

Tidyverse 优雅编程：从向量化、泛函式到数据思维（加量版）

狗熊会案例教学线上研讨

张敬信 博士，副教授

2021 年 12 月 17 日 20:00

哈尔滨商业大学 数学与应用数学系

一. Tidyverse 简介

Tidyverse([Hadley Wickham, 2021](#)) 包是 Hadley Wickham 及团队的集大成之作，是专为数据科学而开发的一系列包的合集，基于整洁数据，提供了一致的底层设计哲学、一致的语法、一致的数据结构。

Tidyverse 用“现代的”、“优雅的”方式，以管道式、泛函式编程技术实现了数据科学的整个流程：数据导入、数据清洗、数据操作、数据可视化、数据建模、可重现与交互报告。

Tidyverse 操作数据的优雅，就体现在：

- 每一步要“做什么”，就写“做什么”，用管道依次做下去，得到最终结果
- 代码读起来，就像是在读文字叙述一样，顺畅自然，毫无滞涩

ggplot2 曾经是 R 语言的一张名片，受到广泛的赞誉；从与时俱进的角度来说，tidyverse 应该成为如今 R 语言的一张名片！

Core Tidy Workflow

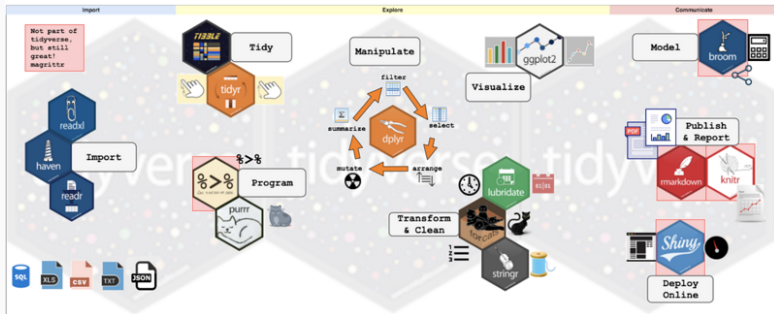


图 1: Tidyverse 核心工作流

加载包

```
library(tidyverse)
```

2021 年, Business Science 公司创始人, 数据科学专家, 也是金融时间序列领域 tidyquant, timetk, modeltime 等包的作者, Matt Dancho 发表博文: R **用于研究**, Python **用于生产** (译) ([Matt Dancho, 2021](#)), 其中这样总结:

对于做研究来说, *R* (*tidyverse*) 是非凡的: 做可视化, 数据洞见, 生成报告以及用 *shiny* 制作 MVP 级应用。从概念 (想法) 到执行 (代码), *R* 用户完成这些任务往往能够比 *Python* (*Pandas*) 用户更快 3 到 5 倍, 从而使研究工作的效率很高。

base R? 可能比 *pandas* 要低的多, 从国内口碑可见一斑。

二. Tidyverse 优雅编程思维

我理解的 tidyverse 优雅编程思维：

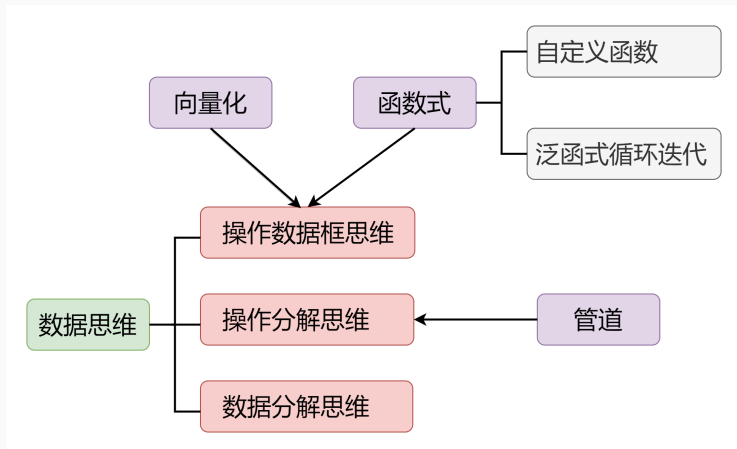


图 2: Tidyverse 优雅编程思维

0. 数据结构

- 数据结构是编程的基础；数据结构是为了便于存储不同类型的数据而设计的数据容器。
- 学习数据结构，就是要把各个数据容器的特点、适合存取什么样的数据理解透彻，只有这样才能在实际中选择最佳的数据容器，数据容器选择的合适与否，直接关系到代码是否高效简洁，甚至能否解决问题。
- 数据框是具有矩阵形式的列表，每列是一个变量（向量），每行是一个样本。
- R 用户需要：**发挥数据框操作数据的优势，避免 `for` 循环逐元素操作、再拼接为数据框**

若从整体的角度来考量，引入矩阵和向量：

$$A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}, \quad x = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad b = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

则可以向量化表示为：

$$Ax = b$$

例 1 计算样本经验熵

```
# 西瓜数据
y = c(rep(" 好瓜", 8), rep(" 坏瓜", 9))
table(y)                                # 计算各类频数，得到向量
#> y
#> 好瓜 坏瓜
#>      8      9
p = table(y) / length(y)  # 向量除以标量
p
#> y
#> 好瓜 坏瓜
#> 0.471 0.529
```

```
log2(p)
```

函数作用向量

```
#> y
```

```
#>   好瓜   坏瓜
```

```
#> -1.087 -0.918
```

```
p * log2(p)
```

向量乘以向量，对应元素做乘法

```
#> y
```

```
#>   好瓜   坏瓜
```

```
#> -0.512 -0.486
```

```
- sum(p * log2(p))
```

向量求和

```
#> [1] 0.998
```

2. 自定义函数

- 要做一件事，拿一个简单实例，调试通过；再改写成函数，就可以一步到位、复用、批量地用、给别人用
- 编程中的函数，其一般形式为：

$$(\text{返回值}1 \dots, \text{返回值}m) = \text{函数名}(\text{输入}1, \dots, \text{输入}n)$$

- 你只要把输入给它，它就能在内部进行相应处理，把你想要的返回值给你
- 这些输入和返回值，在函数定义时，都要有固定的类型（模具）限制，叫作形参（形式上的参数）；在函数调用时，必须给它对应类型的具体数值，才能真正的去做处理，这叫作实参（实际的参数）。
- 定义函数就好比创造一个模具，调用函数就好比用模具批量生成产品

例 2 向量归一化

- 以做归一化为例，设计简单实例 $x = [15, 8, 26, \text{NA}, 12]$:

```
x = c(15, 8, 26, NA, 12)
type = "pos" # 标记正向指标
rng = range(x, na.rm = TRUE) # 计算最小值最大值
if(type == "pos") {
  (x - rng[1]) / (rng[2] - rng[1])
} else {
  (rng[2] - x) / (rng[2] - rng[1])
}
#> [1] 0.389 0.000 1.000    NA 0.222
```

- 改写（封装）为函数

```
Rescale = function(x, type = "pos") {  
  rng = range(x, na.rm = TRUE)  # 计算最小值最大值  
  if(type == "pos") {  
    (x - rng[1]) / (rng[2] - rng[1])  
  } else {  
    (rng[2] - x) / (rng[2] - rng[1])  
  }  
}  
  
Rescale(x)  
#> [1] 0.389 0.000 1.000    NA 0.222  
  
Rescale(x, "neg")  
#> [1] 0.611 1.000 0.000    NA 0.778
```

3. 泛函式循环迭代

- 数学上，函数的函数称为泛函；编程中，表示函数作用在函数上，或者说函数包含其他函数作为参数
- 循环迭代，本质上就是将一个函数依次应用（映射）到序列的每一个元素上，表示出来即 `purrr::map_*(x, f)`
- 两点说明：
 - 序列：由一系列可以根据位置索引的元素构成，元素可以很复杂和不同类型；向量、列表、数据框都是序列
 - 将 `x` 作为第一个参数，是便于使用管道

- purrr 泛函式编程解决循环迭代问题的逻辑：
 - 针对序列每个单独的元素，怎么处理它得到正确的结果，将之定义为函数，再 map 到序列中的每一个元素，将得到的多个结果¹打包到一起返回，并且可以根据想让结果返回什么类型选用 map 后缀。
- 循环迭代返回类型的控制：
 - map_chr, map_lgl, map_dbl, map_int: 返回相应类型向量
 - map_dfr, map_dfc: 返回数据框列表，再按行、按列合并为一个数据框

¹每个元素作用后返回一个结果.

- purrr 风格公式 (匿名函数): 函数参数 .f 的一种简写; 只需要写清楚它是如何操作序列参数 .x 的
 - 一元函数序列参数为 .x, 例如 $f(x) = x^2 + 1$ 表示为 .f = ~ .x ^ 2 + 1
 - 二元函数序列参数为 .x, .y, 例如 $f(x, y) = x^2 - 3y$ 表示为 .f = ~ .x ^ 2 - 3 * .y
 - 还有三元函数序列参数: ..1, ..2, ..3, 所有序列参数...

注: .x 是序列中的一个 (代表) 元素。这也是**分解**的思维, 循环迭代要依次对序列中每个元素做某操作, 只需要把对一个元素做的操作写清楚 (即 .f), 剩下的交给 map_*() 就行了。

- `map_*(.x, .f, ...)`: 依次应用一元函数 `.f` 到一个序列 `.x` 的每个元素, ... 可设置 `.f` 的其它参数

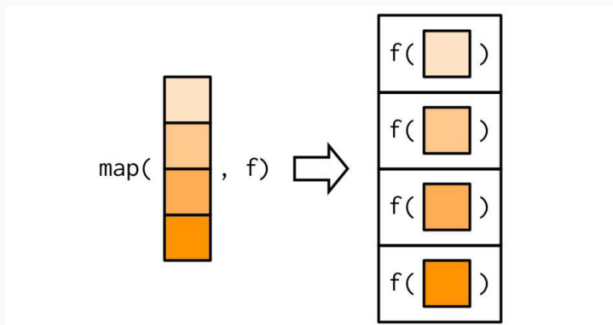


图 3: `map` 函数作用机制示意图

- `map2_*(.x, .y, .f, ...)`: 依次应用二元函数 `.f` 到两个序列 `.x`, `.y` 的每对元素, `...` 可设置 `.f` 的其它参数

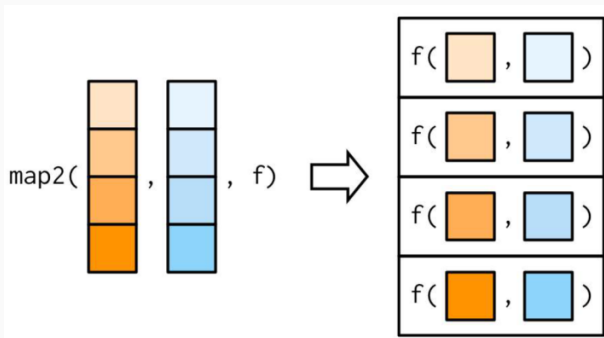


图 4: `map2` 函数作用机制示意图

- `pmap_*(.l, .f, ...)`: 依次应用多元函数 `.f` 到多个序列 `.l` 的每层元素, 可实现对数据框逐行迭代, ... 可设置 `.f` 的其它参数

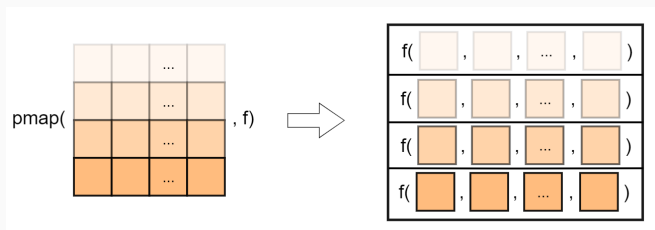


图 5: `pmap` 函数作用机制示意图

- 只循环迭代做事不返回结果, 还有 `walk_*` 系列, 比如批量保存数据/图形到文件; 元素与索引一起迭代有 `imap_*` 系列

例 3 对数据框逐列迭代

- 数据框是序列，第 1 个元素是第 1 列 `df[[1]]`，第 2 个元素是第 2 列 `df[[2]]`，.....

```
df = iris[,1:4]
map_dbl(df, mean)          # 求各列均值
#> Sepal.Length Sepal.Width Petal.Length  Petal.Width
#>           5.84           3.06           3.76           1.20
map_chr(df, mean)
#> Sepal.Length Sepal.Width Petal.Length  Petal.Width
#>  "5.843333"   "3.057333"   "3.758000"   "1.199333"
```

- 对各列做归一化, 若均为正向指标:

```
map_dfc(df, Rescale)
```

```
#> # A tibble: 150 x 4
```

```
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width
```

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

```
#> 1         0.222         0.625         0.0678        0.0417
```

```
#> 2         0.167         0.417         0.0678        0.0417
```

```
#> 3         0.111         0.5          0.0508        0.0417
```

```
#> 4         0.0833        0.458         0.0847        0.0417
```

```
#> 5         0.194         0.667         0.0678        0.0417
```

```
#> # ... with 145 more rows
```

```
# 同 map_dfc(df, Rescale, type = "pos")
```

```
# 同 map_dfc(df, ~ Rescale(.x, "pos"))
```

- 对各列做归一化, 若各列分别为正向, 负向, 负向, 正向

```
type = c("pos", "neg", "neg", "pos")
```

```
map2_dfc(df, type, Rescale)
```

```
#> # A tibble: 150 x 4
```

```
#>   Sepal.Length Sepal.Width Petal.Length Petal.Width
```

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

```
#> 1      0.222      0.375      0.932      0.0417
```

```
#> 2      0.167      0.583      0.932      0.0417
```

```
#> 3      0.111      0.5      0.949      0.0417
```

```
#> 4      0.0833     0.542      0.915      0.0417
```

```
#> 5      0.194      0.333      0.932      0.0417
```

```
#> # ... with 145 more rows
```

例 4 对数据框逐行迭代

```
pmap_dbl(df[1:10,], ~ mean(c(...)))      # 逐行平均
#> [1] 2.55 2.38 2.35 2.35 2.55 2.85 2.42 2.52 2.23 2.40
map_dbl(asplit(df[1:10,], 1), mean)
#>   1    2    3    4    5    6    7    8    9   10
#> 2.55 2.38 2.35 2.35 2.55 2.85 2.42 2.52 2.23 2.40
pmap_dbl(df[1:10,], max)                  # 逐行最大
#> [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
apply(df[1:10,], 1, max)
#>   1    2    3    4    5    6    7    8    9   10
#> 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
```

例 5 批量读取数据并按行合并

TidyProgramming > datas > read_datas			▼	🔄	
^	名称	^			
	新建文件夹	2021/11/27 17:22			
	六1班学生成绩.csv	2021/8/18 22:36			
	六3班学生成绩.csv	2021/8/18 22:36			
	六4班学生成绩.csv	2021/8/18 22:37			
	六5班学生成绩.csv	2021/8/18 22:37			

图 6: pmap 函数作用机制示意图

- 获取文件路径
- 循环迭代读取，同时合并结果


```
files = list.files("datas/read_datas", pattern = "csv",  
                  full.names = TRUE, recursive = TRUE)  
  
files  
#> [1] "datas/read_datas/六 1 班学生成绩.csv"  
#> [2] "datas/read_datas/六 3 班学生成绩.csv"  
#> [3] "datas/read_datas/六 4 班学生成绩.csv"  
#> [4] "datas/read_datas/六 5 班学生成绩.csv"  
#> [5] "datas/read_datas/新建文件夹/六 2 班学生成绩.csv"
```

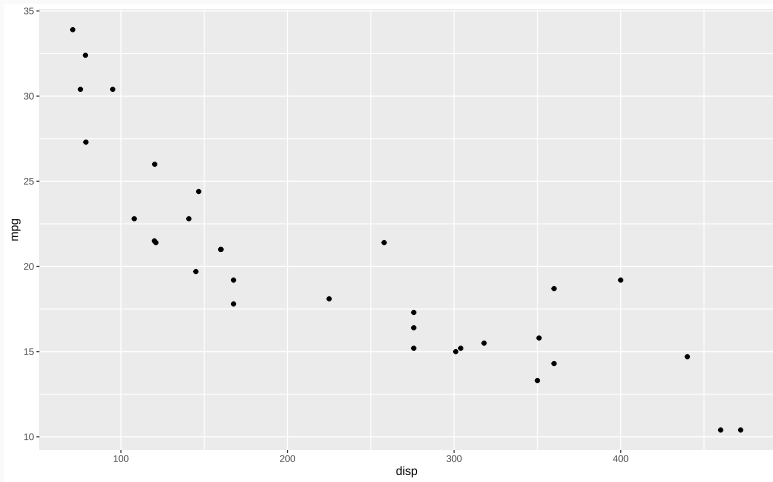
```
map_dfr(files, read_csv)
#> # A tibble: 20 x 6
#>   班级  姓名  性别  语文  数学  英语
#>   <chr> <chr> <chr> <dbl> <dbl> <dbl>
#> 1 六 1 班 何娜 女      87     92     79
#> 2 六 1 班 黄才菊 女      95     77     75
#> 3 六 1 班 陈芳妹 女      79     87     66
#> 4 六 1 班 陈学勤 男      82     79     66
#> 5 六 3 班 江佳欣 女      80     69     75
#> # ... with 15 more rows
```

例 6 批量绘图并保存图片

以 `mtcars` 为例，用不同的数值列作为 `x` 轴，以 `mpg` 列作为 `y` 轴，批量绘制散点图，并保存为以列名命名的 `png` 文件。

- 需要在多个列迭代，将列名作为传递参数是个好主意，还可以用来命名 `png` 文件。
- 先对一个列名完成绘制散点图

```
x = "disp"
mtcars %>%
  ggplot(aes(.data[[x]], mpg)) + # 管道中列名传参方式
  geom_point()
```



- 改写为函数

```
plot_scatter = function(x) {  
  mtcars %>%  
    ggplot(aes(.data[[x]], mpg)) +  
    geom_point()  
}
```

- 批量绘图

```
cols = names(mtcars)[2:7]      # 要绘制的多个列名  
ps = map(cols, plot_scatter)    # 批量绘图
```

- 批量导出到 png 文件

```
files = str_c("images/", cols, ".png") # 准备多个文件路径  
walk2(files, ps, ggsave)
```



4. 管道

- magrittr 包引入了管道操作 %>% (现在 R 4.1 也开始支持管道 |>), 能够通过管道将数据从一个函数传给另一个函数, 从而用若干函数构成的管道依次变换你的数据

```
x %>% f() %>% g()
```

- 表示依次对数据进行若干操作: 先对数据 x 进行 f 操作, 接着对结果数据进行 g 操作
- 使用管道的好处:
 - 避免使用过多的中间变量
 - 程序可读性大大增强: 对数据集依次进行一系列操作
- 数据默认传递给下一个函数的第一参数, 否则需用 . 代替数据

5. 数据思维 I: 操作数据框的思维

- 将向量化和函数式（自定义函数 + 泛函式循环迭代）编程思维，纳入到数据框中来：
 - 向量化编程同时操作一个向量的数据，变成在数据框中操作一系列的数据，或者同时操作数据框的多列，甚至分别操作数据框每个分组的多列；
 - 函数式编程变成想做的操作自定义函数（或现成函数），再依次应用到数据框的多个列上，以修改列或做汇总

注：数据框是指 tibble, 支持列表列（嵌套数据框）；

- 记住：**每次至少操作一列数据！**

```
df = as_tibble(iris) %>%  
  set_names(str_c("x", 1:4), "Species")
```


例 7 操作数据框

- 操作一列 (计算一个新列)

```
df %>%  
  mutate(x1 = x1 * 10)  
#> # A tibble: 150 x 5  
#>       x1     x2     x3     x4 Species  
#>   <dbl> <dbl> <dbl> <dbl> <fct>  
#> 1     51   3.5   1.4   0.2 setosa  
#> 2     49    3    1.4   0.2 setosa  
#> 3     47   3.2   1.3   0.2 setosa  
#> 4     46   3.1   1.5   0.2 setosa  
#> 5     50   3.6   1.4   0.2 setosa  
#> # ... with 145 more rows
```

```
df %>%
  mutate(avg = pmap_dbl(.[1:4], ~ mean(c(...))))
#> # A tibble: 150 x 6
#>       x1     x2     x3     x4 Species   avg
#>   <dbl> <dbl> <dbl> <dbl> <fct>   <dbl>
#> 1   5.1   3.5   1.4   0.2 setosa   2.55
#> 2   4.9    3   1.4   0.2 setosa   2.38
#> 3   4.7   3.2   1.3   0.2 setosa   2.35
#> 4   4.6   3.1   1.5   0.2 setosa   2.35
#> 5    5    3.6   1.4   0.2 setosa   2.55
#> # ... with 145 more rows
```

- 操作多列

```
df %>%
```

```
  mutate(across(1:4, ~ .x * 10))
```

```
#> # A tibble: 150 x 5
```

```
#>       x1     x2     x3     x4 Species
```

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

```
#> 1     51     35     14      2 setosa
```

```
#> 2     49     30     14      2 setosa
```

```
#> 3     47     32     13      2 setosa
```

```
#> 4     46     31     15      2 setosa
```

```
#> 5     50     36     14      2 setosa
```

```
#> # ... with 145 more rows
```

```
df %>%  
  mutate(across(1:4, Rescale))  
#> # A tibble: 150 x 5  
#>       x1      x2      x3      x4 Species  
#>   <dbl> <dbl> <dbl> <dbl> <fct>  
#> 1 0.222  0.625 0.0678 0.0417 setosa  
#> 2 0.167  0.417 0.0678 0.0417 setosa  
#> 3 0.111  0.5    0.0508 0.0417 setosa  
#> 4 0.0833 0.458 0.0847 0.0417 setosa  
#> 5 0.194  0.667 0.0678 0.0417 setosa  
#> # ... with 145 more rows
```

6. 数据思维 II: 操作分解的思维

- 复杂数据操作都可以分解为若干简单的基本数据操作：
 - 数据连接
 - 数据重塑（长宽转换、拆分/合并列）
 - 筛选行
 - 排序行
 - 选择列
 - 修改列
 - 分组汇总
- 一旦完成问题的梳理和分解，又熟悉每个基本数据操作，用“管道”流依次对数据做操作即可

例 8 管道分解操作

```
load("datas/SQL50datas.rda")
head(score, 2)
#> # A tibble: 2 x 3
#>   学号  课程编号  成绩
#>   <chr> <chr>    <dbl>
#> 1 01     01         80
#> 2 01     02         90
head(student, 2)
#> # A tibble: 2 x 4
#>   学号  姓名  生日      性别
#>   <chr> <chr> <chr>    <chr>
#> 1 01    赵雷 1990-01-01 男
#> 2 02    钱电 1990-12-21 男
```

- **问题：**查询平均成绩 ≥ 60 分的学生学号、姓名和平均成绩。

- **分解问题：**

- 先按学号分组汇总计算平均成绩
- 然后根据条件筛选行
- 再根据学号连接学生信息
- 最后选择想要的列

```
score %>%  
  group_by(学号) %>%  
  summarise(平均成绩 = mean(成绩)) %>%  
  filter(平均成绩 >= 60) %>%  
  left_join(student, by = "学号") %>%  
  select(学号, 姓名, 平均成绩)
```

```
#> # A tibble: 5 x 3
```

```
#>   学号  姓名  平均成绩
```

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

```
#> 1 01    赵雷    89.7
```

```
#> 2 02    钱电    70
```

```
#> 3 03    孙风    80
```

```
#> 4 05    周梅    81.5
```

```
#> 5 07    郑竹    93.5
```


7. 数据思维 III: 数据分解的思维

- (1) **分组修改**: 想对数据框进行分组, 分别对每组数据做操作, 整体来想这是不容易想透的复杂事情, 实际上只需做 `group_by()` 分组, 然后把你要对一组数据做的操作实现
- `group_by + summarise`: 分组汇总, 结果是“**有几个分组就有几个样本**”
 - `group_by + mutate`: 分组修改, 结果是“**原来几个样本还是几个样本**”
- (2) **同时操作多列**: `across()` 同时操作多列, 实际上只需把对一列要做的操作实现

(3) `if_any()/if_all()`: 是配合 `filter()` 根据多列的值, 存在/都满足某条件筛选行, 你只需要把该条件表示清楚

(4) `slider::slide_*()`: 滑窗迭代, 构造想要的滑动窗口, 你只需要思考对每个窗口数据做什么操作写成函数

这些都是数据分解的操作思维, 这些函数会帮你**分解 + 分别操作 + 合并结果**, 你只需要关心**分别操作**的部分, 它就是一件简单的事情。

例 9 分组修改数据

```
load("datas/stocks.rda")
stocks
#> # A tibble: 753 x 3
#>   Date      Stock  Close
#>   <date>    <chr> <dbl>
#> 1 2017-01-03 Google  786.
#> 2 2017-01-03 Amazon  754.
#> 3 2017-01-03 Apple   116.
#> 4 2017-01-04 Google  787.
#> 5 2017-01-04 Amazon  757.
#> # ... with 748 more rows
```

- **问题：**分别计算每支股票的收盘价与前一天的差价。

- **分解的逻辑：**只要对 Stock 分组，对一组数据（一支股票）怎么计算收盘价与前一天的差价，就怎么写代码即可：

```
stocks %>%  
  group_by(Stock) %>%  
  mutate(delta = Close - lag(Close))  
#> # A tibble: 753 x 4  
#> # Groups:   Stock [3]  
#>   Date      Stock  Close  delta  
#>   <date>    <chr> <dbl> <dbl>  
#> 1 2017-01-03 Google  786.  NA  
#> 2 2017-01-03 Amazon  754.  NA  
#> 3 2017-01-03 Apple   116.  NA  
#> 4 2017-01-04 Google  787.  0.760  
#> 5 2017-01-04 Amazon  757.  3.51  
#> # ... with 748 more rows
```

例 10 数据重塑 + 批量 t 检验

```
dt = read_csv("datas/demo_t.test.csv")
dt
#> # A tibble: 38 x 7
#>   compoundID case_1 case_2 case_3 control_1 control_2 cont
#>   <chr>         <dbl> <dbl> <dbl>         <dbl>         <dbl>
#> 1 com_001         485   154   268           350           432
#> 2 com_002         208   372   219           457           324
#> 3 com_003         219   125   345           473           480
#> 4 com_004         289   356   116           489           376
#> 5 com_005         248   456   279           457           426
#> # ... with 33 more rows
```

问题：数据共 38 行，每行是一组，包括 3 个实验样本，3 个控制样本；需求是批量地对每行，按实验组和控制组做 t 检验。

Tidyverse 能够优雅操作的都是整洁数据。什么是整洁数据？

- 每个变量构成一列
- 每个观测构成一行
- 每个观测的每个变量值构成一个单元格

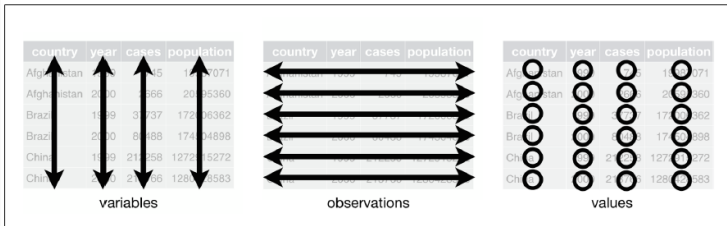


图 7: 整洁数据的 3 个特点

- 第1次宽变长

```
dt = dt %>%  
  pivot_longer(-1, names_pattern = "(.*)_",  
               names_to = ".value")
```

```
dt
```

```
#> # A tibble: 114 x 3  
#>   compoundID case control  
#>   <chr>      <dbl>   <dbl>  
#> 1 com_001      485     350  
#> 2 com_001      154     432  
#> 3 com_001      268     425  
#> 4 com_002      208     457  
#> 5 com_002      372     324  
#> # ... with 109 more rows
```

- 第2次宽变长

```
dt = dt %>%  
  pivot_longer(-1, names_to = "grp", values_to = "val")  
dt  
#> # A tibble: 228 x 3  
#>   compoundID grp      val  
#>   <chr>      <chr>  <dbl>  
#> 1 com_001    case    485  
#> 2 com_001    control 350  
#> 3 com_001    case    154  
#> 4 com_001    control 432  
#> 5 com_001    case    268  
#> # ... with 223 more rows
```


- group_by 分组 + t 检验

```
library(rstatix)  # 整洁统计
```

```
dt %>%
```

```
  group_by(compoundID) %>%
```

```
  t_test(val ~ grp)
```

```
#> # A tibble: 38 x 9
```

```
#>   compoundID .y.   group1 group2     n1     n2 statistic
```

```
#> * <chr>      <chr> <chr>  <chr>   <int> <int>      <dbl> <dbl>
```

```
#> 1 com_001    val   case   control     3     3    -0.994  2.0
```

```
#> 2 com_002    val   case   control     3     3    -1.91   3.0
```

```
#> 3 com_003    val   case   control     3     3    -3.25   2.0
```

```
#> 4 com_004    val   case   control     3     3    -2.46   3.0
```

```
#> 5 com_005    val   case   control     3     3    -1.71   2.0
```

```
#> # ... with 33 more rows
```

例 11 across 同时修改多列

注：f将长度为n的向量，映射为长度为n的向量

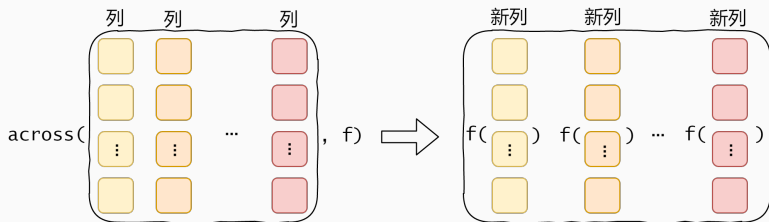


图 8: across 函数作用机制示意图

- **问题：**对数据框的数值列做归一化。
- **分解的逻辑：**定义函数对一系列数据做归一化，剩下的交给 `across()`

```
df %>%  
  mutate(across(where(is.numeric), Rescale))  
#> # A tibble: 150 x 5  
#>       x1      x2      x3      x4 Species  
#>   <dbl> <dbl> <dbl> <dbl> <fct>  
#> 1 0.222  0.625 0.0678 0.0417 setosa  
#> 2 0.167  0.417 0.0678 0.0417 setosa  
#> 3 0.111  0.5    0.0508 0.0417 setosa  
#> 4 0.0833 0.458 0.0847 0.0417 setosa  
#> 5 0.194  0.667 0.0678 0.0417 setosa  
#> # ... with 145 more rows
```

例 12 根据多列筛选行

- 选择 1:4 列存在值 > 7.5 的行

```
df %>%  
  filter(if_any(1:4, ~ .x > 7.5))  
#> # A tibble: 6 x 5  
#>       x1     x2     x3     x4 Species  
#>   <dbl> <dbl> <dbl> <dbl> <fct>  
#> 1   7.6    3     6.6    2.1 virginica  
#> 2   7.7    3.8    6.7    2.2 virginica  
#> 3   7.7    2.6    6.9    2.3 virginica  
#> 4   7.7    2.8    6.7    2    virginica  
#> 5   7.9    3.8    6.4    2    virginica  
#> # ... with 1 more row
```

- 选择 1:4 列所有值 > 2.4 的行

```
df %>%
```

```
  filter(if_all(1:4, ~ .x > 2.4))
```

```
#> # A tibble: 3 x 5
```

```
#>       x1     x2     x3     x4 Species
```

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

```
#> 1   6.3   3.3    6     2.5 virginica
```

```
#> 2   7.2   3.6   6.1     2.5 virginica
```

```
#> 3   6.7   3.3   5.7     2.5 virginica
```

例 13 计算每支股票收盘价的 3 日滑动平均

```
library(slider)
df = group_by(stocks, Stock)
slide(1:5, ~ .x, .before = 1, .after = 1)
#> [[1]]
#> [1] 1 2
#>
#> [[2]]
#> [1] 1 2 3
#>
#> [[3]]
#> [1] 2 3 4
#>
#> [[4]]
#> [1] 3 4 5
#>
#> [[5]]
```

- 按样本

```
df %>%
```

```
  mutate(avg_3 = slide_dbl(Close, mean,  
                           .before = 1, .after = 1))
```

```
#> # A tibble: 753 x 4
```

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

```
#>   Date          Stock  Close avg_3
```

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

```
#> 1 2017-01-03 Google   786.   787.
```

```
#> 2 2017-01-03 Amazon   754.   755.
```

```
#> 3 2017-01-03 Apple    116.   116.
```

```
#> 4 2017-01-04 Google   787.   789.
```

```
#> 5 2017-01-04 Amazon   757.   764.
```

```
#> # ... with 748 more rows
```

- 按日期

```
df %>%
```

```
  mutate(avg_3 = slide_index_dbl(Close, Date, mean,  
                                  .before = 1, .after = 1))
```

```
#> # A tibble: 753 x 4
```

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

```
#>   Date      Stock  Close avg_3
```

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

```
#> 1 2017-01-03 Google  786.   787.
```

```
#> 2 2017-01-03 Amazon  754.   755.
```

```
#> 3 2017-01-03 Apple   116.   116.
```

```
#> 4 2017-01-04 Google  787.   789.
```

```
#> 5 2017-01-04 Amazon  757.   764.
```

```
#> # ... with 748 more rows
```


例 14 综合案例：朝阳医院销售数据探索

- 读取 Excel 数据

```
library(readxl)      # 读取 Excel
library(lubridate)    # 处理日期时间
library(naniar)       # 探索处理缺失值
df = read_xlsx("datas/朝阳医院 2018 年销售数据.xlsx")
head(df, 3)
```

#> # A tibble: 3 x 7

#>	购药时间	社保卡号	商品编码	商品名称	销售
#>	<chr>	<chr>	<chr>	<chr>	<
#> 1	2018-01-01 星期五	001616528	236701	强力 VC 银翘片	
#> 2	2018-01-02 星期六	001616528	236701	清热解毒口服 ~	
#> 3	2018-01-06 星期三	0012602828	236701	感康	

- 购药时间列重命名为销售时间
- 删除前两列存在缺失的行
- 对数值列做均值插补
- 销售时间列从字符型转化为日期型
- 删除销售时间列为缺失的行
- 按根据销售时间对行排序
- 删除销售数量为负数的行

df %>%

rename(销售时间 = 购药时间) %>%	# 重命名列
filter(!if_any(1:2, is.na)) %>%	# 删除缺失值
mutate(across(5:7, impute_mean)) %>%	# 插补缺失值
mutate(销售时间 = ymd(销售时间)) %>%	# 修改日期列
drop_na(销售时间) %>%	# 删除缺失行
arrange(销售时间) %>%	# 按行排序
filter(销售数量 > 0)	# 删除异常值行

```

#> # A tibble: 6,509 x 7
#>   销售时间   社保卡号   商品编码 商品名称   销售数量 应中
#>   <date>    <chr>        <chr>   <chr>        <dbl>    <dbl>
#> 1 2018-01-01 001616528   236701 强力VC银翘片         6      8
#> 2 2018-01-01 00101470528 236709 心痛定             4     179
#> 3 2018-01-01 0010072612028 2367011 开博通             1      2
#> 4 2018-01-01 0010074599128 2367011 开博通             5     14
#> 5 2018-01-01 0011743428   861405 苯磺酸氨氯地平片~     1
#> # ... with 6,504 more rows

```

最后，总结一下贯穿始终的**分解**思维：

- 解决无从下手的复杂问题：分解为若干可上手的简单问题
- 循环迭代：分解为把解决一个元素的过程写成函数，再 map 到一系列的元素
- 复杂的数据操作：分解为若干简单数据操作，再用管道连接
- 操作多组数据：分解为 group_by 分组 + 操作明白一组数据
- 修改多列：分解为 across 选择列 + 操作明白一列数据

以上内容，是我的 R 语言新书 (张敬信, 2022) 第 1、2 章所涉及编程思维的脉络梳理，感谢黄湘云 (黄湘云, 2021) 在 Github 提供的 R markdown (谢益辉, 2021) 模板。

掌握 tidyverse 优雅编程，用最 tidy 的方式学习 R 语言！
《R 语言编程：基于 tidyverse》，张敬信，人民邮电出版社

预计 2022 年上半年上市，敬请期待！

我的知乎专栏：

https://www.zhihu.com/people/huc_zhangjingxin/columns

Email: zhjx_19@hrbcu.edu.cn

注意：这只是临时封面！



群名称:tidy-R语言
群 号:875664831

读者群



哈爾濱商業大學
Harbin University of Commerce

谢 谢 观 看!



参考文献

Hadley Wickham, R. (2021). *tidyverse: R packages for data science*. R package version 1.3.1.

Matt Dancho, J. C. (2021). *R is for Research, Python is for Production*.

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

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

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