R 与 tidyverse——数据分析入门

石天熠

2019-06-20

目录

欢迎

简介

本书为 R(R Core Team 2019) 和 tidyverse(Wickham 2017) 的入门向教程。教学视频在 b 站 ()。

使用说明

左上角的菜单可以选择收起/展开目录，搜索，和外观，字体调整。

如果你对某一段文字有修改意见，可以选择那段文字，并通过 Hypothesis 留言（选择 “annotate”）。右上角可以展开显示公开的留言。

如果你熟悉Bookdown和 Github，可以在此提交 pull request.

Chapter 1

R 和 RStudio 介绍和安装教程

1.1 什么是 R

R(R Core Team 2019) 包含 R 语言（编程语言）和一个有着强大的统计分析及作图功能的软件系统，由新西兰奥克兰大学统计学系的 Ross Ihaka 和 Robert Gentleman 共同创立。

安装了 R 之后，你可以在其自带的 “R” 软件中（也可以直接在命令行使用），但是那个软件界面比较简单，需要记住一些命令来执行 “查看当前存储的数据”，“导出 jpg 格式的图像” 等操作，对新手不太友好。因此我们使用 RStudio.

RStudio(RStudio Team 2015) 是 R 语言官方的 IDE（集成开发环境），它的一系列功能使得编辑，整理和管理 R 代码和项目方便很多。

了解 R 的优势，请看第1.3节

1.2 安装 R 和 RStudio

1.2.1 安装 R

https://cran.r-project.org

前往CRAN，根据自己的操作系统（Linux，MacOS 或 Windows）选择下载安装 R.

1.2.2 安装 RStudio

https://www.rstudio.com/products/rstudio/download/

前往RStudio 下载页，选择最左边免费的开源版本，然后选择对应自己的操作系统的版本，下载并安装。

1.3 为什么使用 R，R 与其他统计软件的比较[1]

（这一小节不影响 R 的学习进度，可以直接跳过到下一章）

SAS，SPSS，Prism，R 和 Python 是数据分析和科研作图常用的软件。

SAS，SPSS 和 Prism 都是收费的，而且不便宜。比如 SAS 第一年需要10000多美元，随后每年要缴纳几千美元的年费。

R 比 SAS 功能更强大。所有 SAS 中的功能，都能在 R 中实现，而很多 R 中的功能无法在 SAS 中实现[2]。

R 和 Python 是开源、免费的。

在数据分析的应用中，R 比 Python 历史更悠久，因此积攒了很多很棒的 packages（包）。

使用 Python 做数据分析近年来愈发流行，网上也有不少教程。

使用 R+Python

至于 Excel，它的定位原本就是办公（而不是学术）软件，用作数据收集和初步整理是可以的，但是做不了严谨的数据分析和大数据，功能也非常局限。有五分之一的使用了 Excel 的遗传学论文，数据都出现了偏差 (Ziemann, Eren, and El-Osta 2016)。

[1] Gentleman, R. (2009). *R Programming for Bioinformatics*. Boca Raton, FL: CRC Press.

[2] https://thomaswdinsmore.com/2014/12/15/sas-versus-r-part-two/

R 是 GNU 计划的一部分，因此 R 是一个自由软件 (Libre software)。你可以在GNU 官网了解更多。

Chapter 2

获取资源与帮助（重要!）

这本书可以帮助你快速学会 R 和 tidyverse 的最常用和最重要的操作，但这仅仅是冰山一角。当你在做自己的研究的时候，会用到很多这本书中没有讲到的方法，因此学会获取资源和帮助是很重要的。以下列举几个常用的获取 R 的帮助的网站/方法：

2.1 论坛类（解答实际操作中的问题）

- 爆栈网 (StackOverflow)是著名计算机技术问答网站（如果你有其他的编程语言基础，一定对它不陌生）。查找问题的时候加上 `[R]`，这样搜索结果就都是与 R 相关的了（为了进一步缩小搜索范围，可以加上其他的 tag，比如 `[ggplot]`，`[dplyr]`）。注意，提问和回答的时候话语尽量精简，不要在任何地方出现与问题无关的话（包括客套话如 “谢谢”），了解更多请查看其新手向导。
- 由谢益辉大佬在 2006 年（竟然比爆栈网更早!）创建的“统计之都” 论坛，是做的最好的一个面向 R 的中文论坛（但是客观地来说活跃度还是没爆栈网高）同样不要忘记读新手指引。

2.2 Reference 类（查找特定的 function 的用法，就像查字典一样）

- 直接在 R console 中执行?+ 函数名称，比如`?t.test`
- RDocumentation上有基础 R 语言和来自 CRAN，GitHub 和 Bioconductor 上的近 18000 个 packages 的所有的函数的说明和使用例。

2.3 教程和书籍类（用来系统地学习）

- R 的官方 Manuals. 其中新手只需要看 *An Introduction to R*，随后选看 *R Language Definition* 即可。部分由丁国徽翻译成中文（点击量其实并不高……要想把握前沿信息还是需要阅读英语的能力的）。
- *R for Data Science* by Garrett Grolemund & Hadley Wickham. `tidyverse` 的作者写的一本书，较为详细地介绍了 `tidyverse` 的用法以及一些更高深的关于编程的内容。
- RStudio Resources是 RStudio 的资源区，有关于 R 和 RStudio 的高质量教程，还可以下载很多方便实用的 Cheat Sheet.
- *The R Book* by Michael J. Crawley
- R 的官方 fAQ
- 存储在 CRAN 上的中文 FAQ（注意这不是英文 FAQ 的翻译，而是一本独立的 R 入门教程）

2.4 速查表 (Cheat sheets)（用来贴墙上）

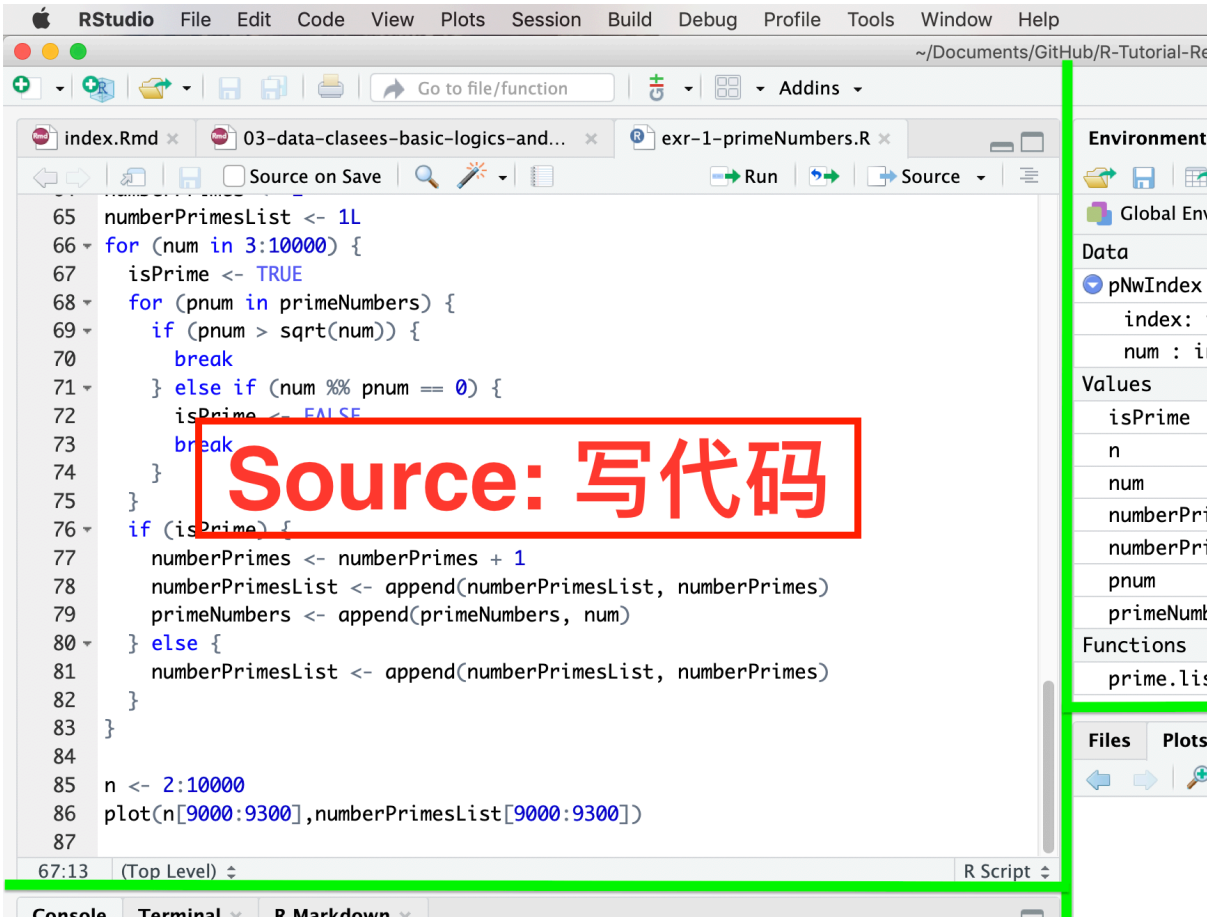
- R Reference Card 2.0 by Mayy Baggott & Tom Short以及其第一版的中文翻译
- RStudio Cheat Sheets包含了 RStudio IDE 和常用 packages 的 cheat sheets。

Chapter 3

RStudio 界面介绍，基本操作，和创建新项目

3.1 界面

3.1.1 概览



3.1.1.1 左下角：Console（控制台）

3.1.1.2 右上角：Environment（环境），History（历史）和 Connections（连接）

3.1.1.3 右下角：Plots（绘图），Help（帮助），Files（文件），Packages（包）和 Viewer（查看器）

3.1.1.4 自定义

3.2 基本操作

3.2.1 执行代码

试着在 console 里输入 `1+1`，并按回车以执行。你的 console 会显示：

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

其中 `2` 是计算结果，`[1]` 我们在下一章再解释。`> 1+1` 是你的 input，`[1] 2` 是 console 给出的 output.

还是用 `1+1` 举例，在本书中，对于 input 和 output 的展示格式是这样的：

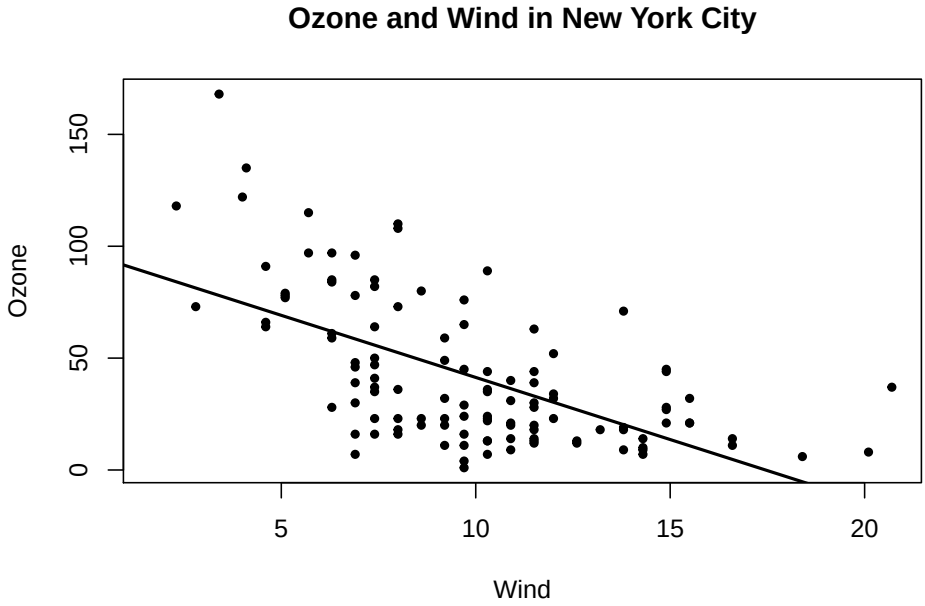
```
1+1
```

```
## [1] 2
```

注意到 input 中的 `>` 被省略了，这意味着你可以直接把代码从本书复制到你的 console 并按回车执行（因为它本身自带了 `>`），类似地，你从其他各种网站上找到的说明书，教程和论坛帖子中看到的 R 代码，也都是这种形式出现，你可以直接复制粘贴然后回车执行，很方便。

再来一个例子，试着在 console 里输入以下代码，打完每一行按回车执行或者换行（你可以直接复制粘贴，但是要脑补这是你一个字一个字打出来的）：

```
library(datasets)
with(airquality, plot(Wind, Ozone, main = "Ozone and Wind in New York City", pch = 20))
model <- lm(Ozone ~ Wind, airquality)
abline(model, lwd = 2)
```



可以看到，在 plots 区，生成了一副漂亮的图表。（先别在意每行代码具体的作用，在之后的章节我会一一讲述）

这时，把 RStudio 关掉，再重新启动，你会发现你之前辛辛苦苦作的图付之一炬了。

3.2.2 记录和管理代码

初学者经常会在 console 里写代码，或者从别处复制代码，并执行。这对于一次性的计算（比如写统计学作业时用 R 来算线性回归的参数）很方便，但是如果你想保存你的工作，你需要把它记录在 R 文件里。如果你的工作比较复杂，比如有一个 excel 表格作为数据源，然后在 R 中用不同的方法分析，导出图表，这时候你会希望这些文件都集中在一起。你可以创建 RProject 来管理它们。

3.2.3 创建 Project

3.2.4 管理

Chapter 4

数据类型，数学运算，逻辑和函数

4.1 数据类型 (Data Types)

R 没有标量，它通过各种类型的向量 (vector) 来存储数据。常用的数据类型有：

类型	含义与说明	例子
numeric	浮点数向量	`3`, `0.5`, `sqrt(2)`, `NaN`, `Inf`
integer	整数向量	`3L`, `100L`
character	字符向量；需被引号包围	`"1"`, `"$"`, `" 你好"`
logical	逻辑向量	`TRUE`, `FALSE`, `NA`
complex	复数向量	`3+5i`, `1i`, `1+0i`

xxx	描述	xxx
factor	因子，用于标记样本	

通过 `class()` 函数，可以查看指定数据的类型。（除此之外，`typeof()`, `mode()`, `storage.mode()` 这三个函数的功能与 `class()` 类似，但有重要区别；为避免造成困惑，此处不展开讨论）。

4.2 数学运算

4.2.1 数的表达

4.2.1.1 浮点数

除非指定作为整数（见下），在 R 中所有的数都被存储为双精度浮点数的格式 (double-precision floating-point format)，其 `class` 为 `numeric`。

```
class(3)
```

```
## [1] "numeric"
```

这会导致一些有趣的现象，比如 $(\sqrt{3})^2 \neq 3$：~~（强迫症患者浑身难受）~~

```
sqrt(3)^2-3
```

```
## [1] -4.440892e-16
```

浮点数的计算比精确数的计算快很多。如果你是第一次接触浮点数，可能会觉得它不可靠，其实不然。在绝大多数情况下，牺牲的这一点点精度并不会影响计算结果（我们的结果所需要的有效数字一般不会超过 10 位）。

`NaN`（非数）和 `Inf`（无限大）也是浮点数！

```
class(NaN)
```

```
## [1] "numeric"
```

```
class(Inf)
```

```
## [1] "numeric"
```

4.2.1.2 科学计数法

R 可以使用科学计数法 (`AeB`$= A \times 10^B$)，比如：

```
3.1e5
```

```
## [1] 310000
```

```
-1.2e-4+1.1e-5
```

```
## [1] -0.000109
```

4.2.1.3 整数

整数的 class 为 `integer`。有两种常见的方法创建整数：1）在数后面加上 `L`；

```
class(2)
```

```
## [1] "numeric"
```

```
class(2L)
```

```
## [1] "integer"
```

2）创建数列

```
1:10 # 公差为 1 的整数向量生成器，包含最小值和最大值
```

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

```
class(1:10)
```

```
## [1] "integer"
```

```
seq(5,50,5) # 自定义公差，首项，末项和公差可以不为整数
```

```
##  [1]  5 10 15 20 25 30 35 40 45 50
```

```
class(seq(5,50,5)) # 因此产生的是一个浮点数向量
```

```
## [1] "numeric"
```

```
seq(5L,50L,5L) # 可以强制生成整数
```

```
##  [1]  5 10 15 20 25 30 35 40 45 50
```

```
class(seq(5L,50L,5L)) # 完美
```

```
## [1] "integer"
```

整数最常见的用处是 indexing（索引）。

4.2.1.3.1 整数变成浮点数的情况

这一小段讲的比较细，初学者可以直接跳到下一节（4.2.2）。

整数与整数之前的加，减，乘，求整数商，和求余数计算会得到整数，其他的运算都会得到浮点数，（阶乘 (`factorial`) 也是，即便现实中不管怎么阶乘都不可能得到非整数）：

```
class(2L+1L)
```

```
## [1] "integer"
```

```
class(2L-1L)
```

```
## [1] "integer"
```

```
class(2L*3L)
```

```
## [1] "integer"
```

```
class(17L%/%3L)
```

```
## [1] "integer"
```

```
class(17L%%3L)
```

```
## [1] "integer"
```

```
class(1000L/1L)
```

```
## [1] "numeric"
```

```
class(3L^4L)
```

```
## [1] "numeric"
```

```
class(sqrt(4L))
```

```
## [1] "numeric"
```

```
class(log(exp(5L)))
```

```
## [1] "numeric"
```

```
class(factorial(5L))
```

```
## [1] "numeric"
```

整数与浮点数之间的运算，显然，全部都会产生浮点数结果，无需举例。

另外一个需要注意的地方是，取整函数4.2.2.3并不会产生整数。如果需要的话，要用 `as.integer()` 函数。

4.2.2 运算

4.2.2.1 二元运算符号

R 中的 binary operators（二元运算符）有：

符号	描述
`+`	加
`-`	减
`*`	乘
`/`	除以
`^` 或 `**`	乘幂
`%/%`	求整数商，比如 `7%%3`= 2
`%%`	求余数，比如 `7%%3`= 1

其中求余/求整数商最常见的两个用法是判定一个数的奇偶性，和时间，角度等单位的转换。（后面再详细介绍）。

4.2.2.2 e^x 和 $\log_x y$

`exp(x)` 便是运算 e^x。如果想要 $e = 2.71828...$ 这个数：

```
exp(1)
```

```
## [1] 2.718282
```

`log(x, base=y)` 便是运算 $\log_y x$，可以简写成 `log(x,y)`（简写需要注意前

后顺序，下面讲函数的时候会解释）。

默认底数为 e：

```
log(exp(5))
```

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

有以 10 和 2 为底的快捷函数，`log10()` 和 `log2()`

```
log10(1000)
```

```
## [1] 3
```

```
log2(128)
```

```
## [1] 7
```

4.2.2.3 近似数（取整，取小数位，取有效数字）

注意，取整函数给出的的结果不是整数！

```
class(ceiling(7.4))
```

```
## [1] "numeric"
```

4.2.2.4 R 中自带的数学函数集合

函数	描述
`exp(x)`	e^x
`log(x,y)`	$\log_y x$
`log(x)`	$\ln(x)$
`sqrt(x)`	$\sqrt{x}$
`factorial(x)`	$x! = x \times (x-1) \times (x-2) \ldots \times 2 \times 1$
`choose(n,k)`	$\binom{n}{k} = \frac{n!}{k!(n-k)!}$（二项式系数）
`gamma(z)`	$\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} dx$（伽马函数）
`lgamma(z)`	$\ln(\Gamma(z))$
`floor(x)`, `ceiling(x)`, `trunc(x)`,	取整；见上一小节。

函数	描述
`round(x, digits = n)`	四舍五入，保留 n 个小数位，n 默认为 0
`signif(x,digits = n)`	四舍五入，保留 n 个有效数字，n 默认为 6）
`sin(x)`, `cos(x)`, `tan(x)`	三角函数
`asin(x)`, `acos(x)`, `atan(x)`	反三角函数
`sinh(x)`, `cosh(x)`, `tanh(x)`	双曲函数
`abs(x)`	$\|x\|$（取绝对值）

4.3 变量

4.4 逻辑

4.4.1 TRUE 和 FALSE

4.4.2 Logical Operators（逻辑运算符）

R 中的 logical operators 有:

符号	描述
`==`	equal to（等于）
`!=`	equal to（不等于）
`<`	less than（小于）
`>`	more than（大于）
`<=`	less than or equal to（小于等于）
`>=`	more than or equal to（大于等于）
`&`	AND（和）
`|`	OR（或）
`!`	反义符号（见下）

4.5 以下是不重要的一些内容

4.5.1 Console 和 R script 编辑器的一些特性

Console 中每个命令开头的 > 叫做 prompt（我不知道它的中文名诶），当它出现在你所编辑的那一行的开头时，按下回车的时候那行的命令才会被执行。有时候它会消失，这时候按 esc 可以将其恢复。

prompt 消失的主要原因是你的代码没有写完，比如括号不完整：

```
> 2+(3+4
```

这时你按回车，它会显示：

```
> 2+(3+4
+
```

+ 号是在提示代码没写完整。这时你把括号补上再按回车：

```
> 2+(3+4
+ )

[1] 9
```

便可以完成计算。

这意味着我们可以把一条很长的命令分成很多行。比如我们可以写这样的代码（在 R script 编辑器中！）

```
if(1 + 1 == 2 & 1 + 2 == 5){
  print(2)
} else{
  print(3)
}
```

然后 Ctrl+Enter 执行。

```
<function>(<argument> = <value>)
```

4.5.2 赋值 (assignment): 使用 <-，而不是 =

与很多其他的计算机语言不同，<-是 R 中的赋值符号：

```
x <- 2
```

4.6 函数

不像很多其他语言的函数有 value.func() 和 func value 等格式，R 中所有函数的通用格式是这样的：

```
function(argument1=value1, argument2=value2, ...)
```

比如

```
x1 <- c(5.1,5.2,4.5,5.3,4.3,5.5,5.7)
t.test(x=x1, mu = 4.5)
```

```
##
##  One Sample t-test
##
## data:  x1
## t = 3.0308, df = 6, p-value = 0.02307
## alternative hypothesis: true mean is not equal to 4.5
## 95 percent confidence interval:
##  4.612840 5.558589
## sample estimates:
## mean of x
##  5.085714
```

4.6.1 关于 "..."

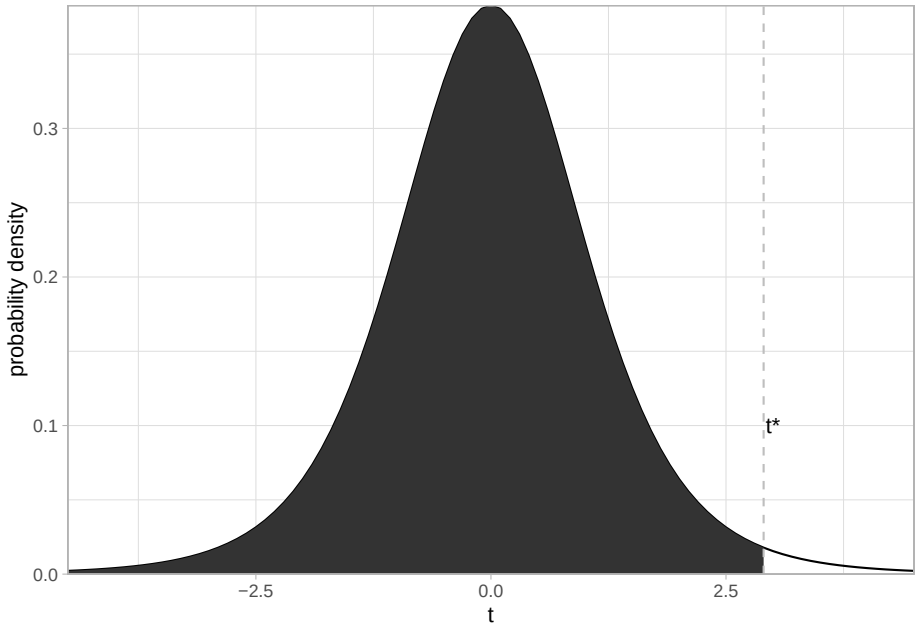
有时候，你想写的函数可能有数量不确定的 arguments

If a function has '...' as a formal argument then any actual arguments that do not match a formal argument are matched with '...'.

4.7 简易的统计学计算

4.7.1 t 分布

下面介绍的这几种方法是以这个 t 分布函数图为基础的：



阴影区域为 P，虚线对应的 t 为 t^*. `qt()` 可以把 P 转化成 t^*,`pt()` 则相反。

举个例子就明白了。假设你需要算一个 confidence interval（置信区间），confidence level（置信等级）为 95%，即 $\alpha = 0.05$，degrees of freedom(自由度)为 12，那么怎么算 t^* 呢?

```
qt(0.975, df = 12)
```

```
## [1] 2.178813
```

对，就是这么简单。(为什么是 0.975? 因为你要把 0.05 分到左右两边，就等同于 t 分布的 $t = 0.095$)

再举一个例子，你在做 t 检验，双尾的，算出来 $t = 1.345$，自由度是 15，那么 p 值怎么算呢?

```
p <- (1-(pt(2.2, df = 15)))*2
p
```

```
## [1] 0.04389558
```

其中 `pt(2.2, df = 15)` 算出阴影面积，1 减去它再乘以二就是对应的双尾 t 检验的 p 值。

4.7.2 z 分布

没有 z 分布专门的函数。可以直接用 t 分布代替，把 `df` 调到很大（比如 `999999`）就行了。比如我们试一下 95% 置信区间所对应的 z：

```
qt(0.975,9999999)
```

```
## [1] 1.959964
```

（果然是 1.96）

4.7.3 t 检验

t test 分为以下几种：

- One sample t test
- Two sample...
  - paired t test
  - Unequal variance t test
  - Equal variance t test

在 R 中做 t 检验，很简单，以上这些 t 检验，都是用 `t.test` 这个函数去完成。

以 one sample 为例：

```
x <- c(2.23,2.24,2.34,2.31,2.35,2.27,2.29,2.26,2.25,2.21,2.29,2.34,2.32)
t.test(x, mu = 2.31)
```

```
##
```

```
##  One Sample t-test
##
## data:  x
## t = -2.0083, df = 12, p-value = 0.06766
## alternative hypothesis: true mean is not equal to 2.31
## 95 percent confidence interval:
##  2.257076 2.312155
## sample estimates:
## mean of x
##  2.284615
```

可以看到 $p = 0.06766$。

R 的默认是双尾检验，你也可以设置成单尾的：

```
x <- c(2.23,2.24,2.34,2.31,2.35,2.27,2.29,2.26,2.25,2.21,2.29,2.34,2.32)

t.test(x, mu = 2.31, alternative = "less")
```

```
##
##  One Sample t-test
##
## data:  x
## t = -2.0083, df = 12, p-value = 0.03383
## alternative hypothesis: true mean is less than 2.31
## 95 percent confidence interval:
##      -Inf 2.307143
## sample estimates:
## mean of x
##  2.284615
```

p 值一下就减了一半。

Two-sample:

```
x <- c(2.23,2.24,2.34,2.31,2.35,2.27,2.29,2.26,2.25,2.21,2.29,2.34,2.32)
y <- c(2.27,2.29,2.37,2.38,2.39,2.25,2.39,2.16,2.55,2.81,2.19,2.44,2.22)
```

```
t.test(x, y)
```

```
##
##  Welch Two Sample t-test
##
## data:  x and y
## t = -1.5624, df = 13.65, p-value = 0.1411
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.18460351  0.02921889
## sample estimates:
## mean of x mean of y
##  2.284615  2.362308
```

r 的默认是 non-paired, unequal variance，你可以通过增加 `paired = TRUE`, `var.equal = TRUE` 这两个参数来改变它。

```
t.test(x, y, paired = TRUE)
```

```
##
##  Paired t-test
##
## data:  x and y
## t = -1.4739, df = 12, p-value = 0.1662
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.19253874  0.03715412
## sample estimates:
## mean of the differences
##              -0.07769231
```

4.7.4 χ^2 检验

χ^2 有两种，good

Chapter 5

安装和使用 packages（包）

5.1 Package 是什么，为什么使用它们?

Package 是别人写好的在 R 中运行的程序（以及附带的数据和文档），你可以免费安装和使用它们。

Packages 可以增加在基础 R 语言中没有的功能，可以精简你代码的语句，或是提升使用体验。比如有个叫做 `tikzDevice` 的 package 可以将 R 中的图表导出成 tikz 语法的矢量图，方便在 LaTeX 中使用。这本书的排版也是依赖于用 R 中的一个叫做 `bookdown` 的 package （它真的超棒，我也许会另出一个课程来讲解它）.

这个课程主要是学习 `tidyverse` 这个 package，

5.2 如何安装 packages

首先我们安装 tidyverse（**很重要，课程接下来的部分都要使用这个 package**）:

```
install.packages("tidyverse")
```

在 console 中运行以上代码，R 就会从 CRAN 中下载 tidyverse 并安装到你电脑上的默认位置。因此安装 packages 需要网络连接。

如果想安装多个 packages，你可以一行一行地安装，或是把多个 packages 的名字合成一列，同时安装，比如：

```
install.packages(c("nycflights13", "gapminder", "Lahman")
```

它其中包含一系列小

5.3 如何使用 packages

安装 packages 后，有两种方法使用它们。以 `tidyverse` 为例：

```
library('tidyverse')
```

或

```
require('tidyverse')
```

两者的效果很大程度上都是一样的，都可以用来读取单个 package。但是它们有两个微妙的不同：

1. `require()` 会返回一个逻辑值。如果 package 读取成功，会返回 `TRUE`，反之则返回 `FALSE`.
2. `library()` 如果读取试图读取不存在的 package，会直接造成错误 (error)，而 `require()` 不会造成错误，只会产生一个警告 (warning).

这意味着 `require()` 可以用来同时读取多个 packages：

```
lapply(c("dplyr","ggplot2"), require, character.only = TRUE)
```

```
## [[1]]
## [1] TRUE
##
## [[2]]
## [1] TRUE
```

或者更精简一点，

```
lapply(c("dplyr","ggplot2"), require, c = T)
```

```
## [[1]]
```

```
## [1] TRUE
##
## [[2]]
## [1] TRUE
```

每次重启 R 的时候，上一次使用的 packages 都会被清空，所以需要重新读取。因此我们要在 R script 里面记录该 script 需要使用的 packages（这算是逼迫你养成好习惯。当你把你的代码分享给别人的时候，要保证在别人的电脑上也能正常运行，就必须要指明要使用哪些 packages）

一种快速读取多个 packages 的方法：

```
lapply(c("dplyr","ggplot2"), require, character.only = TRUE)
```

```
## [[1]]
## [1] TRUE
##
## [[2]]
## [1] TRUE
```

5.4 如何找 packages

5.4.1 通过他人的经验

5.4.2 在 CRAN 中通过分类寻找

Chapter 6

dataframe（数据框）和 tibble

6.1 基础

dataframe 是 R 中存储复杂数据的格式，它直观易操作。tibble 是 tidyverse 的一部分，它是 dataframe 的进化版，功能更强大，更易操作。

在 dataframe/tibble 中，每一行代表的是一个 observation（硬翻译的话是“观测单位”，但是我觉得这个翻译不好），每一列代表的是一个 variable（变量）。举个例子：

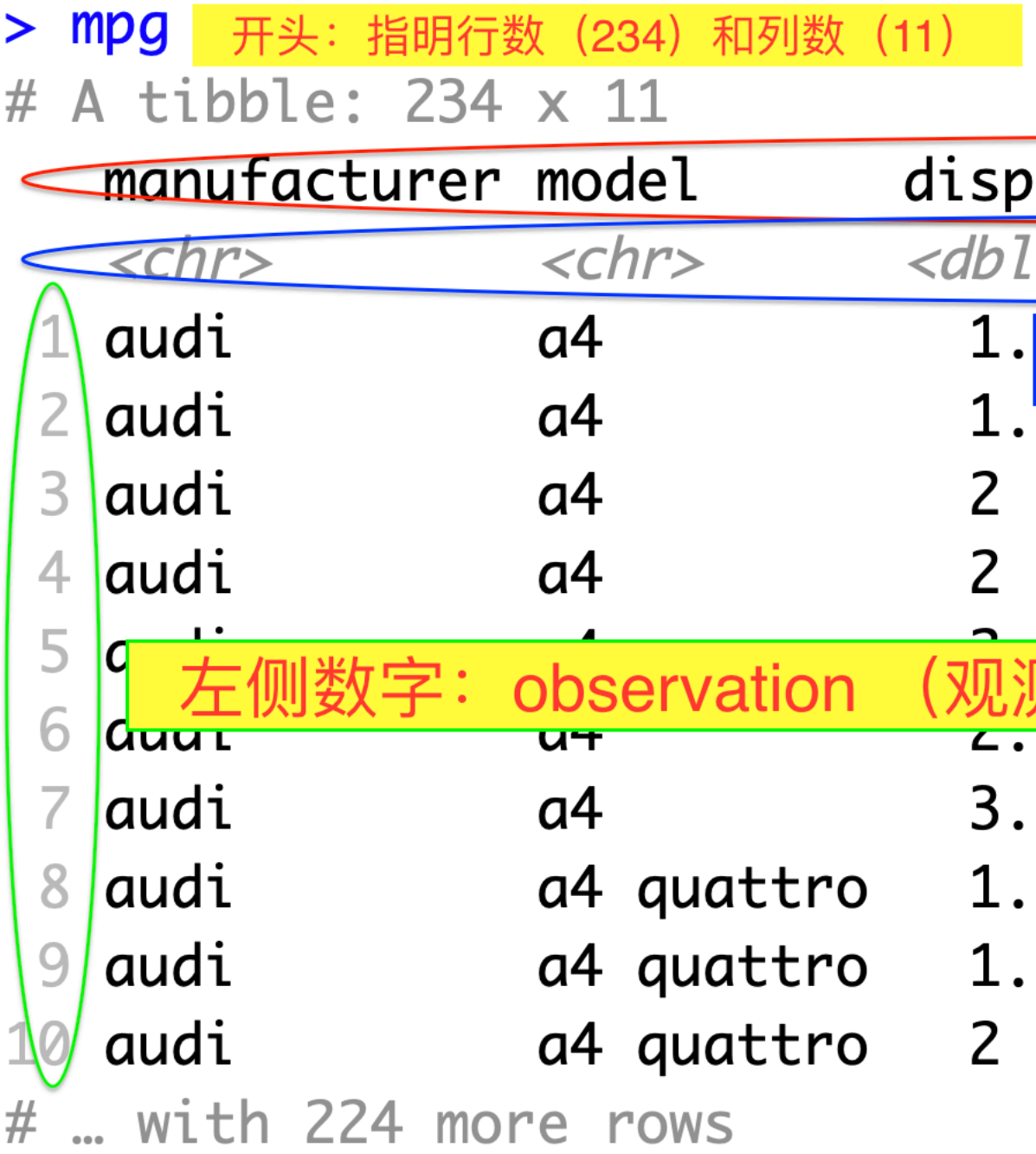
我们先加载 tidyverse：

```
require(tidyverse)
```

以后每次跟着本书使用 R 的时候，都要先加载 tidyverse，不再重复提醒了。

tidyverse 中自带一些范例数据，比如我们输入：

```
mpg
```


> mpg
开头：指明行数（234）和列数（11）
A tibble: 234 x 11
manufacturer model
<chr> <chr>
1 audi a4
2 audi a4
3 audi a4
4 audi a4
7 audi a4
8 audi a4 quattro
9 audi a4 quattro
10 audi a4 quattro
… with 224 more rows
左侧数字：observation （观

6.2

6.3 进阶内容

这一节为进阶内容，不用看。可以直接跳到 @??

其中的很多操作和 dataframe 或 tibble 中的操作是等效的。一般，tibble 中的操作更直观，更容易上手。

6.3.1 arrays（数组）和 matrices（矩阵）简介

Vector 是一维的数据。Array 是多维的数据。Matrix 是二维的数据，因此 matrix 是 array 的一种特殊情况。

Dataframe 不是 matrix. A matrix is a two-dimensional **array** containing numbers. A dataframe is a two-dimensional **list** containing (potentially a mix of) numbers, text or logical variables in different columns.

我们可以用 `dim()` 来创建 arrays:

```
A <- 1:48 # 创建一个 (1,2,3,...24) 的 numeric vector
dim(A) <- c(6,8) # 给 A assign 一个 6 乘 4 的 dimensions
A
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1    7   13   19   25   31   37   43
## [2,]    2    8   14   20   26   32   38   44
## [3,]    3    9   15   21   27   33   39   45
## [4,]    4   10   16   22   28   34   40   46
## [5,]    5   11   17   23   29   35   41   47
## [6,]    6   12   18   24   30   36   42   48
```

可以看到我们创建了一个二维的，array, 因此它也是一个（4 行 6 列的）matrix。

```
is.array(A)
```

```
## [1] TRUE
```

```
is.matrix(A)
```

```
## [1] TRUE
```

注意 24 个数字排列的方式。第一个维度是行，所以先把 4 行排满，随后再使用下一个维度（列），使用第 2 列继续排 4 行，就像数字一样，（十进制中）先把个位从零数到 9，再使用第二个位数（十位），以此类推。下面三维和四维的例子可能会更清晰。

同时注意最左边和最上边的 [1,], [,3] 之类的标记。你应该猜出来了，这些是 index. 假设你要抓取第五行第三列的数值：

```
A[5,3]
```

```
## [1] 17
```

或者第三行的全部数值：

```
A[3,]
```

```
## [1]  3  9 15 21 27 33 39 45
```

或者第四列的全部数值：

```
A[,4]
```

```
## [1] 19 20 21 22 23 24
```

接下来我们再看一个三维的例子（还是用 1-48）：

```
dim(A) <- c(2,8,3)
A
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]    1    3    5    7    9   11   13   15
## [2,]    2    4    6    8   10   12   14   16
##
## , , 2
##
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]   17   19   21   23   25   27   29   31
## [2,]   18   20   22   24   26   28   30   32
##
## , , 3
##
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
## [1,]   33   35   37   39   41   43   45   47
## [2,]   34   36   38   40   42   44   46   48
```

它生成了三个二维的矩阵。在每个 2*8 的矩阵存储满 16 个元素后，第三个维度就要加一了。每个矩阵开头的，，x 正是第三个维度的值。同理，我们可以生成四维的 array：

```
dim(A) <- c(3,4,2,2)
A
```

```
## , , 1, 1
##
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
##
## , , 2, 1
##
##      [,1] [,2] [,3] [,4]
## [1,]   13   16   19   22
## [2,]   14   17   20   23
## [3,]   15   18   21   24
##
## , , 1, 2
##
##      [,1] [,2] [,3] [,4]
## [1,]   25   28   31   34
```

```
## [2,]   26   29   32   35
## [3,]   27   30   33   36
##
## , , 2, 2
##
##      [,1] [,2] [,3] [,4]
## [1,]   37   40   43   46
## [2,]   38   41   44   47
## [3,]   39   42   45   48
```

观察每个矩阵开头的，，x，y. x 是第三个维度，y 是第四个维度。每个二位矩阵存满后，第三个维度（x）加一。x 达到上限后，第四个维度（y）再加一。

类似二维矩阵，你可以通过 index 任意抓取数据，比如：

```
A[ ,3 , , ] # 每个矩第 3 每列的数据，即所有第二个维度为 3 的数值
```

```
## , , 1
##
##      [,1] [,2]
## [1,]    7   19
## [2,]    8   20
## [3,]    9   21
##
## , , 2
##
##      [,1] [,2]
## [1,]   31   43
## [2,]   32   44
## [3,]   33   45
```

6.3.2 给 matrices 和 arrays 命名

假设我们记录了 3 种药物（chloroquine, artemisinin, doxycycline) 对 5 种疟原虫 (P. falciparum, P. malariae, P. ovale, P. vivax, P. knowlesi) 的疗效，其

中每个药物对每种疟原虫做 6 次实验。为了记录数据，我们可以做 3 个 6*5 的矩阵：（这里只是举例子，用的是随机生成的数字）

```
B <- runif(90, 0, 1) # 从均匀分布中取 100 个 0 到 1 之间的数
dim(B) <- c(6, 5, 3) # 注意顺序
B
```

```
## , , 1
##
##           [,1]      [,2]       [,3]      [,4]       [,5]
## [1,] 0.7637297 0.9417252 0.18629634 0.9819489 0.86094387
## [2,] 0.1761423 0.4339366 0.71466158 0.8647651 0.34622092
## [3,] 0.6256021 0.1409803 0.51433185 0.9279917 0.32084294
## [4,] 0.5331692 0.3891523 0.02194286 0.7527327 0.59751732
## [5,] 0.7022421 0.3753248 0.39488831 0.9203920 0.02206705
## [6,] 0.7737602 0.7012732 0.20683602 0.2581793 0.16131253
##
## , , 2
##
##           [,1]      [,2]      [,3]       [,4]      [,5]
## [1,] 0.7139004 0.8001794 0.6132669 0.06479105 0.9293273
## [2,] 0.5685787 0.5460036 0.2339729 0.62023436 0.6454746
## [3,] 0.7027480 0.4374074 0.7392465 0.91019803 0.8801891
## [4,] 0.8279210 0.2116584 0.8232521 0.23876267 0.1254521
## [5,] 0.1762345 0.4734587 0.5952760 0.31882640 0.9853531
## [6,] 0.1419426 0.9712234 0.1846387 0.34450433 0.2106055
##
## , , 3
##
##            [,1]       [,2]      [,3]       [,4]       [,5]
## [1,] 0.18345084 0.09348055 0.9080185 0.07549864 0.98888850
## [2,] 0.98280272 0.64447401 0.8692285 0.16080412 0.81898364
## [3,] 0.18639984 0.36951297 0.4780550 0.43440840 0.86578291
## [4,] 0.47462182 0.08827655 0.7123358 0.73116668 0.29359883
```

```
## [5,] 0.02951967 0.35800396 0.3631958 0.20203451 0.03943304
## [6,] 0.92779483 0.68160766 0.2089293 0.98570428 0.83195895
```

然后我们用 dimnames() 来命名：

```
dimnames(B) <- list(paste("trial.", 1:6), c('P. falciparum', 'P. malariae', 'P. ov
B
```

```
## , , chloroquine
##
##          P. falciparum P. malariae   P. ovale  P. vivax P. knowlesi
## trial. 1     0.7637297   0.9417252 0.18629634 0.9819489  0.86094387
## trial. 2     0.1761423   0.4339366 0.71466158 0.8647651  0.34622092
## trial. 3     0.6256021   0.1409803 0.51433185 0.9279917  0.32084294
## trial. 4     0.5331692   0.3891523 0.02194286 0.7527327  0.59751732
## trial. 5     0.7022421   0.3753248 0.39488831 0.9203920  0.02206705
## trial. 6     0.7737602   0.7012732 0.20683602 0.2581793  0.16131253
##
## , , artemisinin
##
##          P. falciparum P. malariae  P. ovale   P. vivax P. knowlesi
## trial. 1     0.7139004   0.8001794 0.6132669 0.06479105   0.9293273
## trial. 2     0.5685787   0.5460036 0.2339729 0.62023436   0.6454746
## trial. 3     0.7027480   0.4374074 0.7392465 0.91019803   0.8801891
## trial. 4     0.8279210   0.2116584 0.8232521 0.23876267   0.1254521
## trial. 5     0.1762345   0.4734587 0.5952760 0.31882640   0.9853531
## trial. 6     0.1419426   0.9712234 0.1846387 0.34450433   0.2106055
##
## , , doxycycline
##
##          P. falciparum P. malariae  P. ovale   P. vivax P. knowlesi
## trial. 1    0.18345084  0.09348055 0.9080185 0.07549864  0.98888850
## trial. 2    0.98280272  0.64447401 0.8692285 0.16080412  0.81898364
## trial. 3    0.18639984  0.36951297 0.4780550 0.43440840  0.86578291
## trial. 4    0.47462182  0.08827655 0.7123358 0.73116668  0.29359883
```

```
## trial. 5    0.02951967  0.35800396 0.3631958 0.20203451  0.03943304
## trial. 6    0.92779483  0.68160766 0.2089293 0.98570428  0.83195895
```

清清楚楚，一目了然。

6.3.3 apply

```
apply(A,1,sum)
```

```
## [1] 376 392 408
```

R Core Team. 2019. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. https://www.R-project.org/.

RStudio Team. 2015. *RStudio: Integrated Development Environment for R*. Boston, MA: RStudio, Inc. http://www.rstudio.com/.

Wickham, Hadley. 2017. *Tidyverse: Easily Install and Load the 'Tidyverse'*. https://CRAN.R-project.org/package=tidyverse.

Ziemann, Mark, Yotam Eren, and Assam El-Osta. 2016. "Gene Name Errors Are Widespread in the Scientific Literature." Journal Article. *Genome Biology* 17 (1): 177. https://doi.org/10.1186/s13059-016-1044-7.