

R 语言编程：基于 tidyverse

第 13 讲 数据操作 III: 其它操作

张敬信

2022 年 3 月 31 日

哈尔滨商业大学

一. 按行汇总

- 通常的数据操作逻辑都是**按列方式**，这使得按行汇总并不容易
- `rowwise()` 为数据框创建**按行方式**，并不是真的改变数据框，只是创建了按行元信息，改变了数据框的操作逻辑
- **按行方式**可以理解为一种特殊的分组：每一行作为一组，执行的是“分组-汇总-合并”，速度较慢。要解除行化模式，用 `ungroup()`
- **按行方式**的一种巧妙的用途是批量建模（见后文）

```
df = readxl::read_xlsx("datas/ExamDatas.xlsx")
```

```

rf = df %>%
  rowwise()
rf
#> # A tibble: 50 x 8
#> # Rowwise:
#>   class name    sex  chinese  math english moral science
#>   <chr> <chr>  <chr>    <dbl> <dbl>   <dbl> <dbl>   <dbl>
#> 1 六 1 班 何娜    女          87    92       79     9    10
#> 2 六 1 班 黄才菊 女          95    77       75     8     9
#> 3 六 1 班 陈芳妹 女          79    87       66     9    10
#> # ... with 47 more rows

```

```
rf %>%
```

```
  mutate(total = sum(chinese, math, english))
```

```
#> # A tibble: 50 x 9
```

```
#> # Rowwise:
```

```
#>   class name    sex  chinese  math english moral science t
```

```
#>   <chr> <chr>  <chr>    <dbl> <dbl>   <dbl> <dbl>   <dbl> <
```

```
#> 1 六 1 班 何娜    女          87    92      79      9      10
```

```
#> 2 六 1 班 黄才菊 女          95    77      75      8      9
```

```
#> 3 六 1 班 陈芳妹 女          79    87      66      9      10
```

```
#> # ... with 47 more rows
```

- `c_across()` 是为**按行方式**在选定的列范围汇总数据而设计的¹，它没有提供 `.fns` 参数，只能选择列

```
rf %>%  
  mutate(total = sum(c_across(where(is.numeric))))  
#> # A tibble: 50 x 9  
#> # Rowwise:  
#>   class name    sex  chinese  math english moral science t  
#>   <chr> <chr>  <chr>    <dbl> <dbl>   <dbl> <dbl>   <dbl> <  
#> 1 六 1 班 何娜    女         87    92      79      9      10  
#> 2 六 1 班 黄才菊 女         95    77      75      8      9  
#> 3 六 1 班 陈芳妹 女         79    87      66      9      10  
#> # ... with 47 more rows
```

¹若只是做按行求和或均值，直接用 `rowSums()/rowMeans()` 速度更快。

总结**逐行迭代**，除了 for 循环的四种做法：

```
iris[1:4] %>%          # apply
  mutate(avg = apply(., 1, mean))
iris[1:4] %>%          # rowwise (慢)
  rowwise() %>%
  mutate(avg = mean(c_across())))
iris[1:4] %>%          # pmap
  mutate(avg = pmap_dbl(., ~ mean(c(...))))
iris[1:4] %>%          # asplit(逐行分割) + map
  mutate(avg = map_dbl(asplit(., 1), mean))
```

二. 窗口函数

- 汇总函数如 `sum()` 和 `mean()` 接受 `n` 个输入, 返回 1 个值
- 而窗口函数是汇总函数的变体: 接受 `n` 个输入, 返回 `n` 个值。例如, `cumsum()`、`cummean()`、`rank()`、`lead()`、`lag()` 等。

1. 排名和排序函数

- 共有 6 个排名函数，只介绍最常用的 `min_rank()`²：默认从小到大排名

```
df %>%      # 按数学成绩从高到低排名次
  mutate(ranks = min_rank(desc(math))) %>%
  arrange(ranks)

#> # A tibble: 50 x 9
#>   class name  sex  chinese  math english moral science r
#>   <chr> <chr>  <chr>    <dbl> <dbl>   <dbl> <dbl>   <dbl> <
#> 1 六 4 班 周婵  女          92    94      77    10      9
#> 2 六 4 班 陈丽丽 女          87    93      61     8      6
#> 3 六 1 班 何娜  女          87    92      79     9     10
#> # ... with 47 more rows
```

²相当于 `ties.method = "min"`.

2. 移位函数

`lag()`: 取前一个值, 数据整体右移一位, 相当于将时间轴滞后一个单位

`lead()`: 取后一个值, 数据整体左移一位, 相当于将时间轴超前一个单位

```
library(lubridate)
```

```
dt = tibble(
```

```
  day = as_date("2019-08-30") + c(0,4:6),
```

```
  wday = weekdays(day),
```

```
  sales = c(2,6,2,3),
```

```
  balance = c(30, 25, -40, 30)
```

```
)
```

```
dt %>%
  mutate(sales_lag = lag(sales),
         sales_delta = sales - lag(sales))
#> # A tibble: 4 x 6
#>   day          wday  sales balance sales_lag sales_delta
#>   <date>      <chr> <dbl>   <dbl>   <dbl>      <dbl>
#> 1 2019-08-30 星期五     2     30     NA         NA
#> 2 2019-09-03 星期二     6     25     2          4
#> 3 2019-09-04 星期三     2    -40     6         -4
#> # ... with 1 more row
```

注：默认是根据行序移位，可用参数 `order_by` 设置根据某变量值大小顺序做移位。

3. 累计汇总

- base R 已经提供了 `cumsum()`、`cummin()`、`cummax()`、`cumprod()`
- dplyr 包又提供了 `cummean()`、`cumany()`、`cumall()`，后两者可与 `filter()` 连用选择行：
 - `cumany(x)`: 用来选择遇到第一个满足条件之后的所有行
 - `cumany(!x)`: 用来选择遇到第一个不满足条件之后的所有行
 - `cumall(x)`: 用来选择所有行直到遇到第一个不满足条件的行
 - `cumall(!x)`: 用来选择所有行直到遇到第一个满足条件的行

```
x = c(1, 3, 5, 2, 2)
cumany(x >= 5) # 从第一个出现 x>=5 选择后面所有值
#> [1] FALSE FALSE TRUE TRUE TRUE
cumany(!x < 5) # 同上，从第一个出现不满足 x<5 选择后面所有值
#> [1] FALSE FALSE TRUE TRUE TRUE
cumall(x < 5) # 依次选择值直到第一个 x<5 不成立
#> [1] TRUE TRUE FALSE FALSE FALSE
cumall(!x >= 5) # 同上，依次选择值直到第一个出现 x>=5
#> [1] TRUE TRUE FALSE FALSE FALSE
```

```
dt %>%      # 选择第一次透支之后的所有行
  filter(cumany(balance < 0))
```

```
#> # A tibble: 2 x 4
```

```
#>   day          wday    sales balance
#>   <date>      <chr>  <dbl>    <dbl>
#> 1 2019-09-04  星期三      2     -40
#> 2 2019-09-05  星期四      3      30
```

```
dt %>%      # 选择所有行直到第一次透支
  filter(cumall(!(balance < 0)))
```

```
#> # A tibble: 2 x 4
```

```
#>   day          wday    sales balance
#>   <date>      <chr>  <dbl>    <dbl>
#> 1 2019-08-30  星期五      2      30
#> 2 2019-09-03  星期二      6      25
```

三. 滑窗迭代

- “窗口函数”术语来自 SQL，意味着逐窗口浏览数据，将某函数重复应用于数据的每个“窗口”。
- 窗口函数的典型应用包括滑动平均、累计和以及更复杂如滑动回归。

- `slider` 包提供了 `slide_*()` 系列函数实现滑窗迭代，其基本格式为：
`slide_*(.x, .f, ..., .before, .after, .step, .complete)`
- `.x`: 为窗口所要滑过的向量
- `.f`: 要应用于每个窗口的函数，支持 `purrr` 风格公式
- `...`: 用来传递 `.f` 的其他参数
- `.before, .after`: 设置窗口范围当前元往前、往后几个元，可以取 `Inf` (往前、往后所有元)
- `.step`: 每次函数调用，窗口往前移动的步长
- `.complete`: 设置两端处是否保留不完整窗口，默认为 `FALSE`
- `slider::slide_*()` 系列函数与 `purrr::map_*()` 是类似的，只是将“逐元素迭代”换成了“逐窗口迭代”。

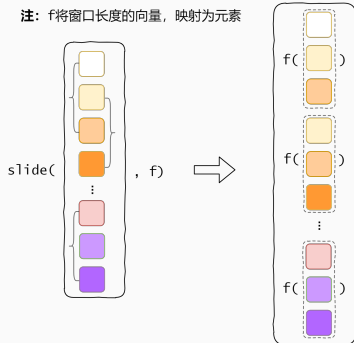


图 1: slide 滑窗迭代示意图

滑窗迭代的逻辑：先利用窗口参数正确设计并生成滑动窗口，每个滑动窗口是一个小向量，函数.f 是作用在一个小向量上，通过后缀控制返回结果类型，返回结果通常作为 mutate 的一系列。

- 金融时间序列数据经常需要计算滑动平均

```
library(slider)
dt %>%
  mutate(avg_3 = slide_dbl(sales, mean,
                           .before = 1, .after = 1))

#> # A tibble: 4 x 5
#>   day          wday  sales balance avg_3
#>   <date>      <chr> <dbl>   <dbl> <dbl>
#> 1 2019-08-30 星期五     2     30    4
#> 2 2019-09-03 星期二     6     25  3.33
#> 3 2019-09-04 星期三     2    -40  3.67
#> # ... with 1 more row
```

- 输出每个滑动窗口更便于理解该 3 日滑动平均是如何计算的：

```
slide(dt$sales, ~ .x, .before = 1, .after = 1)
#> [[1]]
#> [1] 2 6
#>
#> [[2]]
#> [1] 2 6 2
#>
#> [[3]]
#> [1] 6 2 3
#>
#> [[4]]
#> [1] 2 3
```

- 前面计算的并不是真正的 3 日，而是连续 3 个值的滑动平均。因为 `slide()` 默认是以行索引来滑动
- `slide_index(.x, .i, .f, ...)`: 按索引向量 `.i` 计算滑动平均，可根据 “`.i` 的当前元 + 其前/后若干元” 创建相应的 `.x` 的滑动窗口

- 来看行索引和日期索引滑动窗口的区别

```
slide(dt$day, ~ .x, .before = 1, .after = 1)
#> [[1]]
#> [1] "2019-08-30" "2019-09-03"
#>
#> [[2]]
#> [1] "2019-08-30" "2019-09-03" "2019-09-04"
#>
#> [[3]]
#> [1] "2019-09-03" "2019-09-04" "2019-09-05"
#>
#> [[4]]
#> [1] "2019-09-04" "2019-09-05"
```

```
slide_index(dt$day, dt$day, ~ .x, .before = 1, .after = 1)
#> [[1]]
#> [1] "2019-08-30"
#>
#> [[2]]
#> [1] "2019-09-03" "2019-09-04"
#>
#> [[3]]
#> [1] "2019-09-03" "2019-09-04" "2019-09-05"
#>
#> [[4]]
#> [1] "2019-09-04" "2019-09-05"
```

- 计算 sales 真正的 3 日滑动平均

```
dt %>%  
  mutate(avg_3 = slide_index_dbl(sales, day, mean,  
                                  .before = 1, .after = 1))  
  
#> # A tibble: 4 x 5  
#>   day          wday    sales balance avg_3  
#>   <date>      <chr>  <dbl>    <dbl> <dbl>  
#> 1 2019-08-30 星期五      2      30    2  
#> 2 2019-09-03 星期二      6      25    4  
#> 3 2019-09-04 星期三      2     -40  3.67  
#> # ... with 1 more row
```

四. 整洁计算

tidyverse 代码之所以“整洁、优雅”，是因为它内部采用了一套**整洁计算**框架。

1. 数据屏蔽与整洁选择

- **整洁计算**的两种基本形式是：
 - 数据屏蔽：使得可以不用带数据框（环境变量）名字，就能使用数据框内的变量（数据变量），便于在数据集内计算值
 - 整洁选择：即各种选择列语法，便于使用数据集中的列
- 数据屏蔽内在的机制是先冻结表达式，然后注入函数，再恢复其计算。

整洁计算已经为此做好了两种封装：

- `{{ }}` (curly-curly 算符)：若只是传递，可将“冻结 + 注入”合成一步
- `enquo()` 和 `!!` (引用与反引用)：不只是传递，冻结后还需要访问

- 自定义函数时，想要像 tidyverse 那样整洁传递变量名，需要用到 `{{ }}`，即用两个大括号将变量括起来：

```
var_summary = function(data, var) {  
  data %>%  
    summarise(n = n(), mean = mean({{var}}))  
}  
  
mtcars %>%  
  group_by(cyl) %>%  
  var_summary(mpg)  
  
#> # A tibble: 3 x 3  
#>   cyl     n mean  
#>   <dbl> <int> <dbl>  
#> 1     4    11  26.7  
#> 2     6     7  19.7  
#> 3     8    14  15.1
```


- 若想用字符串形式传递变量名，在访问数据时需要借助.data[[var]]，这里.data 是代替数据集

```
var_summary = function(data, var) {  
  data %>%  
    summarise(n = n(), mean = mean(.data[[var]]))  
}  
mtcars %>%  
  group_by(cyl) %>%  
  var_summary("mpg")  
#> # A tibble: 3 x 3  
#>   cyl      n  mean  
#>   <dbl> <int> <dbl>  
#> 1     4    11  26.7  
#> 2     6     7  19.7  
#> 3     8    14  15.1
```

- 还可用于对列名向量做循环迭代，比如对因子型各列计算各水平值频数：

```
mtcars[,9:10] %>%  
  names() %>%  
  map(~ count(mtcars, .data[[.x]]))  
#> [[1]]  
#>   am   n  
#> 1   0 19  
#> 2   1 13  
#>  
#> [[2]]  
#>   gear  n  
#> 1     3 15  
#> 2     4 12  
#> 3     5  5
```

- 将整洁选择作为函数参数传递，也需要用到 `{{ }}`

```
summarise_mean = function(data, vars) {  
  data %>%  
    summarise(n = n(), across({{vars}}, mean))  
}  
mtcars %>%  
  group_by(cyl) %>%  
  summarise_mean(where(is.numeric))
```

```
#> # A tibble: 3 x 12
```

```
#>   cyl      n  mpg  disp   hp  drat    wt  qsec    vs  
#>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
#> 1     4    11  26.7  105.  82.6  4.07  2.29  19.1  0.909  0.  
#> 2     6     7  19.7  183. 122.   3.59  3.12  18.0  0.571  0.  
#> 3     8    14  15.1  353. 209.   3.23  4.00  16.8  0      0.
```

- 若传递是多个列名构成的字符向量，则需要借助函数 `all_of()` 或 `any_of()`

```
vars = c("mpg", "vs")  
mtcars %>% select(all_of(vars))  
mtcars %>% select(!all_of(vars))
```

- 使用 `{{ }}` 或整洁选择同时修改列名

```
my_summarise = function(data, mean_var, sd_var) {  
  data %>%  
    summarise("mean_{{mean_var}}" := mean({{mean_var}}),  
              "sd_{{sd_var}}" := mean({{sd_var}}))  
}  
mtcars %>%  
  group_by(cyl) %>%  
  my_summarise(mpg, disp)  
#> # A tibble: 3 x 3  
#>   cyl mean_mpg sd_disp  
#>   <dbl>   <dbl>   <dbl>  
#> 1     4    26.7    105.  
#> 2     6    19.7    183.  
#> 3     8    15.1    353.
```

```

my_summarise = function(data, group_var, summarise_var) {
  data %>%
    group_by(across({{group_var}})) %>%
    summarise(across({{summarise_var}}, mean,
                      .names = "mean_{.col}"))
}

mtcars %>%
  my_summarise(c(am, cyl), where(is.numeric))

#> # A tibble: 6 x 11
#> # Groups:   am [2]
#>       am    cyl mean_mpg mean_disp mean_hp mean_drat mean_wt
#>   <dbl> <dbl>   <dbl>     <dbl>   <dbl>   <dbl>   <dbl>
#> 1     0     4    22.9      136.    84.7     3.77    2.94
#> 2     0     6    19.1      205.   115.     3.42    3.39
#> 3     0     8    15.0      358.   194.     3.12    4.10
#> # ... with 3 more rows, and 2 more variables: mean_gear <dbl>,

```

- 传递列名向量，同时修改列名

```
var_summary = function(data, var) {  
  data %>%  
    summarise(n = n(),  
              !!enquo(var) := mean(.data[[var]]))  
}  
mtcars %>%  
  group_by(cyl) %>%  
  var_summary("mpg")  
#> # A tibble: 3 x 3  
#>   cyl      n    mpg  
#>   <dbl> <int> <dbl>  
#> 1     4     11  26.7  
#> 2     6      7  19.7  
#> 3     8     14  15.1
```

```

var_summary = function(data, var) {
  data %>%
    summarise(n = n(),
              !!str_c("mean_", var) := mean(.data[[var]]))
}

mtcars %>%
  group_by(cyl) %>%
  var_summary("mpg")
#> # A tibble: 3 x 3
#>   cyl      n mean_mpg
#>   <dbl> <int>   <dbl>
#> 1     4    11    26.7
#> 2     6     7    19.7
#> 3     8    14    15.1

```


2. 引用与反引用

引用与反引用将冻结和注入分成两步，在使用上更加灵活：

- 用 `enquo()` 让函数自动引用其参数
- 用 `!!` 反引用该参数

- 以自定义计算分组均值函数为例：

```
grouped_mean = function(data, summary_var, group_var) {  
  summary_var = enquo(summary_var)  
  group_var = enquo(group_var)  
  data %>%  
    group_by(!!group_var) %>%  
    summarise(mean = mean(!!summary_var))  
}  
  
grouped_mean(mtcars, mpg, cyl)  
#> # A tibble: 3 x 2  
#>   cyl  mean  
#>   <dbl> <dbl>  
#> 1     4  26.7  
#> 2     6  19.7  
#> 3     8  15.1
```

- 要想修改结果列名, 可借助 `as_label()` 从引用中提取名字

```
grouped_mean = function(data, summary_var, group_var) {  
  summary_var = enquo(summary_var)  
  group_var = enquo(group_var)  
  
  summary_nm = str_c("mean_", as_label(summary_var))  
  group_nm = str_c("group_", as_label(group_var))  
  
  data %>%  
    group_by(!!group_nm := !!group_var) %>%  
    summarise(!!summary_nm := mean(!!summary_var))  
}
```

```
grouped_mean(mtcars, mpg, cyl)
```

```
#> # A tibble: 3 x 2
```

```
#>   group_cyl mean_mpg
```

```
#>   <dbl>    <dbl>
```

```
#> 1      4    26.7
```

```
#> 2      6    19.7
```

```
#> 3      8    15.1
```

- 要传递多个参数可以用特殊参数...。比如，还想让用于计算分组均值的 `group_var` 可以是任意多个

```
grouped_mean = function(.data, .summary_var, ...) {  
  summary_var = enquo(.summary_var)  
  .data %>%  
    group_by(...) %>%  
    summarise(mean = mean(!!summary_var))  
}
```

```
grouped_mean(mtcars, disp, cyl, am)
```

```
#> # A tibble: 6 x 3
```

```
#> # Groups:   cyl [3]
```

```
#>   cyl    am  mean
```

```
#>   <dbl> <dbl> <dbl>
```

```
#> 1     4     0 136.
```

```
#> 2     4     1  93.6
```

```
#> 3     6     0 205.
```

```
#> # ... with 3 more rows
```

- ... 参数不需要做引用和反引用就能正确工作，但若要修改结果列名仍需要借助引用和反引用，但是要用 `enques()` 和!!!

```
grouped_mean = function(.data, .summary_var, ...) {  
  summary_var = enquos(.summary_var)  
  group_vars = enquos(..., .named = TRUE)  
  summary_nm = str_c("avg_", as_label(summary_var))  
  names(group_vars) = str_c("groups_", names(group_vars))  
  .data %>%  
    group_by(!!!group_vars) %>%  
    summarise(!!summary_nm := mean(!!summary_var))  
}
```

```
grouped_mean(mtcars, disp, cyl, am)
#> # A tibble: 6 x 3
#> # Groups:   groups_cyl [3]
#>   groups_cyl groups_am avg_disp
#>         <dbl>     <dbl>   <dbl>
#> 1         4         0    136.
#> 2         4         1     93.6
#> 3         6         0    205.
#> # ... with 3 more rows
```


- 参数... 还可以传递表达式

```
filter_fun = function(df, ...) {  
  filter(df, ...)  
}
```

```
mtcars %>%
```

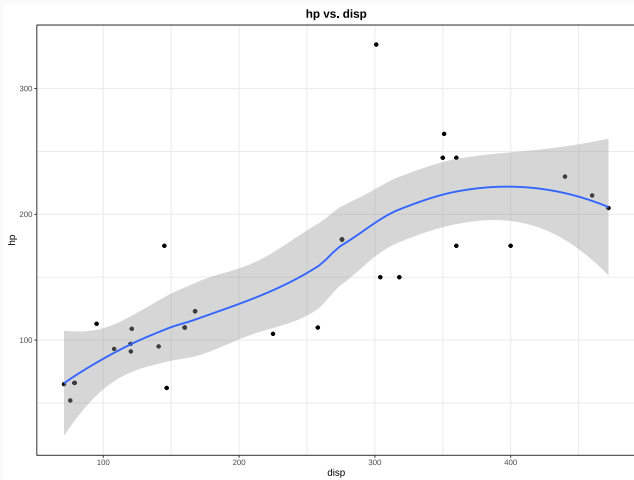
```
  filter_fun(mpg > 25 & disp > 90)
```

```
#>           mpg cyl  disp  hp drat   wt  qsec vs am gear  
#> Porsche 914-2 26.0   4 120.3  91 4.43 2.14 16.7  0  1     5  
#> Lotus Europa 30.4   4  95.1 113 3.77 1.51 16.9  1  1     5
```

- 最后看一个自定义绘制散点图的模板函数

```
scatter_plot = function(df, x_var, y_var) {  
  x_var = enquo(x_var)  
  y_var = enquo(y_var)  
  ggplot(data = df, aes(x = !!x_var, y = !!y_var)) +  
    geom_point() +  
    theme_bw() +  
    theme(plot.title = element_text(lineheight = 1,  
                                     face = "bold",  
                                     hjust = 0.5)) +  
    geom_smooth() +  
    ggtitle(str_c(as_label(y_var), " vs. ", as_label(x_var)))  
}
```

```
scatter_plot(mtcars, disp, hp)
```



本篇主要参阅 (张敬信, 2022), (Lionel Henry, 2020), 以及包文档、R-Bloggers 文章, 模板感谢 (黄湘云, 2021), (谢益辉, 2021).

参考文献

Lionel Henry, H. W. (2020). *Tidy evaluation*. tidyverse.org.

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*.