

NOTE on Data Analysis

zy

2021 年 3 月 6 日

目录

I Head first on data analysis	3
1 数据分析引言: 分解数据	4
1.1 数据分析的基本流程:	4
1.1.1 确定问题	4
1.1.2 分解问题	4
1.1.3 评估组块	5
1.1.4 提出建议	5
2 实验: 检验你的理论	6
2.1 随机调查	6
2.2 比较法与观察研究法	6
2.3 拆分数据块, 管理混杂因素	7
2.4 实验以控制组为基准	7
2.5 随机选择相似组	7
3 最优化: 寻找最大值	8
3.1 需要哪些数据	8
3.2 最优化问题	8
3.2.1 借助目标函数发现目标	9
3.2.2 约束条件	9
3.2.3 excel 实现最优化	9
3.2.4 按照分析目标校正假设	9
3.2.5 提防负相关变量	9
3.2.6 假设立足于不断变化的实际情况	10
4 数据图形化: 图形让你更精明	11
4.1 体现数据	11
4.2 数据图形化的根本在于正确比较	11
4.3 最优秀的图形都是多元图形	12
II R 数据科学实战—A. 工具详解	13
5 数据导入工具	14

5.1	utils—数据读取	14
5.1.1	read.csv/csv2—逗号分隔数据读取	14
5.1.2	read.delim/delim2—特定分隔符数据读取	16
5.1.3	read.table—任意分隔符数据读取	16
5.1.3.1	空白行	17
5.1.3.2	默认值, 空白	19
5.2	readr—进阶数据读取	20
5.3	pdfrools—pdf 文件	20
6	数据清理工具	24
6.1	基本概念	24
6.2	tibble 包—数据集准备	24
6.2.1	为什么使用 tibble	25
6.2.2	创建 tbl 格式	26
6.2.3	as_tibble—转换已有格式的数据集	26
6.2.3.1	as_tibble 函数直接将 vector 格式转换成数据框格式	27
6.2.3.2	矩阵格式转换	27
6.2.3.3	列表格式转换	27
6.2.4	add_row/column—实用小工具	28
6.3	tidyr—数据清道夫	30
6.3.1	为什么使用 tidyr 包	30
6.3.2	gather/spread—“长””宽”数据转换	30
6.3.2.1	gather—“宽”变“长”	30
6.3.2.2	spread—“长”数据变“宽”	32
6.3.3	separate/unite—拆分合并列	33
6.3.4	replace_na/drop_na—默认值处理工具	34
6.3.5	fill/complete—填坑神器	35
6.3.6	separate_rows/nest/unest—行数据处理	35
6.3.6.1	separate_rows—拆分单元格	35
6.3.6.2	nest/unest—压缩和解压缩行数据	36
6.4	lubridate 日期时间处理	37
6.4.1	为什么使用 lubridate	37
6.4.2	ymd/ymd_hms—年月日还是日月年	37
6.4.3	year/month/week/day/hour/minute/second—时间单位提取	39
6.4.4	guess_formats/parse_date_time—时间日期格式分析	39
6.5	stringr 字符处理工具	41
6.5.1	baseR vs stringr	41
6.5.2	正则表达式基础	41
6.5.3	简易正则表达式创建	42
6.5.4	文本挖掘浅析	44

Part I

Head first on data analysis

Chapter 1

数据分析引言：分解数据

1.1 数据分析的基本流程:

1. 确定: 了解问题, 确定问题.
2. 分解: 数据分析总的来说就是分解问题和数据, 使其成为更小的组成部分.
3. 评估: 在这一步对前两步了解到的情况做出各种结论.
4. 决策: 把这些结论重新组合在一起, 作出建议/决策.

1.1.1 确定问题

客户将根据你的分析做出决策, 你需要尽量从他那里了解多一些信息, 才能确定问题.

- 你对客户了解越深, 分析越有可能派上用场.
- 没必要先在脑子里形成问题再去浏览数据. 数据分析总的来说就是认清问题, 继而解决问题.
- 优秀的数据分析师帮助客户思考自己的问题.

你对外界的假设和你确信的观点就是你的心智模型. 心智模型决定你的观察结果, 是你观察现实的棱镜. 你无法看到一切, 因此大脑必须做出选择, 以便集中注意力, 这就是所谓的心智模型大大决定观察结果. 如果你了解你的心智模型, 那么你发现重点, 开发最相关最有用的统计模型的可能性就更大.

心智模型应当包括你不确定的因素. 一定要指出不确定的因素, 只要能明确不确定因素, 你就会小心防范并想办法填补知识的空白, 继而提出更好的建议. 考虑不确定因素及盲点会让人感觉不爽, 但回报显著.

许多由心智模型完成的工作都是为了帮助你填补信息空白. 好的一面是, 数据分析工具让你有能力以系统而自信的方式填补这些空白, 因此指出大量不确定因素, 这一做法就是帮助你发现盲点, 这要求拥有过硬的数据工作经验.

1.1.2 分解问题

你面对的问题常常含糊不清, 需要将问题划分为可管理, 可解决的组块.

从数据开始时, 尝试分解最重要的因子的最好的起步方法就是找出高效的比较因子. 进行有效的比较是数据分析的核心.

1.1.3 评估组块

评估分解组块的关键就是比较.(针对问题的观察结果/因子)(针对数据的观察结果/因子)

让自己介入分析的意思是做出自己明确的假设, 并且以自己的信用为自己的结论打赌. 在撰写最终报告的时候, 一定要提到你自己, 这样客户才知道你的结论出自何处.

1.1.4 提出建议

你的工作是确保自己的意见传达到位, 让人们根据你的意见做出正确的决策.

最初错误假设注定了分析会得出错误的答案, 因此, 从一开始就务必要基于正确的假设建立模型显得如此重要, 并且要做好准备, 一旦所得到的数据有违你的假设, 就要立即回头重新详加思考.

Chapter 2

实验：检验你的理论

一个好实验往往能让你摆脱对观察数据的无限依赖, 帮助你理清因果关系; 可靠的实证证据将让你的分析判断更有说服力.

以咖啡销售为例.

2.1 随机调查

进行客户调查, 随机性.

2.2 比较法与观察研究法

1. 统计与分析最基本的原理之一就是比较法, 数据只有通过相互比较才会有意义.
2. 统计只有与其他统计相关联, 才能给人带来启发. 必须进行明确的比较. 如果一份统计数据看起来颇有意思, 或看来很有用, 你就需要针对这份统计数据与其他数据的比较情况, 解释为什么会有这种作用. 如果不搞清楚这一点, 就等于在假设客户会自己进行这种比较, 这是一个不合格的分析. 比较越多, 分析结果越正确, 对于观察研究尤其如此. 通过观察数据, 你仅仅是在观察人们自己决定所属的群体. 搜集观察数据往往是通过实验取得更有用数据的第一步.

观察研究法: 被研究的人自行决定自己属于哪个群体的一种研究方法

3. 分析师们的一个很好的经验法则是, 当你开始怀疑因果关系的走向时, 如价值感的下降导致销量下降, 请进行反向思考, 如销量下降导致价值感下降, 看看结果怎么样.
4. 当涉及判定因果关系时, 观察研究法并不是那么强大有力. 观察数据无处不在, 但是要了解观察研究法的局限性, 这样才不会得出错误的结论.
5. 混杂因素是观察研究法绕不开的问题. 作为分析师, 你的工作就是不断考虑混杂因素对分析结果的影响. 如果你认为混杂因素的影响微不足道, 很好; 但如果有理由相信这些混杂因素正在引起问题, 那么就需要相应的调整自己的结论. 混杂因素通常不会故意在你面前晃悠, 为了让自己的数据尽量有说服力, 你需要自己动手把这些隐藏的混杂因素挖出来.

那要做到什么程度才算是查清楚了混杂因素?

这与其说是科学, 莫如说是艺术. 你不妨就自己正在研究的问题问自己一些常识性的问题, 借此想象哪些变量可能会影响你的分析结果. 正如数据分析和统计学中的各种手段一样, 无论你的量化技术多么出神入化, 真正的重点永远是分析的结论要有意义. 只要结论有意义, 而且你已经彻头彻尾地查找过混杂因素, 那么你就已经做了观察研究法要求你做的一切工作.

2.3 拆分数据块, 管理混杂因素

为了控制观察研究混杂因素, 有时将数据拆分为更小的数据块是个好想法.

这些小数据块更具同质性, 换句话说, 这些小数据块不包含那些有可能扭曲你的分析结果及让你产生错误想法的内部偏差. 如将不同地区分店的调查数据分开.

2.4 实验以控制组为基准

控制组, 一组体现现状的处理对象, 未经过任何新的处理 (也称对照组). 好的实验总是有一个控制组 (对照组), 使分析师能够将检验情况与现状进行比较.

控制组和实验组应尽量除了处理因素之外, 其余保持相同.

2.5 随机选择相似组

从对象池中随机选择对象是避免混杂因素的极好办法. 在将对象随机分配到各组里以后, 最终的结果是: 可能成为混杂因素的那些因素最终在控制组和实验组中具有同票同权.

通过随机选择组成各个组的成员, 组与组之间将非常相似, 因而具有可比性.

假定在实验中利用随机性将人群分进实验组和控制组, 结果是, 两个组中的混杂因素 X 最终分量一样. 如果总人数中有半数人含有这种隐性因素, 那么划分的每个组中也有半数的人含有这种隐性因素. 这就是随机法的力量.

随机控制是各种实验的黄金标准. 没有它你也能做实验, 但是有了它, 你就做得最好. 随即控制实验能让你最大限度地接近数据分析的核心: 证明因果关系.

Chapter 3

最优化: 寻找最大值

问题背景: 帮忙找出理想的产品组合使得利润提高.

3.1 需要哪些数据

最好能知道橡皮鸭和橡皮鱼的盈利能力; 约束这个问题的其他因素; 生产原料需求量; 生产所需时间...

可以将所需要的数据分为两类: A: 无法控制的因素 B: 可以控制的因素.

A:

- 橡皮鱼的利润如何, 橡皮鸭的利润如何
- 厂家有多少橡胶可以用来生产橡皮鱼, 生产橡皮鱼需要多长时间
- 厂家有多少橡胶可以用来生产橡皮鸭, 生产橡皮鸭需要多长时间

B:

- 生产多少橡皮鸭
- 生产多少橡皮鱼

1. 你需要得到有关能控制因素和不能控制因素的可靠数字.
2. A 这些考虑事项被称为约束条件, 因为它们将决定问题的有关参数. 我们的追求是利润.
3. 决策变量是你能控制的因素, 即 B.

3.2 最优化问题

当你希望尽量多获得/少获得某种东西, 而为了实现这个目的需要改变其他一些量的数值, 你就碰到了一个最优化问题.

3.2.1 借助目标函数发现目标

最简单的目标函数:

$$c_1 x_1 + c_2 x_2 = P \quad (3.1)$$

c_i 是约束条件, x_i 是决策变量, P 是目标, 即期望的最大化对象.

总利润 = 鸭利润 + 鱼利润

3.2.2 约束条件

划出可行域.

3.2.3 excel 实现最优化

3.2.4 按照分析目标校正假设

实际上按照以上所得的结果安排生产未必得到希望的结果, 你的模型告诉你如何实现最大利润, 但仅仅是在你所规定的约束条件下. 模型中没有任何因素表明人们真正会购买此产品, 有人购买, 模型才会生效.

利用历史销售数据可以了解人们愿意买什么, 什么时候愿意买.

3.2.5 提防负相关变量

查看数据, 可以猜测甚至肯定橡皮鸭和橡皮鱼是负相关关系. 在节假日销售高峰期间, 两种产品会同时出现上升趋势, 但永远有一种产品比另一种更领先.

不要假定两种变量是不相关的. 创建模型时, 务必要规定假设中的各种变量的相互关系.

你需要增加新约束条件用于估计某个月的橡皮鸭和橡皮鱼的需求量.

3.2.6 假设立足于不断变化的实际情况

你所使用的数据都是观察数据, 你无法预知未来. 你的模型现在是起作用, 但是可能会突然失灵, 需要做好准备, 以便在必要的时候重新构建分析方法.

反复不断地进行构建正是分析师的工作.

Chapter 4

数据图形化：图形让你更精明

4.1 体现数据

创建优秀数据图形的第一要务就是促使客户谨慎思考并制定正确地决策，优秀的数据分析始终都离不开“用数据思考”。

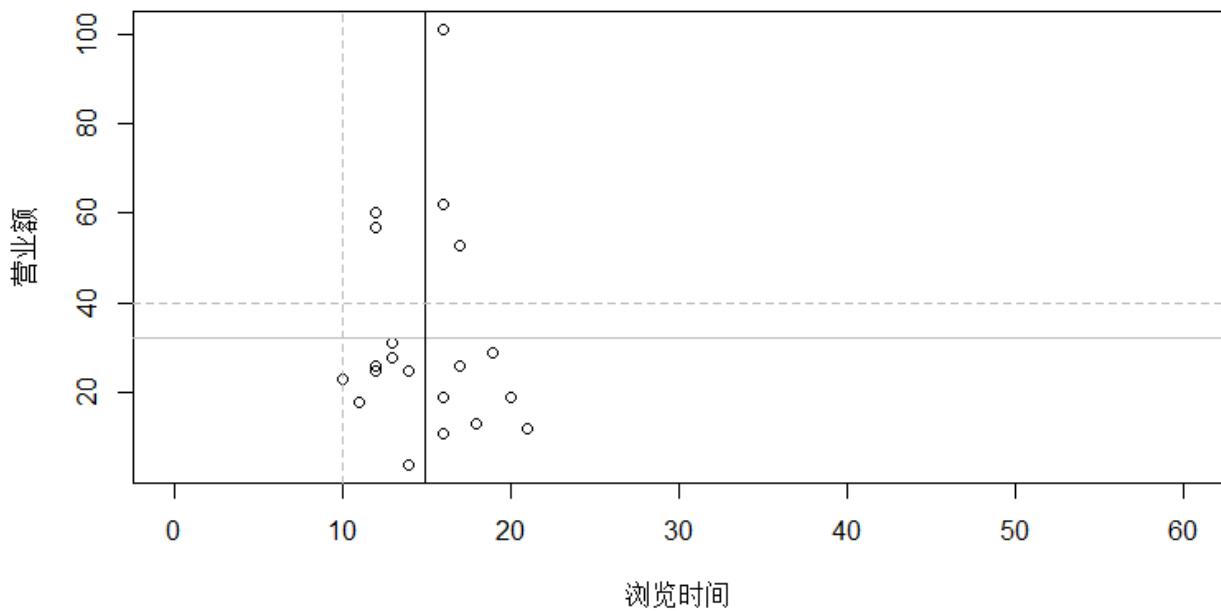
数据太多会给绘制优秀图形带来困难？

并非全无道理。数据分析的根本在于总结数据，而一些总结工具如求均值，不困数据多少，都同样有效。要是你手头有林林总总的数据可供相互比较，这的确很妙。

要是手头数据庞杂，而且对于如何处理这些数据没有把握，这时候只要记住你的分析目标：记住目标，目光停留在和目标有关的数据。而且只要数据图形可以解决客户的问题，那么不管是图形设计精美还是扎眼，都会对客户有吸引力。

4.2 数据图形化的根本在于正确比较

```
1 > page<-read.csv("E:/数据分析/headfirst/HeadFirstDataAnalysisCode/
2           hfda_data/hfda_ch04_home_page1.csv",
3           stringsAsFactors = FALSE)
4 > plot(page$TimeOnSite,page$Revenue,
5 +       xlab = "浏览时间",ylab = "营业额",xlim=c(0,60))
6 > abline(h=mean(page$Revenue),col="grey")
7 > abline(v=mean(page$TimeOnSite,col="grey")))
8 > abline(h=40,lty=2,col="grey")
9 > abline(v=10,lty=2,col="grey")
```



优秀图形的特点:

- 展示数据
- 作了高明的比较
- 展示了多个变量

散点图是探索性数据分析的工具, 分析师喜欢用散点图发现因果关系.

4.3 最优秀的图形都是多元图形

如果一个图形可以对三个以上的变量进行比较, 那么就是多元图形. 只要加上有效的数据分析的基础, 于是尽量让图形多元化最有可能促成最有效的比较.

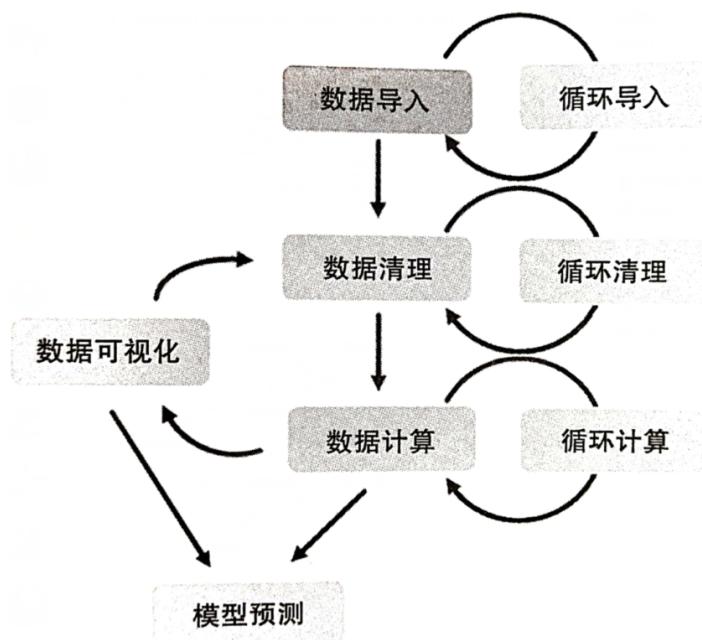
有一个办法让图形多元化, 即多张相似的散点图相邻排放.

Part II

R 数据科学实战—A. 工具详解

Chapter 5

数据导入工具



5.1 utils—数据读取

5.1.1 read.csv/csv2—逗号分隔数据读取

即便是以.csv 为拓展名的文件也并非一定是以逗号进行分隔的, 文件的拓展名也并非必须. 处理无拓展名的文本文件数据时, 最简单的办法就是使用 data.table 包中的 fread 函数.

utils 包里的 read.csv/csv2 是专门用来设置快速读取逗号分隔 (read.csv) 或分号分隔 (read.csv2).
注意: 这两个函数对小数点的处理: 前者默认小数点为"?", 后者","

```
1 > d8<-read.csv("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-Action-R-Tools-and-Case-Studies-master/chapter1/RawData/flights.csv",header=TRUE)
```

数据文件被读取到 R 工作环境中第一步通常为调用 str 函数来对该数据对象进行初步检视.

```
1 > str(d8)
2 'data.frame': 6 obs. of 6 variables:
```

```

3 $ carrier : Factor w/ 4 levels "AA","B6","DL",...: 4 4 1 2 3 4
4 $ flight : int 1545 1714 1141 725 461 1696
5 $ tailnum : Factor w/ 6 levels "N14228","N24211",...: 1 2 4 6 5 3
6 $ origin : Factor w/ 3 levels "EWR","JFK","LGA": 1 3 2 2 3 1
7 $ dest : Factor w/ 5 levels "ATL","BQN","IAH",...: 3 3 4 2 1 5
8 $ air_time: int 227 227 160 183 116 150

```

其他用来检视数据的函数有 head,tail,view 等.

前文提到, .csv 并非一定是以逗号进行分隔, 如果遇到以非逗号分隔数据值的情况, 加之未指定分隔符, 会出现读取错误.

```

1 > d9<-read.csv("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-Action-R-
   Tools-and-Case-Studies-master/chapter1/RawData/flights1.csv",header=TRUE)
2 > str(d9)
3 'data.frame': 6 obs. of 1 variable:
4 $ carrier.flight.tailnum.origin.dest.air_time: Factor w/ 6 levels "AA\t1141\
   \tN619AA\tJFK\tMIA\t160",...: 4 6 1 2 3 5
5
6
7 > d9<-read.csv("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-Action-R-
   Tools-and-Case-Studies-master/chapter1/RawData/flights1.csv",sep="\t",header=
   TRUE)
8 > str(d9)
9 'data.frame': 6 obs. of 6 variables:
10 $ carrier : Factor w/ 4 levels "AA","B6","DL",...: 4 4 1 2 3 4
11 $ flight : int 1545 1714 1141 725 461 1696
12 $ tailnum : Factor w/ 6 levels "N14228","N24211",...: 1 2 4 6 5 3
13 $ origin : Factor w/ 3 levels "EWR","JFK","LGA": 1 3 2 2 3 1
14 $ dest : Factor w/ 5 levels "ATL","BQN","IAH",...: 3 3 4 2 1 5
15 $ air_time: int 227 227 160 183 116 150

```

根据实际情况, 字符型数据有时会是因子, 有时不会. 如果使用 read.csv 默认的读取方式, 那么字符型数据全因子化, 会对后续的数据分析带来很多麻烦. 所以最好是将字符因子化关掉.(stringsAsFactor)

```

1 > d10<-read.csv("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-Action-R-
   Tools-and-Case-Studies-master/chapter1/RawData/flights1.csv",stringsAsFactors
   =F,sep="\t",header=TRUE)
2 > str(d10)
3 'data.frame': 6 obs. of 6 variables:
4 $ carrier : chr "UA" "UA" "AA" "B6" ...
5 $ flight : int 1545 1714 1141 725 461 1696
6 $ tailnum : chr "N14228" "N24211" "N619AA" "N804JB" ...
7 $ origin : chr "EWR" "LGA" "JFK" "JFK" ...
8 $ dest : chr "IAH" "IAH" "MIA" "BQN" ...
9 $ air_time: int 227 227 160 183 116 150

```

5.1.2 read.delim/delim2—特定分隔符数据读取

这两个函数是专门用来处理以 tab 分割数据的文件的.

delim 可用来读取小数点是”.”的数据,delim2 则用来处理小数点”,.”的数据.

5.1.3 read.table—任意分隔符数据读取

read.table 函数会将文件读成数据框的格式, 将分隔符作为区分变量的依据.

参数名称	功能描述
file	数据文件路径 + 文件名, 也可以是一个 url, 或者是文字数据
header	设置逻辑值来指定函数是否将数据文件的第一列作为列名。默认为假
sep	不同变量之间的分隔符, 特指分隔列数据的分隔符。默认值为空, 可以是 “,”、“\t” 等
quote	单双引号规则的设置。如果不希望设置该参数, 则需要指定其为空: quote = ""
dec	用作小数点的符号, 一般为句点或者逗号
row.names	行名。可以通过指定一组向量来进行设置。如果文件中的第一行比数据整体的列数量少一时, 则会默认使用第一列来作为行名
col.names	列名。可以通过指定一组向量来进行列名设置
na.strings	对默认值的处理
colClasses	变量类型的设置。通过指定一组向量来指定每列的变量数据类型, 具体使用方式为: colClasses = c ("character", "numeric", …)
fill	设置逻辑值来处理空白值部分, 使用方法请参见代码演示部分
strip.white	设置逻辑值来处理空白列。某些数据文件内可能会预留一些变量列, 但数据采集后这些预留的列并未被填满, 而是仍然保留着制表符, 该参数就是用来处理掉这些意义不大的制表符
blank.lines.skip	空白行是否跳过, 默认为真, 即跳过
stringsAsFactors	字符串是否作为因子, 推荐设置为否
skip	跳过几行读取原始数据文件, 默认设置为 0, 表示不跳过任何一行, 从文件第一行开始读取, 可以传参任意数字

```

1 > d11<-read.table("E:/7. 数据分析/ headfirst&R 数据科学实战 /Data-Science-in-Action-
  R-Tools-and-Case-Studies-master /chapter1 /RawData /flights.csv")
2 > head(d11)
3 V1
4 1 carrier,flight,tailnum,origin,dest,air_time
5 2           UA,1545,N14228,EWR,IAH,227
6 3           UA,1714,N24211,LGA,IAH,227
7 4           AA,1141,N619AA,JFK,MIA,160
8 5           B6,725,N804JB,JFK,BQN,183
9 6           DL,461,N668DN,LGA,ATL,116

```

因为函数默认的分隔符是空白 (不是空格), 所以应有的 6 个变量都被堵在一列中.

指定 header 参数为真:

```

1 > d11<-read.table("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-Action-
   R-Tools-and-Case-Studies-master/chapter1/RawData/flights.csv",header=T)
2 > head(d11)
3   carrier.flight.tailnum.origin.dest.air_time
4   1           UA,1545,N14228,EWR,IAH,227
5   2           UA,1714,N24211,LGA,IAH,227
6   3           AA,1141,N619AA,JFK,MIA,160
7   4           B6,725,N804JB,JFK,BQN,183
8   5           DL,461,N668DN,LGA,ATL,116
9   6           UA,1696,N39463,EWR,ORD,150

```

指定分隔符:

```

1 > d11<-read.table("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-Action-
   R-Tools-and-Case-Studies-master/chapter1/RawData/flights.csv",header=T,sep=",",
   "")
2 > head(d11)
3   carrier flight tailnum origin dest air_time
4   1     UA    1545  N14228    EWR   IAH     227
5   2     UA    1714  N24211    LGA   IAH     227
6   3     AA    1141  N619AA    JFK   MIA     160
7   4     B6     725  N804JB    JFK   BQN     183
8   5     DL     461  N668DN    LGA   ATL     116
9   6     UA    1696  N39463    EWR   ORD     150

```

5.1.3.1 空白行

read.table 对于空白行的默认处理: 跳过.

但在有些特殊情况下, 如一个表有两个数据集, 或者空白行以上为元数据, 也即解释数据的数据.

```

1 > d12<-read.table("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-Action-
   R-Tools-and-Case-Studies-master/chapter1/RawData/airlines.csv",header=T,sep="",
   \t",blank.lines.skip = FALSE,stringsAsFactors = FALSE)
2 > head(d12,n=8)
3   carrier          name
4   1     AA American Airlines Inc.
5   2     B6      JetBlue Airways
6   3     DL  Delta Air Lines Inc.
7   4
8   5 carrier          flight
9   6 tailnum          origin
10  7 dest            air_time
11  8 AA                  1141

```

可是又出现新问题: 函数按照第一部分的两类变量将后续的所有数据也都写入了两列.

解决:

```

1 > d12<-read.table("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-Action-
  R-Tools-and-Case-Studies-master/chapter1/RawData/airlines.csv",header=T,sep="\\t",
  col.names=paste0("V",1:6),blank.lines.skip = FALSE,stringsAsFactors =
  FALSE)
2 Warning message:
3 In read.table("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-Action-R-
  Tools-and-Case-Studies-master/chapter1/RawData/airlines.csv", :
4 表头名称和'col.names'的长度不一样
5 > head(d12,n=9)
6
7      V1                  V2      V3      V4      V5      V6
8 1     AA American Airlines Inc.
9 2     B6 JetBlue Airways
10 3    DL Delta Air Lines Inc.
11 4
12 5 carrier                  flight tailnum origin dest air_time
13 6   AA                      1141  N619AA    JFK  MIA      160
14 7   B6                      725   N804JB    JFK  BQN      183
15 8   DL                      461   N668DN    LGA  ATL      116

```

这里用 paste0 来创建新的变量名称.paste0 可以理解为胶水函数, 用于将需要的字符串粘合在一起. 这里演示的意思是创建 6 个以 V 开头, 从 V1 到 V6 的字符串作为变量名. 这种处理方式足以应付平时练习用的小型数据集.

下面代码演示了如何实现自动检测数据及所需的变量数:

```

1 > num_of_col<-max(count.fields("E:/7. 数据分析/headfirst&R数据科学实战/Data-
  Science-in-Action-R-Tools-and-Case-Studies-master/chapter1/RawData/airlines.
  csv"))
2
3 > d13 <- read.table("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-
  Action-R-Tools-and-Case-Studies-master/chapter1/RawData/airlines.csv",header=
  T,sep="\\t",col.names=seq_len(num_of_col)),blank.lines.skip = FALSE
  ,stringsAsFactors = FALSE)
4 Warning message:
5 In read.table("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-Action-R-
  Tools-and-Case-Studies-master/chapter1/RawData/airlines.csv", :
6 表头名称和'col.names'的长度不一样
7
8 > head(d13)
9
10      V1                  V2      V3      V4      V5      V6
11 1     AA American Airlines Inc.
12 2     B6 JetBlue Airways
13 3    DL Delta Air Lines Inc.
14 4
15 5 carrier                  flight tailnum origin dest air_time
16 6   AA                      1141  N619AA    JFK  MIA      160

```

count.fields 用于自动检测数据集中每一行数据的观测值个数.

max 用于找出 count.fields 输出结果中的最大值.

seq_len 用于以最大值为参照生成 1 到最大值的整数序列.

paste0 用于定义变量名称.

5.1.3.2 默认值, 空白

理论上来讲, 默认值仍是数据观测值的一种, 虽然在原始数据中其可能与空白一样没有显示, 但是它可以通过其他手段来进行补齐.

而空白可能并不是数据, 比如上面的演示中, V3-V6 的前四行都是空白, 这些空白不属于任何实际数据变量, 是真正的空白, 因而不能说这些空白是默认值.

如何进行简单地默认值, 空白的预处理:

```

1 > d14 <- read.table("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-
2   Action-R-Tools-and-Case-Studies-master/chapter1/RawData/flights Uneven.csv",
3   header=F,sep="\t",na.strings = c(""),
4   fill=T,stringsAsFactors = FALSE)
5 > head(d14)
6
7   V1      V2      V3      V4      V5      V6      V7
8 1 carrier flight tailnum origin dest air_time <NA>
9 2     UA    1545  N14228    EWR    IAH      227 <NA>
10 3     UA    1714  N24211    LGA    IAH      227 测试1
11 4     AA    1141  N619AA    JFK    MIA      160 测试2
12 5     B6     725  N804JB    JFK    BQN      183 测试3
13 6     DL     461  N668DN    LGA    ATL      116 <NA>

```

第七列的数据在指定将空白替换成"NA" 之后, 原有的空白位置被写入了"NA". 也就是说, 第七列的空白属于数据的一部分. 根据实际情况, 也可以将多余的数据部分或全部替换成 NA, 以方便后续的处理和分析.

```

1 > d14 <- read.table("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-
2   Action-R-Tools-and-Case-Studies-master/chapter1/RawData/flights Uneven.csv",
3   header=F,sep="\t",na.strings = c(paste0("测试",1:3),""),
4   fill=T,
5   stringsAsFactors = FALSE)
6 > head(d14)
7
8   V1      V2      V3      V4      V5      V6 V7
9 1 carrier flight tailnum origin dest air_time NA
10 2     UA    1545  N14228    EWR    IAH      227 NA
11 3     UA    1714  N24211    LGA    IAH      227 NA
12 4     AA    1141  N619AA    JFK    MIA      160 NA
13 5     B6     725  N804JB    JFK    BQN      183 NA
14 6     DL     461  N668DN    LGA    ATL      116 NA

```

当数据集行数较多, 不能直接判断 NA 的个数时, 可以配合 unique 函数找到指定列中的非重复观测值, 选取指定观测值并保存到一个向量内, 然后将向量指定给 na.strings 参数来进行替换:

```

1 > d14 <- read.table("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-
2   Action-R-Tools-and-Case-Studies-master/chapter1/RawData/flights Uneven.csv",
3   header=F,sep="\t",fill=T,stringsAsFactors = FALSE)
4 > replace <- unique(d14$V7)
5 > replace

```

```

4 [1] ""      "测试1" "测试2" "测试3"
5 > d14 <- read.table("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-
  Action-R-Tools-and-Case-Studies-master/chapter1/RawData/flights_uneven.csv",
  header=F,sep="\t",fill=T,stringsAsFactors = FALSE,na.strings = c(replace[c
  (1,3)]))
6 > head(d14)
7      V1      V2      V3      V4      V5      V6      V7
8 1 carrier flight tailnum origin dest air_time <NA>
9 2     UA    1545  N14228    EWR    IAH      227 <NA>
10 3     UA    1714  N24211    LGA    IAH      227 测试1
11 4     AA    1141  N619AA    JFK    MIA      160 <NA>
12 5     B6     725  N804JB    JFK    BQN      183 测试3
13 6     DL     461  N668DN    LGA    ATL      116 <NA>

```

全部替换:

```

1 > d14 <- read.table("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-
  Action-R-Tools-and-Case-Studies-master/chapter1/RawData/flights_uneven.csv",
  header=F,sep="\t",fill=T,stringsAsFactors = FALSE,na.strings = c(replace))
2 > head(d14)
3      V1      V2      V3      V4      V5      V6 V7
4 1 carrier flight tailnum origin dest air_time NA
5 2     UA    1545  N14228    EWR    IAH      227 NA
6 3     UA    1714  N24211    LGA    IAH      227 NA
7 4     AA    1141  N619AA    JFK    MIA      160 NA
8 5     B6     725  N804JB    JFK    BQN      183 NA
9 6     DL     461  N668DN    LGA    ATL      116 NA

```

5.2 readr—进阶数据读取

优势:

- 更快. readr 包里的 read_csv 一般要比 read.csv 快三到十倍.
- 默认设置更简洁. 默认情况下 readr 包会自动解析每列的数据类型, 显示解析结果, 无需设置 stringsAsFactors
- 对数据类型的解析更准确.

当一个.csv 数据中前面有很多空白行时, skip 参数可以直接跳过空白行来读取数据. 具体设置非常简单, skip=3 即表示跳过前三行数据, 从第四行开始读取. 这个参数并非只用于跳过空白行, 也可以用来读取原始数据的一部分, 配合 n_max 使用可以做到随心所欲读取任何一部分数据.

5.3 pdfroools—pdf 文件

一般计量型数据分析很少会遇到读取 pdf 文件的情况, 但是在文本挖掘和主题模型预测中, pdf tools 包是必备的. 这个包只有两个母函数: 一个用来从 pdf 文件中提取数据, 另一个则用来将文件渲染成 pdf 格式, 本节只讨论前一个 *pdf_info*

它包括 6 个子函数:

表 1-15 数据导入函数 'pdf_info' 子函数一览

名称	功能描述
pdf_info	读取 PDF 文件的基本信息, 例如, 何时创建、更改, 版本信息, 是否有密码, 页数等, 洋见代码演示部分
pdf_text	提取文件中的所有文字或非文字信息, 包括分页符、换行符
pdf_data	提取数字型数据, 这个提取的结果会因 PDF 文件而异, 有时可以直接将期刊中的数据完整地提取出来, 有时又会因为 PDF 文档在创建时使用了不一致的分隔符而导致数据提取不完整
pdf_fonts	提取文档的字体信息

(续)

名称	功能描述
pdf_attachments	提取文档附件
pdf_toc	提取文档目录

表 1-16 数据导入函数 'pdf_info' 参数详解

参数名称	功能描述
pdf	PDF 文件路径, 可以是网络链接
opw	PDF 文件所有者的密码
upw	PDF 文件用户的密码

读取文档的代码如下:

```

1 > library(pdftools)
2 Using poppler version 0.73.0
3 > pdf_info(pdf = "E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-Action-R
   -Tools-and-Case-Studies-master/chapter1/RawData/pdftools.pdf")
4 $version
5 [1] "1.5"
6
7 $pages
8 [1] 5
9
10 $encrypted
11 [1] FALSE
12
13 $linearized
14 [1] FALSE
15
16 $keys

```

```

17 $keys$Author
18 [1] """
19
20 $keys>Title
21 [1] """
22
23 $keys$Subject
24 [1] """
25
26 $keys$Creator
27 [1] "LaTeX with hyperref package"
28
29 $keys$Producer
30 [1] "pdfTeX-1.40.15"
31
32 $keys$Keywords
33 [1] """
34
35 $keys$Trapped
36 [1] """
37
38 $keys$PTEX.Fullbanner
39 [1] "This is pdfTeX, Version 3.14159265-2.6-1.40.15 (TeX Live 2015/dev/Debian)
      kpathsea version 6.2.1dev"
40
41
42 $created
43 [1] "2018-05-27 21:56:10 CST"
44
45 $modified
46 [1] "2018-05-27 21:56:10 CST"
47
48 $metadata
49 [1] """
50
51 $locked
52 [1] FALSE
53
54 $attachments
55 [1] FALSE
56
57 $layout
58 [1] "no_layout"

```

当使用 `pdf_text` 读取文档内容时, 全部内容被提取为一个字符串向量, 每页的内容都被单独放置于一个字符串中.

帮助文档的 `pdf` 格式一共包含 5 页, 所以这里会得到一个长度为 5 的字符串向量.

有两个方式可以用于查看提取的文本: 可以直接将结果显示在 console 中, 也可以通过 [] 来指定显示某一页的内容.

文档一共包括 6 种字体, pdf fonts 会给出字体的名称, 类型, 是否嵌入文档中这三类信息:

```
1 > pdf_fonts("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-Action-R-  
Tools-and-Case-Studies-master/chapter1/RawData/pdftools.pdf")  
2 # A tibble: 6 x 4  
3   name                      type  embedded  file  
4   <chr>                     <chr> <lg1>     <chr>  
5 1 DSHWTW+NimbusRomNo9L-Medi    type1  TRUE     ""  
6 2 UTHPMJ+NimbusRomNo9L-Regu    type1  TRUE     ""  
7 3 DSQFGA+Inconsolata-zi4r     type1  TRUE     ""  
8 4 LVIJIF+NimbusSanL-Regu      type1  TRUE     ""  
9 5 DQRZJT+NimbusRomNo9L-Regu-Slant_167 type1  TRUE     ""  
10 6 YIECHJ+NimbusRomNo9L-ReguItal type1  TRUE     ""
```

Chapter 6

数据清理工具

一般来讲, 从数据收集到最后报告的整个过程中, 数据清理会占用整个流程 80% 的时间. 如此耗时的原因是数据清理并非一次性工作, 数据清理, 计算和可视化是一个动态的循环. 根据分析需求的不同, 需要应用不同的清理思路和方式.

本章分享数据清理的一些基本原则, 作为框架来指导数据清理工作.

6.1 基本概念

国际上公认的”干净”的数据可以总结为如下三点:

- 属性相同的变量自成一列
- 单一观测自成一行
- 每个数据值必须独立存在

元数据 (metadata): 解释变量名称或数据背景的数据.

6.2 tibble 包-数据集准备

tibble 既是 R 包的名字也是数据在 R 中的一种存储格式. 可以将 tibble 包理解为 R 中最常见的 data.frame 格式的升级版.

如果使用 read.csv 读取数据, 那么数据就会被储存在 data.frame 格式中. 但是当调用 read_csv 时, 数据就会存在三种适用格式:tbl_df, tbl, data.frame.

因为 tibble 和 readr 包都源自 Hadley 的 tidy 系列, 所以使用 readr 包时自动植入了 tibble 的数据格式.

```
1 > iris <- read.csv("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-Action
2   -R-Tools-and-Case-Studies-master/chapter2/RawData/iris.csv", stringsAsFactors
3   = F)
4
5 > class(iris)
[1] "data.frame"
```

```

6 > iris <- read_csv("E:/7. 数据分析/headfirst&R数据科学实战/Data-Science-in-Action
7 -R-Tools-and-Case-Studies-master/chapter2/RawData/iris.csv")
8 Parsed with column specification:
9 cols(
10 Sepal.L..Setosa = col_double(),
11 Sepal.W..Setosa = col_double(),
12 Petal.L..Setosa = col_double(),
13 Petal.W..Setosa = col_double(),
14 Sepal.L..Versicolor = col_double(),
15 Sepal.W..Versicolor = col_double(),
16 Petal.L..Versicolor = col_double(),
17 Petal.W..Versicolor = col_double(),
18 Sepal.L..Virginica = col_double(),
19 Sepal.W..Virginica = col_double(),
20 Petal.L..Virginica = col_double(),
21 Petal.W..Virginica = col_double()
22 )
22 > class(iris)
23 [1] "spec_tbl_df" "tbl_df"        "tbl"           "data.frame"

```

6.2.1 为什么使用 tibble

三点优势:

- 稳定性更好, 可完整保存变量名称及属性
- 更多的信息展示, 警示提醒
- 新的输出方式使得浏览数据时, 屏幕的利用率极佳.

第一章中提到查看 data.frame 中的变量类型时, 通常需要调用 str 函数. 但是在 tbl 格式中, 无需调用任何函数.

默认情况下, tbl 格式会根据 console 窗口的大小自动调整内容的展示. 内容会包括数据格式, 列总数, 行总数, 变量名称和类型, 以及无法完全展示部分的变量信息.

```

1 > iris
2 # A tibble: 50 x 12
3   Sepal.L..Setosa Sepal.W..Setosa Petal.L..Setosa Petal.W..Setosa
4     <dbl>       <dbl>       <dbl>       <dbl>
5 1       5.1        3.5        1.4        0.2
6 2       4.9        3          1.4        0.2
7 3       4.7        3.2        1.3        0.2
8 4       4.6        3.1        1.5        0.2
9 5       5          3.6        1.4        0.2
10 6      5.4        3.9        1.7        0.4
11 7      4.6        3.4        1.4        0.3
12 8      5          3.4        1.5        0.2
13 9      4.4        2.9        1.4        0.2

```

```

14 10          4.9          3.1          1.5          0.1
15 # ... with 40 more rows, and 8 more variables: Sepal.L..Versicolor <dbl>,
16 #   Sepal.W..Versicolor <dbl>, Petal.L..Versicolor <dbl>,
17 #   Petal.W..Versicolor <dbl>, Sepal.L..Virginica <dbl>,
18 #   Sepal.W..Virginica <dbl>, Petal.L..Virginica <dbl>,
19 #   Petal.W..Virginica <dbl>

```

6.2.2 创建 tbl 格式

可以通过函数 `tibble` 或者 `tribble` 来创建新的数据框.

`<int>` 表示 `integer` 整数, `<dbl>` 表示 `double` 浮点型.

创建数据框:

```

1 > library(tibble)
2 > tibble(a=1:6,b=a^2)
3 # A tibble: 6 x 2
4   a     b
5 <int> <dbl>
6 1     1     2
7 2     2     4
8 3     3     6
9 4     4     8
10 5    5    10
11 6    6    12

```

`tibble` 函数比较适合用来创建小型数据集.

6.2.3 `as_tibble`—转换已有格式的数据集

可以先使用 `is_tibble` 来测试目标对象是否已是 `tbl` 格式.

可以通过 `as_tibble` 函数将对象已有的格式 (`vector`、`matrix`、`list` 和 `data.frame` 等) 转换成 `tbl`。表 2-8 中列出了常见对象格式的转换注解。

表 2-8 常见 R 对象与 `tibble` 格式的转换注解

对象格式	解 释
<code>vector</code> (向量)	将 <code>vector</code> 格式转换成“ <code>tbl</code> ”格式, 可以理解成在 Excel 表中, 将一行数据转置成按列进行排放
<code>matrix</code> (矩阵)	将矩阵转换成“ <code>tbl</code> ”。这里需要注意的是, 矩阵格式的对象通常会有 <code>rownames</code> (行名), <code>as_tibble</code> 的默认设置是去除行名, 如需保留行名, 则需要指定参数 <code>rownames = NA</code> 来实现
<code>data.frame</code> (数据框)	将传统数据框转换成“ <code>tbl</code> ”。这里会默认保留 <code>data.frame</code> 中原有的行名, 若想移除行名, 则需指定参数 <code>rownames = NULL</code>
<code>list</code> (列表, 数据)	将列表转换成“ <code>tbl</code> ”。这里必须注意的是, 列表中的每个 <code>element</code> (要素) 都必须要有相同数量的数据值。例外的情况是, 当一个列表中, 一个或多个要素只有一个数值时, 该数值会按照最长要素的长度自动循环补齐, 具体请看代码演示

6.2.3.1 as_tibble 函数直接将 vector 格式转换成数据框格式

```

1 > y <- 1:3
2 > y
3 [1] 1 2 3
4 > as_tibble(x=y)
5 # A tibble: 3 x 1
6   value
7   <int>
8   1     1
9   2     2
10  3     3

```

6.2.3.2 矩阵格式转换

```

1 > b <- matrix(data=1:9, nrow=3, byrow = T, dimnames = list(paste0("Row", 1:3), paste0
   ("col", 1:3)))
2 > b
3   col1 col2 col3
4 Row1    1    2    3
5 Row2    4    5    6
6 Row3    7    8    9
7
8 > as_tibble(x=b, rownames=NULL)
9 # A tibble: 3 x 3
10  col1  col2  col3
11  <int> <int> <int>
12  1     1     2     3
13  2     4     5     6
14  3     7     8     9
15
16 > as_tibble(x=b, rownames=NA)
17 # A tibble: 3 x 3
18  col1  col2  col3
19  * <int> <int> <int>
20  1     1     2     3
21  2     4     5     6
22  3     7     8     9

```

6.2.3.3 列表格式转换

```

1 > A<-list(a=1:3, b=letters[2:4], c=1)
2 > A
3 $a
4 [1] 1 2 3
5

```

```

6 $b
7 [1] "b" "c" "d"
8
9 $c
10 [1] 1
11
12 > as_tibble(x=A)
13 # A tibble: 3 x 3
14   a   b     c
15   <int> <chr> <dbl>
16 1     1   b     1
17 2     2   c     1
18 3     3   d     1

```

每个要素成为一个变量, 原列表中的要素 c 中的数值将被重复使用三次以对应其他变量的长度. 如果要素 c 的长度为 2, 即包含两个数值, 那么转换会失败.

6.2.4 add_row/column—实用小工具

创建一个 tibble, 使用 \$ 来为数据新增一列名为 k 的变量.

```

1 > f <- tibble(i=1:3, j=c("John", "Sam", "Joy"))
2 > f
3 # A tibble: 3 x 2
4   i   j
5   <int> <chr>
6 1     1 John
7 2     2 Sam
8 3     3 Joy
9
10 > f$k <- 3:1
11 > f
12 # A tibble: 3 x 3
13   i   j     k
14   <int> <chr> <int>
15 1     1 John     3
16 2     2 Sam      2
17 3     3 Joy      1

```

在数据框的末尾加入一行新数据也可以实现新增列的功能.

```

> f
# A tibble: 3 x 3
  i   j     k
  <int> <chr> <int>
1   1 John     3
2   2 Sam      2
3   3 Joy      1
> f[nrow(f)+1,] <- c(4, "Jon", 0)
错误: Assigned data `c(4, "Jon", 0)` must be compatible with row subscript `nrow(f) + 1`.
x 1 row must be assigned.
x Assigned data has 3 rows.
i Row updates require a list value. Do you need `list()` or `as.list()`?
Run `rlang::last_error()` to see where the error occurred.

```

```

1 > library(tibble)
2 > f
3 # A tibble: 3 x 3
4   i     j     k
5   <int> <chr> <int>
6 1     1 John     3
7 2     2 Sam      2
8 3     3 Joy      1
9
10 > add_row(f,i=4,j="Jon",k=0)
11 # A tibble: 4 x 3
12   i     j     k
13   <dbl> <chr> <dbl>
14 1     1 John     3
15 2     2 Sam      2
16 3     3 Joy      1
17 4     4 Jon      0

```

在第三行之前插入一行新数据:

```

1 > library(tibble)
2 > f
3 # A tibble: 3 x 3
4   i     j     k
5   <int> <chr> <int>
6 1     1 John     3
7 2     2 Sam      2
8 3     3 Joy      1
9 > add_row(f,i=4,j="Jon",.before = 3)
10 # A tibble: 4 x 3
11   i     j     k
12   <dbl> <chr> <int>
13 1     1 John     3
14 2     2 Sam      2
15 3     4 Jon      NA
16 4     3 Joy      1

```

在第一行之后插入新数据:

```

1 > add_row(f,i=4,j="Jon",.after = 1)
2 # A tibble: 4 x 3
3   i     j     k
4   <dbl> <chr> <int>
5 1     1 John     3
6 2     4 Jon      NA
7 3     2 Sam      2
8 4     3 Joy      1

```

在第一列之后插入新变量:

```

1 > add_column(f, 1=nrow(f):1, .after=1)
2 # A tibble: 3 x 4
3   i     l   j     k
4   <int> <int> <chr> <int>
5 1     1     3 John     3
6 2     2     2 Sam      2
7 3     3     1 Joy      1

```

6.3 tidyverse—数据清道夫

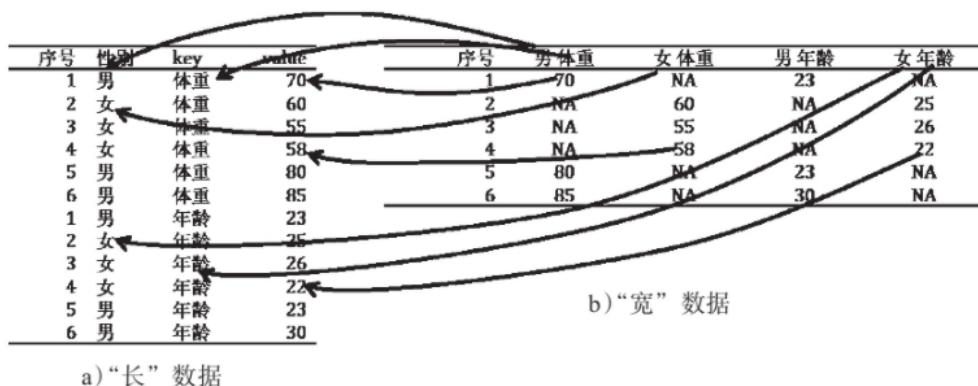
6.3.1 为什么使用 tidyverse 包

- 简洁直观的函数名称, 可读性极强-易上手
- 默认设置可以满足大部分使用需求, 无需时刻参考帮助文档-易使用
- 不同函数中的参数设置结构清晰-易于记忆
- 处理数据过程中完整保留了变量属性及数据格式-不易出现未知错误

6.3.2 gather/spread—“长””宽” 数据转换

6.3.2.1 gather—“宽”变”长”

在理想的情况下, 整洁的数据框应为如下图格式, 因子一列, 变量一列, 剩余所有数值型数据一列.



建立以下脏数据, 保存在名为 df 的数据框中:

```

1 > library(tibble)
2 > df <- tribble(~id, ~"b weight", ~"g weight", ~"b age", ~"g age",
3 +           1, 70, NA, 23, NA,
4 +           2, NA, 60, NA, 25,
5 +           3, NA, 55, NA, 26,
6 +           4, NA, 58, NA, 22,
7 +           5, 80, NA, 23, NA,
8 +           6, 85, NA, 30, NA)

```

```

9 > df
10 # A tibble: 6 x 5
11   id `b weight` `g weight` `b age` `g age`
12   <dbl>      <dbl>      <dbl>      <dbl>      <dbl>
13 1     1         70        NA        23        NA
14 2     2         NA        60        NA        25
15 3     3         NA        55        NA        26
16 4     4         NA        58        NA        22
17 5     5         80        NA        23        NA
18 6     6         85        NA        30        NA

```

然后使用管道函数 `%>%` 将 `df` 传递给 `gather` 函数, 因为管道函数的存在, 所以无需引用 `df`, 而以””来代替, 指定指标列 `key`, 数值列为 `weight`, 保留序号 (保留列需要使用负号加列名的形式进行设置), 并移除默认值. 之后会得到一个中间产物数据框, 该数据框指标列中的”gender”和指标虽然以空格分割开, 但仍然在一列中, 不满足”干净”数据的原则. 所以再次使用管道函数将中间产物的数据框, 传递给函数 `separate`, 将 `key` 列拆分成两列, 分别为性别和 `key`:

```

1 > library(magrittr)
2 > library(tidyr)
3 > df1 <- df %>% gather(data=.,key,weight,-id,na.rm=T) %>% separate(data=.,key,
   into=c("gender","key"))
4 # A tibble: 12 x 4
5   id gender key     weight
6   <dbl> <chr>  <chr>    <dbl>
7   1     b      weight     70
8   2     g      weight     80
9   3     b      weight     85
10  4     2     g      weight     60
11  5     3     g      weight     55
12  6     4     g      weight     58
13  7     1     b      age      23
14  8     5     b      age      23
15  9     6     b      age      30
16  10    2     g      age      25
17  11    3     g      age      26
18  12    4     g      age      22

```

代码清单2-1 gather和separate函数基本使用示例

```

1 > df %>%
2   gather(data = ., key = key, value = value, ... = -序号, na.rm = T) %>%
  separate(data = ., key, into = c("性别", "key"))

```

小知识

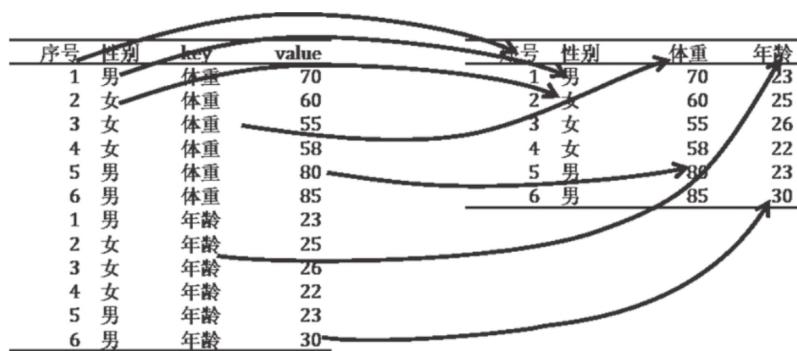
上述代码中的“%>%”为'magrittr'包中的forward-pipe operator，中文可以理解为管道函数。该函数能够与'tidyverse'内的所有函数完美结合使用，且易于理解记忆。有兴趣的读者可以尝试运行指令“? "%>%"”来查看具体的英文帮助。

表2-13 gather函数中参数中的及功能说明

参数名称	中文释义
data	数据框，接收 data.frame 和 tbl 格式
key, value	新的变量名，可以是字符串或者字符。key 参数可以为指标列设置新名称，而 value 参数则是为了设置数值所在的列名
...	需要转换的列，既可以是列名，也可以是列所在的阿拉伯数字位置，支持使用冒号来选择连续列，比如选择列 2 到 5，指定该参数为“2:5”即可实现选择。也可以使用减号来实现反向选择，比如若不想选择第一列，则指该参数为“-1”即可达到需求。需要注意的两点是：使用列名的时候是不需要双引号的；默认设置为选择所有列并进行转换
na.rm	对默认值的处理。默认设置为保留默认值，可以设置为真来移除默认值
convert	该参数可用于引用 utils 中的 type.convert 函数。默认设置为假，即不引用，若设置为真，则会引用 type.convert 函数，对变量名称进行属性转换，具体转换规则请参看函数帮助
factor_key	新指标列中的指标是否转换成因子，默认设置为假，即不设置成因子，若设置为真则转换为因子

6.3.2.2 spread—“长”数据变“宽”

函数 spread 是 gather 函数的逆向函数。



```

1 > df1 %>% spread(data=.,key,weight)
2 # A tibble: 6 x 4
3   id gender  age weight
4   <dbl> <chr>   <dbl>  <dbl>
5 1     1 b       23     70
6 2     2 g       25     60
7 3     3 g       26     55

```

8	4	4	g	22	58
9	5	5	b	23	80
10	6	6	b	30	85

表2-14 函数spread参数及功能说明对照表

参数名称	中文释义
data	数据框，接收 data.frame 和 tbl 格式
key, value	参数 key 是想要作为变量名称的列，表 2-5 中的 key 列中含有可以作为变量名称的“指标”。 参数 value 是与变量名相对应的数值。 两个参数可以是字符串或者数字位置

(续)	
参数名称	中文释义
fill	可以指定该参数来填补默认值，默认设置为以 NA 填补空白
convert	该参数可用于引用 utils 中的 type.convert 函数。默认设置为假，即不引用；若设置为真，则会引用 type.convert 函数，对变量名称进行属性转换，具体转换规则请参看函数帮助
drop	是否对转置后的数据中的保留因子水平，默认为真——仅保留有具体数值的因子水平，若为假，则保留全部因子水平，即使有一个或多个因子水平无真实数值
sep	是否对拆分后的列进行前缀重命名

6.3.3 separate/unite—拆分合并列

前面已经展示了函数 separate 的具体用法，该函数完全可以理解为是 excel 中的拆分列，该函数无法对一个单独的数值位置进行操作。

表2-15 函数separate参数及功能说明对照表

参数名称	中文释义
data	数据框，接收 data.frame 和 tbl 格式
col	需要拆分的列名或数字位置，无须双引号
into	想要分成的列名，必须是字符串向量，详见 2.3.4 节中代码演示
sep	分隔符。接收正则表达式，也可以利用数字位置进行拆分，正 1 代表从左边第一位置开始拆分，负 1 则为从右起第一位置开始，利用数字位置进行变量拆分时，该参数的长度应该比参数 into 少一位
remove	是否保留原列，默认为真，即拆分后移除原列
convert	该参数可用于引用 utils 中的 type.convert 函数。默认设置为假，即不引用；若设置为真，则会引用 type.convert 函数，对变量名称进行属性转换，具体转换规则请参看函数帮助
extra	当遇到拆分列中数据的长度不相等的情况，有以下 3 种处理方式。 1) 默认设置为丢掉多余的数值，并发出警告。 2) 设置为“drop”时，丢掉多余的数值但不发出警告——强烈不推荐使用。 3) 设置为“merge”时，仅拆分成参数 into 中指定的列数，但是会保留多出的数据
fill	同为处理拆分列中数据长度不等的情况，与 extra 处理的方式相反，fill 的 3 种处理方式如下。 1) 默认设置为发出警告，提示拆分列中数据长度不相等，并提示具体是哪一行数据不等，以 NA 来替补。 2) 设置为“right”时，表示从右侧开始填补 NA。 3) 设置为“left”时，表示从左侧开始填补 NA。
...	额外参数设定

6.3.4 replace_na/drop_na—默认值处理工具

一旦明确了默认值的替代方式，replace_na和drop_na两个函数就可以通过对指定列的查询来将NA替换成需要的数值，例如，去掉所有存在默认值的观察值。表

函数名称	功能简介	使用注意事项
replace_na	按列查询并替换默认值	只有两个参数需要设置，data参数为所需处理的数据框，replace参数可用来指定查询列及NA值的具体处理方式。参数replace只接收列表格式，所以后面必须使用list()函数将列名及替换值列表化，列表内是以等号分隔列名和替换值，左侧为列名，右侧为值，不同列之间以逗号相隔
drop_na	按列去除默认值，与baseR中na.omit函数功能类似	可以指定列名来去掉默认值NA，也可以不指定任何列名来去掉所有默认值。需要注意的一点是，当数据框中不同行不同列中都有默认值的情况时，使用该函数可能会造成数据框中数据过少而无法进行后续分析的情况

下面的代码列出了如何使用两个函数：

```

1 > df %>%
2   gather(key, value, -序号) %>%
3   separate(key, c("性别", "key")) %>%
4   replace_na(list(value = "missing"))
5   > df %>%
6   gather(key, value, -序号) %>%
7   separate(key, c("性别", "key")) %>%
  drop_na()

```

这里必须提醒一下读者关于默认值全部替换的情况，将所有默认值替换成0是很危险的行为，不推荐使用这种做法，因为0代表该数据是存在的，只是数值为0，而默认值则可能代表数据不存在和存在两种情况，只是因为某些原因而导致数据采集失败。因此对默认值的处理一定要视具体情况而定。

```

1 > df <- df %>%
2 +   gather(key, weight, -id) %>%
3 +   separate(key, c("gender", "key")) %>%
4 +   drop_na(weight)
5 > df
6 # A tibble: 12 x 4
7   id gender key     weight
8   <dbl> <chr>  <chr>   <dbl>
9     1     b     weight     70
10    2     b     weight     80
11    3     b     weight     85
12    4     g     weight     60
13    5     g     weight     55
14    6     g     weight     58
15    7     1     age       23
16    8     5     age       23
17    9     6     age       30

```

```

18 10      2 g      age      25
19 11      3 g      age      26
20 12      4 g      age      22

```

6.3.5 fill/complete—填坑神器

在处理日期或者计算累积值的时候，如果中间有一个默认数值，则意味着值不完整或累积值无法计算。fill函数可以自动填补默认的日期或者日期，类似于Excel中拖动鼠标来完成单元格数值的复制或序列填充功能。complete函数是将三个函数揉在一起，这三个函数分别为：expand、dplyr::left_join和replace_na。主要功能是将变量和因子的各种组合可能性全部罗列出来，并用指定的数值替代默认值部分。complete函数在日常练习中并不常用，所以这里不做过多介绍，感兴趣的读者可以参考帮助文档进行练习。

表2-17 函数fill参数及功能说明对照

参数名称	中文释义
data	数据框，接受 data.frame 和 tbl 格式
...	需要填补的列
.direction	两种选择，向上或者向下填补，只可以选择其中的一种，默认向下

6.3.6 separate_rows/nest/unest—行数据处理

6.3.6.1 separate_rows—拆分单元格

当遇到一个数据单位中出现多个数值的情况时，该函数就会很有用。将一个数据单位中的不同数值按照参数进行 sep 中给出的参数拆分，然后将拆分之后的结果顺序地放在同一列的不同行中，并自动增加行数。



参数名称	中文释义
data	数据框，接收 data.frame 和 tbl 格式
...	要拆分的列名，选择规则与包中的其他函数类似
sep	分隔符，默认为非字母形式的任何符号，所以使用默认设置可以处理绝大部分的情况，这也是 t 序号 yr 包的优势，对于还不是很熟悉各种参数功能的初学者来说，这些预先设置好的默认设置可以让上手变得容易，从而让用户在后续的大量实践中积累经验，慢慢了解各种复杂的参数设置
convert	同表 2-15

6.3.6.2 nest/unnest—压缩和解压缩行数据

nest 和 unnest 是两个互逆数据, 它们最重要的功能是将一个数据框按照用户自定义的规则, 将其压缩成一个新的数据框, 新的数据框中包含列表型数据.

```

1 > df
2 # A tibble: 6 x 4
3   id gender weight   age
4   <dbl> <chr>    <dbl> <dbl>
5   1   男        70    23
6   2   女        60    25
7   3   女        55    26
8   4   女        58    22
9   5   男        80    23
10  6   男        85    30
11 > df_tidy <- df
12 > df_tidy %>% nest(-gender)
13 # A tibble: 2 x 2
14   gender data
15   <chr>  <list>
16   1 男    <tibble [3 x 3]>
17   2 女    <tibble [3 x 3]>
18 Warning message:
19 All elements of '...' must be named.
20 Did you want 'data = c(id, weight, age)'?

```

单独使用nest函数没有任何实际价值, 但是当配合循环 (第4章) 和purrr包 (第5章) 中的map函数家族时, nest函数就会显示出强大的功能性。对dplyr包有一定了解的读者可以跳过第3章, 直接查看nest与其他具有循环功能的函数结合使用的例子。

表2-21和表2-22中列举了nest/unnest这一对函数的参数及功能说明。需要提醒读者一点的是, 如果需要使用unnest函数“解压缩”两列及以上时, 那么每一列中数据框的行数都必须相等, 否则无法成功“解压”。

表2-21 函数nest参数及功能说明对照

参数名称	中文释义
data	数据框, 接收 data.frame 和 tbl 格式
...	需要拆分的列名, 选择规则与包中的其他函数类似
.key	新的列名, 可以是字符, 或者是符号, 但不推荐使用符号。不需要双引号

表2-22 函数unnest参数中英文对照

参数名称	中文释义
data	数据框, 接收 data.frame 和 tbl 格式
...	需要拆分的列名, 选择规则与包中的其他函数类似
.drop	是否去掉格外的列? 默认设置为去掉
.id	增加一列识别码, 用于标识每一行数据来自的数据框, 设置该参数之后, 新列名即为传参值
.sep	对新列的名字进行操作, 如果指定了参数, 则以该值作为分隔符, 将原列和压缩的数据框名结合在一起, 形成新列名

6.4 lubridate 日期时间处理

6.4.1 为什么使用 lubridate

日期的格式各种各样, 在处理与时间有关的数值时, 解析日期和时间变量往往无可避免.

6.4.2 ymd/ymd_hms—年月日还是日月年

一般情况下, ymd及其子函数可以完整地解析以数字或字符串形式出现的日期形式, 只有当日期中对不同的成分以类似双引号作为分隔符的情况, 或者是对象为奇数的情况对 (详见代码演示), ymd等函数可能会无法直接进行解析, 而是需要进行额外处理。ymd函数即代表年月日, ymd_hms函数则代表年月日时分秒。两类函数的参数名称、结构和位置完全一致, 具体函数名称见表2-23。默认时区为世界标准时间UTC。表2-23中列出了两组函数所有的子函数。读者在解析时间时应当注意时区, 因为北京时间比UTC早8个小时, 所以是UTC+8。所有对象经过解析后都会输出为年月日 (时分秒) 的标准日期格式, 并且类别为 “Date” 。

一般情况下, ymd及其子函数可以完整地解析以数字或字符串形式出现的日期形式, 只有当日期中对不同的成分以类似双引号作为分隔符的情况, 或者是对象为奇数的情况对 (详见代码演示), ymd等函数可能会无法直接进行解析, 而是需要进行额外处理。ymd函数即代表年月日, ymd_hms函数则代表年月日时分秒。两类函数的参数名称、结构和位置完全一致, 具体函数名称见表2-23。默认时区为世界标准时间UTC。表2-23中列出了两组函数所有的子函数。读者在解析时间时应当注意时区, 因为北京时间比UTC早8个小时, 所以是UTC+8。所有对象经过解析后都会输出为年月日 (时分秒) 的标准日期格式, 并且类别为 “Date” 。

表2-23 lubridate包中日期格式解析主要函数一览

日期	日期时间
ymd	ymd_hms
ydm	ymd_hm
mdy	ymd_h
dmy	dmy_hms
dym	dmy_hm
yq	dmy_h
	mdy_hms
	mdy_hm
	mdy_h
	ydm_hms
	ydm_hm
	ydm_h

lubridate函数可以仅使用默认设置轻松解析偶数位的字符型向量，必须要注意的是偶数位必须大于6位，否则会产生NA。在下列的代码中“2018 1 2”因为其中存在空格，所以被默认解析为6位。同样的逻辑也适用于解析日期时间对象。参数tz用于设置时区，示例代码如下：

```

1 > library(lubridate)
2
3 载入程辑包: 'lubridate'
4
5 The following objects are masked from 'package:base':
6
7 date, intersect, setdiff, union
8
9 > ymd(c(20210306,"2022-03-06","2023 3 6"))
10 [1] "2021-03-06" "2022-03-06" "2023-03-06"
11
12 > dmy_h(c(1802201810,"20-10-2018 24")),tz="Asia/Shanghai")
13 [1] "2018-02-18 10:00:00 CST" "2018-10-21 00:00:00 CST"

```

小提示

如果函数没有自动解析正确的时区，那么读者可以使用Sys.timezone()或Olson-Na mes()来寻找正确的时区，并传参设置时区。

6.4.3 year/month/week/day/hour/minute/second—时间单位提取

气象领域通常会计算若干年的月、日平均降雨量或气温等指标，这时就会涉及月和日的提取要求。`lubridate`包中的函数，包含了提取从年到秒所有单位的功能。而为了方便记忆，这些函数的名称也都与相应的组件一一对应。需要读者注意的一点是，该组函数只能提取时间日期格式的对象，这些对象可以是常见的“Date”“POSIXct”“POSIXlt”“Period”等，或者是其他日期时间处理R包中的格式“chron”“yearmon”“yearqtr”“zoo”“zooreg”等。

```

1 > date <- ymd(c(20210306, "2022-03-06", "2023 3 6"))
2 > year(date)
3 [1] 2021 2022 2023
4 > month(date)
5 [1] 3 3 3
6 > week(date)
7 [1] 10 10 10
8 > day(date)
9 [1] 6 6 6
10 > hour(date)
11 [1] 0 0 0

```

6.4.4 guess_formats/parse_date_time—时间日期格式分析

当遇到使用英文月份简写的日期，比如24 Jan 2018/Jan 24，或者其他更糟糕的情况时，如果使用传统的baseR中的函数，诸如`strptime`或是`format`之类，那么用户可能会浪费很多时间去猜测和组装正确的日期时间格式，因为只有顺序和格式都正确的时候，baseR中提供的相应函数才可以正确解析日期时间，否则就会不停地返回NA值。幸运的是，`guess_formats`和`parse_date_time`两个函数的存在，完全颠覆了以往的解析模式，从而使得这一过程变得简单有趣。

使用这两个函数解析日期时间的大体思路具体如下。

- 1) 执行`guess_formats`函数以用于猜测需要解析对象的可能日期时间顺序及格式，用户必须指定可能存在的格式顺序。
- 2) 复制`guess_formats`函数的返回结果。
- 3) 执行`parse_date_time`，并将复制的内容以字符串向量的格式传参给函数。
- 4) 若遇到解析不成功或不彻底的情况，则需要手动组建日期时间格式（组件列表请参看表2-24），并加入到`guess_formats`中的`order`参数中。

下面的代码简要解释了`guess_formats`和`parse_date_time`两个函数配合使用解析日期时间的流程。首先生成一个名为`example_messyDate`的练习字符串向量，然后对该向量运行`guess_formats`，第二位参数`orders`中包含了可能存在的日期时间格式，并函数的返回结果中会报告匹配的顺序格式。将报告结果复制到`parse_date_time`的第二位参数中。至此解析成功。

```

1 > example_messydate <- c("24 Jan 2018", 1802201810)
2 > library(lubridate)
3
4 载入程辑包: 'lubridate'
5
6 The following objects are masked from 'package:base' :
7
8 date, intersect, setdiff, union
9
10 > guess_formats(example_messydate, c("mdY", "BdY", "Bdy", "bdY", "bdy", "dbY", "dmYH"))
11 d0bY      d0mYH      dmYH
12 "%d %0b %Y" "%d%0m%Y%H" "%d%m%Y%H"
13 > parse_date_time(example_messydate, orders = c("d0bY", "d0mYH", "dmYH"))
14 [1] "2018-01-24 00:00:00 UTC" "2018-02-18 10:00:00 UTC"

```

表2-24 日期时间组组成部分解释

缩写	中文释义
a	工作日缩写, 例如 Mon、Tus 等, 会根据系统语言调整显示
A	工作日全名
b or B	月份的全名或缩写, 同样会根据系统语言自动决定语言模式
d	月份中的日
H	24 小时制的小时
I	12 小时制的小时
j	天

缩写	中文释义
q	四分之一年
m	月份
M	60 分钟制中的分
p	英制上午或下午
S	秒
OS	微秒
U	美制, 单位周, 从每周日开始算一周, 一年按 53 周算
w	一周的七天, 取值范围为 0~6, 0 为周日
W	英制单位周, 同样为一年 53 周, 但周一为每周的第一天
y	两位数的年份, 取值范围为 0~99
Y	带世纪的年份, 即四位数年
Om	匹配数字月份和对应的英文字母顺序
Op	匹配英制 AM/PM
r*	匹配 12 小时和 24 小时制的顺序
R*	匹配 24 小时制中的时分和 12 小时制中的时分
T*	匹配 12 小时制中的时分秒和 24 小时制中的时分秒

6.5 stringr 字符处理工具

6.5.1 baseR vs stringr

baseR中已存在一些使用正则表达式处理字符串的函数，例如，以grep为母函数的一众函数，包括最常用的gsub，等等。熟悉Linux系统的读者可能会觉得grep看起来很眼熟，这是因为R语言与其他编程语言一样，都借鉴了各种计算机语言的精华部分。表2-25列出了baseR中与字符串有关的函数及其与stringr包中相应函数的对比及小结。该表的意义在于，可以通过学习stringr包中的主要函数来帮助了解baseR包中的对应函数。因为stringr虽然简单易上手，但是在实际处理应用数据时，其在速度上会比baseR又略逊一筹，读者可以通过stringr包中的函数来练习字符处理的能力，在实际工作中使用baseR中的函数来执行具体任务。

表2-25 baseR和stringr包中字符串处理函数对比

BaseR	stringr	小结
sub	str_replace	查询并替换第一个匹配的字符模式
gsub	str_replace_all	查询并替换所有匹配的字符模式
grep	str_detect	检测指定字符串中是否存在匹配的模式，是测试正则表达式的必备函数
regexp	str_locate	检测第一个匹配字符模式的位置并报告
gregexpr	str_locate_all	检测所有匹配字符模式的位置并报告
regmatches	str_extract/_all	检测匹配字符模式并提取成一个独立的返回值
	str_match/_all	同上

```

1 > library(stringr)
2
3 > example_txt <- "sub and gsub perform replacement of the first and all matches
   respectively."
4 > str_replace(string=example_txt, pattern = "a", replacement = "@")
5 [1] "sub @nd gsub perform replacement of the first and all matches respectively.
   "
6 > str_replace_all(string=example_txt, pattern = "a", replacement = "@")
7 [1] "sub @nd gsub perform repl@acement of the first @nd @ll m@tches respectively.
   "

```

6.5.2 正则表达式基础

Regular expression (正则表达式) 在目前主流的统计语言上都有应用。使用符号型字符串大规模查找和替换数据，不仅可以提高工作效率，同时还能保证规则的一致性。R中正则表达式的符号意义，请参看表2-26。表2-26中列出了最常见的正则表达式基础单位，读者可以将这些符号想象成儿时乐高积木的小构件，由简到繁地慢慢组合搭配这些构件，以实现不同的数据处理目标。简单构建正则表

表2-26 正则表达式符号及解释

符号	意 义
[:alnum:]	英文字母和数字, 字母包括大写和小写
[:alpha:]	英文字母, 不区分大小写
[:blank:]	空白, 可以是空格、tab 或者其他会形成空白的符号
[:cntrl:]	控制符
[:digit:]	阿拉伯数字, 0-9
[:graph:]	制表符, 包括标点符号和字母数字
[:lower:]	小写字母, 或根据 locale 调整字母查询所基于的语言
[:print:]	标点符号, 数字、字母和空格键
[:punct:]	各种标点符号: ! " # \$ % & ' () * + , - . / : ; < = > ? @ [\] ^ _ ' { } ~
[:space:]	空格键, 包括 tab、分页符、垂直的 tab、回车等
[:upper:]	大写字母
[:xdigit:]	十六进制数字
^	字符起始位置
\$	字符结束位置
	或, 用来构建条件选择
[], ^]	查询中括号内的匹配项目, 如果中括号中存在 “^”, 则查询不包括中括号中的匹配内容
()	分组查询, 括号内为一组, 可以将一个字符模式分为多组查询, 然后对其中一组或多组匹配字符进行处理
.	除换行符以外的任何字符、字母、数字
?	最多匹配一次
*	至少匹配 0 次
+	至少匹配一次
{n}	匹配 n 次
{n,}	至少匹配 n 次
{n,m}	匹配 n 到 m 次

6.5.3 简易正则表达式创建

数据集 df2 是笔者从网络上获取的一组英文期刊作者名和年份.

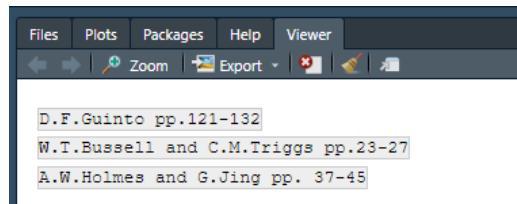
```

1 > library(tibble)
2 > df2 <- tribble(~year, ~authors,
3 +                 2016, "D.F.Guinto pp.121-132",
4 +                 2017, "W.T.Bussell and C.M.Triggs pp.23-27",
5 +                 2017, "A.W.Holmes and G.Jing pp. 37-45")
6
7 > df2
8 # A tibble: 3 x 2
9   year authors
10  <dbl> <chr>
11 1 2016 D.F.Guinto pp.121-132
12 2 2017 W.T.Bussell and C.M.Triggs pp.23-27
13 3 2017 A.W.Holmes and G.Jing pp. 37-45

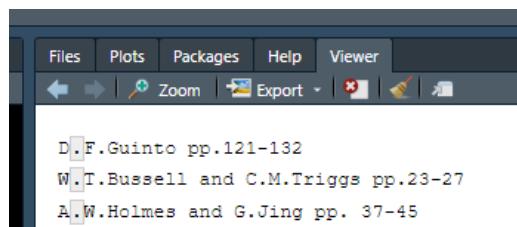
```

函数str_view/_all可以很直观地反应出数据内部匹配的项目。 “.+” 组合的意思是匹配除换行符 “\n” 以外的所有字符，字符至少出现一次，所以全部的字符都被匹配出来。若只希望匹配 “.”，则需要使用反斜杠来告知函数，这是因为独立存在的 “.” 会被解析为任何除换行符以外的字符、字母和数字（见表2-26）。所以第二行代码就的意思是匹配第一个出现的英文句号：

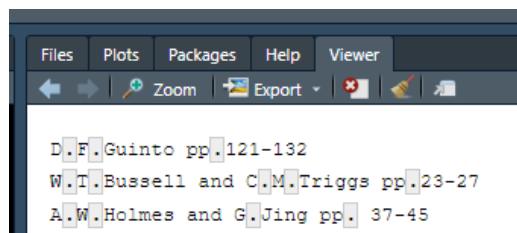
```
1 > library(stringr)
2 > str_view(df2$authors, pattern = ".+")
```



```
1 > str_view(df2$authors, pattern = "\\.")
```

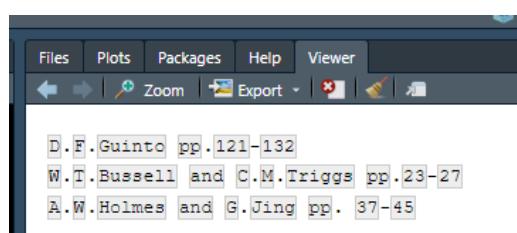


```
1 > str_view_all(df2$authors, pattern = "\\.")
```



匹配所有字母和数字：

```
1 > str_view_all(df2$authors, pattern = "[[:alnum:]]+")
```



当处理数据较多时，str_view/_all的速度可能会很慢，可以使用str_detect来检测所使用的表达式在数据中是否有匹配。该函数只返回逻辑判断，代码如下：

```

1 > str_detect(df2$authors, pattern = "\\".)
2 [1] TRUE TRUE TRUE

```

在df中，页码可以被归类为无用信息，所以需要清理掉。下面的代码使用了str_replace来将页码的部分完全替换掉。匹配模式为"pp\..\..+[: digit:]{2, 3}\-\[: digit:]{2, 3}"。分解这个正则表达式：“pp”匹配“pp”；“\\.”匹配“.”；“..+”代表“pp.”后面的任何字符串；“[: digit:]{2, 3}”代表2到3位数字；“\\-”匹配“-”；最后的“[: digit:]{2, 3}”表示数字出现2到3位。

```

1 > df2$authors <- str_replace(df2$authors, pattern = "pp\\..\..+[:digit:]{2,3}\\-[[:digit:]{2,3}]", replacement = "")
2 > df2
3 # A tibble: 3 x 2
4   year  authors
5   <dbl> <chr>
6     1 2016 "D.F.Guinto"
7     2 2017 "W.T.Bussell and C.M.Triggs pp.23-27"
8     3 2017 "A.W.Holmes and G.Jing"

```

6.5.4 文本挖掘浅析

文本（包括但不仅限于书刊）挖掘，或者更通俗地讲——自然语言处理（Natural Language Processing），是人工智能领域必不可少的一项技术。每一秒钟，世界范围内都有不计其数的新文本在以各种形式记录或保存起来。但这些以人类语言书写或录制下来的“数据”，并不像二进制的表格式数据那样容易被电脑接受并处理。如何分析人类历史中这些以文本形式保存的数据，就是文本挖掘需要解决的问

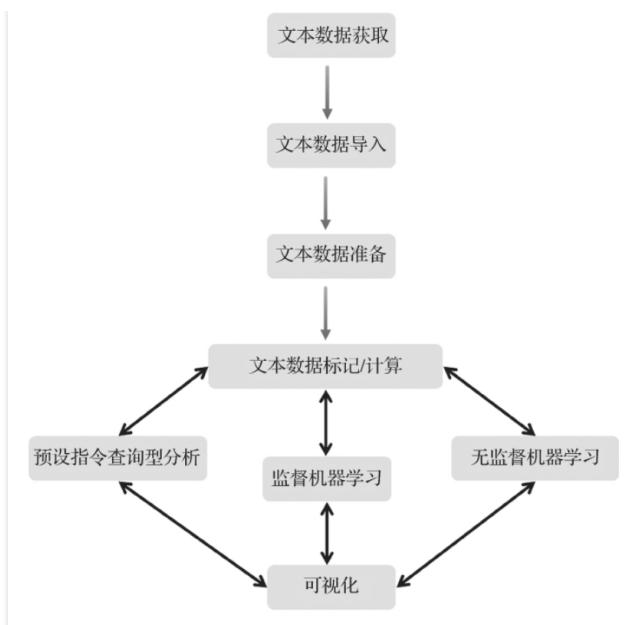


图2-4 文本挖掘的一般流程

