数据可视化与 R 语言

Data Visualization with R

黄湘云

2019-07-26 19:44:43 CST

To my parents

目录

前言

关于本书

这里写每章的主要内容介绍

本书结构

关于作者

热心开源事业，统计之都副主编，经常混迹于统计之都论坛、Github 和爆栈网。个人主页 https://www.xiangyunhuang.com.cn/

第一章　数据搬运工

导入数据与导出数据，各种数据格式，数据库

1.1 导入数据

Base R 针对不同的数据格式文件，提供了大量的数据导入和导出函数，不愧是专注数据分析 20 余年的优秀统计软件。除了函数 `write.ftable` 和 `read.ftable` 来自 stats 包，都来自 base 和 utils 包

```
# 当前环境的搜索路径
searchpaths()
#> [1] ".GlobalEnv"                    "/usr/lib/R/library/stats"
#> [3] "/usr/lib/R/library/graphics"   "/usr/lib/R/library/grDevices"
#> [5] "/usr/lib/R/library/utils"      "/usr/lib/R/library/datasets"
#> [7] "/usr/lib/R/library/methods"    "Autoloads"
#> [9] "/usr/lib/R/library/base"
# 返回匹配结果及其所在路径的编号
apropos("^(read|write)", where = TRUE, mode = "function")
#>                   5                   5                   9
#>         "read.csv"         "read.csv2"         "read.dcf"
#>                   5                   5                   5
#>       "read.delim"      "read.delim2"         "read.DIF"
#>                   5                   2                   5
#>     "read.fortran"      "read.ftable"         "read.fwf"
#>                   5                   5                   9
#>      "read.socket"       "read.table"          "readBin"
#>                   9                   5                   9
#>         "readChar" "readCitationFile"         "readline"
#>                   9                   9                   9
```

```
#>        "readLines"           "readRDS"     "readRenviron"
#>                  9                   5                   5
#>            "write"         "write.csv"        "write.csv2"
#>                  9                   2                   5
#>        "write.dcf"      "write.ftable"      "write.socket"
#>                  5                   9                   9
#>      "write.table"          "writeBin"         "writeChar"
#>                  9
#>       "writeLines"
```

1.1.1 scan

```
scan(file = "", what = double(), nmax = -1, n = -1, sep = "",
     quote = if(identical(sep, "\n")) "" else "'\"", dec = ".",
     skip = 0, nlines = 0, na.strings = "NA",
     flush = FALSE, fill = FALSE, strip.white = FALSE,
     quiet = FALSE, blank.lines.skip = TRUE, multi.line = TRUE,
     comment.char = "", allowEscapes = FALSE,
     fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)
```

首先让我们用 cat 函数创建一个练习数据集 ex.data

```
cat("TITLE extra line", "2 3 5 7", "11 13 17")
#> TITLE extra line 2 3 5 7 11 13 17
cat("TITLE extra line", "2 3 5 7", "11 13 17", file = "data/ex.data", sep = "\n")
```

以此练习数据集，介绍 scan 函数最常用的参数

```
scan("data/ex.data")
#> Error in scan("data/ex.data"): scan() expected 'a real', got 'TITLE'
```

从上面的报错信息，我们发现 scan 函数只能读取同一类型的数据，如布尔型 logical，整型 integer，数值型 numeric(double)，复数型 complex，字符型 character，raw 和列表 list。所以我们设置参数 skip = 1 把第一行跳过，就成功读取了数据

```
scan("data/ex.data", skip = 1)
#> [1]  2  3  5  7 11 13 17
```

如果设置参数 quiet = TRUE 就不会报告读取的数据量

```
scan("data/ex.data", skip = 1, quiet = TRUE)
#> [1]  2  3  5  7 11 13 17
```

参数 nlines = 1 表示只读取一行数据

```
scan("data/ex.data", skip = 1, nlines = 1) # only 1 line after the skipped one
#> [1] 2 3 5 7
```

默认参数 flush = TRUE 表示读取最后一个请求的字段后，刷新到行尾，下面对比一下读取的结果

```
scan("data/ex.data", what = list("", "", "")) # flush is F -> read "7"
#> Warning in scan("data/ex.data", what = list("", "", "")): number of items
#> read is not a multiple of the number of columns
#> [[1]]
#> [1] "TITLE" "2"     "7"     "17"
#>
#> [[2]]
#> [1] "extra" "3"     "11"    ""
#>
#> [[3]]
#> [1] "line" "5"    "13"   ""
scan("data/ex.data", what = list("", "", ""), flush = TRUE)
#> [[1]]
#> [1] "TITLE" "2"     "11"
#>
#> [[2]]
#> [1] "extra" "3"     "13"
#>
#> [[3]]
#> [1] "line" "5"    "17"
```

临时文件 ex.data 用完了，我们调用 unlink 函数将其删除，以免留下垃圾文件

```
unlink("data/ex.data") # tidy up
```

1.1.2 read.table

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
           dec = ".", numerals = c("allow.loss", "warn.loss", "no.loss"),
```

```
           row.names, col.names, as.is = !stringsAsFactors,
           na.strings = "NA", colClasses = NA, nrows = -1,
           skip = 0, check.names = TRUE, fill = !blank.lines.skip,
           strip.white = FALSE, blank.lines.skip = TRUE,
           comment.char = "#",
           allowEscapes = FALSE, flush = FALSE,
           stringsAsFactors = default.stringsAsFactors(),
           fileEncoding = "", encoding = "unknown", text, skipNul = FALSE)

read.csv(file, header = TRUE, sep = ",", quote = "\"",
         dec = ".", fill = TRUE, comment.char = "", ...)

read.csv2(file, header = TRUE, sep = ";", quote = "\"",
          dec = ",", fill = TRUE, comment.char = "", ...)

read.delim(file, header = TRUE, sep = "\t", quote = "\"",
           dec = ".", fill = TRUE, comment.char = "", ...)

read.delim2(file, header = TRUE, sep = "\t", quote = "\"",
            dec = ",", fill = TRUE, comment.char = "", ...)
```

变量名是不允许以下划线开头的，同样在数据框里，列名也不推荐使用下划线开头。默认情况下，read.table 都会通过参数 check.names 检查列名的有效性，该参数实际调用了函数 make.names 去检查。如果想尽量保持数据集原来的样子可以设置参数 check.names = FALSE，stringsAsFactors = FALSE。默认情形下，read.table 还会将字符串转化为因子变量，这是 R 的历史原因，作为一门统计学家的必备语言，在统计模型中，字符常用来描述类别，而类别变量在 R 环境中常用因子类型来表示，而且大量内置的统计模型也是将它们视为因子变量，如 lm、glm 等

```
dat1 = read.table(header = TRUE, check.names = TRUE, text = "
_a _b _c
1 2 a1
3 4 a2
")
dat1
#>   X_a X_b X_c
#> 1   1   2  a1
#> 2   3   4  a2
dat2 = read.table(header = TRUE, check.names = FALSE, text = "
```

```
_a _b _c
1 2 a1
3 4 a2
")
dat2
#>   _a _b _c
#> 1  1  2 a1
#> 2  3  4 a2
dat3 = read.table(header = TRUE, check.names = FALSE, stringsAsFactors = FALSE, text = "
_a _b _c
1 2 a1
3 4 a2
")
dat3
#>   _a _b _c
#> 1  1  2 a1
#> 2  3  4 a2
```

1.1.3 readLines

```
readLines(con = stdin(), n = -1L, ok = TRUE, warn = TRUE,
          encoding = "unknown", skipNul = FALSE)
```

让我们折腾一波，读进来又写出去，只有 R 3.5.3 以上才能保持原样的正确输入输出，因为这里有一个之前版本包含的 BUG

```
writeLines(readLines(system.file("DESCRIPTION", package = "splines")), "data/DESCRIPTION")
# 比较一下
identical(
  readLines(system.file("DESCRIPTION", package = "splines")),
  readLines("data/DESCRIPTION")
)
#> [1] TRUE
```

这次我们创建一个真的临时文件，因为重新启动 R 这个文件和文件夹就没有了，回收掉了

```
fil <- tempfile(fileext = ".data")
cat("TITLE extra line", "2 3 5 7", "", "11 13 17", file = fil,
    sep = "\n")
```

```
fil
#> [1] "/tmp/RtmpulzaSd/filedb561e15da.data"
```

设置参数 n = -1 表示将文件 fil 的内容从头读到尾

```
readLines(fil, n = -1)
#> [1] "TITLE extra line" "2 3 5 7"          ""
#> [4] "11 13 17"
```

作为拥有良好习惯的 R 用户，这种垃圾文件最好用后即焚

```
unlink(fil) # tidy up
```

再举个例子，我们创建一个新的临时文件 fil，文件内容只有

```
cat("123\nabc")
#> 123
#> abc
```

```
fil <- tempfile("test")
cat("123\nabc\n", file = fil, append = TRUE)
fil
#> [1] "/tmp/RtmpulzaSd/testdb2e010d94"
readLines(fil)
#> [1] "123" "abc"
```

这次读取文件的过程给出了警告，原因是 fil 没有以空行结尾，warn = TRUE 表示这种情况要给出警告，如果设置参数 warn = FALSE 就没有警告。我们还是建议大家尽量遵循规范。

再举一个例子，从一个连接读取数据，建立连接的方式有很多，参见 ?file，下面设置参数 blocking

```
con <- file(fil, "r", blocking = FALSE)
readLines(con)
#> [1] "123" "abc"

cat(" def\n", file = fil, append = TRUE)
readLines(con)
#> [1] " def"
# 关闭连接
close(con)
# 清理垃圾文件
unlink(fil)
```

1.1.4 readRDS

序列化数据操作，Mark Klik 开发的 fst 和 Travers Ching 开发的 qs，Hadley Wickham 开发的 feather 包实现跨语言环境快速的读写数据

表 1.1: fst 序列化数据框对象性能比较 BaseR、data.table 和 feather[2]

Method	Format	Time (ms)	Size (MB)	Speed (MB/s)	N
readRDS	bin	1577	1000	633	112
saveRDS	bin	2042	1000	489	112
fread	csv	2925	1038	410	232
fwrite	csv	2790	1038	358	241
read_feather	bin	3950	813	253	112
write_feather	bin	1820	813	549	112
read_fst	**bin**	**457**	**303**	**2184**	**282**
write_fst	**bin**	**314**	**303**	**3180**	**291**

目前比较好的是 qs 和 fst 包

1.2 其它数据格式

来自其它格式的数据形式，如 JSON、XML、YAML 需要转化清理成 R 中数据框的形式 data.frame

1. Data Rectangling with jq
2. Mongolite User Manual introduction to using MongoDB with the mongolite client in R

jsonlite 读取 `*.json` 格式的文件，`jsonlite::write_json` 函数将 R 对象保存为 JSON 文件，`jsonlite::fromJSON` 将 json 字符串或文件转化为 R 对象，`jsonlite::toJSON` 函数正好与之相反

```
library(jsonlite)
# 从 json 格式的文件导入
# jsonlite::read_json(path = "path/to/filename.json")
# A JSON array of primitives
json <- '["Mario", "Peach", null, "Bowser"]'

# 简化为原子向量 atomic vector
fromJSON(json)
```

```
#> [1] "Mario"  "Peach"  NA       "Bowser"

# 默认返回一个列表
fromJSON(json, simplifyVector = FALSE)
#> [[1]]
#> [1] "Mario"
#>
#> [[2]]
#> [1] "Peach"
#>
#> [[3]]
#> NULL
#>
#> [[4]]
#> [1] "Bowser"
```

yaml 包读取 `*.yml` 格式文件，返回一个列表，`yaml::write_yaml` 函数将 R 对象写入 yaml 格式

```
library(yaml)
yaml::read_yaml(file = '_bookdown.yml')
#> $delete_merged_file
#> [1] TRUE
#>
#> $language
#> $language$label
#> $language$label$fig
#> [1] " 图 "
#>
#> $language$label$tab
#> [1] " 表 "
#>
#>
#> $language$ui
#> $language$ui$edit
#> [1] " 编辑"
#>
#> $language$ui$chapter_name
#> [1] " 第 " " 章"
#>
```

```
#>
#>
#> $output_dir
#> [1] "_book"
#>
#> $new_session
#> [1] TRUE
#>
#> $before_chapter_script
#> [1] "_common.R"
#>
#> $rmd_files
#> [1] "index.Rmd"            "preface.Rmd"          "dm-import-export.Rmd"
#> [4] "dm-base-r.Rmd"        "dm-dplyr.Rmd"         "99-references.Rmd"
```

表 1.2: 导入来自其它数据分析软件产生的数据集

统计软件	R 函数	R 包
ERSI ArcGIS	`read.shapefile`	shapefiles
Matlab	`readMat`	R.matlab
minitab	`read.mtp`	foreign
SAS (permanent data)	`read.ssd`	foreign
SAS (XPORT format)	`read.xport`	foreign
SPSS	`read.spss`	foreign
Stata	`read.dta`	foreign
Systat	`read.systat`	foreign
Octave	`read.octave`	foreign

表 1.3: 导入来自其它格式的数据集

文件格式	R 函数	R 包
列联表数据	`read.ftable`	stats
二进制数据	`readBin`	base
字符串数据	`readChar`	base
剪贴板数据	`readClipboard`	utils

`read.dcf` 函数读取 Debian 控制格式文件，这种类型的文件以人眼可读的形式在存储数据，如 R 包的 DESCRIPTION 文件或者包含所有 CRAN 上 R 包描述的文件 https://cran.r-project.org/src/contrib/PACKAGES

```
x <- read.dcf(file = system.file("DESCRIPTION", package = "splines"),
              fields = c("Package", "Version", "Title"))
x
#>      Package   Version Title
#> [1,] "splines" "3.6.1" "Regression Spline Functions and Classes"
```

最后要提及拥有瑞士军刀之称的 rio 包，它集合了当前 R 可以读取的所有统计分析软件导出的数据。

1.3　导入大数据集

在不使用数据库的情况下，从命令行导入大数据集，如几百 M 或几个 G 的 csv 文件。利用 data.table 包的 `fread` 去读取

https://stackoverflow.com/questions/1727772/

1.4　从数据库导入

Hands-On Programming with R 数据读写章节[3] 以及 R, Databases and Docker

将大量的 txt 文本存进 MySQL 数据库中，通过操作数据库来聚合文本，极大降低内存消耗[4]，而 ODBC 与 DBI 包是其它数据库接口的基础，knitr 提供了一个支持 SQL 代码的引擎，它便是基于 DBI，因此可以在 R Markdown 文档中直接使用 SQL 代码块[5]。这里制作一个归纳表格，左边数据库右边对应其 R 接口，两边都包含链接，如表 1.4 所示

1.4.1　PostgreSQL

odbc 可以支持很多数据库，下面以连接 PostgreSQL 数据库为例介绍其过程

首先在某台机器上，拉取 PostgreSQL 的 Docker 镜像

[3] https://rstudio-education.github.io/hopr/dataio.html

[4] https://brucezhaor.github.io/blog/2016/08/04/batch-process-txt-to-mysql

[5] https://bookdown.org/yihui/rmarkdown/language-engines.html#sql [rstudio-spark]: https://spark.rstudio.com/ [rmarkdown-teaching-demo]: https://stackoverflow.com/questions/35459166

表 1.4: 数据库接口

数据库	官网	R 接口	开发仓
MySQL	https://www.mysql.com/	RMySQL	https://github.com/r-dbi/RMySQL
SQLite	https://www.sqlite.org	RSQLite	https://github.com/r-dbi/RSQLite
PostgreSQL	https://www.postgresql.org/	RPostgres	https://github.com/r-dbi/RPostgres
MariaDB	https://mariadb.org/	RMariaDB	https://github.com/r-dbi/RMariaDB

```
docker pull postgres
```

在 Docker 上运行 PostgreSQL，主机端口号 8181 映射给数据库 PostgreSQL 的默认端口号 5432 (或其它你的 DBA 分配给你的端口)

```
docker run --name psql -d -p 8181:5432 -e ROOT=TRUE \
   -e USER=xiangyun -e PASSWORD=cloud postgres
```

在主机 Ubuntu 上配置

```
sudo apt-get install unixodbc unixodbc-dev odbc-postgresql
```

端口 5432 是分配给 PostgreSQL 的默认端口，`host` 可以是云端的地址，如你的亚马逊账户下的 PostgreSQL 数据库地址 `<ec2-54-83-201-96.compute-1.amazonaws.com>`，也可以是本地局域网 IP 地址，如 `<192.168.1.200>`。通过参数 `dbname` 连接到指定的 PostgreSQL 数据库，如 Heroku，这里作为演示就以默认的数据库 `postgres` 为例

查看配置系统文件路径

```
odbcinst -j
```

```
unixODBC 2.3.6
DRIVERS............: /etc/odbcinst.ini
SYSTEM DATA SOURCES: /etc/odbc.ini
FILE DATA SOURCES..: /etc/ODBCDataSources
USER DATA SOURCES..: /root/.odbc.ini
SQLULEN Size.......: 8
SQLLEN Size........: 8
SQLSETPOSIROW Size.: 8
```

不推荐修改全局配置文件,可设置 `ODBCSYSINI` 环境变量指定配置文件路径,如 `ODBCSYSINI=~/ODBC`
http://www.unixodbc.org/odbcinst.html

安装完驱动程序，`/etc/odbcinst.ini` 文件内容自动更新，我们可以不必修改，如果你想自定义不妨手动修改，我们查看在 R 环境中注册的数据库，可以看到 PostgreSQL 的驱动已经配置好

```
odbc::odbcListDrivers()
```

```
                 name   attribute                                     value
1     PostgreSQL ANSI Description    PostgreSQL ODBC driver (ANSI version)
2     PostgreSQL ANSI      Driver                                psqlodbca.so
3     PostgreSQL ANSI       Setup                            libodbcpsqlS.so
4     PostgreSQL ANSI       Debug                                         0
5     PostgreSQL ANSI     CommLog                                         1
6     PostgreSQL ANSI  UsageCount                                         1
7  PostgreSQL Unicode Description PostgreSQL ODBC driver (Unicode version)
8  PostgreSQL Unicode      Driver                                psqlodbcw.so
9  PostgreSQL Unicode       Setup                            libodbcpsqlS.so
10 PostgreSQL Unicode       Debug                                         0
11 PostgreSQL Unicode     CommLog                                         1
12 PostgreSQL Unicode  UsageCount                                         1
```

系统配置文件 `/etc/odbcinst.ini` 已经包含有 PostgreSQL 的驱动配置，无需再重复配置

```
[PostgreSQL ANSI]
Description=PostgreSQL ODBC driver (ANSI version)
Driver=psqlodbca.so
Setup=libodbcpsqlS.so
Debug=0
CommLog=1
UsageCount=1

[PostgreSQL Unicode]
Description=PostgreSQL ODBC driver (Unicode version)
Driver=psqlodbcw.so
Setup=libodbcpsqlS.so
Debug=0
CommLog=1
UsageCount=1
```

只需将如下内容存放在 `~/.odbc.ini` 文件中，

```
[PostgreSQL]
Driver              = PostgreSQL Unicode
Database            = postgres
Servername          = 172.17.0.1
UserName            = postgres
```

```
Password            = default
Port                = 8080
```

最后，一行命令 DNS 配置连接 https://github.com/r-dbi/odbc 这样就实现了代码中无任何敏感信息，这里为了展示这个配置过程故而把相关信息公开。

> 注意下面的内容需要在容器中运行，Windows 环境下的配置 PostgreSQL 的驱动有点麻烦就不搞了，意义也不大，现在数据库基本都是跑在 Linux 系统上

`docker-machine.exe ip default` 可以获得本地 Docker 的 IP，比如 192.168.99.101。Travis 上 `ip addr` 可以查看 Docker 的 IP，如 172.17.0.1

```
library(DBI)
con <- dbConnect(RPostgres::Postgres(),
  dbname = "postgres",
  host = ifelse(is_on_travis, Sys.getenv("DOCKER_HOST_IP"), "192.168.99.101"),
  port = 8080,
  user = "postgres",
  password = "default"
)
```

```
library(DBI)
con <- dbConnect(odbc::odbc(), "PostgreSQL")
```

列出数据库中的所有表

```
dbListTables(con)
#> [1] "mtcars"
```

第一次启动从 Docker Hub 上下载的镜像，默认的数据库是 postgres 里面没有任何表，所以将 R 环境中的 mtcars 数据集写入 postgres 数据库

将数据集 mtcars 写入 PostgreSQL 数据库中，基本操作，写入表的操作也不能缓存，即不能缓存数据库中的表 mtcars

```
dbWriteTable(con, "mtcars", mtcars, overwrite = TRUE)
```

现在可以看到数据表 mtcars 的各个字段

```
dbListFields(con, "mtcars")
#>  [1] "row_names" "mpg"       "cyl"       "disp"      "hp"
#>  [6] "drat"      "wt"        "qsec"      "vs"        "am"
#> [11] "gear"      "carb"
```

最后执行一条 SQL 语句

```
res <- dbSendQuery(con, "SELECT * FROM mtcars WHERE cyl = 4") # 发送 SQL 语句
dbFetch(res) # 获取查询结果
#>          row_names  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
#> 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        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
#> 6  Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
#> 7   Toyota Corona 21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
#> 8       Fiat X1-9 27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
#> 9   Porsche 914-2 26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
#> 10   Lotus Europa 30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
#> 11     Volvo 142E 21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
dbClearResult(res) # 清理查询通道
```

或者一条命令搞定

```
dbGetQuery(con, "SELECT * FROM mtcars WHERE cyl = 4")
#>          row_names  mpg cyl  disp  hp drat    wt  qsec vs am gear carb
#> 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        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
#> 6  Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
#> 7   Toyota Corona 21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
#> 8       Fiat X1-9 27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
#> 9   Porsche 914-2 26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
#> 10   Lotus Europa 30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
#> 11     Volvo 142E 21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
```

再复杂一点的 SQL 查询操作

```
dbGetQuery(con, "SELECT cyl, AVG(mpg) AS mpg FROM mtcars GROUP BY cyl ORDER BY cyl")
#>   cyl      mpg
#> 1   4 26.66364
#> 2   6 19.74286
#> 3   8 15.10000
aggregate(mpg ~ cyl, data = mtcars, mean)
```

表 1.5: 表格标题

cyl	mpg
4	26.66364
6	19.74286
8	15.10000

```
#>   cyl      mpg
#> 1   4 26.66364
#> 2   6 19.74286
#> 3   8 15.10000
```

得益于 knitr (Xie, 2015) 开发的钩子，这里直接写 SQL 语句块，打印出来见表 1.5，值得注意的是 SQL 代码块不能启用缓存，数据库连接通道也不能缓存，如果数据库中还没有写入表，那么写入表的操作也不能缓存

```
SELECT cyl, AVG(mpg) AS mpg FROM mtcars GROUP BY cyl ORDER BY cyl
```

如果将查询结果导出到变量，在 Chunk 设置 `output.var = "agg_cyl"` 可以使用缓存，下面将 mpg 按 cyl 分组聚合的结果打印出来

```
agg_cyl
#>   cyl      mpg
#> 1   4 26.66364
#> 2   6 19.74286
#> 3   8 15.10000
```

这种基于 odbc 的方式的好处就不需要再安装 R 包 RPostgres 和相关系统依赖，最后关闭连接通道

```
dbDisconnect(con)
```

1.4.2 MySQL

MySQL 是一个很常见，应用也很广泛的数据库，数据分析的常见环境是在一个 R Notebook 里，我们可以在正文之前先设定数据库连接信息

````
```{r setup}
library(DBI)
# 指定数据库连接信息
db <- dbConnect(RMySQL::MySQL(),
  dbname = 'dbtest',
````
````

```
  username = 'user_test',
  password = 'password',
  host = '10.10.101.10',
  port = 3306
)
# 创建默认连接
knitr::opts_chunk$set(connection = 'db')
# 设置字符编码，以免中文查询乱码
DBI::dbSendQuery(db, 'SET NAMES utf8')
# 设置日期变量，以运用在SQL中
idate <- '2019-05-03'
```

SQL 代码块中使用 R 环境中的变量，并将查询结果输出为 R 环境中的数据框

```
```{sql, output.var='data_output'}
SELECT * FROM user_table where date_format(created_date,'%Y-%m-%d')>=?idate
```
```

以上代码会将 SQL 的运行结果存在 `data_output` 这是数据库中，idate 取之前设置的日期 `2019-05-03`，`user_table` 是 MySQL 数据库中的表名，`created_date` 是创建 `user_table` 时，指定的日期名。

如果 SQL 比较长，为了代码美观，把带有变量的 SQL 保存为 `demo.sql` 脚本，只需要在 SQL 的 chunk 中直接读取 SQL 文件[6]。

```
```{sql, code=readLines('demo.sql'), output.var='data_output'}
```
```

如果我们需要每天或者按照指定的日期重复地运行这个 R Markdown 文件，可以在 YAML 部分引入参数[7]

```
---
params:
  date: "2019-05-03"  # 参数化日期
---
```

```
```{r setup, include=FALSE}
idate = params$date # 将参数化日期传递给 idate 变量
```
```

[6] https://d.cosx.org/d/419974

[7] https://bookdown.org/yihui/rmarkdown/params-knit.html

我们将这个 Rmd 文件命名为 MyDocument.Rmd，运行这个文件可以从 R 控制台执行或在 RStudio 点击 knit。

```
rmarkdown::render("MyDocument.Rmd", params = list(
  date = "2019-05-03"
))
```

如果在文档的 YAML 位置已经指定日期，这里可以不指定。注意在这里设置日期会覆盖 YAML 处指定的参数值，这样做的好处是可以批量化操作。

1.4.3 Spark

当数据分析报告遇上 Spark 时，就需要 SparkR、sparklyr、arrow 或 rsparking 接口了，Javier Luraschi 写了一本书 The R in Spark: Learning Apache Spark with R 详细介绍了相关扩展和应用

首先安装 sparklyr 包，RStudio 公司 Javier Lurasch 开发了 sparklyr 包，作为 Spark 与 R 语言之间的接口，安装完 sparklyr 包，还是需要 Spark 和 Hadoop 环境

```
install.packages('sparklyr')
library(sparklyr)
spark_install()
# Installing Spark 2.4.0 for Hadoop 2.7 or later.
# Downloading from:
# - 'https://archive.apache.org/dist/spark/spark-2.4.0/spark-2.4.0-bin-hadoop2.7.tgz'
# Installing to:
# - '~/spark/spark-2.4.0-bin-hadoop2.7'
# trying URL 'https://archive.apache.org/dist/spark/spark-2.4.0/spark-2.4.0-bin-hadoop2.7.tgz'
# Content type 'application/x-gzip' length 227893062 bytes (217.3 MB)
# ==================================================
# downloaded 217.3 MB
#
# Installation complete.
```

既然 sparklyr 已经安装了 Spark 和 Hadoop 环境，安装 SparkR 后，只需配置好路径，就可以加载 SparkR 包

```
install.packages('SparkR')
if (nchar(Sys.getenv("SPARK_HOME")) < 1) {
  Sys.setenv(SPARK_HOME = "~/spark/spark-2.4.0-bin-hadoop2.7")
}
library(SparkR, lib.loc = c(file.path(Sys.getenv("SPARK_HOME"), "R", "lib")))
```

```
sparkR.session(master = "local[*]", sparkConfig = list(spark.driver.memory = "2g"))
```

rscala 架起了 R 和 Scala 两门语言之间交流的桥梁，使得彼此之间可以互相调用

> 是否存在这样的可能，Spark 提供了大量的 MLib 库的调用接口，R 的功能支持是最少的，Java/Scala 是原生的，那么要么自己开发新的功能整合到 SparkR 中，要么借助 rscala 将 scala 接口代码封装进来

1.5　批量导入数据

```
library(tidyverse)
```

```
read_list <- function(list_of_datasets, read_func) {
  read_and_assign <- function(dataset, read_func) {
    dataset_name <- as.name(dataset)
    dataset_name <- read_func(dataset)
  }

  # invisible is used to suppress the unneeded output
  output <- invisible(
    sapply(list_of_datasets,
      read_and_assign,
      read_func = read_func, simplify = FALSE, USE.NAMES = TRUE
    )
  )

  # Remove the extension at the end of the data set names
  names_of_datasets <- c(unlist(strsplit(list_of_datasets, "[.]"))[c(T, F)])
  names(output) <- names_of_datasets
  return(output)
}
```

批量导入文件扩展名为 .csv 的数据文件，即逗号分割的文件

```
data_files <- list.files(path = "path/to/csv/dir",pattern = ".csv", full.names = TRUE)
print(data_files)
```

相比于 Base R 提供的 read.csv 函数，使用 readr 包的 read_csv 函数可以更快地读取 csv 格式文件，特别是在读取 GB 级数据文件时，效果特别明显。

```
list_of_data_sets <- read_list(data_files, readr::read_csv)
```

使用 tibble 包的 glimpse 函数可以十分方便地对整个数据集有一个大致的了解，展示方式和信息量相当于 str 加 head 函数

```
tibble::glimpse(list_of_data_sets)
```

1.6 批量导出数据

假定我们有一个列表，其每个元素都是一个数据框，现在要把每个数据框分别存入 xlsx 表的工作薄中，以 mtcars 数据集为例，将其按分类变量 cyl 分组拆分，获得一个列表 list

```
dat <- split(mtcars, mtcars$cyl)
dat
#> $`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
#> Honda Civic    30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
#> Toyota Corolla 33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
#> Toyota Corona  21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
#> Fiat X1-9      27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
#> Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
#> Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
#> Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
#>
#> $`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
#> Merc 280       19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
#> Merc 280C      17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
#> Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
#>
```

```
#> $`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
#> Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
#> Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
#> Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
#> Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
#> Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
#> AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
#> Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
#> Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
#> Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
#> Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
```

将 xlsx 表格初始化，创建空白的工作薄，openxlsx 包不依赖 Java 环境，读写效率也高

```
## 加载 openxlsx 包
library(openxlsx)
## 创建空白的工作薄
wb <- createWorkbook()
```

将列表里的每张表分别存入 xlsx 表格的每个 worksheet，worksheet 的名字就是分组变量的名字

```
Map(function(data, name){
    addWorksheet(wb, name)
    writeData(wb, name, data)

}, dat, names(dat))
```

最后保存数据到磁盘，见图 1.1

```
saveWorkbook(wb, file = "data/matcars.xlsx", overwrite = TRUE)
```

处理 Excel 2003 (XLS) 和 Excel 2007 (XLSX) 文件还可以使用 WriteXLS 包，不过它依赖于 Perl，另一个 R 包 xlsx 与之功能类似，依赖 Java 环境。Jennifer Bryan 和 Hadley Wickham 开发的 readxl 包和 Jeroen Ooms 开发的 writexl 包专门处理 xlsx 格式并且无任何系统依赖

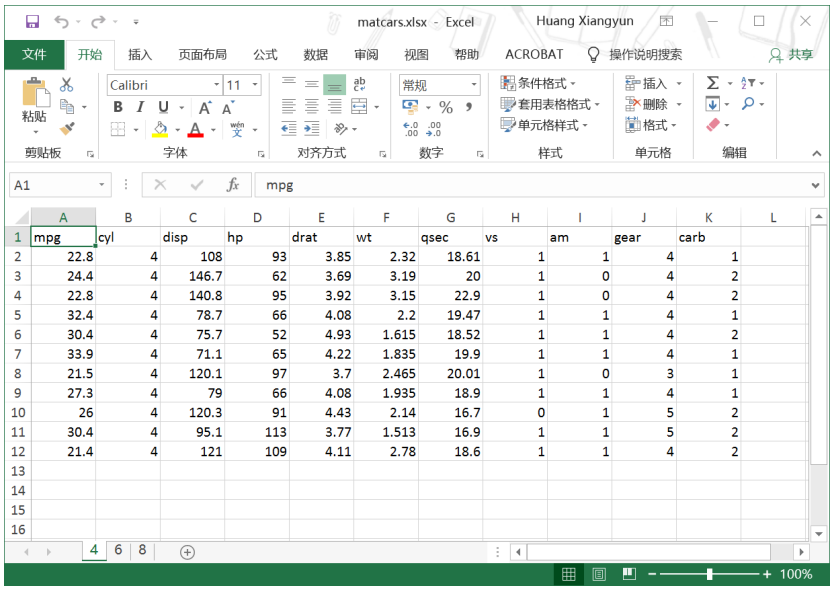


图 1.1: 批量导出数据

1.7　导出数据

1.7.1　导出运行结果

```
capture.output(..., file = NULL, append = FALSE,
               type = c("output", "message"), split = FALSE)
```

capture.output 将一段 R 代码执行结果，保存到文件，参数为表达式。capture.output 和 sink 的关系相当于 with 和 attach 的关系。

```
glmout <- capture.output(summary(glm(case ~ spontaneous + induced,
  data = infert, family = binomial()
)), file = "data/capture.txt")
capture.output(1 + 1, 2 + 2)
#> [1] "[1] 2" "[1] 4"
capture.output({
  1 + 1
  2 + 2
})
#> [1] "[1] 4"
```

sink 函数将控制台输出结果保存到文件，只将 outer 函数运行的结果保存到 ex-sink.txt 文件，outer 函数计算的是直积，在这里相当于 seq(10) %*% t(seq(10))，而在 R 语言中，更加有

效的计算方式是 tcrossprod(seq(10),seq(10))

```
sink("data/ex-sink.txt")
i <- 1:10
outer(i, i, "*")
#>       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
#>  [1,]    1    2    3    4    5    6    7    8    9    10
#>  [2,]    2    4    6    8   10   12   14   16   18    20
#>  [3,]    3    6    9   12   15   18   21   24   27    30
#>  [4,]    4    8   12   16   20   24   28   32   36    40
#>  [5,]    5   10   15   20   25   30   35   40   45    50
#>  [6,]    6   12   18   24   30   36   42   48   54    60
#>  [7,]    7   14   21   28   35   42   49   56   63    70
#>  [8,]    8   16   24   32   40   48   56   64   72    80
#>  [9,]    9   18   27   36   45   54   63   72   81    90
#> [10,]   10   20   30   40   50   60   70   80   90   100
sink()
```

1.7.2 导出数据对象

```
load(file, envir = parent.frame(), verbose = FALSE)

save(..., list = character(),
     file = stop("'file' must be specified"),
     ascii = FALSE, version = NULL, envir = parent.frame(),
     compress = isTRUE(!ascii), compression_level,
     eval.promises = TRUE, precheck = TRUE)

save.image(file = ".RData", version = NULL, ascii = FALSE,
           compress = !ascii, safe = TRUE)
```

load 和 save 函数加载或保存包含工作环境信息的数据对象，save.image 保存当前工作环境到磁盘，即保存工作空间中所有数据对象，数据格式为 .RData，即相当于

```
save(list = ls(all.names = TRUE), file = ".RData", envir = .GlobalEnv)
```

dump 保存数据对象 AirPassengers 到文件 AirPassengers.txt，文件内容是 R 命令，可把 AirPassengers.txt 看作代码文档执行，dput 保存数据对象内容到文件 AirPassengers.dat，文件中不包含变量名 AirPassengers。注意到 dump 输入是一个字符串，而 dput 要求输入数据对

象的名称，source 函数与 dump 对应，而 dget 与 dput 对应。

```
# 加载数据
data(AirPassengers, package = "datasets")
# 将数据以 R 代码块的形式保存到文件
dump('AirPassengers', file = 'data/AirPassengers.txt')
# source(file = 'data/AirPassengers.txt')
```

接下来，我们读取 AirPassengers.txt 的文件内容，可见它是一段完整的 R 代码，可以直接复制到 R 的控制台中运行，并且得到一个与原始 AirPassengers 变量一样的结果

```
cat(readLines('data/AirPassengers.txt'), sep = "\n")
#> AirPassengers <-
#> structure(c(112, 118, 132, 129, 121, 135, 148, 148, 136, 119,
#> 104, 118, 115, 126, 141, 135, 125, 149, 170, 170, 158, 133, 114,
#> 140, 145, 150, 178, 163, 172, 178, 199, 199, 184, 162, 146, 166,
#> 171, 180, 193, 181, 183, 218, 230, 242, 209, 191, 172, 194, 196,
#> 196, 236, 235, 229, 243, 264, 272, 237, 211, 180, 201, 204, 188,
#> 235, 227, 234, 264, 302, 293, 259, 229, 203, 229, 242, 233, 267,
#> 269, 270, 315, 364, 347, 312, 274, 237, 278, 284, 277, 317, 313,
#> 318, 374, 413, 405, 355, 306, 271, 306, 315, 301, 356, 348, 355,
#> 422, 465, 467, 404, 347, 305, 336, 340, 318, 362, 348, 363, 435,
#> 491, 505, 404, 359, 310, 337, 360, 342, 406, 396, 420, 472, 548,
#> 559, 463, 407, 362, 405, 417, 391, 419, 461, 472, 535, 622, 606,
#> 508, 461, 390, 432), .Tsp = c(1949, 1960.91666666667, 12), class = "ts")
```

dput 函数类似 dump 函数，保存数据对象到磁盘文件

```
# 将 R 对象保存/导出到磁盘
dput(AirPassengers, file = 'data/AirPassengers.dat')
AirPassengers
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1949 112 118 132 129 121 135 148 148 136 119 104 118
1950 115 126 141 135 125 149 170 170 158 133 114 140
1951 145 150 178 163 172 178 199 199 184 162 146 166
1952 171 180 193 181 183 218 230 242 209 191 172 194
1953 196 196 236 235 229 243 264 272 237 211 180 201
1954 204 188 235 227 234 264 302 293 259 229 203 229
1955 242 233 267 269 270 315 364 347 312 274 237 278
1956 284 277 317 313 318 374 413 405 355 306 271 306
```

```
1957 315 301 356 348 355 422 465 467 404 347 305 336
1958 340 318 362 348 363 435 491 505 404 359 310 337
1959 360 342 406 396 420 472 548 559 463 407 362 405
1960 417 391 419 461 472 535 622 606 508 461 390 432
# dget 作用与 dput 相反
AirPassengers2 <- dget(file = 'data/AirPassengers.dat')
AirPassengers2
     Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
1949 112 118 132 129 121 135 148 148 136 119 104 118
1950 115 126 141 135 125 149 170 170 158 133 114 140
1951 145 150 178 163 172 178 199 199 184 162 146 166
1952 171 180 193 181 183 218 230 242 209 191 172 194
1953 196 196 236 235 229 243 264 272 237 211 180 201
1954 204 188 235 227 234 264 302 293 259 229 203 229
1955 242 233 267 269 270 315 364 347 312 274 237 278
1956 284 277 317 313 318 374 413 405 355 306 271 306
1957 315 301 356 348 355 422 465 467 404 347 305 336
1958 340 318 362 348 363 435 491 505 404 359 310 337
1959 360 342 406 396 420 472 548 559 463 407 362 405
1960 417 391 419 461 472 535 622 606 508 461 390 432
```

同样地，现在我们观察 dput 函数保存的文件 AirPassengers.dat 内容，和 dump 函数保存的文件 AirPassengers.txt 相比，就缺一个赋值变量

```
cat(readLines('data/AirPassengers.dat'), sep = "\n")
structure(c(112, 118, 132, 129, 121, 135, 148, 148, 136, 119,
104, 118, 115, 126, 141, 135, 125, 149, 170, 170, 158, 133, 114,
140, 145, 150, 178, 163, 172, 178, 199, 199, 184, 162, 146, 166,
171, 180, 193, 181, 183, 218, 230, 242, 209, 191, 172, 194, 196,
196, 236, 235, 229, 243, 264, 272, 237, 211, 180, 201, 204, 188,
235, 227, 234, 264, 302, 293, 259, 229, 203, 229, 242, 233, 267,
269, 270, 315, 364, 347, 312, 274, 237, 278, 284, 277, 317, 313,
318, 374, 413, 405, 355, 306, 271, 306, 315, 301, 356, 348, 355,
422, 465, 467, 404, 347, 305, 336, 340, 318, 362, 348, 363, 435,
491, 505, 404, 359, 310, 337, 360, 342, 406, 396, 420, 472, 548,
559, 463, 407, 362, 405, 417, 391, 419, 461, 472, 535, 622, 606,
508, 461, 390, 432), .Tsp = c(1949, 1960.91666666667, 12), class = "ts")
```

1.8 运行环境

```
xfun::session_info(c("jsonlite", "yaml", "odbc"))
#> R version 3.6.1 (2019-07-05)
#> Platform: x86_64-pc-linux-gnu (64-bit)
#> Running under: Debian GNU/Linux 10 (buster)
#>
#> Locale:
#>   LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
#>   LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
#>   LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
#>   LC_PAPER=en_US.UTF-8       LC_NAME=C
#>   LC_ADDRESS=C               LC_TELEPHONE=C
#>   LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
#>
#> Package version:
#>   assertthat_0.2.1  backports_1.1.4   BH_1.69.0.1
#>   bit_1.1.14        bit64_0.9.7       blob_1.2.0
#>   DBI_1.0.0         digest_0.6.20     ellipsis_0.2.0.1
#>   glue_1.3.1        graphics_3.6.1    grDevices_3.6.1
#>   hms_0.5.0         jsonlite_1.6      magrittr_1.5
#>   methods_3.6.1     odbc_1.1.6        pkgconfig_2.0.2
#>   prettyunits_1.0.2 Rcpp_1.0.2        rlang_0.4.0
#>   stats_3.6.1       tools_3.6.1       utils_3.6.1
#>   vctrs_0.2.0       yaml_2.2.0        zeallot_0.1.0
```

第二章　数据操作手

参考 Data Manipulation With R (Spector, 2008) 重新捋一遍本章本章的操作对象是 data.frame

介绍 Base R 提供的数据操作，关于采用 Base R 还是 tidyverse 做数据操作的 讨论

数据操作的动画展示参考 https://github.com/gadenbuie/tidyexplain 提供 Base R 对应的实现

什么是 Base R? Base R 指的是 R 语言/软件的核心组件，由 R Core Team 维护

```
Pkgs <- sapply(list.files(R.home("library")), function(x)
  packageDescription(pkg = x, fields = "Priority"))
names(Pkgs[Pkgs == "base" & !is.na(Pkgs)])
#>  [1] "base"      "compiler"  "datasets"  "graphics"  "grDevices"
#>  [6] "grid"      "methods"   "parallel"  "splines"   "stats"
#> [11] "stats4"    "tcltk"     "tools"     "utils"
```

```
names(Pkgs[Pkgs == "recommended" & !is.na(Pkgs)])
#> character(0)
```

数据变形，分组统计聚合等，用以作为模型的输入，绘图的对象，操作的数据对象是数据框 (data.frame) 类型的，而且如果没有特别说明，文中出现的数据集都是 Base R 内置的，第三方 R 包或者来源于网上的数据集都会加以说明。

2.1　查看数据

查看属性

```
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 0.2 0.4 0.3 0.2 0.2 0.1 ...
#>  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1..
```

查看部分数据集

```
head(iris, 5)
#>   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
tail(iris, 5)
#>     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
#> 146          6.7         3.0          5.2         2.3 virginica
#> 147          6.3         2.5          5.0         1.9 virginica
#> 148          6.5         3.0          5.2         2.0 virginica
#> 149          6.2         3.4          5.4         2.3 virginica
#> 150          5.9         3.0          5.1         1.8 virginica
```

查看文件前（后）5 行

```
head -n 5 test.csv
tail -n 5 test.csv
```

对象的类型，存储方式

```
class(iris)
#> [1] "data.frame"
mode(iris)
#> [1] "list"
typeof(iris)
#> [1] "list"
```

查看对象在 R 环境中所占空间的大小

```
object.size(iris)
#> 7256 bytes
object.size(letters)
#> 1712 bytes
object.size(ls)
```

```
#> 89904 bytes
format(object.size(library), units = "auto")
#> [1] "1.8 Mb"
```

2.2　数据变形

重复测量数据的变形 Reshape Grouped Data，将宽格式 wide 的数据框变长格式 long 的，反之也行。reshape 还支持正则表达式

```
str(Indometh)
#> Classes 'nfnGroupedData', 'nfGroupedData', 'groupedData' and 'data.frame':   66 obs. of  3 variabl
#>  $ Subject: Ord.factor w/ 6 levels "1"<"4"<"2"<"5"<..: 1 1 1 1 1 1 1 1 1 ..
#>  $ time   : num  0.25 0.5 0.75 1 1.25 2 3 4 5 6 ...
#>  $ conc   : num  1.5 0.94 0.78 0.48 0.37 0.19 0.12 0.11 0.08 0.07 ...
#>  - attr(*, "formula")=Class 'formula'  language conc ~ time | Subject
#>   .. ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
#>  - attr(*, "labels")=List of 2
#>   ..$ x: chr "Time since drug administration"
#>   ..$ y: chr "Indomethacin concentration"
#>  - attr(*, "units")=List of 2
#>   ..$ x: chr "(hr)"
#>   ..$ y: chr "(mcg/ml)"
summary(Indometh)
#>  Subject      time           conc
#>  1:11    Min.   :0.250   Min.   :0.0500
#>  4:11    1st Qu.:0.750   1st Qu.:0.1100
#>  2:11    Median :2.000   Median :0.3400
#>  5:11    Mean   :2.886   Mean   :0.5918
#>  6:11    3rd Qu.:5.000   3rd Qu.:0.8325
#>  3:11    Max.   :8.000   Max.   :2.7200
```

```
# 长的变宽
wide <- reshape(Indometh,
  v.names = "conc", idvar = "Subject",
  timevar = "time", direction = "wide"
)
wide[, 1:6]
#>    Subject conc.0.25 conc.0.5 conc.0.75 conc.1 conc.1.25
```

```
#> 1        1      1.50     0.94      0.78   0.48      0.37
#> 12       2      2.03     1.63      0.71   0.70      0.64
#> 23       3      2.72     1.49      1.16   0.80      0.80
#> 34       4      1.85     1.39      1.02   0.89      0.59
#> 45       5      2.05     1.04      0.81   0.39      0.30
....

# 宽的变长
reshape(wide, direction = "long")
#>        Subject time conc
#> 1.0.25       1 0.25 1.50
#> 2.0.25       2 0.25 2.03
#> 3.0.25       3 0.25 2.72
#> 4.0.25       4 0.25 1.85
#> 5.0.25       5 0.25 2.05
....
```

宽的格式变成长的格式 https://stackoverflow.com/questions/2185252

长的格式变成宽的格式 https://stackoverflow.com/questions/5890584/

```
set.seed(45)
dat <- data.frame(
    name = rep(c("Orange", "Apple"), each=4),
    numbers = rep(1:4, 2),
    value = rnorm(8))
dat
#>     name numbers      value
#> 1 Orange       1  0.3407997
#> 2 Orange       2 -0.7033403
#> 3 Orange       3 -0.3795377
#> 4 Orange       4 -0.7460474
#> 5  Apple       1 -0.8981073
#> 6  Apple       2 -0.3347941
#> 7  Apple       3 -0.5013782
#> 8  Apple       4 -0.1745357

reshape(dat, idvar = "name", timevar = "numbers", direction = "wide")
#>     name    value.1    value.2    value.3    value.4
```

```
#> 1 Orange  0.3407997 -0.7033403 -0.3795377 -0.7460474
#> 5  Apple -0.8981073 -0.3347941 -0.5013782 -0.1745357
```

更加复杂的例子，gambia 数据集，重塑的效果是使得个体水平的长格式变为村庄水平的宽格式

```
# data(gambia, package = "geoR")
# 在线下载数据集
gambia <- read.table(
  file =
    paste("http://www.leg.ufpr.br/lib/exe/fetch.php",
      "pessoais:paulojus:mbgbook:datasets:gambia.txt",
      sep = "/"
    ), header = TRUE
)
head(gambia)
#>          x       y pos  age netuse treated green phc
#> 1 349631.3 1458055   1 1783      0       0 40.85   1
#> 2 349631.3 1458055   0  404      1       0 40.85   1
#> 3 349631.3 1458055   0  452      1       0 40.85   1
#> 4 349631.3 1458055   1  566      1       0 40.85   1
#> 5 349631.3 1458055   0  598      1       0 40.85   1
#> 6 349631.3 1458055   1  590      1       0 40.85   1
# Building a "village-level" data frame
ind <- paste("x", gambia[, 1], "y", gambia[, 2], sep = "")
village <- gambia[!duplicated(ind), c(1:2, 7:8)]
village$prev <- as.vector(tapply(gambia$pos, ind, mean))
head(village)
#>            x       y green phc      prev
#> 1   349631.3 1458055 40.85   1 0.5151515
#> 34  358543.1 1460112 40.85   1 0.3015873
#> 97  360308.1 1460026 40.10   0 0.4117647
#> 114 363795.7 1496919 40.85   0 0.3333333
#> 138 366400.5 1460248 40.85   0 0.3846154
#> 164 366687.5 1463002 40.85   0 0.3888889
```

2.3　数据转换

transform 对数据框中的某些列做计算，取对数，将计算的结果单存一列加到数据框中

```
transform(iris, scale.sl = (max(Sepal.Length) - Sepal.Length) / (max(Sepal.Length) - min(Sepal.Length
#>     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
....
```

验证一下 scale.sl 变量的第一个值

```
(max(iris$Sepal.Length) - 5.1) / (max(iris$Sepal.Length) - min(iris$Sepal.Length))
#> [1] 0.7777778
```

Warning: This is a convenience function intended for use interactively. For programming it is better to use the standard subsetting arithmetic functions, and in particular the non-standard evaluation of argument transform can have unanticipated consequences.

2.4 提取子集

```
subset(x, subset, select, drop = FALSE, ...)
```

参数 subset 代表行操作，select 代表列操作，函数 subset 从数据框中提取部分数据

```
subset(iris, Species == "virginica")
#>     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
#> 101          6.3         3.3          6.0         2.5 virginica
#> 102          5.8         2.7          5.1         1.9 virginica
#> 103          7.1         3.0          5.9         2.1 virginica
#> 104          6.3         2.9          5.6         1.8 virginica
#> 105          6.5         3.0          5.8         2.2 virginica
....
# summary(iris$Sepal.Length)  mean(iris$Sepal.Length)
# 且的逻辑
# subset(iris, Species == "virginica" & Sepal.Length > 5.84333)
subset(iris, Species == "virginica" &
  Sepal.Length > mean(Sepal.Length))
#>     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
```

```
#> 101          6.3         3.3          6.0         2.5 virginica
#> 103          7.1         3.0          5.9         2.1 virginica
#> 104          6.3         2.9          5.6         1.8 virginica
#> 105          6.5         3.0          5.8         2.2 virginica
#> 106          7.6         3.0          6.6         2.1 virginica
....
# 在行的子集范围内
subset(iris, Species %in% c("virginica", "versicolor") &
  Sepal.Length > mean(Sepal.Length))
#>     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
#> 51           7.0         3.2          4.7         1.4 versicolor
#> 52           6.4         3.2          4.5         1.5 versicolor
#> 53           6.9         3.1          4.9         1.5 versicolor
#> 55           6.5         2.8          4.6         1.5 versicolor
#> 57           6.3         3.3          4.7         1.6 versicolor
....
# 在列的子集内 先选中列
subset(iris, Sepal.Length > mean(Sepal.Length),
  select = c("Sepal.Length", "Species")
)
#>     Sepal.Length    Species
#> 51           7.0 versicolor
#> 52           6.4 versicolor
#> 53           6.9 versicolor
#> 55           6.5 versicolor
#> 57           6.3 versicolor
....
```

高级操作：加入正则表达式筛选

```
## sometimes requiring a logical 'subset' argument is a nuisance
nm <- rownames(state.x77)
start_with_M <- nm %in% grep("^M", nm, value = TRUE)
subset(state.x77, start_with_M, Illiteracy:Murder)
#>               Illiteracy Life Exp Murder
#> Maine                0.7    70.39    2.7
#> Maryland             0.9    70.22    8.5
#> Massachusetts        1.1    71.83    3.3
#> Michigan             0.9    70.63   11.1
```

```
#> Minnesota            0.6    72.96    2.3
#> Mississippi          2.4    68.09   12.5
#> Missouri             0.8    70.69    9.3
#> Montana              0.6    70.56    5.0
# 简化
subset(state.x77, subset = grepl("^M", rownames(state.x77)), select = Illiteracy:Murder)
#>               Illiteracy Life Exp Murder
#> Maine                0.7    70.39    2.7
#> Maryland             0.9    70.22    8.5
#> Massachusetts        1.1    71.83    3.3
#> Michigan             0.9    70.63   11.1
#> Minnesota            0.6    72.96    2.3
#> Mississippi          2.4    68.09   12.5
#> Missouri             0.8    70.69    9.3
#> Montana              0.6    70.56    5.0
# 继续简化
subset(state.x77, grepl("^M", rownames(state.x77)), Illiteracy:Murder)
#>               Illiteracy Life Exp Murder
#> Maine                0.7    70.39    2.7
#> Maryland             0.9    70.22    8.5
#> Massachusetts        1.1    71.83    3.3
#> Michigan             0.9    70.63   11.1
#> Minnesota            0.6    72.96    2.3
#> Mississippi          2.4    68.09   12.5
#> Missouri             0.8    70.69    9.3
#> Montana              0.6    70.56    5.0
```

警告：这是一个为了交互使用打造的便捷函数。对于编程，最好使用标准的子集函数，如 `[`，特别地，参数 subset 的非标准计算 (non-standard evaluation)[1] 可能带来意想不到的后果。

使用索引 `[`

```
iris[iris$Species == "virginica", ]
#>     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
#> 101          6.3         3.3          6.0         2.5 virginica
#> 102          5.8         2.7          5.1         1.9 virginica
#> 103          7.1         3.0          5.9         2.1 virginica
#> 104          6.3         2.9          5.6         1.8 virginica
```

```
#> 105          6.5         3.0          5.8         2.2 virginica
....
iris[iris$Species == "virginica" &
  iris$Sepal.Length > mean(iris$Sepal.Length), ]
#>     Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
#> 101          6.3         3.3          6.0         2.5 virginica
#> 103          7.1         3.0          5.9         2.1 virginica
#> 104          6.3         2.9          5.6         1.8 virginica
#> 105          6.5         3.0          5.8         2.2 virginica
#> 106          7.6         3.0          6.6         2.1 virginica
....

iris[
  iris$Species == "virginica" &
    iris$Sepal.Length > mean(iris$Sepal.Length),
  c("Sepal.Length", "Species")
]
#>     Sepal.Length   Species
#> 101          6.3 virginica
#> 103          7.1 virginica
#> 104          6.3 virginica
#> 105          6.5 virginica
#> 106          7.6 virginica
....
```

2.5　按列排序

在数据框内，根据 (order) 某一列或几列对行进行排序 (sort)，根据鸢尾花 (iris) 的类别 (Species) 对萼片 (sepal) 的长度进行排序，其余的列随之变化

```
# 对萼片的长度排序
iris[order(iris$Species, iris$Sepal.Length), ]
#>     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
#> 14          4.3         3.0          1.1         0.1     setosa
#> 9           4.4         2.9          1.4         0.2     setosa
#> 39          4.4         3.0          1.3         0.2     setosa
#> 43          4.4         3.2          1.3         0.2     setosa
```

```
#> 42           4.5         2.3          1.3         0.3     setosa
....
# 对花瓣的长度排序
iris[order(iris$Species, iris$Petal.Length), ]
#>     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
#> 23           4.6         3.6          1.0         0.2     setosa
#> 14           4.3         3.0          1.1         0.1     setosa
#> 15           5.8         4.0          1.2         0.2     setosa
#> 36           5.0         3.2          1.2         0.2     setosa
#> 3            4.7         3.2          1.3         0.2     setosa
....
# 先对花瓣的宽度排序，再对花瓣的长度排序
iris[order(iris$Petal.Width, iris$Petal.Length), ]
#>     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species
#> 14           4.3         3.0          1.1         0.1     setosa
#> 13           4.8         3.0          1.4         0.1     setosa
#> 38           4.9         3.6          1.4         0.1     setosa
#> 10           4.9         3.1          1.5         0.1     setosa
#> 33           5.2         4.1          1.5         0.1     setosa
....
```

sort/ordered 排序，默认是升序

```
dd <- data.frame(
  b = factor(c("Hi", "Med", "Hi", "Low"),
    levels = c("Low", "Med", "Hi"), ordered = TRUE
  ),
  x = c("A", "D", "A", "C"), y = c(8, 3, 9, 9),
  z = c(1, 1, 1, 2)
)
str(dd)
#> 'data.frame':    4 obs. of  4 variables:
#>  $ b: Ord.factor w/ 3 levels "Low"<"Med"<"Hi": 3 2 3 1
#>  $ x: chr  "A" "D" "A" "C"
#>  $ y: num  8 3 9 9
#>  $ z: num  1 1 1 2
dd[order(-dd[,4], dd[,1]), ]
#>     b x y z
#> 4 Low C 9 2
```

```
#> 2 Med D 3 1
#> 1  Hi A 8 1
#> 3  Hi A 9 1
```

根据变量 z

```
dd[order(dd$z, dd$b), ]
#>     b x y z
#> 2 Med D 3 1
#> 1  Hi A 8 1
#> 3  Hi A 9 1
#> 4 Low C 9 2
```

2.6　数据拆分

数据拆分通常是按找某一个分类变量分组，分完组就是计算，计算完就把结果按照原来的分组方式合并

```
## Notice that assignment form is not used since a variable is being added
g <- airquality$Month
l <- split(airquality, g) # 分组
l <- lapply(l, transform, Oz.Z = scale(Ozone)) # 计算：按月对 Ozone 标准化
aq2 <- unsplit(l, g) # 合并
head(aq2)
#>   Ozone Solar.R Wind Temp Month Day       Oz.Z
#> 1    41     190  7.4   67     5   1  0.7822293
#> 2    36     118  8.0   72     5   2  0.5572518
#> 3    12     149 12.6   74     5   3 -0.5226399
#> 4    18     313 11.5   62     5   4 -0.2526670
#> 5    NA      NA 14.3   56     5   5         NA
#> 6    28      NA 14.9   66     5   6  0.1972879
```

tapply 自带分组的功能，按月份 Month 对 Ozone 中心标准化，其返回一个列表

```
with(airquality, tapply(Ozone, Month, scale))
#> $`5`
#>              [,1]
#>  [1,]  0.78222929
#>  [2,]  0.55725184
#>  [3,] -0.52263993
```

```
#>  [4,] -0.25266698
#>  [5,]          NA
#>  [6,]  0.19728792
#>  [7,] -0.02768953
#>  [8,] -0.20767149
....
```

上面的过程等价于

```
do.call("rbind", lapply(split(airquality, airquality$Month), transform, Oz.Z = scale(Ozone)))
#>       Ozone Solar.R Wind Temp Month Day         Oz.Z
#> 5.1      41     190  7.4   67     5   1  0.782229293
#> 5.2      36     118  8.0   72     5   2  0.557251841
#> 5.3      12     149 12.6   74     5   3 -0.522639926
#> 5.4      18     313 11.5   62     5   4 -0.252666984
#> 5.5      NA      NA 14.3   56     5   5           NA
#> 5.6      28      NA 14.9   66     5   6  0.197287919
#> 5.7      23     299  8.6   65     5   7 -0.027689532
#> 5.8      19      99 13.8   59     5   8 -0.207671494
#> 5.9       8      19 20.1   61     5   9 -0.702621887
....
```

由于上面对 Ozone 正态标准化，所以标准化后的 `Oz.z` 再按月分组计算方差自然每个月都是 1，而均值都是 0。

```
with(aq2, tapply(Oz.Z, Month, sd, na.rm = TRUE))
#> 5 6 7 8 9
#> 1 1 1 1 1
with(aq2, tapply(Oz.Z, Month, mean, na.rm = TRUE))
#>             5             6             7             8             9
#> -4.240273e-17  1.052760e-16  5.841432e-17  5.898060e-17  2.571709e-17
```

> 循着这个思路，我们可以用 tapply 实现分组计算，上面函数 `sd` 和 `mean` 完全可以用自定义的更加复杂的函数替代

`cut` 函数可以将连续型变量划分为分类变量

```
set.seed(2019)
Z <- stats::rnorm(10)
cut(Z, breaks = -6:6)
#>  [1] (0,1]   (-1,0]  (-2,-1] (0,1]   (-2,-1] (0,1]   (-1,0]  (0,1]
#>  [9] (-2,-1] (-1,0]
```

```
#> 12 Levels: (-6,-5] (-5,-4] (-4,-3] (-3,-2] (-2,-1] (-1,0] (0,1] ... (5,6]
# labels = FALSE 返回每个数所落的区间位置
cut(Z, breaks = -6:6, labels = FALSE)
#>  [1] 7 6 5 7 5 7 6 7 5 6
```

我们还可以指定参数 dig.lab 设置分组的精度，ordered 将分组变量看作是有序的，breaks 传递单个数时，表示分组数，而不是断点

```
cut(Z, breaks = 3, dig.lab = 4, ordered = TRUE)
#>  [1] (0.06396,0.9186]  (-0.7881,0.06396] (-1.643,-0.7881]
#>  [4] (0.06396,0.9186]  (-1.643,-0.7881]  (0.06396,0.9186]
#>  [7] (-0.7881,0.06396] (0.06396,0.9186]  (-1.643,-0.7881]
#> [10] (-0.7881,0.06396]
#> Levels: (-1.643,-0.7881] < (-0.7881,0.06396] < (0.06396,0.9186]
```

此时，统计每组的频数，如图 2.1

```
# 条形图
plot(cut(Z, breaks = -6:6))
# 直方图
hist(Z, breaks = -6:6)
```

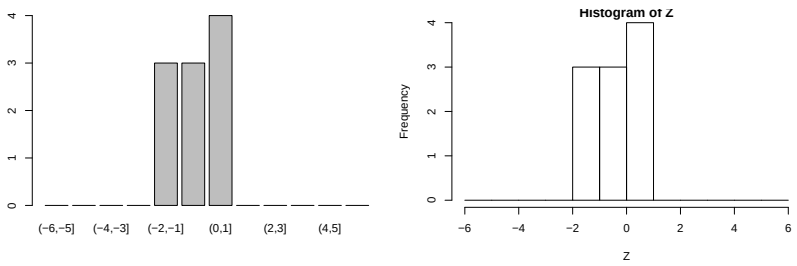


图 2.1: 连续型变量分组统计

在指定分组数的情况下，我们还想获取分组的断点

```
labs <- levels(cut(Z, 3))
labs
#> [1] "(-1.64,-0.788]" "(-0.788,0.064]" "(0.064,0.919]"
```

用正则表达式抽取断点

```
cbind(
  lower = as.numeric(sub("\\((.+),.*", "\\1", labs)),
  upper = as.numeric(sub("[^,]*,([^]]*)\\]", "\\1", labs))
```

```
)
#>       lower  upper
#> [1,] -1.640 -0.788
#> [2,] -0.788  0.064
#> [3,]  0.064  0.919
```

更多相关函数可以参考 `findInterval` 和 `embed`

tabulate 和 table 有所不同，它表示排列

```
t(combn(8, 4, tabulate, nbins = 8))
#>       [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]
#>  [1,]    1    1    1    1    0    0    0    0
#>  [2,]    1    1    1    0    1    0    0    0
#>  [3,]    1    1    1    0    0    1    0    0
#>  [4,]    1    1    1    0    0    0    1    0
#>  [5,]    1    1    1    0    0    0    0    1
....
```

2.7　数据合并

merge 合并两个数据框

```
authors <- data.frame(
  ## I(*) : use character columns of names to get sensible sort order
  surname = I(c("Tukey", "Venables", "Tierney", "Ripley", "McNeil")),
  nationality = c("US", "Australia", "US", "UK", "Australia"),
  deceased = c("yes", rep("no", 4))
)
authorN <- within(authors, {
  name <- surname
  rm(surname)
})
books <- data.frame(
  name = I(c(
    "Tukey", "Venables", "Tierney",
    "Ripley", "Ripley", "McNeil", "R Core"
  )),
  title = c(
```

```
    "Exploratory Data Analysis",
    "Modern Applied Statistics ...",
    "LISP-STAT",
    "Spatial Statistics", "Stochastic Simulation",
    "Interactive Data Analysis",
    "An Introduction to R"
  ),
  other.author = c(
    NA, "Ripley", NA, NA, NA, NA,
    "Venables & Smith"
  )
)

authors
#>    surname nationality deceased
#> 1    Tukey          US      yes
#> 2 Venables   Australia       no
#> 3  Tierney          US       no
#> 4   Ripley          UK       no
#> 5   McNeil   Australia       no
authorN
#>   nationality deceased     name
#> 1          US      yes    Tukey
#> 2   Australia       no Venables
#> 3          US       no  Tierney
#> 4          UK       no   Ripley
#> 5   Australia       no   McNeil
books
#>       name                          title     other.author
#> 1    Tukey     Exploratory Data Analysis             <NA>
#> 2 Venables Modern Applied Statistics ...           Ripley
#> 3  Tierney                     LISP-STAT             <NA>
#> 4   Ripley            Spatial Statistics             <NA>
#> 5   Ripley         Stochastic Simulation             <NA>
#> 6   McNeil     Interactive Data Analysis             <NA>
#> 7   R Core          An Introduction to R Venables & Smith
```

默认找到同名的列，然后是同名的行合并，多余的没有匹配到的就丢掉

```
merge(authorN, books)
#>       name nationality deceased                       title other.author
#> 1   McNeil   Australia       no    Interactive Data Analysis         <NA>
#> 2   Ripley          UK       no           Spatial Statistics         <NA>
#> 3   Ripley          UK       no        Stochastic Simulation         <NA>
#> 4  Tierney          US       no                    LISP-STAT         <NA>
#> 5    Tukey          US      yes    Exploratory Data Analysis         <NA>
#> 6 Venables   Australia       no Modern Applied Statistics ...       Ripley
```

还可以指定合并的列，先按照 surname 合并，留下 surname

```
merge(authors, books, by.x = "surname", by.y = "name")
#>    surname nationality deceased                       title other.author
#> 1   McNeil   Australia       no    Interactive Data Analysis         <NA>
#> 2   Ripley          UK       no           Spatial Statistics         <NA>
#> 3   Ripley          UK       no        Stochastic Simulation         <NA>
#> 4  Tierney          US       no                    LISP-STAT         <NA>
#> 5    Tukey          US      yes    Exploratory Data Analysis         <NA>
#> 6 Venables   Australia       no Modern Applied Statistics ...       Ripley
```

留下的是 name

```
merge(books, authors, by.x = "name", by.y = "surname")
#>       name                       title other.author nationality deceased
#> 1   McNeil    Interactive Data Analysis         <NA>   Australia       no
#> 2   Ripley           Spatial Statistics         <NA>          UK       no
#> 3   Ripley       Stochastic Simulation         <NA>          UK       no
#> 4  Tierney                   LISP-STAT         <NA>          US       no
#> 5    Tukey    Exploratory Data Analysis         <NA>          US      yes
#> 6 Venables Modern Applied Statistics ...       Ripley   Australia       no
```

为了比较清楚地观察几种合并的区别，这里提供对应的动画展示 https://github.com/gadenbuie/tidyexplain

(inner, outer, left, right, cross) join 共 5 种合并方式详情请看 https://stackoverflow.com/questions/1299871

cbind 和 rbind 分别是按列和行合并数据框

2.8　数据去重

单个数值型向量去重，此时和 unique 函数作用一样

```
(x <- c(9:20, 1:5, 3:7, 0:8))
#>  [1]  9 10 11 12 13 14 15 16 17 18 19 20  1  2  3  4  5  3  4  5  6  7  0
#> [24]  1  2  3  4  5  6  7  8
## extract unique elements
x[!duplicated(x)]
#>  [1]  9 10 11 12 13 14 15 16 17 18 19 20  1  2  3  4  5  6  7  0  8
unique(x)
#>  [1]  9 10 11 12 13 14 15 16 17 18 19 20  1  2  3  4  5  6  7  0  8
```

数据框类型数据中，去除重复的行，这个重复可以是多个变量对应的向量

```
set.seed(123)
df <- data.frame(
  x = sample(0:1, 10, replace = T),
  y = sample(0:1, 10, replace = T),
  z = 1:10
)
df
#>    x y  z
#> 1  0 1  1
#> 2  0 1  2
#> 3  0 1  3
#> 4  1 0  4
#> 5  0 1  5
#> 6  1 0  6
#> 7  1 1  7
#> 8  1 0  8
#> 9  0 0  9
#> 10 0 0 10
df[!duplicated(df[, 1:2]), ]
#>   x y z
#> 1 0 1 1
#> 4 1 0 4
#> 7 1 1 7
#> 9 0 0 9
```

2.9　数据聚合

分组求和 https://stackoverflow.com/questions/1660124

主要是分组统计

```
apropos("apply")
#>  [1] "apply"      "dendrapply" "eapply"     "kernapply"  "lapply"
#>  [6] "mapply"     "rapply"     "sapply"     "tapply"     "vapply"
```

```
# 分组求和 colSums colMeans max
unique(iris$Species)
#> [1] setosa     versicolor virginica
#> Levels: setosa versicolor virginica
# 分类求和
# colSums(iris[iris$Species == "setosa", -5])
# colSums(iris[iris$Species == "virginica", -5])
colSums(iris[iris$Species == "versicolor", -5])
#> Sepal.Length  Sepal.Width Petal.Length  Petal.Width
#>        296.8        138.5        213.0         66.3
# apply(iris[iris$Species == "setosa", -5], 2, sum)
# apply(iris[iris$Species == "setosa", -5], 2, mean)
# apply(iris[iris$Species == "setosa", -5], 2, min)
# apply(iris[iris$Species == "setosa", -5], 2, max)
apply(iris[iris$Species == "setosa", -5], 2, quantile)
#>      Sepal.Length Sepal.Width Petal.Length Petal.Width
#> 0%            4.3       2.300        1.000         0.1
#> 25%           4.8       3.200        1.400         0.2
#> 50%           5.0       3.400        1.500         0.2
#> 75%           5.2       3.675        1.575         0.3
#> 100%          5.8       4.400        1.900         0.6
```

aggregate: Compute Summary Statistics of Data Subsets

```
# 按分类变量 Species 分组求和
# aggregate(subset(iris, select = -Species), by = list(iris[, "Species"]), FUN = sum)
aggregate(iris[, -5], list(iris[, 5]), sum)
#>      Group.1 Sepal.Length Sepal.Width Petal.Length Petal.Width
#> 1     setosa        250.3       171.4         73.1        12.3
#> 2 versicolor        296.8       138.5        213.0        66.3
#> 3  virginica        329.4       148.7        277.6       101.3
```

```
# 先确定位置，假设有很多分类变量
ind <- which("Species" == colnames(iris))
# 分组统计
aggregate(iris[, -ind], list(iris[, ind]), sum)
#>      Group.1 Sepal.Length Sepal.Width Petal.Length Petal.Width
#> 1     setosa        250.3       171.4         73.1        12.3
#> 2 versicolor        296.8       138.5        213.0        66.3
#> 3  virginica        329.4       148.7        277.6       101.3
```

按照 Species 划分的类别，分组计算，使用公式表示形式，右边一定是分类变量，否则会报错误或者警告，输出奇怪的结果，请读者尝试运行 `aggregate(Species ~ Sepal.Length, data = iris, mean)`。公式法表示分组计算，`~` 左手边可以做加 `+` 减 `-` 乘 `*` 除 `/` 取余 `%%` 等数学运算。下面以数据集 iris 为例，只对 Sepal.Length 按 Species 分组计算

```
aggregate(Sepal.Length ~ Species, data = iris, mean)
#>      Species Sepal.Length
#> 1     setosa        5.006
#> 2 versicolor        5.936
#> 3  virginica        6.588
```

与上述分组统计结果一样的命令，在大数据集上，与 aggregate 相比，tapply 要快很多，by 是 tapply 的包裹，处理速度差不多。读者可以构造伪随机数据集验证。

```
# tapply(iris$Sepal.Length, list(iris$Species), mean)
with(iris, tapply(Sepal.Length, Species, mean))
#>     setosa versicolor  virginica
#>      5.006      5.936      6.588
by(iris$Sepal.Length, iris$Species, mean)
#> iris$Species: setosa
#> [1] 5.006
#> ------------------------------------------------------------
#> iris$Species: versicolor
#> [1] 5.936
#> ------------------------------------------------------------
#> iris$Species: virginica
#> [1] 6.588
```

对所有变量按 Species 分组计算

```
aggregate(. ~ Species, data = iris, mean)
#>      Species Sepal.Length Sepal.Width Petal.Length Petal.Width
```

```
#> 1     setosa        5.006       3.428        1.462       0.246
#> 2 versicolor        5.936       2.770        4.260       1.326
#> 3  virginica        6.588       2.974        5.552       2.026
```

对变量 Sepal.Length 和 Sepal.Width 求和后，按 Species 分组计算

```
aggregate(Sepal.Length + Sepal.Width ~ Species, data = iris, mean)
#>      Species Sepal.Length + Sepal.Width
#> 1     setosa                      8.434
#> 2 versicolor                      8.706
#> 3  virginica                      9.562
```

对多个分类变量做分组计算，在数据集 ChickWeight 中 Chick 和 Diet 都是数字编码的分类变量，其中 Chick 是有序的因子变量，Diet 是无序的因子变量，而 Time 是数值型的变量，表示小鸡出生的天数。

```
# 查看数据
str(ChickWeight)
#> Classes 'nfnGroupedData', 'nfGroupedData', 'groupedData' and 'data.frame':   578 obs. of  4 variab
#>  $ weight: num  42 51 59 64 76 93 106 125 149 171 ...
#>  $ Time  : num  0 2 4 6 8 10 12 14 16 18 ...
#>  $ Chick : Ord.factor w/ 50 levels "18"<"16"<"15"<..: 15 15 15 15 15 15 1..
#>  $ Diet  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
#>  - attr(*, "formula")=Class 'formula'  language weight ~ Time | Chick
#>   .. ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
#>  - attr(*, "outer")=Class 'formula'  language ~Diet
#>   .. ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
#>  - attr(*, "labels")=List of 2
#>   ..$ x: chr "Time"
#>   ..$ y: chr "Body weight"
#>  - attr(*, "units")=List of 2
#>   ..$ x: chr "(days)"
#>   ..$ y: chr "(gm)"
```

查看数据集 ChickWeight 的前几行

```
head(ChickWeight)
#>   weight Time Chick Diet
#> 1     42    0     1    1
#> 2     51    2     1    1
#> 3     59    4     1    1
```

```
#> 4     64    6     1    1
#> 5     76    8     1    1
....
str(ChickWeight)
#> Classes 'nfnGroupedData', 'nfGroupedData', 'groupedData' and 'data.frame':   578 obs. of  4 variab
#>  $ weight: num  42 51 59 64 76 93 106 125 149 171 ...
#>  $ Time  : num  0 2 4 6 8 10 12 14 16 18 ...
#>  $ Chick : Ord.factor w/ 50 levels "18"<"16"<"15"<..: 15 15 15 15 15 15 1..
#>  $ Diet  : Factor w/ 4 levels "1","2","3","4": 1 1 1 1 1 1 1 1 1 1 ...
#>  - attr(*, "formula")=Class 'formula'  language weight ~ Time | Chick
....
```

对于数据集 ChickWeight 中的有序变量 Chick，aggregate 会按照既定顺序返回分组计算的结果

```
aggregate(weight ~ Chick, data = ChickWeight, mean)
#>    Chick    weight
#> 1     18  37.00000
#> 2     16  49.71429
#> 3     15  60.12500
#> 4     13  67.83333
#> 5      9  81.16667
....
aggregate(weight ~ Diet, data = ChickWeight, mean)
#>   Diet   weight
#> 1    1 102.6455
#> 2    2 122.6167
#> 3    3 142.9500
#> 4    4 135.2627
```

分类变量没有用数字编码，以 CO2 数据集为例，该数据集描述草植对二氧化碳的吸收情况，Plant 是具有 12 个水平的有序的因子变量，Type 表示植物的源头分别是魁北克 (Quebec) 和密西西比 (Mississippi)，Treatment 表示冷却 (chilled) 和不冷却 (nonchilled) 两种处理方式，conc 表示周围环境中二氧化碳的浓度，uptake 表示植物吸收二氧化碳的速率。

```
# 查看数据集
head(CO2)
#>   Plant   Type  Treatment conc uptake
#> 1   Qn1 Quebec nonchilled   95   16.0
#> 2   Qn1 Quebec nonchilled  175   30.4
#> 3   Qn1 Quebec nonchilled  250   34.8
```

```
#> 4   Qn1 Quebec nonchilled  350   37.2
#> 5   Qn1 Quebec nonchilled  500   35.3
#> 6   Qn1 Quebec nonchilled  675   39.2
str(CO2)
#> Classes 'nfnGroupedData', 'nfGroupedData', 'groupedData' and 'data.frame':   84 obs. of  5 variabl
#>  $ Plant    : Ord.factor w/ 12 levels "Qn1"<"Qn2"<"Qn3"<..: 1 1 1 1 1 1 1..
#>  $ Type     : Factor w/ 2 levels "Quebec","Mississippi": 1 1 1 1 1 1 1 ..
#>  $ Treatment: Factor w/ 2 levels "nonchilled","chilled": 1 1 1 1 1 1 1 ..
#>  $ conc     : num  95 175 250 350 500 675 1000 95 175 250 ...
#>  $ uptake   : num  16 30.4 34.8 37.2 35.3 39.2 39.7 13.6 27.3 37.1 ...
#>  - attr(*, "formula")=Class 'formula'  language uptake ~ conc | Plant
#>   .. ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
#>  - attr(*, "outer")=Class 'formula'  language ~Treatment * Type
#>   .. ..- attr(*, ".Environment")=<environment: R_EmptyEnv>
#>  - attr(*, "labels")=List of 2
#>   ..$ x: chr "Ambient carbon dioxide concentration"
#>   ..$ y: chr "CO2 uptake rate"
#>  - attr(*, "units")=List of 2
#>   ..$ x: chr "(uL/L)"
#>   ..$ y: chr "(umol/m^2 s)"
```

对单个变量分组统计

```
aggregate(uptake ~ Plant, data = CO2, mean)
#>    Plant   uptake
#> 1    Qn1 33.22857
#> 2    Qn2 35.15714
#> 3    Qn3 37.61429
#> 4    Qc1 29.97143
#> 5    Qc3 32.58571
#> 6    Qc2 32.70000
#> 7    Mn3 24.11429
#> 8    Mn2 27.34286
#> 9    Mn1 26.40000
#> 10   Mc2 12.14286
#> 11   Mc3 17.30000
#> 12   Mc1 18.00000
aggregate(uptake ~ Type, data = CO2, mean)
#>          Type   uptake
```

```
#> 1      Quebec 33.54286
#> 2 Mississippi 20.88333
aggregate(uptake ~ Treatment, data = CO2, mean)
#>    Treatment   uptake
#> 1 nonchilled 30.64286
#> 2    chilled 23.78333
```

对多个变量分组统计，查看二氧化碳吸收速率 uptake 随类型 Type 和处理方式 Treatment

```
aggregate(uptake ~ Type + Treatment, data = CO2, mean)
#>          Type  Treatment   uptake
#> 1      Quebec nonchilled 35.33333
#> 2 Mississippi nonchilled 25.95238
#> 3      Quebec    chilled 31.75238
#> 4 Mississippi    chilled 15.81429
tapply(CO2$uptake, list(CO2$Type, CO2$Treatment), mean)
#>             nonchilled  chilled
#> Quebec        35.33333 31.75238
#> Mississippi   25.95238 15.81429
by(CO2$uptake, list(CO2$Type, CO2$Treatment), mean)
#> : Quebec
#> : nonchilled
#> [1] 35.33333
#> ------------------------------------------------------------
#> : Mississippi
#> : nonchilled
#> [1] 25.95238
#> ------------------------------------------------------------
#> : Quebec
#> : chilled
#> [1] 31.75238
#> ------------------------------------------------------------
#> : Mississippi
#> : chilled
#> [1] 15.81429
```

在这个例子中 tapply 和 by 的输出结果的表示形式不一样，aggregate 返回一个 data.frame 数据框，tapply 返回一个表格 table，by 返回特殊的数据类型 by。

Function by is an object-oriented wrapper for `tapply` applied to data frames.

```r
# 分组求和
# by(iris[, 1], INDICES = list(iris$Species), FUN = sum)
# by(iris[, 2], INDICES = list(iris$Species), FUN = sum)
by(iris[, 3], INDICES = list(iris$Species), FUN = sum)
#> : setosa
#> [1] 73.1
#> ------------------------------------------------------------
#> : versicolor
#> [1] 213
#> ------------------------------------------------------------
#> : virginica
#> [1] 277.6
by(iris[1:3], INDICES = list(iris$Species), FUN = sum)
#> : setosa
#> [1] 494.8
#> ------------------------------------------------------------
#> : versicolor
#> [1] 648.3
#> ------------------------------------------------------------
#> : virginica
#> [1] 755.7
by(iris[1:3], INDICES = list(iris$Species), FUN = summary)
#> : setosa
#>   Sepal.Length    Sepal.Width     Petal.Length
#>  Min.   :4.300   Min.   :2.300   Min.   :1.000
#>  1st Qu.:4.800   1st Qu.:3.200   1st Qu.:1.400
#>  Median :5.000   Median :3.400   Median :1.500
#>  Mean   :5.006   Mean   :3.428   Mean   :1.462
#>  3rd Qu.:5.200   3rd Qu.:3.675   3rd Qu.:1.575
#>  Max.   :5.800   Max.   :4.400   Max.   :1.900
#> ------------------------------------------------------------
#> : versicolor
#>   Sepal.Length    Sepal.Width     Petal.Length
#>  Min.   :4.900   Min.   :2.000   Min.   :3.00
#>  1st Qu.:5.600   1st Qu.:2.525   1st Qu.:4.00
#>  Median :5.900   Median :2.800   Median :4.35
#>  Mean   :5.936   Mean   :2.770   Mean   :4.26
```

```
#>  3rd Qu.:6.300   3rd Qu.:3.000   3rd Qu.:4.60
#>  Max.   :7.000   Max.   :3.400   Max.   :5.10
#> ------------------------------------------------------------
#> : virginica
#>   Sepal.Length    Sepal.Width     Petal.Length
#>  Min.   :4.900   Min.   :2.200   Min.   :4.500
#>  1st Qu.:6.225   1st Qu.:2.800   1st Qu.:5.100
#>  Median :6.500   Median :3.000   Median :5.550
#>  Mean   :6.588   Mean   :2.974   Mean   :5.552
#>  3rd Qu.:6.900   3rd Qu.:3.175   3rd Qu.:5.875
#>  Max.   :7.900   Max.   :3.800   Max.   :6.900
by(iris, INDICES = list(iris$Species), FUN = summary)
#> : setosa
#>   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
#>  Min.   :4.300   Min.   :2.300   Min.   :1.000   Min.   :0.100
#>  1st Qu.:4.800   1st Qu.:3.200   1st Qu.:1.400   1st Qu.:0.200
#>  Median :5.000   Median :3.400   Median :1.500   Median :0.200
#>  Mean   :5.006   Mean   :3.428   Mean   :1.462   Mean   :0.246
#>  3rd Qu.:5.200   3rd Qu.:3.675   3rd Qu.:1.575   3rd Qu.:0.300
#>  Max.   :5.800   Max.   :4.400   Max.   :1.900   Max.   :0.600
#>        Species
#>  setosa    :50
#>  versicolor: 0
#>  virginica : 0
#>
#>
#>
#> ------------------------------------------------------------
#> : versicolor
#>   Sepal.Length    Sepal.Width     Petal.Length   Petal.Width
#>  Min.   :4.900   Min.   :2.000   Min.   :3.00   Min.   :1.000
#>  1st Qu.:5.600   1st Qu.:2.525   1st Qu.:4.00   1st Qu.:1.200
#>  Median :5.900   Median :2.800   Median :4.35   Median :1.300
#>  Mean   :5.936   Mean   :2.770   Mean   :4.26   Mean   :1.326
#>  3rd Qu.:6.300   3rd Qu.:3.000   3rd Qu.:4.60   3rd Qu.:1.500
#>  Max.   :7.000   Max.   :3.400   Max.   :5.10   Max.   :1.800
#>        Species
```

```
#>  setosa    : 0
#>  versicolor:50
#>  virginica : 0
#>
#>
#>
#> ------------------------------------------------------------
#> : virginica
#>   Sepal.Length    Sepal.Width     Petal.Length    Petal.Width
#>  Min.   :4.900   Min.   :2.200   Min.   :4.500   Min.   :1.400
#>  1st Qu.:6.225   1st Qu.:2.800   1st Qu.:5.100   1st Qu.:1.800
#>  Median :6.500   Median :3.000   Median :5.550   Median :2.000
#>  Mean   :6.588   Mean   :2.974   Mean   :5.552   Mean   :2.026
#>  3rd Qu.:6.900   3rd Qu.:3.175   3rd Qu.:5.875   3rd Qu.:2.300
#>  Max.   :7.900   Max.   :3.800   Max.   :6.900   Max.   :2.500
#>        Species
#>  setosa    : 0
#>  versicolor: 0
#>  virginica :50
#>
#>
#>
```

Group Averages Over Level Combinations of Factors 分组平均

```
str(warpbreaks)
#> 'data.frame':    54 obs. of  3 variables:
#>  $ breaks : num  26 30 54 25 70 52 51 26 67 18 ...
#>  $ wool   : Factor w/ 2 levels "A","B": 1 1 1 1 1 1 1 1 1 1 ...
#>  $ tension: Factor w/ 3 levels "L","M","H": 1 1 1 1 1 1 1 1 1 2 ...
head(warpbreaks)
#>   breaks wool tension
#> 1     26    A       L
#> 2     30    A       L
#> 3     54    A       L
#> 4     25    A       L
#> 5     70    A       L
#> 6     52    A       L
```

```r
ave(warpbreaks$breaks, warpbreaks$wool)
#>  [1] 31.03704 31.03704 31.03704 31.03704 31.03704 31.03704 31.03704
#>  [8] 31.03704 31.03704 31.03704 31.03704 31.03704 31.03704 31.03704
#> [15] 31.03704 31.03704 31.03704 31.03704 31.03704 31.03704 31.03704
#> [22] 31.03704 31.03704 31.03704 31.03704 31.03704 31.03704 25.25926
#> [29] 25.25926 25.25926 25.25926 25.25926 25.25926 25.25926 25.25926
#> [36] 25.25926 25.25926 25.25926 25.25926 25.25926 25.25926 25.25926
#> [43] 25.25926 25.25926 25.25926 25.25926 25.25926 25.25926 25.25926
#> [50] 25.25926 25.25926 25.25926 25.25926 25.25926
with(warpbreaks, ave(breaks, tension, FUN = function(x) mean(x, trim = 0.1)))
#>  [1] 35.6875 35.6875 35.6875 35.6875 35.6875 35.6875 35.6875 35.6875
#>  [9] 35.6875 26.3125 26.3125 26.3125 26.3125 26.3125 26.3125 26.3125
#> [17] 26.3125 26.3125 21.0625 21.0625 21.0625 21.0625 21.0625 21.0625
#> [25] 21.0625 21.0625 21.0625 35.6875 35.6875 35.6875 35.6875 35.6875
#> [33] 35.6875 35.6875 35.6875 35.6875 26.3125 26.3125 26.3125 26.3125
#> [41] 26.3125 26.3125 26.3125 26.3125 26.3125 21.0625 21.0625 21.0625
#> [49] 21.0625 21.0625 21.0625 21.0625 21.0625 21.0625
# 分组求和
with(warpbreaks, ave(breaks, tension, FUN = function(x) sum(x)))
#>  [1] 655 655 655 655 655 655 655 655 655 475 475 475 475 475 475 475 475
#> [18] 475 390 390 390 390 390 390 390 390 390 655 655 655 655 655 655 655
#> [35] 655 655 475 475 475 475 475 475 475 475 475 390 390 390 390 390 390
#> [52] 390 390 390
# 分组求和
with(iris, ave(Sepal.Length, Species, FUN = function(x) sum(x)))
#>   [1] 250.3 250.3 250.3 250.3 250.3 250.3 250.3 250.3 250.3 250.3 250.3
#>  [12] 250.3 250.3 250.3 250.3 250.3 250.3 250.3 250.3 250.3 250.3 250.3
#>  [23] 250.3 250.3 250.3 250.3 250.3 250.3 250.3 250.3 250.3 250.3 250.3
#>  [34] 250.3 250.3 250.3 250.3 250.3 250.3 250.3 250.3 250.3 250.3 250.3
#>  [45] 250.3 250.3 250.3 250.3 250.3 250.3 296.8 296.8 296.8 296.8 296.8
#>  [56] 296.8 296.8 296.8 296.8 296.8 296.8 296.8 296.8 296.8 296.8 296.8
#>  [67] 296.8 296.8 296.8 296.8 296.8 296.8 296.8 296.8 296.8 296.8 296.8
#>  [78] 296.8 296.8 296.8 296.8 296.8 296.8 296.8 296.8 296.8 296.8 296.8
#>  [89] 296.8 296.8 296.8 296.8 296.8 296.8 296.8 296.8 296.8 296.8 296.8
#> [100] 296.8 329.4 329.4 329.4 329.4 329.4 329.4 329.4 329.4 329.4 329.4
#> [111] 329.4 329.4 329.4 329.4 329.4 329.4 329.4 329.4 329.4 329.4 329.4
#> [122] 329.4 329.4 329.4 329.4 329.4 329.4 329.4 329.4 329.4 329.4 329.4
```

```
#> [133] 329.4 329.4 329.4 329.4 329.4 329.4 329.4 329.4 329.4 329.4 329.4
#> [144] 329.4 329.4 329.4 329.4 329.4 329.4 329.4
```

2.10　表格统计

介绍操作表格的 table, addmargins, prop.table, xtabs, margin.table, ftabe 等函数

table 多个分类变量分组计数统计

- 介绍 warpbreaks 和 airquality 纽约空气质量监测数据集二维的数据框
- UCBAdmissions 1973 年加州大学伯克利分校的院系录取数据集 3 维的列联表
- Titanic 4 维的列联表数据泰坦尼克号幸存者数据集

```
with(warpbreaks, table(wool, tension))
#>     tension
#> wool L M H
#>    A 9 9 9
#>    B 9 9 9
```

以 iris 数据集为例，table 的第一个参数是自己制造的第二个分类变量，原始分类变量是 Species

```
with(iris, table(Sepal.check = Sepal.Length > 7, Species))
#>            Species
#> Sepal.check setosa versicolor virginica
#>       FALSE     50         50        38
#>       TRUE       0          0        12
with(iris, table(Sepal.check = Sepal.Length > mean(Sepal.Length), Species))
#>            Species
#> Sepal.check setosa versicolor virginica
#>       FALSE     50         24         6
#>       TRUE       0         26        44
```

以 airquality 数据集为例，看看月份中臭氧含量比较高的几天

```
aiq.tab <- with(airquality, table(Oz.high = Ozone > 80, Month))
aiq.tab
#>        Month
#> Oz.high  5  6  7  8  9
#>   FALSE 25  9 20 19 27
#>   TRUE   1  0  6  7  2
```

对表格按行和列求和，即求表格的边际，查看总体情况

```
addmargins(aiq.tab, 1:2)
#>        Month
#> Oz.high   5   6   7   8   9 Sum
#>   FALSE  25   9  20  19  27 100
#>   TRUE    1   0   6   7   2  16
#>   Sum    26   9  26  26  29 116
```

臭氧含量超 80 的天数在每个月的占比，addmargins 的第二个参数 1 表示对列求和

```
aiq.prop <- prop.table(aiq.tab, 2)
aiq.prop
#>        Month
#> Oz.high          5          6          7          8          9
#>   FALSE 0.96153846 1.00000000 0.76923077 0.73076923 0.93103448
#>   TRUE  0.03846154 0.00000000 0.23076923 0.26923077 0.06896552
aiq.marprop <- addmargins(aiq.prop, 1)
aiq.marprop
#>        Month
#> Oz.high          5          6          7          8          9
#>   FALSE 0.96153846 1.00000000 0.76923077 0.73076923 0.93103448
#>   TRUE  0.03846154 0.00000000 0.23076923 0.26923077 0.06896552
#>   Sum   1.00000000 1.00000000 1.00000000 1.00000000 1.00000000
```

转换成百分比，将小数四舍五入转化为百分数，保留两位小数点

```
round(100 * aiq.marprop, 2)
#>        Month
#> Oz.high      5      6      7      8      9
#>   FALSE  96.15 100.00  76.92  73.08  93.10
#>   TRUE    3.85   0.00  23.08  26.92   6.90
#>   Sum   100.00 100.00 100.00 100.00 100.00
```

```
pairs(airquality, panel = panel.smooth, main = "airquality data")
```

以 UCBAdmissions 数据集为例，使用 xtabs 函数把数据组织成列联表，先查看数据的内容

```
UCBAdmissions
#> , , Dept = A
#>
#>          Gender
```

```
#> Admit      Male Female
#>   Admitted  512     89
#>   Rejected  313     19
....
UCBA2DF <- as.data.frame(UCBAdmissions)
UCBA2DF
#>       Admit Gender Dept Freq
#> 1  Admitted   Male    A  512
#> 2  Rejected   Male    A  313
#> 3  Admitted Female    A   89
#> 4  Rejected Female    A   19
#> 5  Admitted   Male    B  353
....
```

接着将 UCBA2DF 数据集转化为表格的形式

```
UCBA2DF.tab <- xtabs(Freq ~ Gender + Admit + Dept, data = UCBA2DF)
ftable(UCBA2DF.tab)
#>                 Dept   A   B   C   D   E   F
#> Gender Admit
#> Male   Admitted      512 353 120 138  53  22
#>        Rejected      313 207 205 279 138 351
#> Female Admitted       89  17 202 131  94  24
#>        Rejected       19   8 391 244 299 317
```

将录取性别和院系进行对比

```
prop.table(margin.table(UCBA2DF.tab, c(1, 3)), 1)
#>         Dept
#> Gender            A          B          C          D          E          F
#>   Male   0.30657748 0.20810108 0.12077295 0.15496098 0.07097733 0.13861018
#>   Female 0.05885559 0.01362398 0.32316076 0.20435967 0.21416894 0.18583106
```

男生倾向于申请院系 A 和 B，女生倾向于申请院系 C 到 F，院系 A 和 B 是最容易录取的。

2.11　索引访问

which 与引用 [性能比较，在区间 $[0,1]$ 上生成 10 万个服从均匀分布的随机数，随机抽取其中 $\frac{1}{4}$。

```
n <- 100000
x <- runif(n)
i <- logical(n)
i[sample(n, n / 4)] <- TRUE
microbenchmark::microbenchmark(x[i], x[which(i)], times = 1000)
```

TODO: 使用 subset 函数与 [比较

2.12　多维数组

多维数组的行列是怎么定义的 ?array 轴的概念，画个图表示数组

```
array(1:27, c(3, 3, 3))
#> , , 1
#>
#>      [,1] [,2] [,3]
#> [1,]    1    4    7
#> [2,]    2    5    8
#> [3,]    3    6    9
#>
#> , , 2
#>
#>      [,1] [,2] [,3]
#> [1,]   10   13   16
#> [2,]   11   14   17
#> [3,]   12   15   18
#>
#> , , 3
#>
#>      [,1] [,2] [,3]
#> [1,]   19   22   25
#> [2,]   20   23   26
#> [3,]   21   24   27
```

垂直于 Z 轴的平面去截三维立方体，3 代表 z 轴，得到三个截面（二维矩阵）

```
asplit(array(1:27, c(3, 3, 3)), 3)
#> [[1]]
#>      [,1] [,2] [,3]
```

```
#> [1,]    1    4    7
#> [2,]    2    5    8
#> [3,]    3    6    9
#>
#> [[2]]
#>      [,1] [,2] [,3]
#> [1,]   10   13   16
#> [2,]   11   14   17
#> [3,]   12   15   18
#>
#> [[3]]
#>      [,1] [,2] [,3]
#> [1,]   19   22   25
#> [2,]   20   23   26
#> [3,]   21   24   27
```

对每个二维矩阵按列求和

```
lapply(asplit(array(1:27, c(3, 3, 3)), 3), apply, 2, sum)
#> [[1]]
#> [1]  6 15 24
#>
#> [[2]]
#> [1] 33 42 51
#>
#> [[3]]
#> [1] 60 69 78
```

`asplit` 和 `lapply` 组合处理多维数组的计算问题，多维数组[2]

三维数组的矩阵运算 abind 包提供更多的数组操作，如合并，替换

2.13　其它操作

成对的数据操作有 `list` 与 `unlist`、`stack` 与 `unstack`、`class` 与 `unclass`、`attach` 与 `detach` 以及 `with` 和 `within`，它们在数据操作过程中有时会起到一定的补充作用。

[2] https://www.brodieg.com/2018/11/23/is-your-matrix-running-slow-try-lists/

2.13.1 列表属性

```
# 创建列表
list(...)
pairlist(...)
# 转化列表
as.list(x, ...)
## S3 method for class 'environment'
as.list(x, all.names = FALSE, sorted = FALSE, ...)
as.pairlist(x)
# 检查列表
is.list(x)
is.pairlist(x)

alist(...)
```

list 函数用来构造、转化和检查 R 列表对象。下面创建一个临时列表对象 tmp ，它包含两个元素 A 和 B，两个元素都是向量，前者是数值型，后者是字符型

```
(tmp <- list(A = c(1, 2, 3), B = c("a", "b")))
#> $A
#> [1] 1 2 3
#>
#> $B
#> [1] "a" "b"
```

```
unlist(x, recursive = TRUE, use.names = TRUE)
```

unlist 函数将给定的列表对象 x 简化为原子向量 (atomic vector)，我们发现简化之后变成一个字符型向量

```
unlist(tmp)
#>  A1  A2  A3  B1  B2
#> "1" "2" "3" "a" "b"
unlist(tmp, use.names = FALSE)
#> [1] "1" "2" "3" "a" "b"
```

unlist 的逆操作是 relist

2.13.2 堆叠向量

```
stack(x, ...)
## Default S3 method:
stack(x, drop = FALSE, ...)
## S3 method for class 'data.frame'
stack(x, select, drop = FALSE, ...)

unstack(x, ...)
## Default S3 method:
unstack(x, form, ...)
## S3 method for class 'data.frame'
unstack(x, form, ...)
```

stack 与 unstack 将多个向量堆在一起组成一个向量

```
# 查看数据集 PlantGrowth
class(PlantGrowth)
#> [1] "data.frame"
head(PlantGrowth)
#>   weight group
#> 1   4.17  ctrl
#> 2   5.58  ctrl
#> 3   5.18  ctrl
#> 4   6.11  ctrl
#> 5   4.50  ctrl
#> 6   4.61  ctrl
# 检查默认的公式
formula(PlantGrowth)
#> weight ~ group
# 根据公式解除堆叠
# 下面等价于 unstack(PlantGrowth, form = weight ~ group)
(pg <- unstack(PlantGrowth))
#>    ctrl trt1 trt2
#> 1  4.17 4.81 6.31
#> 2  5.58 4.17 5.12
#> 3  5.18 4.41 5.54
#> 4  6.11 3.59 5.50
#> 5  4.50 5.87 5.37
```

```
#> 6  4.61 3.83 5.29
#> 7  5.17 6.03 4.92
#> 8  4.53 4.89 6.15
#> 9  5.33 4.32 5.80
#> 10 5.14 4.69 5.26
```

现在再将变量 pg 堆叠起来，还可以指定要堆叠的列

```
stack(pg)
#>    values  ind
#> 1    4.17 ctrl
#> 2    5.58 ctrl
#> 3    5.18 ctrl
#> 4    6.11 ctrl
#> 5    4.50 ctrl
....
stack(pg, select = -ctrl)
#>    values  ind
#> 1    4.81 trt1
#> 2    4.17 trt1
#> 3    4.41 trt1
#> 4    3.59 trt1
#> 5    5.87 trt1
....
```

形式上和 reshape 有一些相似之处，数据框可以由长变宽或由宽变长

2.13.3 属性转化

```
class(x)
class(x) <- value
unclass(x)
inherits(x, what, which = FALSE)

oldClass(x)
oldClass(x) <- value
```

class 和 unclass 函数用来查看、设置类属性和取消类属性，常用于面向对象的编程设计中

```
class(iris)
#> [1] "data.frame"
class(iris$Species)
#> [1] "factor"
unclass(iris$Species)
#>   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
#>  [36] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
#>  [71] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3
#> [106] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
#> [141] 3 3 3 3 3 3 3 3 3 3
#> attr(,"levels")
....
```

2.13.4 绑定环境

```
attach(what,
  pos = 2L, name = deparse(substitute(what), backtick = FALSE),
  warn.conflicts = TRUE
)
detach(name,
  pos = 2L, unload = FALSE, character.only = FALSE,
  force = FALSE
)
```

attach 和 detach 是否绑定数据框的列名，不推荐操作，推荐使用 with

```
attach(iris)
head(Species)
#> [1] setosa setosa setosa setosa setosa setosa
#> Levels: setosa versicolor virginica
detach(iris)
```

2.13.5 数据环境

```
with(data, expr, ...)
within(data, expr, ...)
## S3 method for class 'list'
```

```
within(data, expr, keepAttrs = TRUE, ...)
```

data 参数 data 用来构造表达式计算的环境。其默认值可以是一个环境，列表，数据框。在 within 函数中 data 参数只能是列表或数据框。

expr 参数 expr 包含要计算的表达式。在 within 中通常包含一个复合表达式，比如

```
{
  a <- somefun()
  b <- otherfun()
  ...
  rm(unused1, temp)
}
```

with 和 within 计算一组表达式，计算的环境是由数据构造的，后者可以修改原始的数据

```
with(mtcars, mpg[cyl == 8 & disp > 350])
#> [1] 18.7 14.3 10.4 10.4 14.7 19.2 15.8
```

和下面计算的结果一样，但是更加简洁漂亮

```
mtcars$mpg[mtcars$cyl == 8 & mtcars$disp > 350]
#> [1] 18.7 14.3 10.4 10.4 14.7 19.2 15.8
```

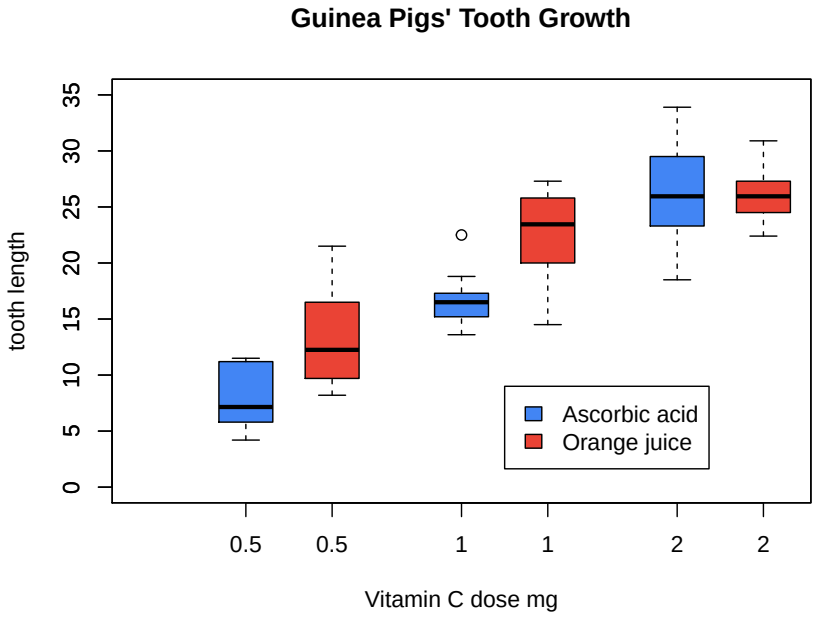
within 函数可以修改原数据环境中的多个变量，比如删除、修改和添加等

```
# 原数据集 airquality
head(airquality)
#>   Ozone Solar.R Wind Temp Month Day
#> 1    41     190  7.4   67     5   1
#> 2    36     118  8.0   72     5   2
#> 3    12     149 12.6   74     5   3
#> 4    18     313 11.5   62     5   4
#> 5    NA      NA 14.3   56     5   5
#> 6    28      NA 14.9   66     5   6
aq <- within(airquality, {
  lOzone <- log(Ozone) # 取对数
  Month <- factor(month.abb[Month]) # 字符串型转因子型
  cTemp <- round((Temp - 32) * 5 / 9, 1) # 从华氏温度到摄氏温度转化
  S.cT <- Solar.R / cTemp # 使用新创建的变量
  rm(Day, Temp)
})
# 修改后的数据集
```

```
head(aq)
#>   Ozone Solar.R Wind Month      S.cT cTemp   lOzone
#> 1    41     190  7.4   May  9.793814  19.4 3.713572
#> 2    36     118  8.0   May  5.315315  22.2 3.583519
#> 3    12     149 12.6   May  6.394850  23.3 2.484907
#> 4    18     313 11.5   May 18.742515  16.7 2.890372
#> 5    NA      NA 14.3   May        NA  13.3       NA
#> 6    28      NA 14.9   May        NA  18.9 3.332205
```

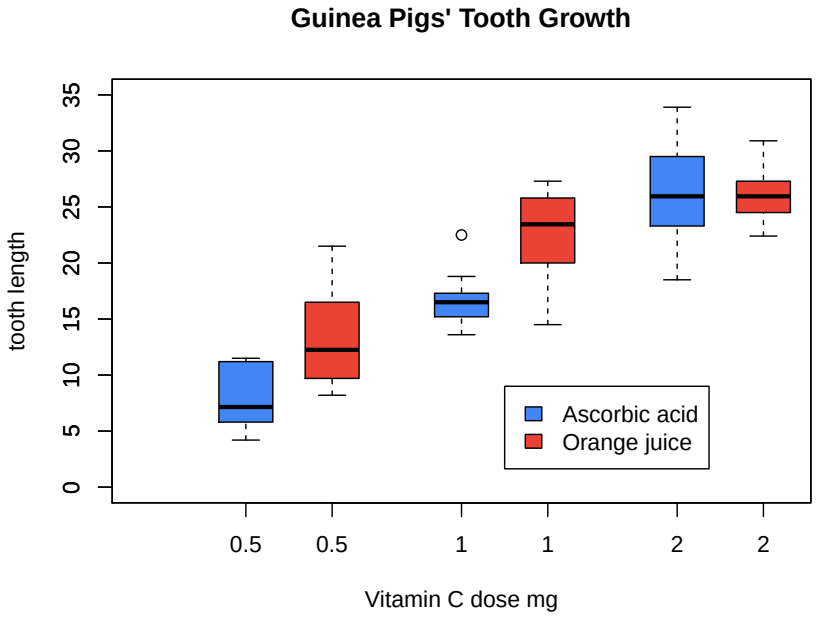
下面再举一个复杂的绘图例子，这个例子来自 boxplot 函数

```
with(ToothGrowth, {
  boxplot(len ~ dose,
    boxwex = 0.25, at = 1:3 - 0.2,
    subset = (supp == "VC"), col = "#4285f4",
    main = "Guinea Pigs' Tooth Growth",
    xlab = "Vitamin C dose mg",
    ylab = "tooth length", ylim = c(0, 35)
  )
  boxplot(len ~ dose,
    add = TRUE, boxwex = 0.25, at = 1:3 + 0.2,
    subset = supp == "OJ", col = "#EA4335"
  )
  legend(2, 9, c("Ascorbic acid", "Orange juice"),
    fill = c("#4285f4", "#EA4335")
  )
})
```



将 boxplot 函数的 subset 参数单独提出来，调用数据子集选择函数 subset，这里 with 中只包含一个表达式，所以也可以不用大括号

```
with(
  subset(ToothGrowth, supp == "VC"),
  boxplot(len ~ dose,
    boxwex = 0.25, at = 1:3 - 0.2,
    col = "#4285f4", main = "Guinea Pigs' Tooth Growth",
    xlab = "Vitamin C dose mg",
    ylab = "tooth length", ylim = c(0, 35)
  )
)
with(
  subset(ToothGrowth, supp == "OJ"),
  boxplot(len ~ dose,
    add = TRUE, boxwex = 0.25, at = 1:3 + 0.2,
    col = "#EA4335"
  )
)
legend(2, 9, c("Ascorbic acid", "Orange juice"),
  fill = c("#4285f4", "#EA4335")
)
```



2.14　运行环境

```
xfun::session_info()
#> R version 3.6.1 (2019-07-05)
#> Platform: x86_64-pc-linux-gnu (64-bit)
#> Running under: Debian GNU/Linux 10 (buster)
#>
#> Locale:
#>   LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
#>   LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
#>   LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
#>   LC_PAPER=en_US.UTF-8       LC_NAME=C
#>   LC_ADDRESS=C               LC_TELEPHONE=C
#>   LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
#>
#> Package version:
#>   base64enc_0.1.3  bookdown_0.12    codetools_0.2-16 compiler_3.6.1
#>   curl_4.0         digest_0.6.20    evaluate_0.14    glue_1.3.1
#>   graphics_3.6.1   grDevices_3.6.1  highr_0.8        htmltools_0.3.6
#>   jsonlite_1.6     knitr_1.23       magrittr_1.5     markdown_1.0
```

```
#>   methods_3.6.1    mime_0.7         Rcpp_1.0.2       rmarkdown_1.14
#>   stats_3.6.1      stringi_1.4.3    stringr_1.4.0    tinytex_0.14
#>   tools_3.6.1      utils_3.6.1      xfun_0.8         yaml_2.2.0
```

第三章　净土化操作

> 常用操作和高频问题需要合并进之前的 data-manipulation，本章只介绍向量化计算以 dplyr 为核心的 tidyverse 风数据操作管道风操作

在不同规模的数据集上，Base R，dplyr 和 data.table 的处理性能应该属于低、中、高档搭配的情形

更加高级的数据变形操作，特别是数据类型的一致性，方便后续的可视化和建模，引入 tidyverse，数据处理或者叫特征工程 Base R vs data.table vs dplyr 它们各有优点，所以都加以介绍参考 Jozef Hajnala 博文。

关于 tidyverse 提供的数据操作不要移动到 Base R 对应的章节，这二者已经越行越远，本章主要讲并行或分布式数据操作工具，如 SparkR sparklyr 针对大数据集上的操

data.table 诞生于 2006 年 4 月 15 日（以在 CRAN 上发布的第一个版本时间为准），是基于 `data.frame` 的扩展和 Base R 的数据操作连贯一些，dplyr 诞生于 2014 年 1 月 29 日，号称数据操作的语法，其实二者套路一致，都是借用 SQL 语言的设计，实现方式不同罢了，前者主要依靠 C 语言完成底层数据操作，总代码量 1.29M，C 占 65.6%，后者主要依靠 C++ 语言完成底层数据操作，总代码量 1.2M，C++ 占 34.4%，上层的高级操作接口都是 R 语言。像这样的大神在写代码，码力应该差不多，编程语言会对数据操作的性能有比较大的影响，我想这也是为什么在很多场合下 data.table 霸榜！

关于 data.table 和 dplyr 的对比，参看爆栈网的帖子 https://stackoverflow.com/questions/21435339

Base R 的数据操作的一致性问题参见统计之都帖子 https://d.cosx.org/d/420763

Malcolm Barrett 以幻灯片的形式呈现 dplyr 和 purrr 的基础用法

Charlotte Wickham 的课程 A introduction to purrr purrr-tutorial

关于引用 quotation

相比于 SQL，dplyr 在数据库操作的不足，这是一些比较难的部分 https://dbi.r-dbi.org/articles/dbi-1#sec:open-issues

函数式编程 Functional Programming Languages 用于数据处理

- rpivotTable 动态数据透视表
- fuzzyjoin Join tables together on inexact matching
- dtplyr dtplyr is the data.table backend for dplyr. It provides S3 methods for data.table objects so that dplyr works the way you expect.
- bplyr basic dplyr and tidyr functionality without the tidyverse dependencies
- SqlRender 基于 Java 语言，借助 rJava 包支持参数化的 SQL 语句，并且可以将一种 SQL 语句 (如 Microsoft SQL Server) 转化为多种 SQL 语句 (如 Oracle, PostgreSQL, Amazon RedShift, Impala, IBM Netezza, Google BigQuery, Microsoft PDW, and SQLite)
- fastmap 实现键值存储，提供新的数据结构
- Roaring bitmaps Bitsets, also called bitmaps, are commonly used as fast data structures.

```
library(tidyverse)
```

数据操作的语法

第一代

1. Base R 数据操作已在第 二 章详细介绍

第二代

1. reshape （退休）使用函数 `melt` 和 `cast` 重构 (restructure) 和聚合 (aggregate) 数据
2. reshape2 （退休）是 reshape 的继任者，功能和 reshape 类似，提供两个函数 `melt` 和 `cast` 聚合数据，因此不再介绍 reshape，而鉴于 reshape2 还在活跃使用中，故而以它为例介绍 `melt` 和 `cast` 函数
3. plyr （退休）统一拆分 (split)，计算 (apply)，合并 (combine) 的数据处理流，由 dplyr（用于 data.frame）和 purrr （用于 list）继任

第三代

1. dplyr 操作数据的语法及其扩展
2. sparklyr 给 dplyr 提供 Spark 接口支持
3. dbplyr 给 dplyr 提供 DBI 数据库接口支持
4. dtplyr 给 dplyr 提供 data.table 支持
5. tidyr 提供 `spread` 和 `gather` 两个函数清洗数据

Garrett Grolemund 在 RStudio 主要从事教育教学，参考 Materials for the Tidyverse Train-the-trainer workshop 和 The Tidyverse Cookbook

Dirk Eddelbuettel 的 Getting Started in R – Tinyverse Edition

3.1 常用操作

dplyr 由 Hadley Wickham 主要由开发和维护，是 Rstudio 公司开源的用于数据处理的一大利器，该包号称 ‘‘数据操作的语法’’，与 ggplot2 对应，也就是说数据处理那一套已经建立完整的和 SQL 一样的功能。它们都遵循同样的处理逻辑，只不过一个用 SQL 写，一个用 R 语言写，处理效率差不多，R 语言写的 SQL 会被翻译为 SQL 语句，再传至数据库查询，当然它也支持内存内的数据操作。目前 dplyr 以 dbplyr 为后端支持的数据库有：MySQL、PostgreSQL，SQLite 等，完整的支持列表请看 这里，连接特定数据库，都是基于 DBI，DBI 即 Database Interface，是使用 C/C++ 开发的底层数据库接口，是一个统一的关系型数据库连接框架，需要根据不同的具体的数据库进行实例化，才可使用。

dplyr 常用的函数是 6 个：`arrange` 排序 `filter` 过滤 `select` 选择 `mutate` 变换 `summarise` 汇总 `group_by` 分组

以 GGplot2 自带的钻石数据集 diamonds 为例介绍

3.1.1 查看

除了直接打印数据集的前几行，tibble 包还提供 glimpse 函数查看数据集，而 Base R 默认查看方式是调用 str 函数

```
diamonds
#> # A tibble: 53,940 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 0.290 Premium   I     VS2      62.4    58   334  4.2   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
#> # ... with 5.393e+04 more rows
glimpse(diamonds)
#> Observations: 53,940
#> Variables: 10
#> $ carat   <dbl> 0.23, 0.21, 0.23, 0.29, 0.31, 0.24, 0.24, 0.26, 0.22, ...
#> $ cut     <ord> Ideal, Premium, Good, Premium, Good, Very Good, Very G...
#> $ color   <ord> E, E, E, I, J, J, I, H, E, H, J, J, F, J, E, E, I, J, ...
#> $ clarity <ord> SI2, SI1, VS1, VS2, SI2, VVS2, VVS1, SI1, VS2, VS1, SI...
```

```
#> $ depth   <dbl> 61.5, 59.8, 56.9, 62.4, 63.3, 62.8, 62.3, 61.9, 65.1, ...
#> $ table   <dbl> 55, 61, 65, 58, 58, 57, 57, 55, 61, 61, 55, 56, 61, 54...
#> $ price   <int> 326, 326, 327, 334, 335, 336, 336, 337, 337, 338, 339,...
#> $ x       <dbl> 3.95, 3.89, 4.05, 4.20, 4.34, 3.94, 3.95, 4.07, 3.87, ...
#> $ y       <dbl> 3.98, 3.84, 4.07, 4.23, 4.35, 3.96, 3.98, 4.11, 3.78, ...
#> $ z       <dbl> 2.43, 2.31, 2.31, 2.63, 2.75, 2.48, 2.47, 2.53, 2.49, ...
```

表 3.1: dplyr 定义的数据对象类型

类型	含义
int	整型 integer
dbl	（单）双精度浮点类型
chr	字符（串）类型
dttm	data-time 类型
lgl	布尔类型
fctr	因子类型 factor
date	日期类型

表 3.1 中 dttm 和 date 类型代指 lubridate 包指定的日期对象 POSIXct、POSIXlt、Date、chron、yearmon、yearqtr、zoo、zooreg、timeDate、xts、its、ti、jul、timeSeries 和 fts。

3.1.2 筛选

3.1.3 排序

3.1.4 聚合

3.1.5 合并

3.1.6 重塑

3.1.7 变换

3.1.8 去重

数据去重在 dplyr 中的实现[1]。

[1] https://stackoverflow.com/questions/22959635/

```r
set.seed(123)
df <- data.frame(
  x = sample(0:1, 10, replace = T),
  y = sample(0:1, 10, replace = T),
  z = 1:10
)
df
#>    x y  z
#> 1  0 1  1
#> 2  0 1  2
#> 3  0 1  3
#> 4  1 0  4
#> 5  0 1  5
#> 6  1 0  6
#> 7  1 1  7
#> 8  1 0  8
#> 9  0 0  9
#> 10 0 0 10
df %>%
  group_by(x, y) %>%
  filter(row_number(z) == 1)
#> # A tibble: 4 x 3
#> # Groups:   x, y [4]
#>       x     y     z
#>   <int> <int> <int>
#> 1     0     1     1
#> 2     1     0     4
#> 3     1     1     7
#> 4     0     0     9
```

3.2 高频问题

常用的数据操作包含

1. 创建空的数据框或者说初始化一个数据框，
2. 按指定的列对数据框排序，
3. 选择特定的一些列，复杂情况是可能需要正则表达式从列名或者值中筛选

4. 合并两个数据框，分为 (inner outer left right) 四种情况

3.2.1 初始化数据框

创建空的数据框，就是不包含任何行、记录 [^create-an-empty-data-frame]

```
empty_df <- data.frame(
  Doubles = double(),
  Ints = integer(),
  Factors = factor(),
  Logicals = logical(),
  Characters = character(),
  stringsAsFactors = FALSE
)
str(empty_df)
#> 'data.frame':    0 obs. of  5 variables:
#>  $ Doubles   : num
#>  $ Ints      : int
#>  $ Factors   : Factor w/ 0 levels:
#>  $ Logicals  : logi
#>  $ Characters: chr
```

如果数据框 df 包含数据，现在要依据它创建一个空的数据框

```
empty_df = df[FALSE,]
```

还可以使用 structure 构造一个数据框，并且我们发现它的效率更高

```
s <- function() structure(list(
    Date = as.Date(character()),
    File = character(),
    User = character()
  ),
  class = "data.frame"
  )
d <- function() data.frame(
    Date = as.Date(character()),
    File = character(),
    User = character(),
    stringsAsFactors = FALSE
```

```
  )
# microbenchmark::microbenchmark(s(), d())
```

3.2.2 移除缺失记录

只要行中包含缺失值，我们就把这样的行移除出去

```
airquality[complete.cases(airquality), ]
#>     Ozone Solar.R Wind Temp Month Day
#> 1      41     190  7.4   67     5   1
#> 2      36     118  8.0   72     5   2
#> 3      12     149 12.6   74     5   3
#> 4      18     313 11.5   62     5   4
#> 7      23     299  8.6   65     5   7
#> 8      19      99 13.8   59     5   8
#> 9       8      19 20.1   61     5   9
#> 12     16     256  9.7   69     5  12
#> 13     11     290  9.2   66     5  13
....
```

3.2.3 数据类型转化

```
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 ...
bob <- PlantGrowth
i <- sapply(bob, is.factor)
bob[i] <- lapply(bob[i], as.character)
str(bob)
#> '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 : chr  "ctrl" "ctrl" "ctrl" "ctrl" ...
```

3.2.4 跨列分组求和

输入是一个数据框 data.frame，按照其中某一变量分组，然后计算任意数量的变量的行和和列和。

空气质量数据集 airquality 按月份 Month 分组，然后求取满足条件的列的和

```
Reduce(rbind, lapply(unique(airquality$Month), function(gv) {
  subdta <- subset(airquality, subset = Month == gv)
  data.frame(
    Colsum = as.numeric(
      colSums(subdta[, grepl("[mM]", names(airquality))], na.rm = TRUE)
    ),
    Month = gv
  )
}))
#>    Colsum Month
#> 1    2032     5
#> 2     155     5
#> 3    2373     6
#> 4     180     6
#> 5    2601     7
#> 6     217     7
#> 7    2603     8
#> 8     248     8
#> 9    2307     9
#> 10    270     9
```

什么是函数式编程，R 语言环境下的函数式编程是如何操作的

3.3 管道操作

Stefan Milton Bache 开发了 magrittr 包实现管道操作，增加代码的可读性和维护性，但是这个 R 包的名字取的太奇葩，因为 记不住，它其实是一个复杂的法语发音，中式英语就叫它马格里特吧！这下应该好记多了吧！

我要查看是否需要新添加一个 R 包依赖，假设该 R 包是 reticulate 没有出现在 DESCRIPTION 文件中，但是可能已经被其中某（个）些 R 包依赖了

```
"reticulate" %in% sort(unique(unlist(tools::package_dependencies(desc::desc_get_deps()$package, recu
#> [1] FALSE
```

安装 pkg 的依赖

```
pkg <- c(
  "bookdown",
  "e1071",
  "formatR",
  "lme4",
  "mvtnorm",
  "prettydoc", "psych",
  "reticulate", "rstan", "rstanarm", "rticles",
  "svglite",
  "TMB", "glmmTMB"
)
# 获取 pkg 的所有依赖
dep_pkg <- tools::package_dependencies(pkg, recursive = TRUE)
# 将列表 list 合并为向量 vector
merge_pkg <- Reduce("c", dep_pkg, accumulate = FALSE)
# 所有未安装的 R 包
miss_pkg <- setdiff(unique(merge_pkg), unique(.packages(TRUE)))
# 除了 pkg 外，未安装的 R 包，安装 pkg 的依赖
sort(setdiff(miss_pkg, pkg))
#> [1] "foreign" "mnormt"
```

转化为管道操作，增加可读性，[^pipe-r]

再举一个关于数据模拟的例子

模拟 0-1 序列，

```
set.seed(2019)
binom_sample <- function(n) {
  sum(sample(x = c(0,1), size = n, prob = c(0.8, 0.2), replace = TRUE))/n
}
# 频率估计概率
one_prob <- sapply(10^(seq(8)), binom_sample)
# 估计的误差
one_abs <- abs(one_prob - 0.2)
one_abs
```

```
#> [1] 1.000e-01 1.000e-02 1.100e-02 4.400e-03 1.460e-03 3.980e-04 4.700e-06
#> [8] 9.552e-05
```

似然估计

3.4 运行环境

```
xfun::session_info('tidyverse')
#> R version 3.6.1 (2019-07-05)
#> Platform: x86_64-pc-linux-gnu (64-bit)
#> Running under: Debian GNU/Linux 10 (buster)
#>
#> Locale:
#>   LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
#>   LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
#>   LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
#>   LC_PAPER=en_US.UTF-8       LC_NAME=C
#>   LC_ADDRESS=C               LC_TELEPHONE=C
#>   LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
#>
#> Package version:
#>   askpass_1.1        assertthat_0.2.1   backports_1.1.4
#>   base64enc_0.1.3    BH_1.69.0.1        broom_0.5.2
#>   callr_3.3.1        cellranger_1.1.0   cli_1.1.0
#>   clipr_0.7.0        colorspace_1.4.1   crayon_1.3.4
#>   curl_4.0           DBI_1.0.0          dbplyr_1.4.2
#>   digest_0.6.20      dplyr_0.8.3        ellipsis_0.2.0.1
#>   evaluate_0.14      fansi_0.4.0        forcats_0.4.0
#>   fs_1.3.1           generics_0.0.2     ggplot2_3.2.0
#>   glue_1.3.1         graphics_3.6.1     grDevices_3.6.1
#>   grid_3.6.1         gtable_0.3.0       haven_2.1.1
#>   highr_0.8          hms_0.5.0          htmltools_0.3.6
#>   httr_1.4.0         jsonlite_1.6       knitr_1.23
#>   labeling_0.3       lattice_0.20.38    lazyeval_0.2.2
#>   lubridate_1.7.4    magrittr_1.5       markdown_1.0
#>   MASS_7.3.51.4      Matrix_1.2.17      methods_3.6.1
#>   mgcv_1.8.28        mime_0.7           modelr_0.1.4
```

```
#>   munsell_0.5.0      nlme_3.1.140       openssl_1.4.1
#>   pillar_1.4.2       pkgconfig_2.0.2    plogr_0.2.0
#>   plyr_1.8.4         prettyunits_1.0.2  processx_3.4.1
#>   progress_1.2.2     ps_1.3.0           purrr_0.3.2
#>   R6_2.4.0           RColorBrewer_1.1.2 Rcpp_1.0.2
#>   readr_1.3.1        readxl_1.3.1       rematch_1.0.1
#>   reprex_0.3.0       reshape2_1.4.3     rlang_0.4.0
#>   rmarkdown_1.14     rstudioapi_0.10    rvest_0.3.4
#>   scales_1.0.0       selectr_0.4.1      splines_3.6.1
#>   stats_3.6.1        stringi_1.4.3      stringr_1.4.0
#>   sys_3.2            tibble_2.1.3       tidyr_0.8.3
#>   tidyselect_0.2.5   tidyverse_1.2.1    tinytex_0.14
#>   tools_3.6.1        utf8_1.1.4         utils_3.6.1
#>   vctrs_0.2.0        viridisLite_0.3.0  whisker_0.3.2
#>   withr_2.1.2        xfun_0.8           xml2_1.2.0
#>   yaml_2.2.0         zeallot_0.1.0
```

参考文献

Spector, P. (2008). *Data Manipulation With R*. Springer, New York, NY.

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.