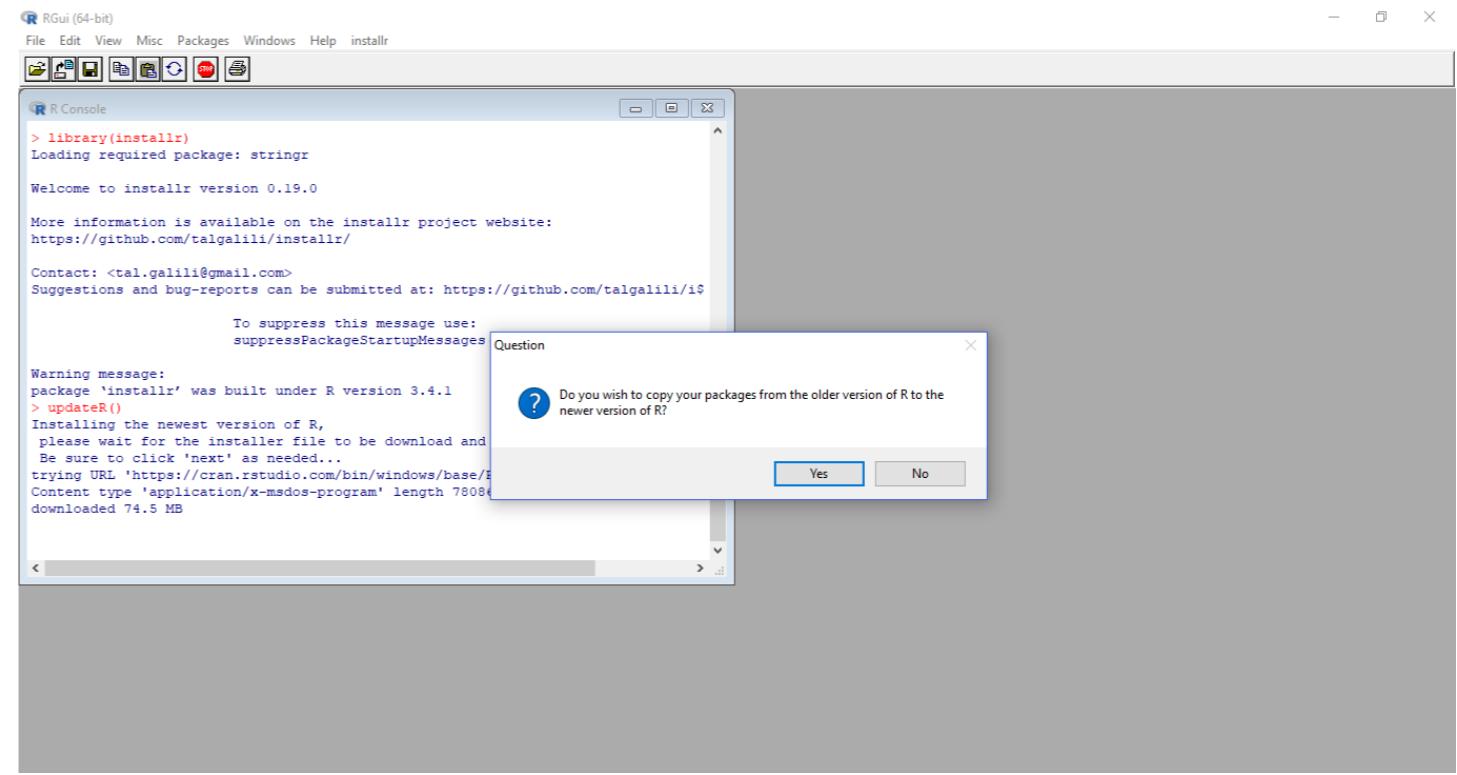
## Section 127.3: Deciding on the old packages

Once the installation is finished click the Finish button.

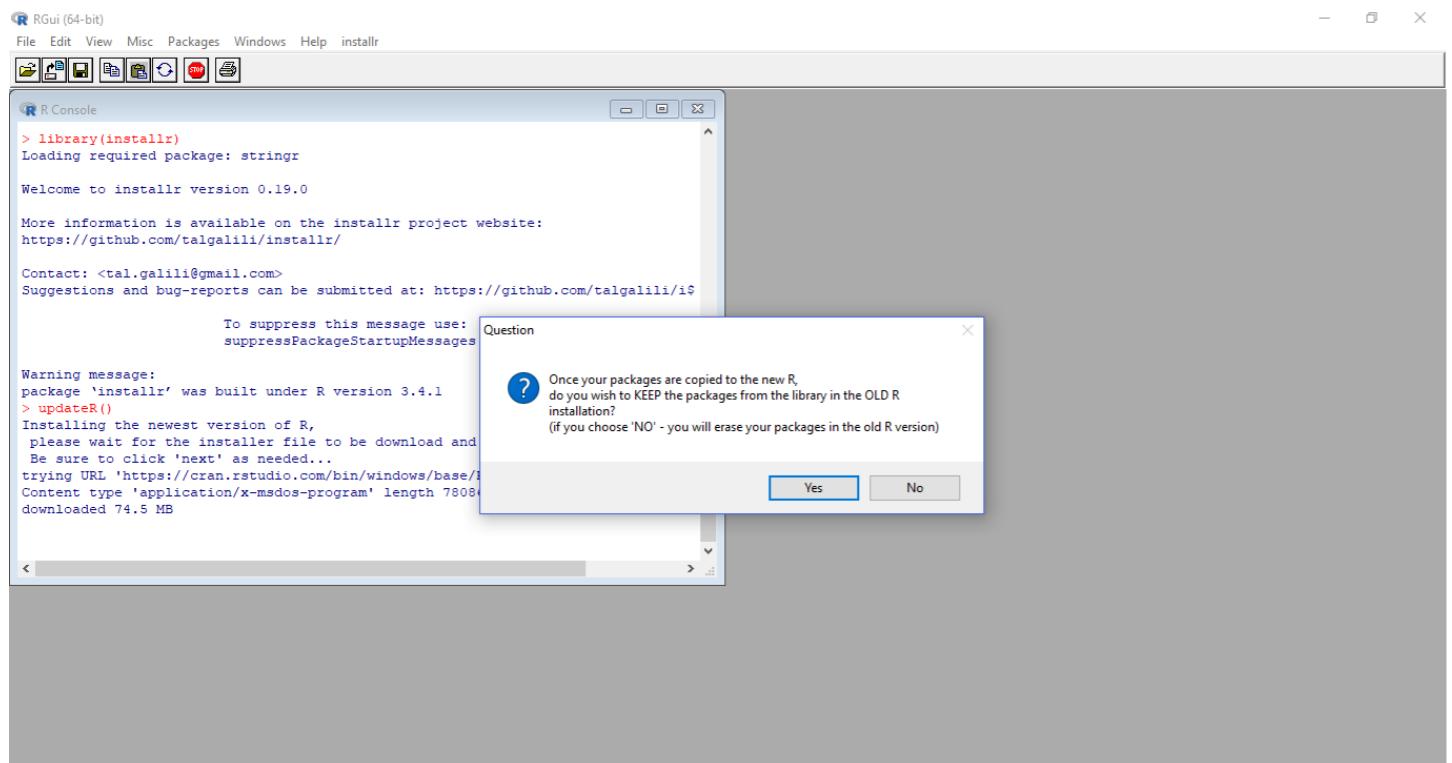
Now it asks if you want to copy your packages fro the older version of R to Newer version of R. Once you choose yes all the package are copied to the newer version of R.



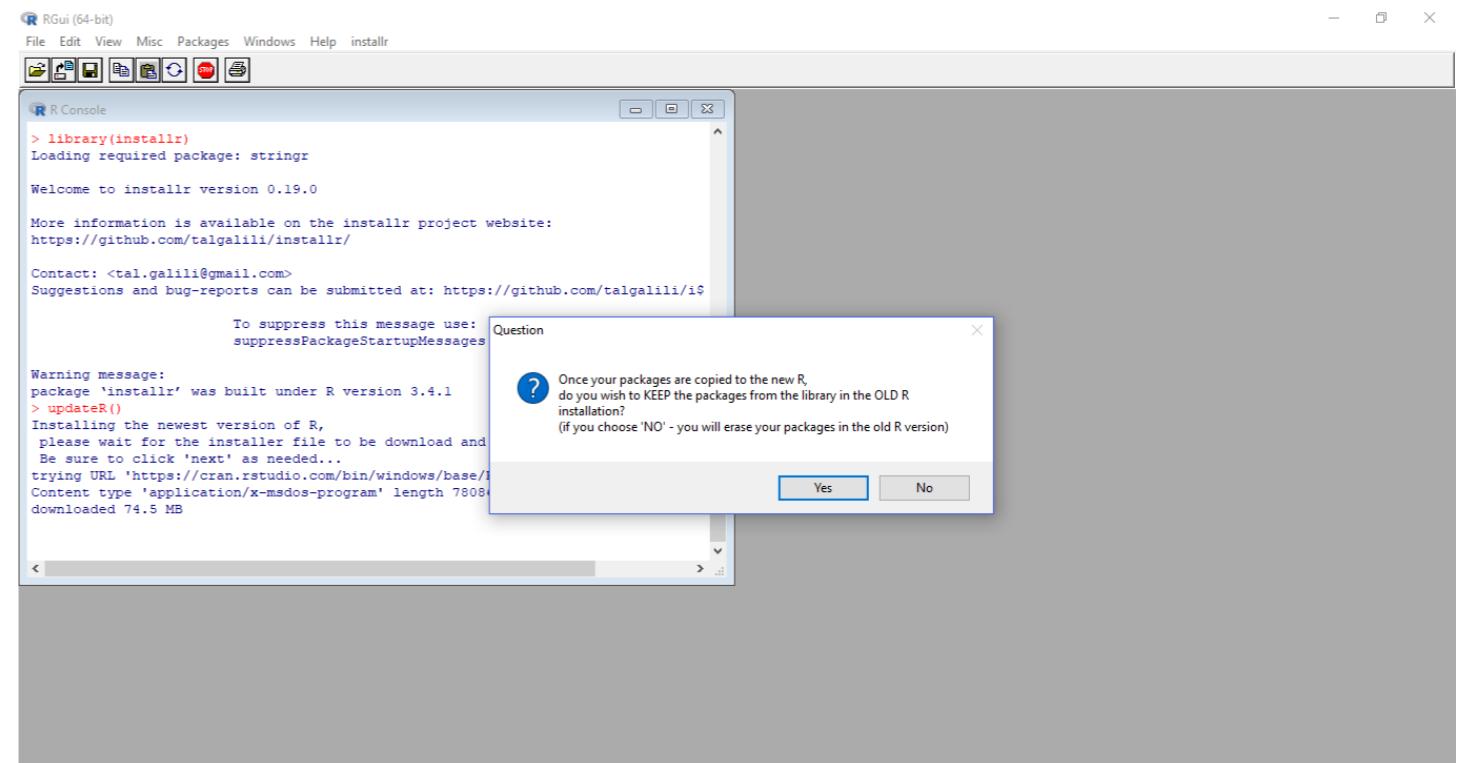
之后你可以选择是否保留旧包或删除它们。



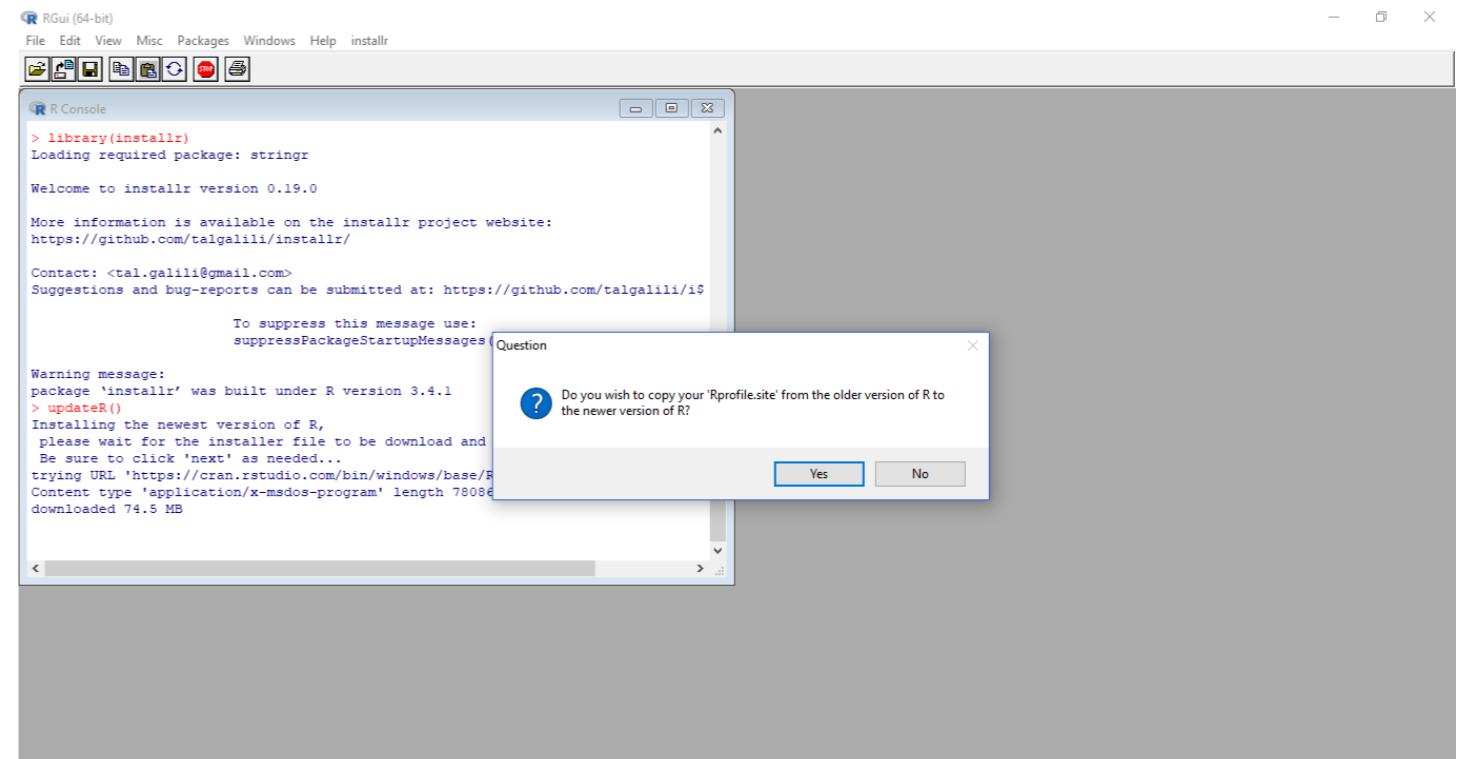
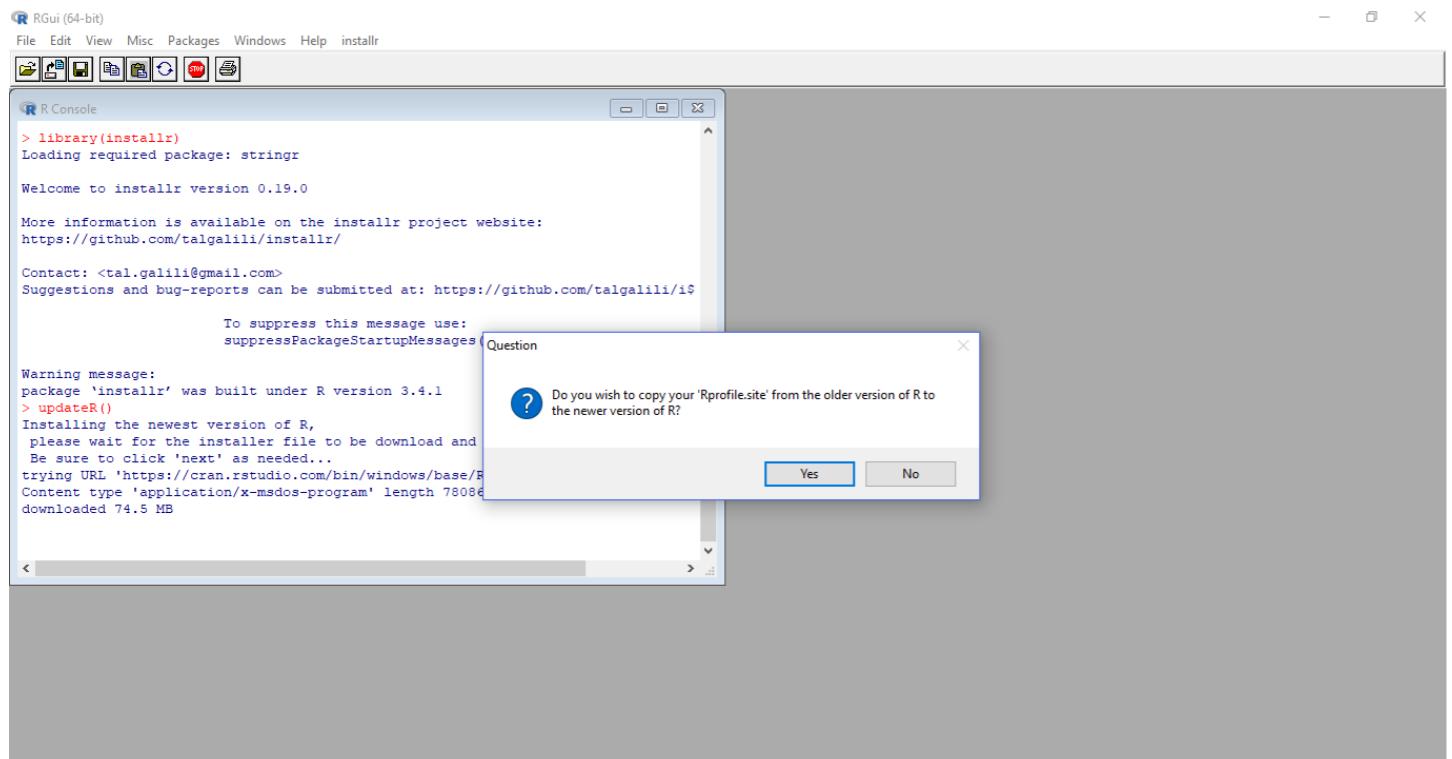
After that you can choose if you still want to keep the old packages or delete.



你甚至可以将旧版本的Rprofile.site移动过来，以保留所有自定义设置。

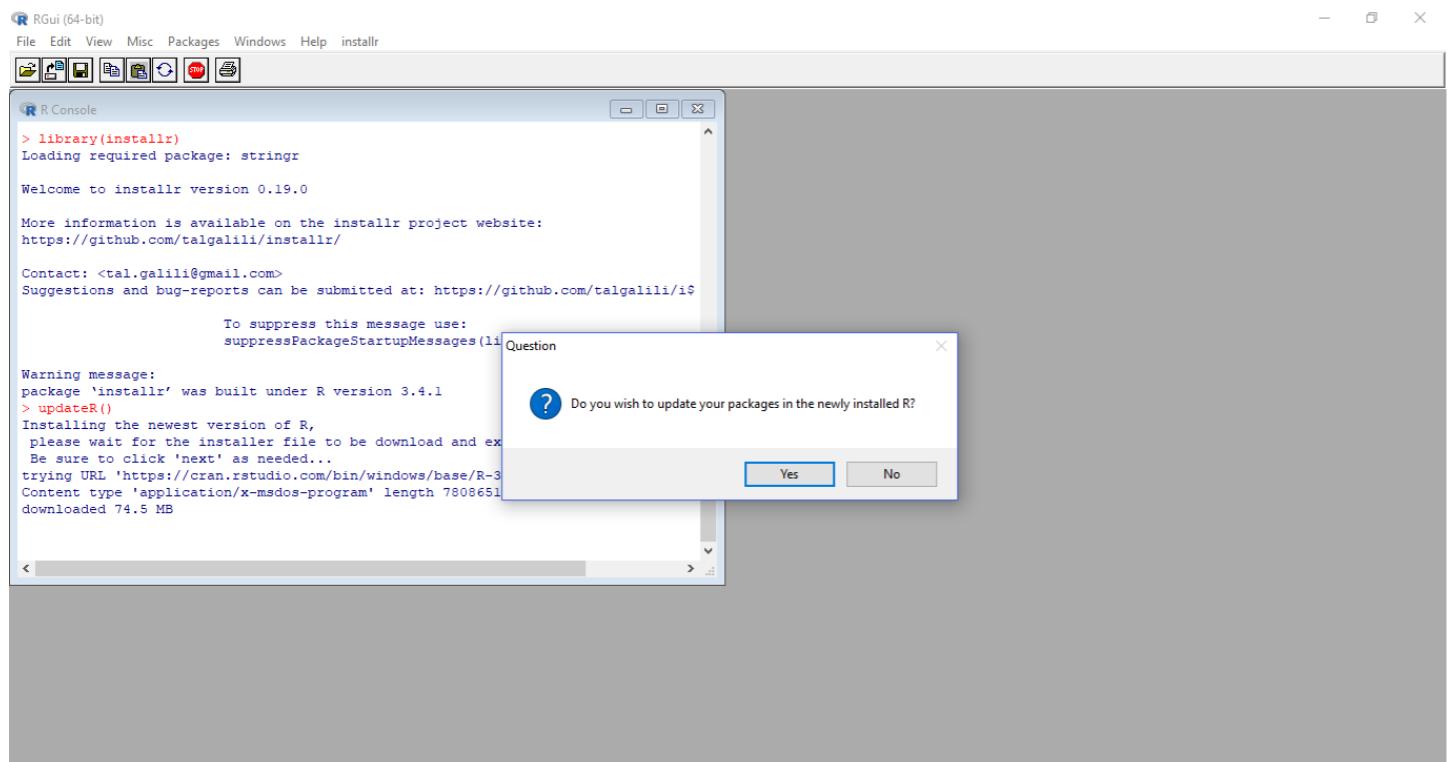


You can even move your Rprofile.site from older version to keep all your customised settings.



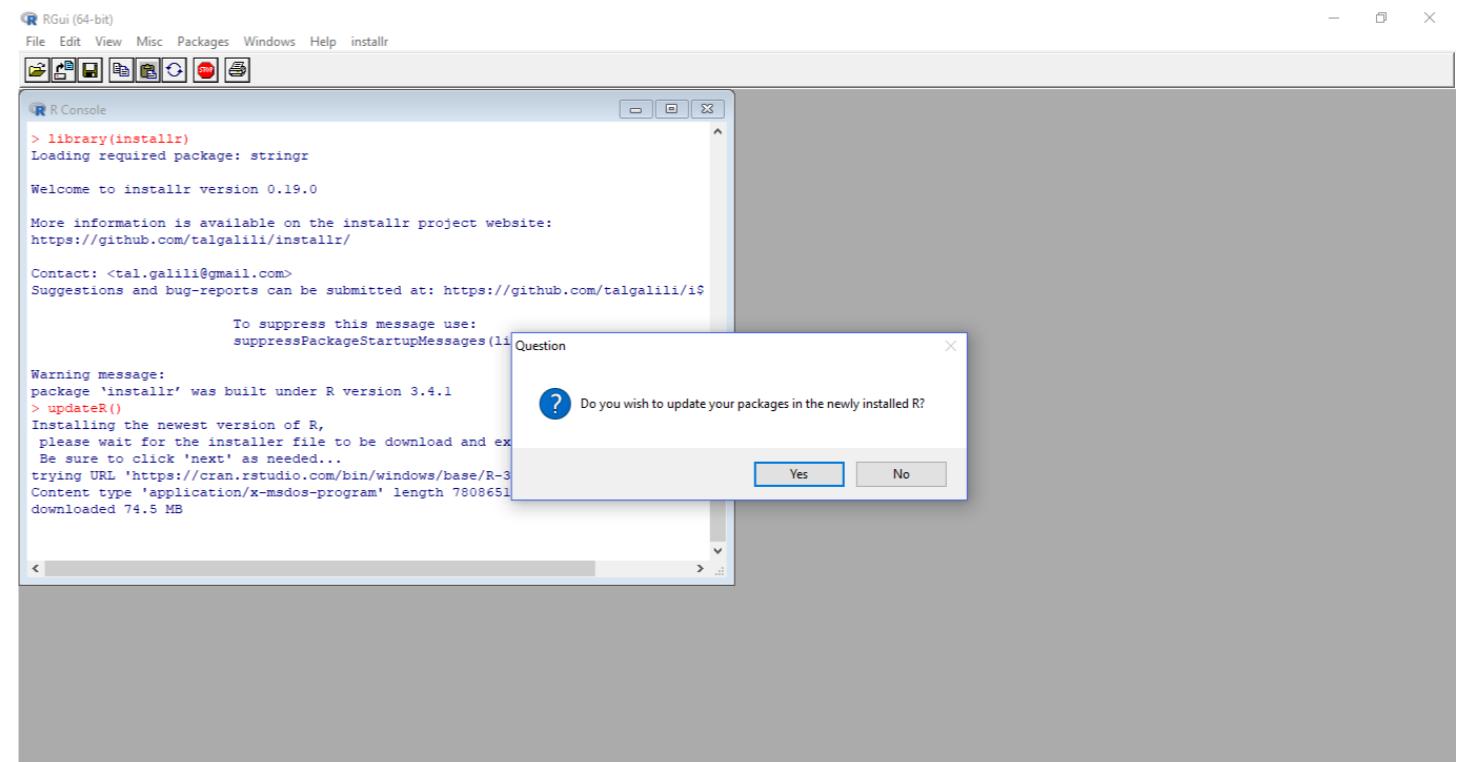
## 第127.4节：更新包

R更新完成后，你可以更新已安装的包。



## Section 127.4: Updating Packages

You can update your installed packages once the updating of R is done.



完成后，重启R，尽情探索吧。

## 第127.5节：检查R版本

您可以通过控制台检查R版本

版本

Once its done Restart R and enjoy exploring.

## Section 127.5: Check R Version

You can check R Version using the console

version

# 鸣谢

非常感谢所有来自Stack Overflow Documentation的人员帮助提供此内容，  
更多更改可以发送至[web@petercv.com](mailto:web@petercv.com)以发布或更新新内容

<a href="#">42</a>	第1、2、5、6、8、17、21、26、29、39、76、92和100章
<a href="#">阿加兹·侯赛因</a>	第20章
<a href="#">阿卜杜</a>	第30章
<a href="#">abhiieor</a>	第6章
<a href="#">农学家</a>	第39章
<a href="#">akraf</a>	第1、28和120章
<a href="#">akrun</a>	第23、52和54章
<a href="#">AkselA</a>	第14、33和105章
<a href="#">Ale</a>	第二章
<a href="#">亚历克斯</a>	第18章、第28章、第92章、第94章和第115章
<a href="#">亚历山德鲁·帕皮乌</a>	第21章
<a href="#">阿列克谢·希克洛马诺夫</a>	第26章和第99章
<a href="#">alexis_laz</a>	第39章
<a href="#">阿里汗·兹纳</a>	第77章
<a href="#">阿利斯泰尔</a>	第12、13、14、20、21、33、55和58章
<a href="#">alko989</a>	第41章
<a href="#">艾伦·王</a>	第23章
<a href="#">安德里亚·奇里洛</a>	第1、28和92章
<a href="#">安德里亚·伊安尼</a>	第9和18章
<a href="#">安德鲁·布扎</a>	第一章
<a href="#">安德鲁·布莱克</a>	第45章和第56章
<a href="#">安德烈·阿金辛</a>	第21章和第85章
<a href="#">安杰洛</a>	第55章
<a href="#">阿尔捷姆·克列夫佐夫</a>	第36章和第38章
<a href="#">阿伦·巴拉克里希南</a>	第41章
<a href="#">阿希什</a>	第88章
<a href="#">阿蒂什</a>	第21章
<a href="#">斧头手</a>	第2、5、21、65和66章
<a href="#">巴蒂斯特</a>	第28章
<a href="#">巴克利BG</a>	第9章
<a href="#">巴特克塔尔塔努斯</a>	第7、11和37章
<a href="#">巴塔尼切克</a>	第3章和第18章
<a href="#">本·博尔克</a>	第22章、第50章、第59章、第67章和第70章
<a href="#">本杰明</a>	第21章、第29章、第51章、第68章和第100章
<a href="#">blmoore</a>	第28章
<a href="#">Boysenb3rry</a>	第86章
<a href="#">Carl</a>	第22章
<a href="#">卡尔·维特霍夫特</a>	第20章
<a href="#">卡洛斯·奇内利</a>	第18、20、21、22、27和92章
<a href="#">卡森</a>	第33章
<a href="#">灾难性的</a>	第20和79章
<a href="#">CClaire</a>	第47章
<a href="#">cderman</a>	第84章
<a href="#">克里斯托夫·D.</a>	第18、27和47章
<a href="#">CL.</a>	第41章
<a href="#">CMichael</a>	第21章
<a href="#">无外套的</a>	第12、13、14和36章

# Credits

Thank you greatly to all the people from Stack Overflow Documentation who helped provide this content,  
more changes can be sent to [web@petercv.com](mailto:web@petercv.com) for new content to be published or updated

<a href="#">42</a>	Chapters 1, 2, 5, 6, 8, 17, 21, 26, 29, 39, 76, 92 and 100
<a href="#">Aaghaz Hussain</a>	Chapter 20
<a href="#">Abdou</a>	Chapter 30
<a href="#">abhiieor</a>	Chapter 6
<a href="#">Agriculturist</a>	Chapter 39
<a href="#">akraf</a>	Chapters 1, 28 and 120
<a href="#">akrun</a>	Chapters 23, 52 and 54
<a href="#">AkselA</a>	Chapters 14, 33 and 105
<a href="#">Ale</a>	Chapter 2
<a href="#">Alex</a>	Chapters 18, 28, 92, 94 and 115
<a href="#">Alexandru Papiu</a>	Chapter 21
<a href="#">Alexey Shiklomanov</a>	Chapters 26 and 99
<a href="#">alexis_laz</a>	Chapter 39
<a href="#">Alihan Zihna</a>	Chapter 77
<a href="#">alistaire</a>	Chapters 12, 13, 14, 20, 21, 33, 55 and 58
<a href="#">alko989</a>	Chapter 41
<a href="#">Allen Wang</a>	Chapter 23
<a href="#">Andrea Cirillo</a>	Chapters 1, 28 and 92
<a href="#">Andrea Ianni</a>	Chapters 9 and 18
<a href="#">Andrew Bréza</a>	Chapter 1
<a href="#">Andrew Bryk</a>	Chapters 45 and 56
<a href="#">AndreyAkinshin</a>	Chapters 21 and 85
<a href="#">Angelo</a>	Chapter 55
<a href="#">Artem Klevtsov</a>	Chapters 36 and 38
<a href="#">Arun Balakrishnan</a>	Chapter 41
<a href="#">Ashish</a>	Chapter 88
<a href="#">Atish</a>	Chapter 21
<a href="#">Axeman</a>	Chapters 2, 5, 21, 65 and 66
<a href="#">baptiste</a>	Chapter 28
<a href="#">BarkleyBG</a>	Chapter 9
<a href="#">bartektartanus</a>	Chapters 7, 11 and 37
<a href="#">Batanichek</a>	Chapters 3 and 18
<a href="#">Ben Bolker</a>	Chapters 22, 50, 59, 67 and 70
<a href="#">Benjamin</a>	Chapters 21, 29, 51, 68 and 100
<a href="#">blmoore</a>	Chapter 28
<a href="#">Boysenb3rry</a>	Chapter 86
<a href="#">Carl</a>	Chapter 22
<a href="#">Carl Witthoft</a>	Chapter 20
<a href="#">Carlos Cinelli</a>	Chapters 18, 20, 21, 22, 27 and 92
<a href="#">Carson</a>	Chapter 33
<a href="#">catastrophic</a>	Chapters 20 and 79
<a href="#">CClaire</a>	Chapter 47
<a href="#">cderman</a>	Chapter 84
<a href="#">Christophe D.</a>	Chapters 18, 27 and 47
<a href="#">CL.</a>	Chapter 41
<a href="#">CMichael</a>	Chapter 21
<a href="#">coatless</a>	Chapters 12, 13, 14 and 36

[CptNemo](#)  
[克雷格·弗米尔](#)  
[农作物](#)  
[d.b](#)  
[dash2](#)  
[DataTx](#)  
[Dave2e](#)  
[DaveRGP](#)  
[大卫](#)  
[大卫·阿伦堡](#)  
[大卫·莱尔](#)  
[大卫·罗宾逊](#)  
[Dawny33](#)  
[dayne](#)  
[迪恩·麦格雷戈](#)  
[德温·麦基里](#)  
[德沃P](#)  
[迪尔克·埃尔布特尔](#)  
[dmail](#)  
[dotancohen](#)  
[DrPositron](#)  
[EDI](#)  
[egnha](#)  
[埃里克·勒库特尔](#)  
[FisherDisinformation](#)  
[弗洛里安](#)  
[折叠染色质](#)  
[弗兰克](#)  
[G5W](#)  
[加文·辛普森](#)  
[乔治·博恩布赖特](#)  
[乔治·奥斯·K](#)  
[格伦·莫特里](#)  
[格雷戈尔](#)  
[Hack](#)  
[哈里祖安·努拉兹曼](#)  
[herbaman](#)  
[高带宽](#)  
[ikashnitsky](#)  
[雅普](#)  
[詹姆斯·埃尔德菲尔德](#)  
[jameselmore](#)  
[Jav](#)  
[jcb](#)  
[杰夫](#)  
[杰罗米·安格林](#)  
[JHowlX](#)  
[josliber](#)  
[Joy](#)  
[JulioSergio](#)  
[JvH](#)  
[LF](#)

[CptNemo](#)  
[Craig Vermeer](#)  
[Crops](#)  
[d.b](#)  
[dash2](#)  
[DataTx](#)  
[Dave2e](#)  
[DaveRGP](#)  
[David](#)  
[David Arenburg](#)  
[David Leal](#)  
[David Robinson](#)  
[Dawny33](#)  
[dayne](#)  
[Dean MacGregor](#)  
[Derwin McGeary](#)  
[DeveauP](#)  
[Dirk Eddelbuettel](#)  
[dmail](#)  
[dotancohen](#)  
[DrPositron](#)  
[EDI](#)  
[egnha](#)  
[Eric Lecoutre](#)  
[FisherDisinformation](#)  
[Florian](#)  
[FoldedChromatin](#)  
[Frank](#)  
[G5W](#)  
[Gavin Simpson](#)  
[George Bonebright](#)  
[Giorgos K](#)  
[Glen Moutrie](#)  
[Gregor](#)  
[Hack](#)  
[Hairizuan Noorazman](#)  
[herbaman](#)  
[highBandWidth](#)  
[ikashnitsky](#)  
[Jaap](#)  
[James Elderfield](#)  
[jameselmore](#)  
[Jav](#)  
[jcb](#)  
[Jeff](#)  
[Jeremy Anglim](#)  
[JHowlX](#)  
[josliber](#)  
[Joy](#)  
[JulioSergio](#)  
[JvH](#)  
[LF](#)

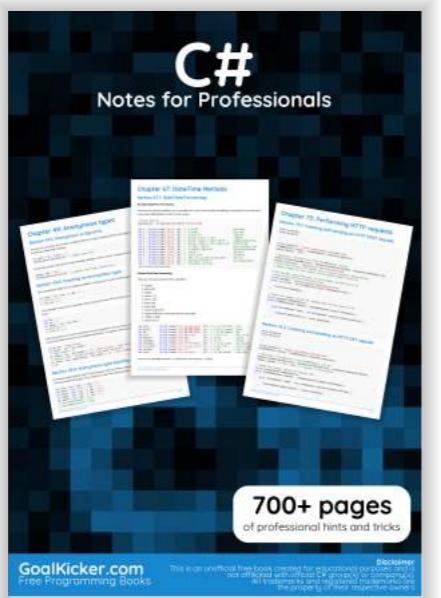
<a href="#">K.Daisey</a>	第20章、第26章、第38章和第62章	<a href="#">K.Daisey</a>	Chapters 20, 26, 38 and 62
<a href="#">kaksat</a>	第18章、第41章、第92章和第105章	<a href="#">kaksat</a>	Chapters 18, 41, 92 and 105
<a href="#">Karolis_Koncevičius</a>	第27章、第37章、第76章、第83章、第87章和第101章	<a href="#">Karolis_Koncevičius</a>	Chapters 27, 37, 76, 83, 87 and 101
<a href="#">Karsten_W.</a>	第11章	<a href="#">Karsten_W.</a>	Chapter 11
<a href="#">土豆沙拉</a>	第一章	<a href="#">kartoffelsalat</a>	Chapter 1
<a href="#">凯·布罗德森</a>	第21章	<a href="#">Kay_Brodersen</a>	Chapter 21
<a href="#">kdopen</a>	第1章和第6章	<a href="#">kdopen</a>	Chapters 1 and 6
<a href="#">肯·S.</a>	第92章	<a href="#">Ken_S.</a>	Chapter 92
<a href="#">kitman0804</a>	第66章和第92章	<a href="#">kitman0804</a>	Chapters 66 and 92
<a href="#">kneijenhuijs</a>	第8章、第12章、第29章、第41章和第70章	<a href="#">kneijenhuijs</a>	Chapters 8, 12, 29, 41 and 70
<a href="#">L.V.拉奥</a>	第1章、第2章、第13章、第17章、第25章、第27章、第39章和第46章	<a href="#">L.V.Rao</a>	Chapters 1, 2, 13, 17, 25, 27, 39 and 46
<a href="#">leogama</a>	第23章	<a href="#">leogama</a>	Chapter 23
<a href="#">Imckeogh</a>	第2章和第41章	<a href="#">Imckeogh</a>	Chapters 2 and 41
<a href="#">lmo</a>	第11章、第38章、第57章、第60章、第66章和第92章	<a href="#">lmo</a>	Chapters 11, 38, 57, 60, 66 and 92
<a href="#">loki</a>	第28章、第38章、第48章、第71章、第95章和第110章	<a href="#">loki</a>	Chapters 28, 38, 48, 71, 95 and 110
<a href="#">Lovy</a>	第1章、第43章和第126章	<a href="#">Lovy</a>	Chapters 1, 43 and 126
<a href="#">马利克·侯赛因</a>	第23章	<a href="#">Mallick_Hossain</a>	Chapter 23
<a href="#">马尔钦·科辛斯基</a>	第72章	<a href="#">Marcin_Kosiński</a>	Chapter 72
<a href="#">马里奥</a>	第96章	<a href="#">Mario</a>	Chapter 96
<a href="#">马丁</a>	第26章和第48章	<a href="#">maRtin</a>	Chapters 26 and 48
<a href="#">马丁·施梅尔策</a>	第61章和第83章	<a href="#">Martin_Schmelzer</a>	Chapters 61 and 83
<a href="#">马克西米连·科尔</a>	第89章	<a href="#">Maximilian_Kohl</a>	Chapter 89
<a href="#">迈克尔·奇里科</a>	第12、13、14和100章	<a href="#">MichaelChirico</a>	Chapters 12, 13, 14 and 100
<a href="#">micstr</a>	第23章	<a href="#">micstr</a>	Chapter 23
<a href="#">米哈</a>	第37、49和92章	<a href="#">Miha</a>	Chapters 37, 49 and 92
<a href="#">mrip</a>	第39章	<a href="#">mrip</a>	Chapter 39
<a href="#">munirbe</a>	第106章	<a href="#">munirbe</a>	Chapter 106
<a href="#">七海</a>	第85章	<a href="#">Nanami</a>	Chapter 85
<a href="#">纳伦德拉</a>	第28章、第41章和第55章	<a href="#">narendra</a>	Chapters 28, 41 and 55
<a href="#">内森·沃斯</a>	第23章和第29章	<a href="#">Nathan_Werth</a>	Chapters 23 and 29
<a href="#">nrussell</a>	第10章、第51章和第68章	<a href="#">nrussell</a>	Chapters 10, 51 and 68
<a href="#">NWaters</a>	第30章	<a href="#">NWaters</a>	Chapter 30
<a href="#">omar</a>	第29章和第68章	<a href="#">omar</a>	Chapters 29 and 68
<a href="#">oshun</a>	第23章	<a href="#">oshun</a>	Chapter 23
<a href="#">PAC</a>	第20章、第100章和第116章	<a href="#">PAC</a>	Chapters 20, 100 and 116
<a href="#">潘卡杰·夏尔马</a>	第80章和第82章	<a href="#">Pankaj_Sharma</a>	Chapters 80 and 82
<a href="#">帕菲</a>	第92章	<a href="#">Parfait</a>	Chapter 92
<a href="#">彼得·洪堡</a>	第9章和第41章	<a href="#">Peter_Humburg</a>	Chapters 9 and 41
<a href="#">皮埃尔·拉福图恩</a>	第22章	<a href="#">Pierre_Lafortune</a>	Chapter 22
<a href="#">波尔卡舞</a>	第1章、第21章和第92章	<a href="#">polka</a>	Chapters 1, 21 and 92
<a href="#">普拉贾迪蒂亚·达斯</a>	第41章	<a href="#">Pragyaditya_Das</a>	Chapter 41
<a href="#">Psidom</a>	第31章	<a href="#">Psidom</a>	Chapter 31
<a href="#">Qaswed</a>	第5章	<a href="#">Qaswed</a>	Chapter 5
<a href="#">拉胡尔·赛尼</a>	第一章	<a href="#">Rahul_Saini</a>	Chapter 1
<a href="#">拉吉·帕德马纳班</a>	第41章	<a href="#">Raj_Padmanabhan</a>	Chapter 41
<a href="#">Rappster</a>	第73章	<a href="#">Rappster</a>	Chapter 73
<a href="#">rcorty</a>	第118章	<a href="#">rcorty</a>	Chapter 118
<a href="#">撤回和退休</a>	第一章	<a href="#">RetractedAndRetired</a>	Chapter 1
<a href="#">罗伯特</a>	第3、11、22、26、29、33、51和77章	<a href="#">Robert</a>	Chapters 3, 11, 22, 26, 29, 33, 51 and 77
<a href="#">罗伯特·麦克</a>	第44章	<a href="#">RobertMc</a>	Chapter 44
<a href="#">罗宾·格滕巴赫</a>	第11章	<a href="#">Robin_Gertenbach</a>	Chapter 11
<a href="#">罗尔·霍格弗斯特</a>	第92章和第114章	<a href="#">Roel_Hogervorst</a>	Chapters 92 and 114
<a href="#">拉塞尔·皮尔斯</a>	第1、3、10、38、40、47和96章	<a href="#">russellpierce</a>	Chapters 1, 3, 10, 38, 40, 47 and 96

[萨姆·菲尔克](#)  
[萨蒂什](#)  
[scoa](#)  
[seasmith](#)  
[肖恩·米汉](#)  
[smci](#)  
[SommerEngineering](#)  
[索姆亚·S·马尼安](#)  
[斯佩斯德曼](#)  
[斯坦尼卡姆](#)  
[斯特迪](#)  
[史蒂夫·科林](#)  
[苏梅德](#)  
[孙比](#)  
[SymbolixAU](#)  
[symbolrush](#)  
[takje](#)  
[塔尔·加利利 \(Tal Galili\)](#)  
[TARehman](#)  
[tenCupMaximum](#)  
[天思拜](#)  
[阿伦](#)  
[晚邮](#)  
[托马斯](#)  
[蒂姆·科克](#)  
[TriskalJM](#)  
[tuomastik](#)  
[翁贝托](#)  
[user2100721](#)  
[user890739](#)  
[USER\\_1](#)  
[乌韦](#)  
[Vedda](#)  
[WAF](#)  
[while](#)  
[YCR](#)  
[云清](#)  
[zacdav](#)  
[zelite](#)  
[zx8754](#)

第20和21章  
第5章  
第18章和第55章  
第21章、第66章和第68章  
第21章  
第11章  
第30章和第93章  
第42章  
第21章  
第23章  
第76章  
第1、8、13、15、16、17、23、29、30、45、66、78和90章  
第18、23和77章  
第24章  
第12、14、33、41、47、50、53、60和114章  
第26和33章  
第11和60章  
第107章  
第117章  
第46章和第63章  
第3章、第19章和第30章  
第1章、第12章、第28章、第66章和第77章  
第14章和第19章  
第6章、第18章、第76章和第92章  
第54章  
第21章  
第47章  
第121章  
第18、21和114章  
第41章  
第22和97章  
第6章  
第22章  
第22章  
第一章  
第59、61、98和109章  
第28章  
第68和96章  
第20章  
第41章和第96章

[Sam Firke](#)  
[Sathish](#)  
[scoa](#)  
[seasmith](#)  
[Shawn Mehan](#)  
[smci](#)  
[SommerEngineering](#)  
[Sowmya S. Manian](#)  
[Spacedman](#)  
[stanekam](#)  
[Stedy](#)  
[Steve\\_Corrin](#)  
[Sumedh](#)  
[Sun Bee](#)  
[SymbolixAU](#)  
[symbolrush](#)  
[takje](#)  
[Tal Galili](#)  
[TARehman](#)  
[tenCupMaximum](#)  
[Tensibai](#)  
[theArun](#)  
[thelatemail](#)  
[Thomas](#)  
[Tim Coker](#)  
[TriskalJM](#)  
[tuomastik](#)  
[Umberto](#)  
[user2100721](#)  
[user890739](#)  
[USER\\_1](#)  
[Uwe](#)  
[Vedda](#)  
[WAF](#)  
[while](#)  
[YCR](#)  
[Yun Ching](#)  
[zacdav](#)  
[zelite](#)  
[zx8754](#)

## 你可能也喜欢



## You may also like

