R 语言编程：基于 tidyverse

第 09 讲　数据连接

张敬信

2022 年 12 月 1 日

哈尔滨商业大学

一. 连接数据库

- R 操作数据是先将数据载入内存，当数据超过内存限制时，一种解决办法是，将大数据存放在远程数据库（远程服务器或本地硬盘），然后建立与 R 的连接，再从 R 中执行查询、探索、建模等。
- dplyr 是 tidyverse 操作数据的最核心包，而 dbplyr 包是用于数据库的 dplyr 后端，能够操作远程数据库中的数据表，就像它们是内存中的数据框一样。
- DBI 包提供了通用的接口，使得能够使用相同的代码与许多不同的数据库连用。

- 常见的主流数据库软件：SQL Server，MySQL，Oracle 等都能支持，但还需要为其安装特定的驱动，比如
  - RMariaDB 包：连接到 MySQL 和 MariaDB
  - RPostgres 包：连接到 Postgres 和 Redshift
  - RSQLite 包：嵌入 SQLite 数据库[1]
  - odbc 包：通过开放数据库连接协议连接到许多商业数据库
  - bigrquery 包：连接到谷歌的 BigQuery

[1] SQLlite 已经嵌入到 R 包中，是不需要额外安装数据库软件就能直接用的轻量级数据库.

案例：R 连接 MySQL 数据库

(1) 配置 MySQL 开发环境（略，可参阅知乎八咫镜：mysql 安装及配置）

(2) 新建 MySQL 连接和数据库

- 在 Navicat 新建 MySQL 连接，输入连接名（随便起名）和配置 MySQL 时设好的用户名及相应密码：

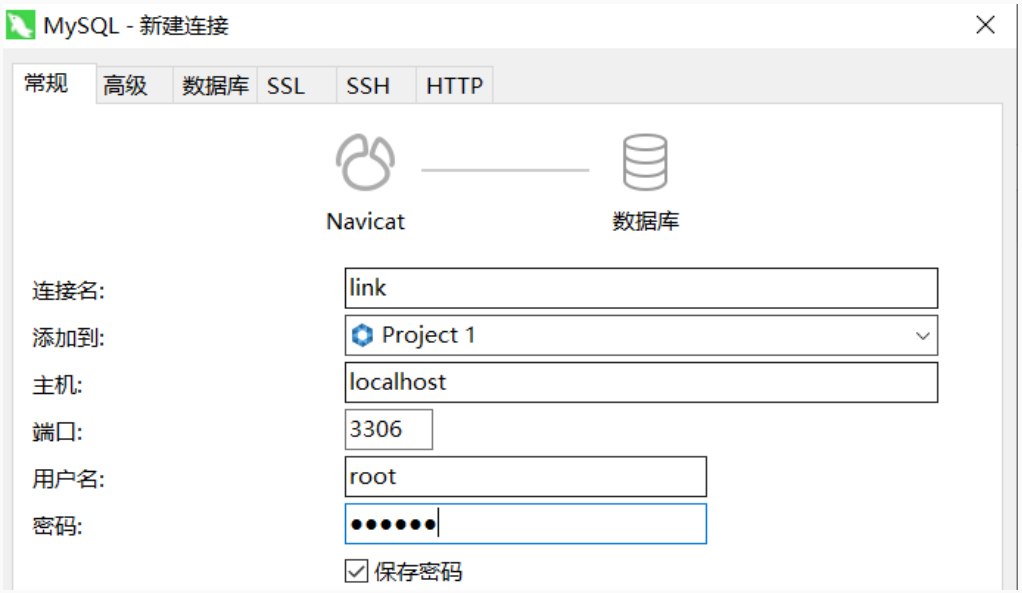

图 1: 在 Navicat 中新建 MySQL 连接

- 打开该连接，右键新建数据库，取名为 mydb，选择字符编码和排序规则：

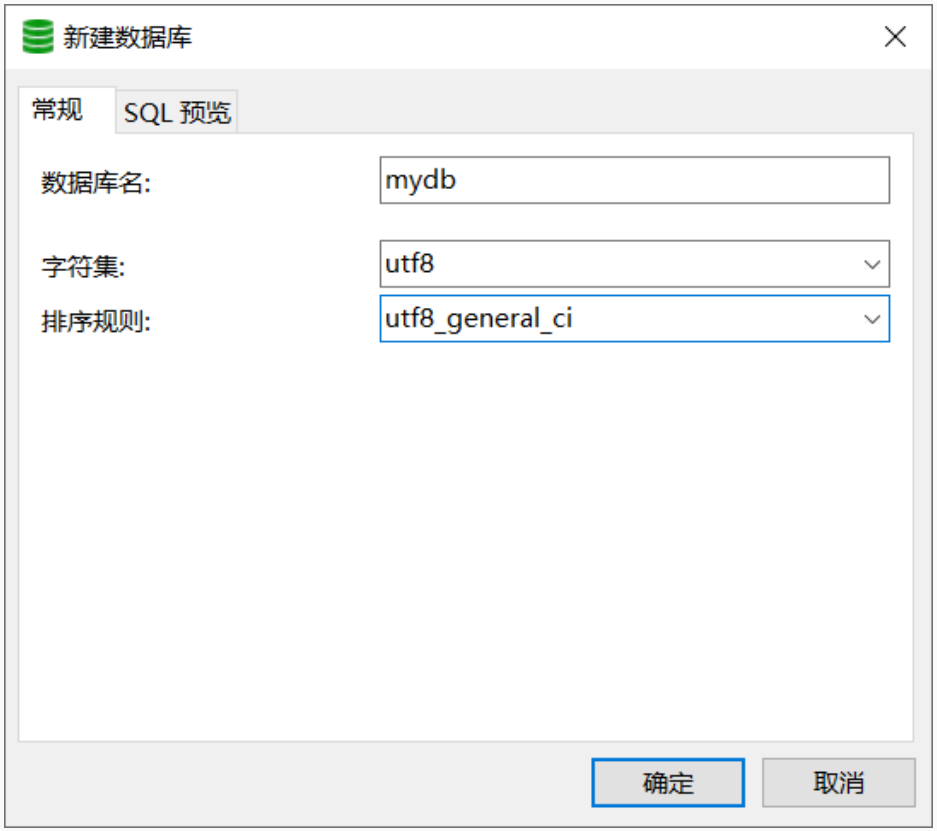

图 2: 新建数据库

(3) 建立 R 与 MySQL 的连接

- 先加载 RMariaDB 包，再用 dbConnect() 来建立连接，需要提供数据库后端、用户名、密码、数据库名、主机：

```
library(RMariaDB)
con = dbConnect(MariaDB(),
                user = "root",
                password = "123456",
                dbname = "mydb",
                host = "localhost")
dbListTables(con)      # 查看 con 连接下的数据表
#> [1] "exam"
```

注：character(0) 表明该连接下还没有数据表。

(4) 创建数据表

- 该连接下，若已有 MySQL 数据表则直接进入下一步，否则有两种方法创建数据表：
  - 在 MySQL 端，从 Navicat 创建表，可从外部数据文件导入到数据表
  - 在 R 端，读取数据，再通过函数 dbWriteTable() 写入到数据表；若是大数据，可以借助循环逐块地读取和追加写入。

```
library(readxl)
data = read_xlsx("data/ExamDatas.xlsx")
dbWriteTable(con, name = "exam", value = data,
             overwrite = TRUE)
dbListTables(con)
#> [1] "exam"
```

(5) 数据表引用

- 用函数 tbl() 获取数据表的引用，引用是一种浅拷贝机制，能够不做物理拷贝而使用数据，一般处理大数据都采用该策略。

```
df = tbl(con, "exam")
df
#> # Source:   table<exam> [?? x 8]
#> # Database: mysql  [root@localhost:NA/mydb]
#>   class name   sex   chinese  math english moral scienc
#>   <chr> <chr>  <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl>
#> 1 六 1 班 何娜   女         87    92      79     9
#> 2 六 1 班 黄才菊 女         95    77      75     8
#> 3 六 1 班 陈芳妹 女         79    87      66     9
#> # ... with more rows
```

- 输出数据表引用，看起来和 tibble 几乎一样，主要区别就是它是来自远程 MySQL 数据库。

(6) 数据表查询

- 与数据库交互，通常是用 SQL（结构化查询语言），几乎所有的数据库都在使用 SQL.
- dbplyr 包让 R 用户用 dplyr 语法就能执行 SQL 查询，就像用在 R 中操作数据框一样：

```
df %>%
  group_by(sex) %>%
  summarise(avg = mean(math, na.rm = TRUE))
#> # Source:   SQL [2 x 2]
#> # Database: mysql  [root@localhost:NA/mydb]
#>   sex     avg
#>   <chr> <dbl>
#> 1 女     69.1
#> 2 男     65.2
```

普通数据框与远程数据库查询之间最重要的区别：

- R 代码被翻译成 SQL 并在远程服务器上的数据库中执行，而不是在本地机器上的 R 中执行。当与数据库一起工作时，dplyr 试图尽可能地懒惰：
  - 除非明确要求（接 collect()），否则它不会把数据拉到 R 中
  - 它把任何工作都尽可能地推迟到最后一刻：把想做的所有事情合在一起，然后一步送到数据库中
- dbplyr 包还提供了将 dplyr 代码翻译成 SQL 查询代码的函数 show_query()。可以进一步用于 MySQL，或 dbSendQuery()，dbGetQuery()：

```r
df %>%
  group_by(sex) %>%
  summarise(avg = mean(math, na.rm = TRUE)) %>%
  show_query()
#> <SQL>
#> SELECT `sex`, AVG(`math`) AS `avg`
#> FROM `exam`
#> GROUP BY `sex`
```

```
dbGetQuery(con, "SELECT `sex`, AVG(`math`) AS `avg`
                 FROM `exam`
                 GROUP BY `sex`")
#>   sex  avg
#> 1  女 69.1
#> 2  男 65.2
```

- 最后，关闭 R 与 MySQL 的连接

```
dbDisconnect(con)
```

二. 关系数据库

- 数据分析经常会涉及相互关联的多个数据表，称为关系数据库。关系数据库通用语言是 SQL（结构化查询语言），dplyr 包提供了一系列类似 SQL 语法的函数，可以很方便地操作关系数据库。
- 关系是指两个数据表之间的关系，更多数据表之间的关系总可以表示为两两之间的关系。
- 一个项目的数据，通常都是用若干数据表分别存放，它们之间通过“键”连接在一起，根据数据分析的需要，通过键匹配进行数据连接。

- 以纽约机场航班数据的关系结构为例：

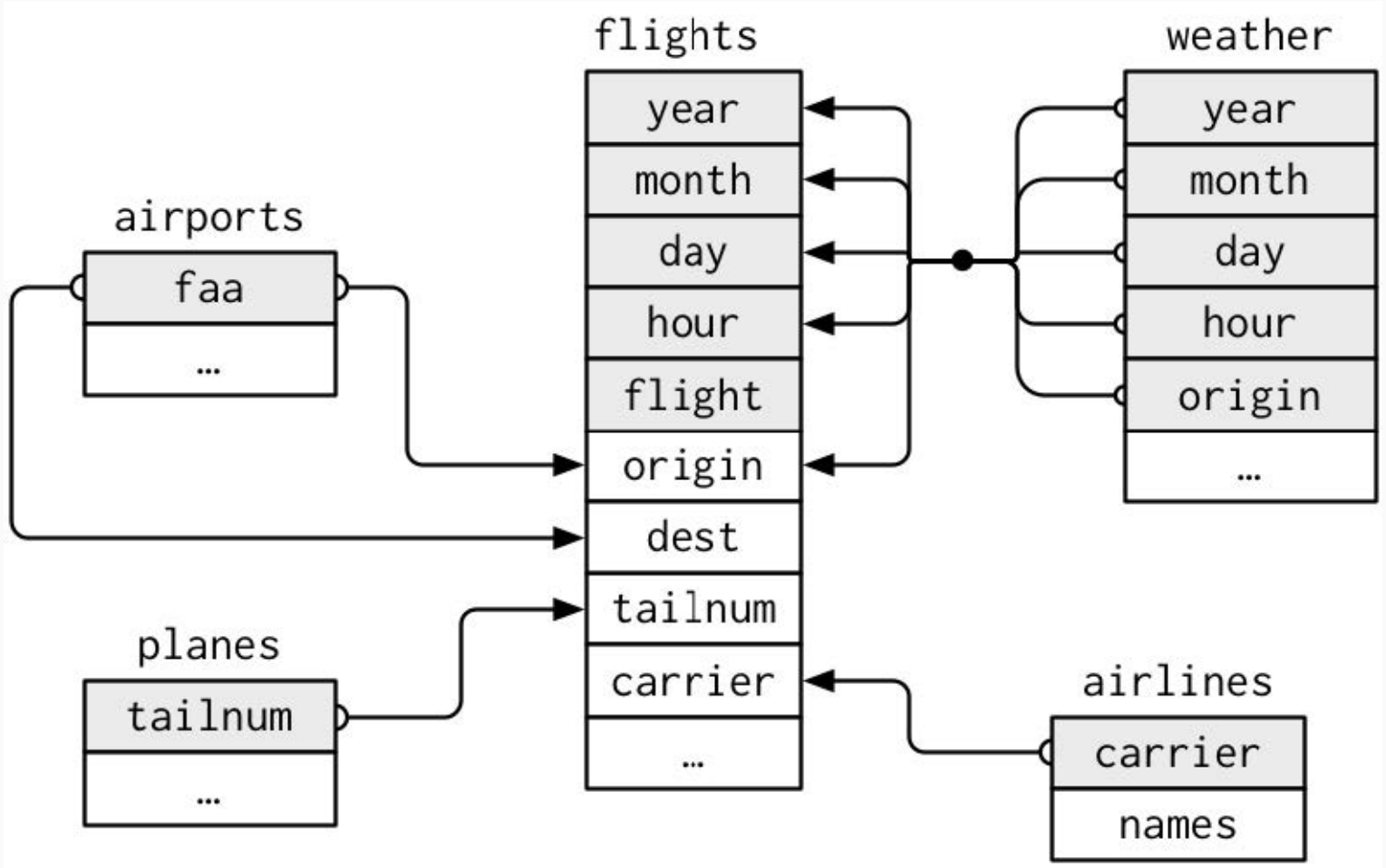

图 3: 数据库中数据表的关系结构示意图

- 想要考察天气状况对航班的影响，就需要先将数据表 `flights` 和 `weather` 根据其键值匹配连接为一个新数据表。
- 键列（可以不止 1 列），能够唯一识别自己或别人数据表的每一个观测。可以这样判断某（些）列是否是键列：

```
load("data/planes.rda")
planes %>%
  count(tailnum) %>%
  filter(n > 1)
#> # A tibble: 0 x 2
#> # ... with 2 variables: tailnum <chr>, n <int>
```

```r
load("data/weather.rda")
weather %>%
  count(year, month, day, hour, origin) %>%
  filter(n > 1)
#> # A tibble: 0 x 6
#> # ... with 6 variables: year <int>, month <int>, day <i
#> #   origin <chr>, n <int>
```

注：不唯一匹配的列，也可以作为键列进行数据连接，只是当有“一对多”关系时，会按“多”重复生成观测，有时候这恰好是需要的。

三. 合并行与合并列

bind_rows() 合并行：下方堆叠新行，根据列名匹配列，注意列名相同，否则作为新列 (NA 填充)

bind_cols() 合并列：右侧拼接新列，根据位置匹配行，行数必须相同。

```
bind_rows(
  sample_n(iris, 2),     # 随机抽取 2 个样本 (行)
  sample_n(iris, 2),
  sample_n(iris, 2))
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width    S
#> 1          6.1         2.8          4.0         1.3 vers
#> 2          6.3         2.8          5.1         1.5  vi
#> 3          5.4         3.7          1.5         0.2
#> 4          5.0         3.2          1.2         0.2
#> 5          4.8         3.0          1.4         0.3
#> 6          5.5         2.4          3.7         1.0 vers
```

```r
one = mtcars[1:4, 1:3]
two = mtcars[1:4, 4:5]
bind_cols(one, two)
#>                    mpg cyl disp  hp drat
#> Mazda RX4         21.0   6  160 110 3.90
#> Mazda RX4 Wag     21.0   6  160 110 3.90
#> Datsun 710        22.8   4  108  93 3.85
#> Hornet 4 Drive    21.4   6  258 110 3.08
```

利用 purrr 包中 map_dfr() 和 map_dfc() 的函数可以在批量读入或生成数据的同时合并行和合并列。还有 add_row(.data, ..., .before, .after) 函数可以根据索引位置插入行。

另外，受 SQL 的 INSERT、UPDATE 和 DELETE 函数的启发，dplyr 包还提供了以下函数实现根据另一个数据框来修改某数据框中的行：

- rows_insert(x, y, by)：插入新行（类似 INSERT）。默认情况下，y 中的键值必须不存在于 x 中。
- rows_append()：与 rowws_insert 类似，但是忽略键值。
- rows_update()：更改现有的行（类似 UPDATE）。y 中的键值必须是唯一的，而且默认情况下，y 中的健值必须存在于 x 中。
- rows patch()：与 rows_update() 类似，但是只覆盖 NA 值。
- rows_upscrt()：根据 y 中的键值是否已经存在于 x 中，对 x 进行插入或更新。
- rows_delete()：删除行（类似 DELETE）。默认情况下，y 中的键值必须存在于 x 中。

四. 根据值匹配合并数据框

最常用的六种合并：**左连接、右连接、全连接、内连接、半连接、反连接**[2]

```
left_join(x, y, by)
right_join(x, y, by)
full_join(x, y, by)
inner_join(x, y, by)
semi_join(x, y, by)
anti_join(x, y, by)
```

[2]前四种连接又称为**修改连接**，后两种连接又称为**过滤连接**.

- 演示数据集

```
band = band_members
band
#> # A tibble: 3 x 2
#>   name  band
#>   <chr> <chr>
#> 1 Mick  Stones
#> 2 John  Beatles
#> 3 Paul  Beatles
```

```
instrument = band_instruments
instrument
#> # A tibble: 3 x 2
#>   name  plays
#>   <chr> <chr>
#> 1 John  guitar
#> 2 Paul  bass
#> 3 Keith guitar
```

1. 左连接：left_join()

- 外连接至少保留一个数据表中的所有观测，分为左连接、右连接、全连接，其中最常用的是左连接：保留 x 所有行，合并匹配的 y 中的列。

```
band %>%
  left_join(instrument, by = "name")
#> # A tibble: 3 x 3
#>   name  band    plays
#>   <chr> <chr>   <chr>
#> 1 Mick  Stones  <NA>
#> 2 John  Beatles guitar
#> 3 Paul  Beatles bass
```

- 若两个表中的键列列名不同，用 by = c("name1" = "name2")；若根据多个键列匹配，用 by = c("name1", "name2").

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
Mick	Stones	<NA>
John	Beatles	guitar
Paul	Beatles	bass

图 4: 左连接示意图

2. 右连接：right_join()

- 保留 y 所有行，合并匹配的 x 中的列。

```
band %>%
  right_join(instrument, by = "name")
#> # A tibble: 3 x 3
#>   name  band    plays
#>   <chr> <chr>   <chr>
#> 1 John  Beatles guitar
#> 2 Paul  Beatles bass
#> 3 Keith <NA>    guitar
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
John	Beatles	guitar
Paul	Beatles	bass
Keith	<NA>	guitar

图 5: 右连接示意图

3. 全连接：full_join()

- 保留 x 和 y 中的所有行，合并匹配的列。

```
band %>%
  full_join(instrument, by = "name")
#> # A tibble: 4 x 3
#>   name  band    plays
#>   <chr> <chr>   <chr>
#> 1 Mick  Stones  <NA>
#> 2 John  Beatles guitar
#> 3 Paul  Beatles bass
#> # ... with 1 more row
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
Mick	Stones	<NA>
John	Beatles	guitar
Paul	Beatles	bass
Keith	<NA>	guitar

图 6: 全连接示意图

4. 内连接：inner_join()

- 内连接是保留两个数据表中所共有的观测：只保留 `x` 中与 `y` 匹配的行，合并匹配的 `y` 中的列。

```
band %>%
  inner_join(instrument, by = "name")
#> # A tibble: 2 x 3
#>   name  band    plays
#>   <chr> <chr>   <chr>
#> 1 John  Beatles guitar
#> 2 Paul  Beatles bass
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band	plays
John	Beatles	guitar
Paul	Beatles	bass

图 7: 内连接示意图

5. 半连接：semi_join()

- 保留 x 表中与 y 表中的行相匹配的所有行，即根据 y 表中有匹配的部分来筛选 x 表中的行

```
band %>%
  semi_join(instrument, by = "name")
#> # A tibble: 2 x 2
#>   name  band
#>   <chr> <chr>
#> 1 John  Beatles
#> 2 Paul  Beatles
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band
John	Beatles
Paul	Beatles

图 8: 半连接示意图

6. 反连接：anti_join()

- 删掉 x 表中与 y 表中的行相匹配的所有行，即根据 y 表中没有匹配的部分筛选 x 表中的行

```
band %>%
  anti_join(instrument, by = "name")
#> # A tibble: 1 x 2
#>   name  band
#>   <chr> <chr>
#> 1 Mick  Stones
```

band

name	band
Mick	Stones
John	Beatles
Paul	Beatles

+

instrument

name	plays
John	guitar
Paul	bass
Keith	guitar

=

name	band
Mick	Stones

图 9: 反连接示意图

- 前面的都是连接两个数据表，若要连接多个数据表，将连接两个数据表的函数结合 `purrr::reduce()` 使用即可。
- 比如 `achieves` 文件夹有 3 个 `Excel` 文件，想要批量读取它们，再依次做全连接（做其他连接也是类似的）。
- `reduce()` 可以实现先将前两个表做全连接，再将结果表与第三个表做全连接（更多表就依次这样做下去）

```r
library(readxl)
files = list.files("data/achieves/", pattern = "xlsx",
                   full.names = TRUE)
map(files, read_xlsx) %>%
  reduce(full_join, by = " 人名")    # 读入并依次做全连接
#> # A tibble: 7 x 4
#>   人名   `3 月业绩` `4 月业绩` `5 月业绩`
#>   <chr>      <dbl>      <dbl>      <dbl>
#> 1 小明          80         NA         NA
#> 2 小李          85         NA         80
#> 3 小张          90         50         NA
#> # ... with 4 more rows
```

- 同样的数据，在一个工作簿的多个工作表中，批量读取并依次做全连接：

```
path = "data/3-5 月业绩.xlsx"
map(excel_sheets(path),
    ~ read_xlsx(path, sheet = .x)) %>%
  reduce(full_join, by = " 人名")    # 读入并依次做全连接
```

五. 集合运算

集合运算有时候很有用，都是针对所有行，通过比较变量的值来实现。这就需要数据表 x 和 y 具有相同的变量，并将观测看成是集合中的元素：

```
intersect(x, y)      # 返回 x 和 y 共同包含的观测
union(x, y)          # 返回 x 和 y 中所有的（唯一）观测
setdiff(x, y)        # 返回在 x 中但不在 y 中的观测
setequal(x, y)       # 判断集合 x 和 y 是否相等
```

本篇主要参阅 (张敬信, 2022), (Hadley Wickham, 2017), (Amelia McNamara, 2020), (Desi Quintans, 2019), 以及 RStudio 博文，模板感谢 (黄湘云, 2021), (谢益辉, 2021).

参考文献

Amelia McNamara, H. W. (2020). *Introduction to Data Science in the Tidyverse*. rstudio::conf 2020.

Desi Quintans, J. P. (2019). *Working in the Tidyverse*. HIE Advanced R workshop.

Hadley Wickham, G. G. (2017). *R for Data Science*. O' Reilly, 1 edition. ISBN 978-1491910399.

张敬信 (2022). *R* 语言编程：基于 *tidyverse*. 人民邮电出版社, 北京.

谢益辉 (2021). *rmarkdown: Dynamic Documents for R*.

黄湘云 (2021). *Github: R-Markdown-Template*.