R 语言数据预处理笔记

缪啸宇

南航经管

更新:2020 年 12 月 25 日

1 数据导入工具

需要掌握：

- R 语言如何读取不同类型的数据文件
- 读取数据常用的 R 包
- 不同 R 包中相似函数的优缺点
- 数据读取函数的常用参数设置
- 处理规则和不规则原始数据文件

1.1 utils——一个自带的基础包

utils 不是专门用来进行数据导入的。utils 比较底层，是用来编程和开发 R 包的。导入数据用到这个包是为了学习时事半功倍。

1.1.1 read.csv/csv2——逗号分隔

.csv = comma-separated values，即每个数据值用逗号隔开，因此本质上是一种.txt。而 utils 包中的`read.csv/csv2`[1]正是用于快速读取 csv 文件。read.csv 的简单实用方式如下[2]：

[1]read.csv 是逗号分隔;read.csv2 是分号分隔。一般用 csv 多，因为 csv 里小数点是“.”，而 csv2 里小数点是“,”。

[2]file 参数可以省略，即 read.csv("flights.csv")

```
flights <- read.csv(file = "flights.csv")
```

要会使用.rproj，这个是 R 项目，将每一次数据分析过程都看作一个独立的项目。其作用是免去设置路径的麻烦，也减少了原始数据文件太多导致的各种隐患。

数据导入，第一步通常是调用 str 函数来进行初步检视：

```
str(object = flights)
```

str 函数用于检视数据的结构、变量名称和类型等。除了 str 外，还有其他检视数据集的函数如 head、tail、view 等。检视数据集的意义在于.csv 文件并不一定是逗号分隔的。例如 read.csv 读取 Tab 分隔的文件就会出现把所有数据读取到了一列中的情况。这在检视结果中可以看出。该问题可以用指定分隔符参数解决，将\t指定给 sep 参数后就可以用 read.csv 读取 Tab 分隔的 csv 文件，如下：

```
flights3 <- read.csv(file = "flights1.csv", sep = "\t")
str(flights3)
```

read.csv 默认将所有字符型数据都读成因子型（Factor），这会对后续处理带来麻烦，因此将字符因子化关掉，用stringsAsFactors参数:

```
flights_str <- read.csv("flightsstrings.csv", sep = "\t", stringsAsFactors = FALSE)
str(flights_str)
```

1.1.2 read.delim/delim2——tab 分隔

read.delim/delim2[3]是专门处理 tab 分隔的数据文件的。

其与 read.csv 的唯一不同只是参数sep = "\t"。因此和前面差不多，read.delim 会在原始数据中不断寻找 tab 分隔符，如果莫得的话就会和前面一样将所有变量都挤在一列中。实例代码如下，可以看出和 read.csv 的参数格式相同；还可以看出其本质是通过 read.table 执行的[4]。

```
1 read.delim
```

1.1.3 read.table——任意分隔符

read.table函数将文件读取成数据框的格式，示例：

```
1 flights <- read.table("flights.csv", header = TRUE)
2 head(x = flights)
```

上面 head 函数类似 str 函数，只不过 head 函数是从上到下显示，str 函数是从左到右显示。一般推荐两个都运行下，head 方便与原始数据比对，str 显示所保存的数据框属性、变量类型等信息。

- 如果不将 header 设置为 TRUE 的话，原数据里变量名称这一行就会变成观测值的第一行，变量名会被函数重新分配一个 V1。

[3]delim 读取小数点是“.”的数据；delim2 读取小数点是“,”的数据

[4]read.csv/csv2/delim/delim2 的母函数都是 read.table

- 如果不将分隔符 sep 参数设置为“,”的话，由于 read.table 的默认分隔符是空白[5]，会导致所有变量挤在一列中。
- 不要忘记加上stringsAsFactors = FALSE，让字符型变量不要变成因子型。
- 关于缺失值的处理感觉这个讲得不太好，需要另外查一下。

数据集里出现默认值（NA）或空白（“”）很常见。理论上，默认值仍是观测值，而空白可能不是数据。这个可以拓展的东西比较多，这边之后可以放一个超链接。

1.2 readr——进阶数据读取

readr的优势：

- 更快。
- 默认设置简洁。会自动解析每列数据类型并显示解析结果，无需设置 stringAsFactors。
- 对数据类型的解析更准确[6]。

readr 中常用的数据读取函数：

- `read_delim`[7]
- read_fwf

[5]空白 空格。那等于啥啊草，书里也没说。

[6]read.table 在甄别数据属性时，只会对起始 5 行的观测值评估；readr 是 1000 行。这在处理空白行时免去了很多麻烦。

[7]read_delim 是 read_csv/csv2/tsv 的母函数，因此可以直接调用子函数。其分别对应 utils 中的 read.csv/csv2/delim。

- read_lines
- read_log
- read_table

`read_delim`中比较重要的参数：

- file：路径 + 文件名[8]。
- delim：每行的分隔符，如“,”和“\t”。不过在三个子函数中就无需设置，只是母函数没有默认值而已。
- skip：跳过几行读取原始数据文件，默认为 0，即不跳过。
- col_names：TRUE 时原始数据文件第一行用作列名，且不在数据集内。其他的自查。
- col_types：列的数据类型，内容较多，自查。

read_csv 运行后会在 console 中显示解析后每一列的属性，可以快速甄别哪一列的属性和期望不符。

```
library(readr)
read_csv("flights.csv")
```

1.3 utils vs readr

- 用 utils 导入数据：莫得网络，装不了包，不想多加载一个包。

[8].gz/bz2/xz/zip 的会自动解压缩；http(s)://和 ftp(s)://会下载到本地

- 用 readr 导入数据：不介意多加载一个包，喜欢用默认设置。
- 用 data.table 包中的 fread 函数：文件个数（>10）和文件大小（>1MB）较大时用。

1.4 readxl——Excel 文件

读取 Excel 文件的必备包，只有五个函数：

- excel_format 和 excel_sheets：探测性函数
- readxl_example：引用例子
- cell-specification：读取特定单元格
- `read_excel`：最重要的，读取 Excel 文件

现在数据会被读取为 tibble 格式，可以理解为提升版的 data.frame。

1.5 DBI——数据库查询下载

DBI 包能够方便地与数据库进行交互，需要有以下的前提：

- 已知数据库类型（如 MySQL、MongoDB 等）
- 安装了相应数据库类型的 R 包
- 数据库服务器地址
- 数据库名称

- 接入数据库的权限、账号、密码
- 安装 dplyr 包用来本地化数据库中的数据

R 与数据库交互的流程为：建立连接 发送查询请求 获取相关数据。需要三个必备的包[9]：

```
1 library(DBI)
2 library(dplyr)
3 library(RPostgreSQL)
```

- dbConnect：连接服务器，需要在单引号中输入服务器地址、名称、权限等信息。
- dbListTables：查询数据库中的详细内容，用字符串向量的格式返回。无内容则返回空值。

这块的内容这里只是简单介绍，数据库交互详细可以查阅其他资料。

1.6 pdftools——PDF 文件

一般很少遇到 PDF 文件，这些大多是学术期刊、杂志、电子书籍等。不过在文本挖掘（Text Mining）和主题模型（Topic Modelling）预测中，pdftools 包必备。该包只有两个母函数：从 PDF 中提取数据（pdf_info）；将文件渲染成 PDF 格式。

pdf_info母函数下一共有 6 个子函数：

- pdf_info：读取 PDF 文件的基本信息。
- pdf_text：提取所有文字和非文字信息，包括分页符、换行符等。

[9]DBI 和 RPostgreSQL 建立与数据库的连接和发送请求；dplyr 将数据保存到本地

- pdf_data：提取数字型数据（实际使用起来效果有差异）。
- pdf_fonts：提取字体信息。
- pdf_attachments：提取文档附件。
- pdf_toc：提取文档目录。

这几个函数所用参数完全相同，只有三个：

- 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。

用 pdf_text 提取文档时，全部内容被提取为一个字符串向量，每页内容单独放置于一个字符串中。

1.7 jsonlite——JSON 文件

JavaScript object Notation（JSON）通常作为不同语言之间互相交流信息的文件。jsonlite 包能够将 JSON 格式文件完整解析读取到 R 语言中，也可以将 R 对象（object）输出为 JSON 格式。

读取 JSON 文件使用的是fromJSON函数。

1.8 本章小结

不同格式数据文件读取所需要的 R 包：

- readr/utils：平面文件格式，如.csv/txt
- readxl：Excel 格式，如.xlsx/xls
- DBI：数据库格式，如.db
- pdftools：PDF 格式
- jsonlite：JSON 格式

2 数据清理工具

原始数据基本都是脏的，无论人工还是传感器采集。原因很多，普遍例如：

- 不同采样人员的记录数据方式不同
- 录入数据时的失误
- 传感器断电导致大段数据默认
- 不同国家地区对时间日期制式的标准不同

在整个数据分析过程中，数据清洗可能占到 80% 的时间。耗时是因为数据清洗并非一次性工作，数据清洗 计算 可视化是一个动态循环。根据分析需求不同，清理的思路和方式也不同。例如对默认值可能采取完全移除、部分移除或替换等方式。

本章需要掌握：

- 数据的“脏”和“干净”的标准是什么？
- 数据清理的指导原则
- 数据清理的相关包（tibble、tidyr、lubridate、stringr等）

2.1 基本概念

只要不满足当前分析要求的数据集都可以说是“脏”的。而对“干净”数据可以总结为：

- 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

在 R 环境中“长”[10]数据比“宽[11]”数据计算速度更快[12]，两者转换只需要gather函数（见 2.3 节）

另外不同来源的数据应该单独成表独立存在。例如元数据[13]（此处有介绍）不应当与原始数据本身同时存在在一个数据集中。只有需要解释原始数据的时候才调用元数据。

2.2 tibble 包——数据集准备

2.2.1 为什么要用 tibble？

tibble 包是用 R 中一个数据存储格式命名的。read.csv 会将数据读取在一个 data.frame（数据框）当中；但 read_csv 有三种数据存储格式：tbl_df、tbl 和 data.frame。那么为什么要用 tbl 格式？原因如下：

- 稳定性好，可以完整保存变量名称和属性。
- 更多的信息展示、警示提醒，有利于及时发现错误。
- 新输出方式，在浏览数据时屏幕利用效率更佳。

[10]同类型变量单独成列
[11]多个同类或者不同类的变量并存
[12]两者区别文字说不清，网上找一下相关列表一眼就能看懂
[13]meta data，解释变量名称或数据背景的数据。通常包含坐标、指标的具体含义等解释性信息

当然 tbl 格式也有缺陷，就是 data.frame 格式用了很久，一些早期函数调用新的 tbl 格式可能不兼容。但是因为 data.frame 在处理变量名称的时候，有时会悄悄改动名称来满足自己的需求，导致会出现难以察觉的意料之外的错误，非常要命。

另外查看 data.frame 中变量类型往往需要 str 函数，但 tbl 无须调用函数，直接输入数据集名称即可。tbl 会根据 console 窗口大小自动调整展示内容；而 data.frame 直接在 console 中打出来，会将小于 1000 行 1000 列的所有内容展示出来，且还不会展示变量属性等内容。孰优孰劣显而易见。

2.2.2 创建 tbl 格式

用tibble或tribble函数创建新数据框，创建方法与 data.frame 函数一致。代码如下[14]：

```
1 library(tibble)
2 tibble(a = 1:6, b = a*2)
```

而 tribble 函数适合创建小型数据集（如元数据）。手动输入数据，变量名称用“~”开头，“,”结束，数据值用逗号分隔。元数据有时比较杂乱，用软件清理费时费力，但肉眼能直接看出关键信息时，直接用 tribble 函数手动生成会更高效。具体例子可以自查。

2.2.3 as_tibble——转换已有格式的数据集

- is_tibble：测试目标对象是否已经是 tbl 格式。

[14]int 代表 integer 整数，dbl 代表 double 浮点型数据类型

- as_tibble：将 vector、matrix、list[15]和 data.frame 转换成 tbl。
- enframe：也是 tibble 中的格式转换函数，优势在于可对向量数据进行编号。

2.2.4 add_row/column

add_row/column函数类似 Excel 中任意插入或删除一行/列数据。

baseR[16]也可以新增行或列：

```
f <- tibble(i = 1:3, j = c("John","Sam","Joy"))    # 创建一个tbl。
f$k <- 3:1                                         # 用$来新增一列名为k的变量，数值为3、2、1。
f[nrow(f)+1, ] <- c(4,"Jon",0)                     # nrow是计算行数，左边即为增加一行的意思。
```

上面;f[nrow(f)+1,] 中，中括号紧跟在数据框后面，可以作为索引来选择数据框中的特定数值。逗号前后分别为行、列索引[17]。因此与之相似的，f[,ncol(f)+1] 就是新增一列的意思。

这 tibble 包中两个小函数实用之处在于，可以随时随地任意新增行或列数据到指定位置，不用像上面 baseR 中只能在数据尾部或已有变量后面新增行或列。tibble 包中add_row和add_column使用例：

```
add_row(f, i = 4, j = "Jon")                       # 因为只指定了两个变量的值，所以未指定部分会自动填
    入默认值NA。
add_row(f, i = 4, j = "Jon", .before = 3)          # 在第三行之前插入一行新数据。
add_row(f, i = 4, j = "Jon", .after = 1)           # 在第一行之后插入新数据。
```

[15]列表格式转换的时候要注意列表中要素的长度，这个具体见书 P.36

[16]baseR 指的应该是 R 语言自带的基础功能，不加任何包

[17][行, 列]

```
4 add_column(f, l = nrow(f):1, .after = 1)          # 在第一列之后插入新变量，注意这里l的意思是从f的行
    数到1，即4到1。想了想确实应该这么写。
```

2.3 tidyr——数据清道夫

2.3.1 为什么用 tidyr?

优势：

- 易上手：函数名称简洁直观，可读性强
- 易使用：默认设置可以满足大部分需求
- 易记忆：不同函数的参数结构清晰
- 不易出现未知错误：处理数据过程中完整保留了变量属性和数据格式

2.3.2 gather/spread——长、宽数据转换

什么是长数据？全部变量名为一列，相关数值为一列。gather函数就是将宽数据转换成长数据。具体流程是用管道函数“%>%[18]”（此处有介绍）将脏数据框 df 传递给 gather 函数。管道函数可以使得无须重新引用 df，直接用“.”代替。指定指标列为 key，数值列为 value，保留列序号[19]，并移除默认值。这会得到一个中间产物数据框，但仍然不干净，因为性别和 key 在同一列中。要用管道函数再次拆分，将 key

[18] forward-pipe operator，需要 magrittr 包，可以通过“?'%>%'”来查看具体用法。

[19] 负号 + 列名进行设置

拆分为两列（性别 +key），传递给函数 separate。具体如下：

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

因为 tidy 系列函数结构简洁清晰，所以熟悉之后可以省略参数名，记住位置即可：

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

如果要反过来，将长数据变换成宽数据，就用spread函数。具体操作类似 gather，略。

2.3.3 separate/unite——拆分合并列

separate函数类似 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。unite是 separate 的逆向函数。

2.3.4 replace_na/drop_na——默认值处理工具

replace_na和drop_na可以通过对指定列的查询来将 NA 替换成需要的数值。前者是替换，后者是去除默认值。使用例：

```
df %>%
gather(key, value, -序号) %>%
separate(key, c("性别", "key")) %>%
```

```
4 replace_na(list(value = "missing"))
```

```
1 df %>%
2 gather(key, value, -序号) %>%
3 separate(key, c("性别", "key")) %>%
4 drop_na()
```

提醒：将所有默认值 NA 替换为 0 是很危险的，不推荐这么做。因为 0 代表数据存在而数值为 0，但默认值 NA 代表数据存在或不存在两种情况，只是因为某些原因导致数据采集失败。

2.3.5 fill/complete——填充工具

处理日期或者计算累计值的时候，如果中间有默认值则意味着不完整或没法计算。fill函数就类似 Excel 的填充功能，自动填补默认的日期或等值。而 complete 函数不常用就不多介绍，主要是将变量和因子的各种组合全部罗列出来。

2.3.6 separate_rows/nest/unnest——行数据处理

- separate_rows：类似 Excel 的拆分单元格。一个数据单位中出现多个数值，就用该函数参照参数 sep 给出的参进行拆分，然后顺序放入同一列的不同行中，会自动增加行数。
- nest/unnest：压缩和解压缩行数据。具体来说是将一个数据框压缩成新的数据框，其中包含列表型数据[20]。

[20]这个找具体例子看一下就懂了，一目了然。

单独使用 nest 函数没有什么意义，但是配合循环或者 purrr 包的 map 函数家族时，功能性很强大。这个具体到后面看。另外注意 unnest 解压缩两列以上时，每一列中数据框的行数必须相等，否则无法成功解压。

2.4 lubridate 日期时间处理

2.4.1 为什么用 lubridate？

传感器记录的数据为了避免闰年导致奇怪的错误，一般都是纯数字的日期格式，其问题是可读性较差。对日期的处理并不是如看似那么简单，而是和处理默认值相当的一大挑战。lubridate包的优势在于：

- 自然语言书写编程语法，易于理解和记忆。
- 融合了其他 R 包的时间处理函数，优化了默认配置。
- 能轻松完成时间日期数据的计算任务。

2.4.2 ymd/ymd_hms——年月日还是日月年？

ymd 及其子函数可以完整解析数字或字符串形式的日期格式，但也有例外[21]。ymd_hms代表年月日时分秒。lubridate 函数可以用默认设置解析偶数位的字符型向量，但偶数位必须大于 6 位，否则会产生

[21]例如日期中对不同成分以类似双引号作为分隔符，或是对象为奇数等。

NA[22]。

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

这些函数只能提取时间日期格式的对象，如常见的 Date、POSIXct、POSIXlt、Period、chron、yearmon、yearqtr、zoo、zooreg 等。

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

baseR 不适合解析英文月份简写的日期（如 24　Jan　2018 这样的），但guess_formats和parse_date_time能够方便解析，思路如下：

- 用 guess_formats 函数猜测解析对象的可能日期顺序及格式，需要先自己指定。
- 复制 guess_formats 函数的返回结果。
- 执行 parse_date_time，将复制的内容以字符串向量格式传参给函数。
- 如果解析不彻底，需要手动组建日期时间格式加入到 guess_formats 中的 order 参数。

使用例：

```
example_messyDate <- c("24 Jan 2018", 1802201810)
guess_formats(example_messyDatem c("mdY", "BdY", "Bdy", "bdY", "bdy", "dbY", "dmYH"))
parse_date_time(example_messyDate, orders = c("dObY", "dOmYH", "dmYH"))
```

[22]如果函数没有解析正确的时区，用 Sys.timezone() 或 OlsonNames() 来寻找并传参正确的时区。

2.5 stringr 字符处理函数

stringr 的优点在于参数很少超过三个，参数在结构和名称上一致，逻辑清晰。简单的字符处理能极大地提高数据清理的效率。使用 stingr 包能：

- 快速上手正则表达式——快速处理数据
- 理解表达式的基本概念——为以后的复杂任务打基础

2.5.1 baseR vs stringr

baseR 中存在一些使用正则表达式处理字符串的函数，如 grep 母函数类（包括了最常用的 gsub，这些在 Linux 中也常见）。stringr 简单上手但是实际处理数据时比 baseR 慢。因此可以通过 P.52 的对照表，用 stringr 包中的函数练习字符处理能力，实际工作中用 baseR 中的函数来执行。使用例：

```
1 library(stringr)
2 example_txt <- "sub and gsub perform replacement of the first and all matches respectively."  # 创建练习用的字符串向量
3 str_repalce(string = example_txt, pattern = "a", replacement = "@") # 参数pattern是查询第一次出现的位置，参数replacement设置为替换
4 str_replace_all(string = example_txt, pattern = "a", replacement = "@") # 与上一行类似，只不过会将所有符合要求的部分全部替换掉
```

baseR 中的 sub 和 gsub 函数逻辑与 str_replace 和 str_replace_all 相同。

2.5.2 正则表达式基础