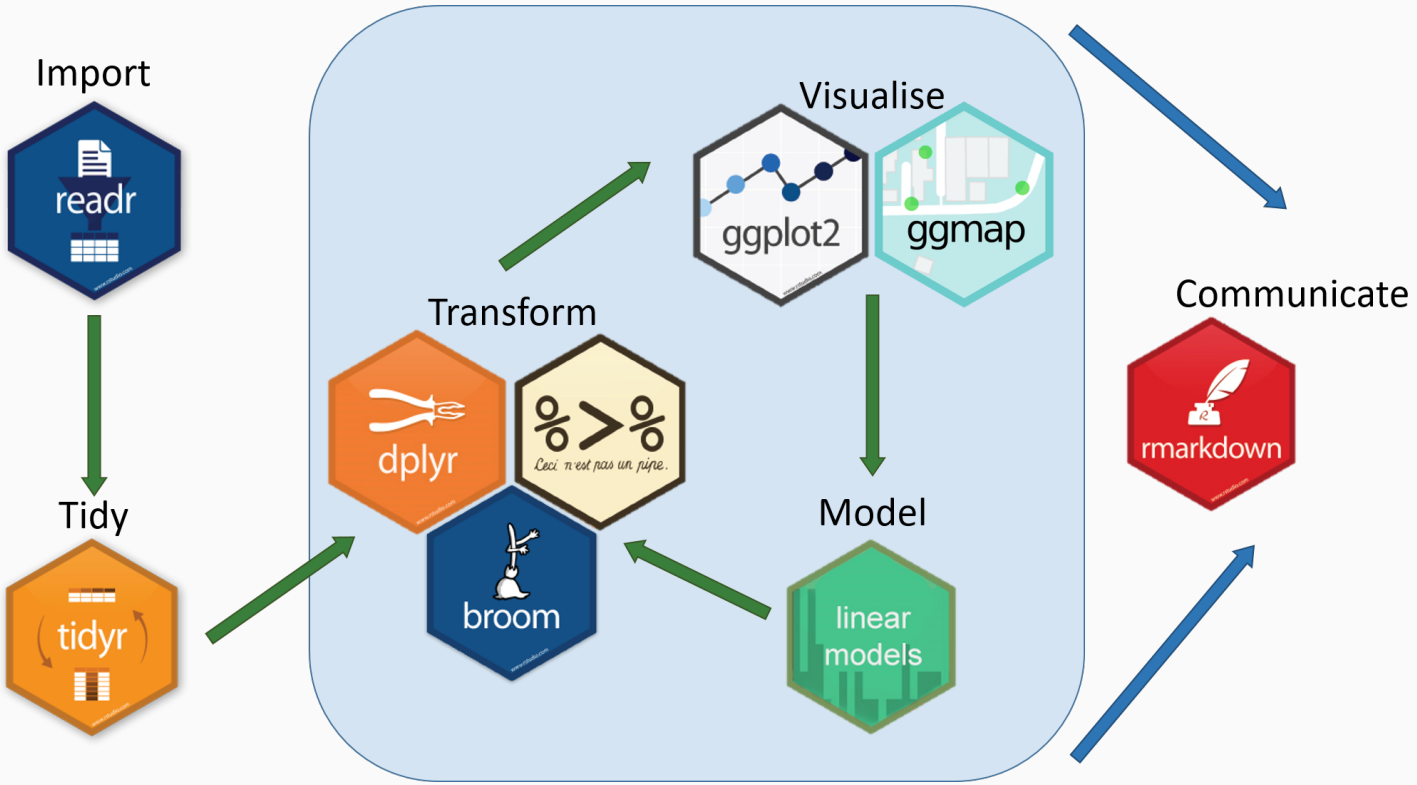
第四章：数据处理

王敏杰

2020 年 7 月 27 日

四川师范大学

正式进入 tidyverse 家族的学习



tidyverse 家族

tidyverse 家族主要成员包括

功能	宏包
有颜值担当	ggplot2
数据处理王者	dplyr
数据转换专家	tidyr
数据载入利器	readr
循环加速器	purrr
强化数据框	tibble

数据读取

读取数据

R 语言提供了很多读取数据的函数。

文件格式	R 函数
.txt	read.table()
.csv	read.csv() and readr::read_csv()
.xls and .xlsx	readxl::read_excel() and openxlsx::read.xlsx()
.sav	foreign::read.spss()
.Rdata or rda	load()
.rds	readRDS() and readr::read_rds()
.dta	haven::read_dta() and haven::read_stata()
Internet	download.file()

范例

```
library(readr)
wages <- read_csv("./demo_data/wages.csv")
head(wages, 6)
#> # A tibble: 6 x 6
#>     earn height sex    race      ed   age
#>    <dbl>  <dbl> <chr>  <chr> <dbl> <dbl>
#> 1 79571.   73.9 male   white    16    49
#> 2 96397.   66.2 female white    16    62
#> 3 48711.   63.8 female white    16    33
#> 4 80478.   63.2 female other    16    95
#> 5 82089.   63.1 female white    17    43
#> 6 15313.   64.5 female white    15    30
```

范例

```
library(readxl)
d <- read_excel("./demo_data/olympics.xlsx")
tail(d, 6)
#> # A tibble: 6 x 3
#>   Olympic_year Men_score Women_score
#>          <dbl>     <dbl>       <dbl>
#> 1         1984      9.99        11.0
#> 2         1988      9.92        10.5
#> 3         1992      9.96        10.8
#> 4         1996      9.84        10.9
#> 5         2000      9.87        10.8
#> 6         2004      9.85        10.9
```

数据处理

tidy 原则

Hadley Wickhamt 提出了数据科学 tidy 原则，我结合自己的理解，tidy 思想体现在:

- 一切都是数据框，任何数据都可以规整
- 数据框的一列代表一个**变量**，数据框的一行代表一次**观察**
- 函数处理数据时，数据框进数据框出（函数的第一个参数始终为**数据框**）

dplyr 宏包

本章我们介绍 tidyverse 里数据处理的神器 dplyr 宏包。首先，我们加载该宏包

```
library(dplyr)
```

dplyr 定义了数据处理的规范语法，其中主要包含以下七个主要的函数。

- `mutate()`, `select()`, `filter()`
- `summarise()`, `group_by()`, `arrange()`
- `left_join()`, `right_join()`, `full_join()`

我们将依次介绍

假定数据

假定我们有一数据框，包含三位学生的英语和数学科目

```
df <- data.frame(
      name = c("Alice", "Alice", "Bob", "Bob", "Carol", "Ca
      type = c("english", "math", "english", "math", "engli
)
df
#>    name    type
#> 1 Alice english
#> 2 Alice    math
#> 3   Bob english
#> 4   Bob    math
#> 5 Carol english
#> 6 Carol    math
```

mutate() 增加一列

这里有他们的最近的考试成绩，想添加到数据框中

```
score2020 <- c(80.2, 90.5, 92.2, 90.8, 82.5, 84.6)
score2020
#> [1] 80.2 90.5 92.2 90.8 82.5 84.6
```

使用传统的方法

```
df$score <- score2020
df
#>    name    type score
#> 1 Alice english  80.2
#> 2 Alice    math  90.5
#> 3   Bob english  92.2
#> 4   Bob    math  90.8
#> 5 Carol english  82.5
#> 6 Carol    math  84.6
```

dplyr 语法这样写

```
#
mutate(df, score = score2020)
#>    name    type score
#> 1 Alice english  80.2
#> 2 Alice    math  90.5
#> 3   Bob english  92.2
#> 4   Bob    math  90.8
#> 5 Carol english  82.5
#> 6 Carol    math  84.6
```

mutate() 增加一列

mutate() 函数

```
mutate(.data = df, score = score2020)
```

- 第一参数是我们要处理的数据框，比如这里的 df，
- 第二个参数是 score = score2020，等号左边的 score 是我们打算创建一个新列，而取的列名；等号右边是装着学生成绩的**向量**（注意，向量的长度要与数据框的行数相等，比如这里长度都是 6）

管道%>%

这里有必要介绍下管道操作符 %>%.

```
c(1:10)
#>  [1]  1  2  3  4  5  6  7  8  9 10
```

```
sum(c(1:10))
#> [1] 55
```

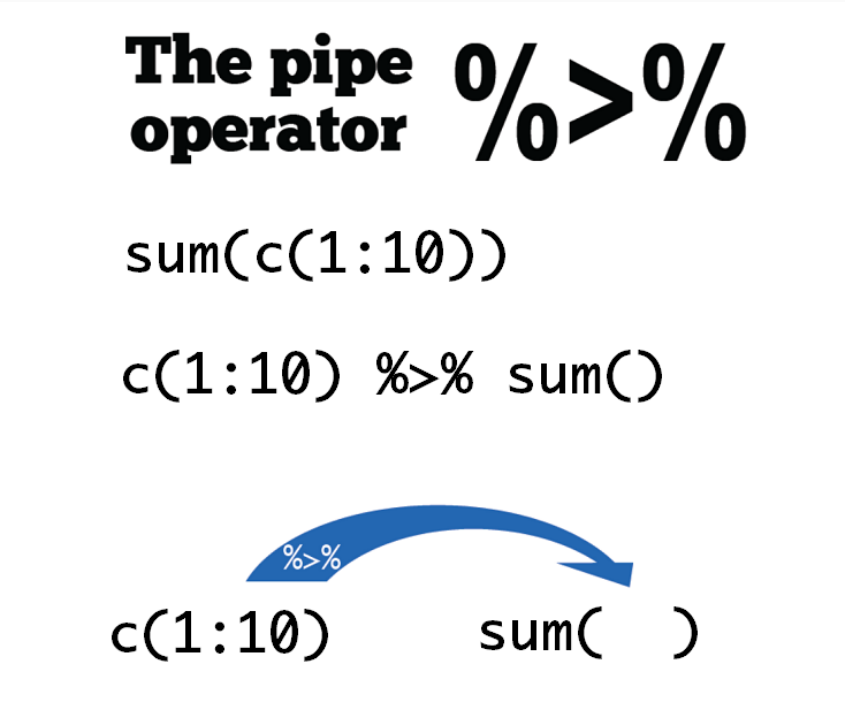
与下面的写法是等价的,

```
c(1:10) %>% sum()
#> [1] 55
```

管道%>%

```
c(1:10) %>% sum()
```

这条语句的意思，向量 c(1:10) 通过管道操作符 %>% ，传递到函数 sum() 的第一个参数位置，即 sum(c(1:10))，这个%>% 管道操作符还是很形象的，



管道%>%

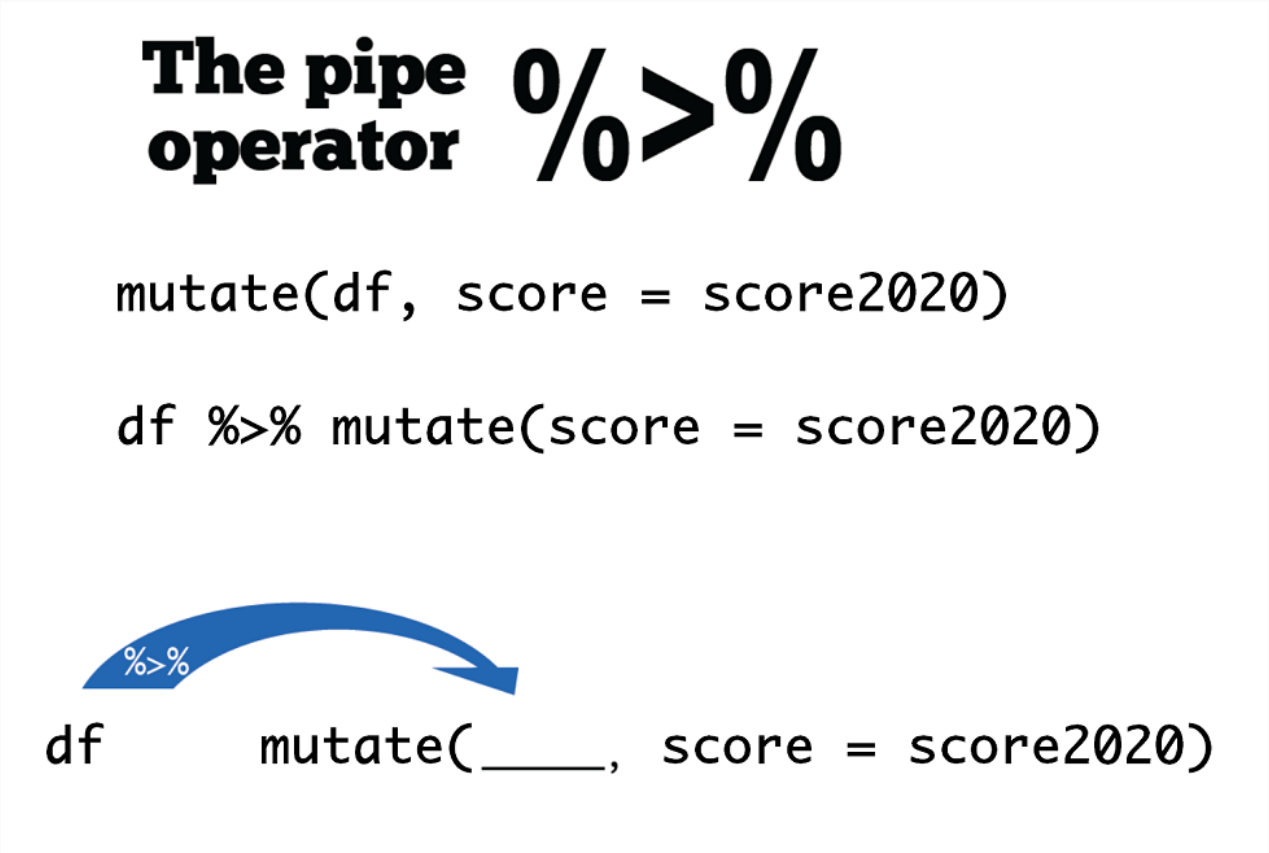
当对执行多个函数操作的时候，就显得格外方便，代码可读性更强。

```
sqrt(sum(abs(c(-10:10))))
#> [1] 10.5
```

```
c(-10:10) %>% abs() %>% sum() %>% sqrt()
#> [1] 10.5
```

管道%>%

那么，上面增加学生成绩的语句 mutate(df, score = score2020) 就可以使用管道



管道%>%

```
# 等价于
df %>% mutate(score = score2020)
#>     name    type score
#> 1 Alice english  80.2
#> 2 Alice    math  90.5
#> 3   Bob english  92.2
#> 4   Bob    math  90.8
#> 5 Carol english  82.5
#> 6 Carol    math  84.6
```

是不是很赞？

select() 选择某列

select()，就是选择数据框的某一列

传统的方法

```
df["name"]
#>     name
#> 1 Alice
#> 2 Alice
#> 3   Bob
#> 4   Bob
#> 5 Carol
#> 6 Carol
```

dplyr 的方法

```
df %>% select(name)
#>     name
#> 1 Alice
#> 2 Alice
#> 3   Bob
#> 4   Bob
#> 5 Carol
#> 6 Carol
```

select() 选择某列

如果选取多列，就再写一个就行了

```
df %>% select(name, score)
#>     name score
#> 1 Alice  80.2
#> 2 Alice  90.5
#> 3   Bob  92.2
#> 4   Bob  90.8
#> 5 Carol  82.5
#> 6 Carol  84.6
```

select() 选择某列

如果不想要某列，可以在变量前面加-，

```
df %>% select(-type)
#>     name score
#> 1 Alice  80.2
#> 2 Alice  90.5
#> 3   Bob  92.2
#> 4   Bob  90.8
#> 5 Carol  82.5
#> 6 Carol  84.6
```

filter() 筛选

我们还可以对数据行方向的选择和筛选，比如这里把成绩高于 90 分的同学筛选出来

```
df %>% filter(score >= 90)
#>    name    type score
#> 1 Alice    math  90.5
#> 2   Bob english  92.2
#> 3   Bob    math  90.8
```

filter() 筛选

我们也可以限定多个条件进行筛选，英语成绩高于 90 分的筛选出来

```
df %>% filter(type == "english", score >= 90)
#>    name    type score
#> 1   Bob english  92.2
```

summarise() 统计

summarise() 主要用于统计，往往与其他函数配合使用

比如，计算所有同学的考试成绩的均值

```
df %>% summarise( mean_score = mean(score))
#>   mean_score
#> 1       86.8
```

比如，计算所有同学的考试成绩的标准差

```
df %>% summarise( mean_score = sd(score))
#>   mean_score
#> 1       5.01
```

summarise() 统计

还可以同时完成多个统计

```
df %>% summarise(
  mean_score = mean(score),
  median_score = median(score),
  n = n(),
  sum = sum(score)
)
#>   mean_score median_score n sum
#> 1       86.8         87.5 6 521
```

group_by() 分组

先分组再统计。比如，我们想统计每个学生的平均成绩，即先按学生 name 分组，然后分别求平均

```
df %>%
  group_by(name) %>%
  summarise(
    mean_score = mean(score),
   sd_score = sd(score)
  )
#> # A tibble: 3 x 3
#>   name  mean_score sd_score
#>   <chr>      <dbl>    <dbl>
#> 1 Alice       85.4    7.28
#> 2 Bob         91.5    0.990
#> 3 Carol       83.6    1.48
```

arrange() 排序

我们按照考试成绩从低到高排序，然后输出

```
df %>% arrange(score)
#>    name    type score
#> 1 Alice english  80.2
#> 2 Carol english  82.5
#> 3 Carol    math  84.6
#> 4 Alice    math  90.5
#> 5   Bob    math  90.8
#> 6   Bob english  92.2
```

arrange() 排序

如果从高到低降序排列呢，有两种方法:

```
df %>% arrange(-score)
#>     name    type score
#> 1    Bob english  92.2
#> 2    Bob    math  90.8
#> 3  Alice    math  90.5
#> 4  Carol    math  84.6
#> 5  Carol english  82.5
#> 6  Alice english  80.2
```

```
df %>% arrange(desc(score))
#>     name    type score
#> 1    Bob english  92.2
#> 2    Bob    math  90.8
#> 3  Alice    math  90.5
#> 4  Carol    math  84.6
#> 5  Carol english  82.5
#> 6  Alice english  80.2
```

哪边可读性更强些？

arrange() 排序

也可对多个变量先后排序。比如，先按学科排，然后按照成绩从高到底排序

```
df %>%
  arrange(type, desc(score))
#>    name    type score
#> 1   Bob english  92.2
#> 2 Carol english  82.5
#> 3 Alice english  80.2
#> 4   Bob    math  90.8
#> 5 Alice    math  90.5
#> 6 Carol    math  84.6
```

left_join 合并

假定我们已经统计了每个同学的平均成绩，存放在数据框 df1

```
df1 <- df %>%
  group_by(name) %>%
  summarise( mean_score = mean(score) )
df1
#> # A tibble: 3 x 2
#>   name  mean_score
#>   <chr>       <dbl>
#> 1 Alice        85.4
#> 2 Bob          91.5
#> 3 Carol        83.6
```

left_join 合并

同时，我们又有新一个数据框 df2，它包含同学们的年龄信息

```
df2 <- tibble(
       name = c("Alice", "Bob"),
       age =  c(12, 13)
)
df2
#> # A tibble: 2 x 2
#>   name    age
#>   <chr> <dbl>
#> 1 Alice    12
#> 2 Bob      13
```

left_join 左合并

通过姓名 name 把两个数据框 df1 和 df2 合并，

```
left_join(df1, df2, by = "name")
#> # A tibble: 3 x 3
#>   name  mean_score   age
#>   <chr>      <dbl> <dbl>
#> 1 Alice       85.4    12
#> 2 Bob         91.5    13
#> 3 Carol       83.6    NA
```

大家注意到最后一行 Carol 的年龄是 NA，大家想想为什么呢？

left_join 左合并

当然，也可以这样写

```
df1 %>% left_join(df2, by = "name")
#> # A tibble: 3 x 3
#>   name  mean_score   age
#>   <chr>      <dbl> <dbl>
#> 1 Alice       85.4    12
#> 2 Bob         91.5    13
#> 3 Carol       83.6    NA
```

right_join 右合并

我们再试试 right_join() 右合并

```
df1 %>% right_join(df2, by = "name")
#> # A tibble: 2 x 3
#>   name  mean_score   age
#>   <chr>      <dbl> <dbl>
#> 1 Alice       85.4    12
#> 2 Bob         91.5    13
```

Carol 同学的信息没有了？大家想想又为什么呢？

延伸阅读

- 推荐`https://dplyr.tidyverse.org/`.
- cheatsheet
- 运行并读懂nycflights.Rmd