

## 第二章 数据操作

### 2.1 tidyverse 简介与管道

#### 2.1.1 tidyverse 包简介

tidyverse 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、语法、数据结构。

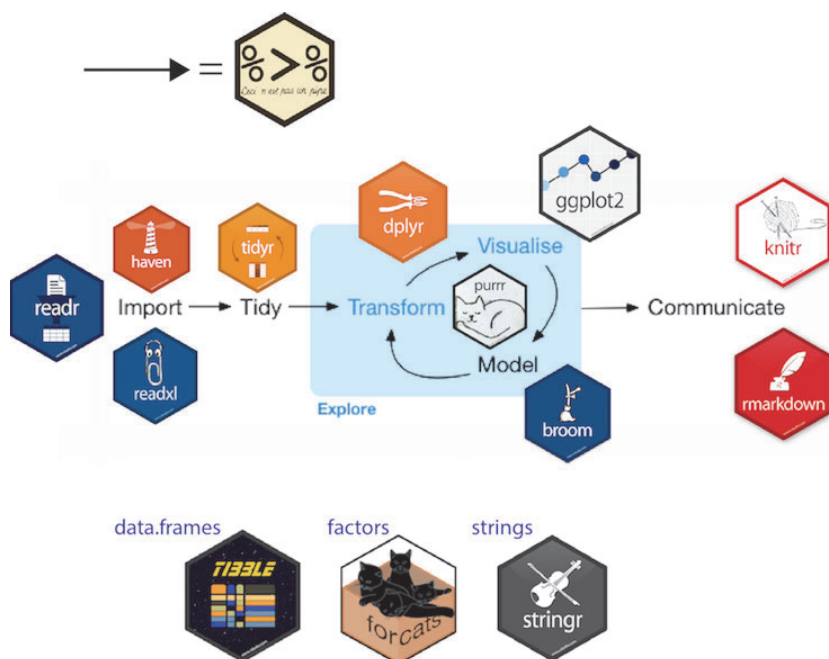


图 2.1: tidyverse 核心包

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。

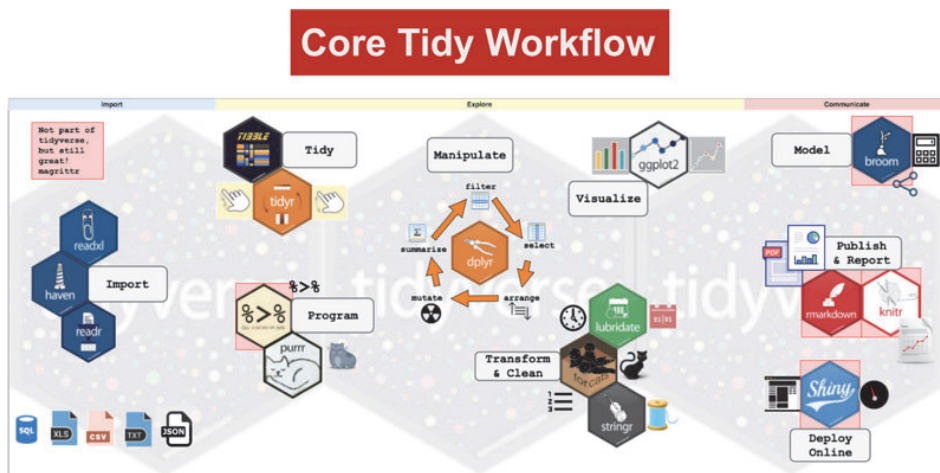


图 2.2: tidyverse 整洁 workflow

在 tidyverse 包的引领下，近年来涌现出一系列具体研究领域的 tidy\* 版本的包，比较综合的有 tidymodels (机器学习)、rstatix (统计)、tidybayes (贝叶斯模型)、tidyquant (金融)、fpp3 (时间序列)、tidytext (文本挖掘)、sf (空间数据分析)、tidybulk (生信) 等。

## tidyverse 与 data.table

tidyverse 操作数据语法优雅、容易上手，但效率与主打高效的 data.table 包不可同日而语，处理几 G 甚至几十 G 的数据，需要用 data.table。

但 data.table 的语法高度抽象、不容易上手。本书不对 data.table 做过多展开，其简单使用可参阅附录 2。另一种不错的方案是使用专门的转化包：有不少包尝试底层用 data.table，上层用 tidyverse 语法包装（转化），如 dtplyr, tidyst 等。

### 2.1.2 管道操作

#### 1. 什么是管道操作？

magrittr 包引入了管道操作，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。

例如，对数据集 mtcars，先按分类变量 cyl 分组，再对连续变量 mpg 做分组汇总计算均值：

```
library(tidyverse)
mtcars %>%
  group_by(cyl) %>%
  summarise(mpg_avg = mean(mpg))
```

```
## # A tibble: 3 x 2
##   cyl mpg_avg
##   <dbl>   <dbl>
## 1     4    26.7
## 2     6    19.7
## 3     8    15.1
```

管道运算符 %>% 的意思是：将左边的运算结果，以输入的方式传给右边函数。若干个函数通过管道链接起来，叫做管道 (pipeline)。

```
x %>% f() %>% g() # 等同于 g(f(x))
```

对该管道示例应该这样理解：依次对数据进行若干操作：先对 x 进行 f 操作，接着对结果进行 g 操作。

管道，也支持 R base 函数：

```
month.abb %>% # 内置月份名缩写字符向量
  sample(6) %>%
  tolower() %>%
  str_c(collapse = "|")
```

```
## [1] "may|jul|apr|sep|feb|nov"
```

使用管道的好处是：

- 避免使用过多的中间变量；
- 程序可读性大大增强：

管道操作的过程，读起来就是对原数据集依次进行一系列操作的过程。而非管道操作，读起来与操作的过程是相反的，比如同样实现上例：

```
str_c(tolower(sample(month.abb, 6)), collapse="|")
```

## 2. 常用管道操作

管道默认将数据传给下一个函数的第 1 个参数，且它可以省略

```
c(1, 3, 4, 5, NA) %>% mean(., na.rm = TRUE) # "." 可以省略
c(1, 3, 4, 5, NA) %>% mean(na.rm = TRUE)    # 建议写法
```

这种机制使得管道代码看起来就是：从数据开始，依次用函数对数据施加一系列的操作（变换数据），各个函数都直接从非数据参数开始写即可，而不用再额外操心数据的事情，数据会自己沿管道向前“流动”。

正是这种管道操作，使得 tidyverse 能够优雅地操作数据。

所以，tidyverse 中的函数都设计为数据作为第 1 个参数，自定义的函数也建议这样做。

### 数据可以在下一个函数中使用多次

数据经过管道默认传递给函数的第 1 个参数（通常直接省略）；若在非第 1 个参数处使用该数据，必须用“.”代替（绝对不能省略），这使得管道作用更加强大和灵活。下面看一些具体实例：

```
# 数据传递给 plot 第一个参数作为绘图数据（. 省略），
# 同时用于拼接成字符串给 main 参数用于图形标题
c(1, 3, 4, 5) %>% plot(main = str_c(., collapse=","))

# 数据传递给第二个参数 data
```

```
mtcars %>% plot(mpg ~ disp, data = .)

# 选择列
iris %>% .$Species           # 选择 Species 列内容
iris %>% .[1:3]              # 选择 1-3 列子集
```

再来看一个更复杂的例子：

```
mtcars %>%
  split(.$cyl) %>%           # . 相当于 mtcars
  map(~ lm(mpg ~ wt, data = .x))
```

`split()` 是将数据框 `mtcars` 根据其 `cyl` 列（包含 3 个水平的分类变量）分组，得到包含 3 个成分的列表；列表接着传递给 `map(.x, .f)` 的第一个参数（直接省略），`~lm(mpg ~ wt, data = .x)` 是第二参数 `.f`，为 `purrr` 风格公式写法。

整体来看，实现的是分组建模：将数据框根据分类变量分组，再用 `map` 循环机制依次对每组数据建立线性回归模型。

建议进行区分：`.` 用于管道操作中代替数据；`.x` 用于 `purrr` 风格公式（匿名函数）。

本节部分内容参阅 (Hadley Wickham 2017), (Desi Quintans 2019).

## 2.2 数据读写

### 2.2.1 数据读写的包与函数

先来罗列一下读写常见数据文件的包和函数，具体使用可查阅其帮助。

#### 1. readr 包

读写带分隔符的文本文件，如 `csv` 和 `tsv`；也能读写序列化的 R 对象 `rds`，若想保存数据集后续再加载回来，`rds` 将保存元数据和该对象的状态，如分组和数据类型。

- 读入数据到数据框：`read_csv()` 和 `read_tsv()`
- 读入欧式格式数据<sup>1</sup>：`read_csv2()` 和 `read_tsv2()`
- 读写 `rds` 数据：`read_rds()` 和 `write_rds()`
- 写出数据到文件：`write_csv()`, `write_tsv()`, `write_csv2()`, `write_tsv2()`
- 转化数据类型：`parse_number()`, `parse_logical()`, `parse_factor()` 等

<sup>1</sup>欧式格式数据以“;”为分隔符，“.”为小数位。

## 2. readxl 包

专门读取 Excel 文件，包括同一个工作簿中的不同工作表。

- `read_excel()`: 自动检测 xls 或xlsx 文件
- `read_xls()`: 读取 xls 文件
- `read_xlsx()`: 读取 xlsx 文件

读写 Excel 文件好用的包，还有 `openxlsx`。

## 3. haven 包

读写 SPSS, Stata, SAS 数据文件。

- 读: `read_spss()`, `read_dta()`, `read_sas()`
- 写: `write_spss()`, `write_stata()`, `write_sas()`

## 4. readtext 包

读取全部文本文件的内容到数据框，每个文件变成一行，常用于文本挖掘<sup>2</sup>或数据收集；`readtext` 还支持读取 csv, tab, json, xml, html, pdf, doc, docx, rtf, xls, xlsx 等。

- `readtext()`: 返回数据框，`doc_id` 列为文档标识，`text` 列为读取的全部文本内容（1 个字符串）。

```
library(readtext)
document = readtext("datas/十年一觉.txt")
document
```

```
## readtext object consisting of 1 document and 0 docvars.
## # Description: df[,2] [1 x 2]
##   doc_id      text
##   <chr>       <chr>
## 1 十年一觉.txt "\      “这位公子爷\“...”
```

### 2.2.2 数据读写实例

以读取 csv 和 Excel 文件为例演示，读取其它类型的数据文件，换成其它读取函数即可。

<sup>2</sup>做文本挖掘 R 包有 `tidytext`，中文文本挖掘相比英文多了 `jiebaR` 包分词的前期步骤。

```
read_csv(file, col_names, col_types, locale, skip, na, n_max, ...)
```

- file: 数据文件所在相对或绝对路径
- col\_names: 第一行是否作为列名
- skip: 开头跳过的行数
- na: 设置什么值解读为缺失值
- n\_max: 读取的最大行数
- col\_types: 设置列类型<sup>a</sup>, 默认 NULL (全部猜测), 可总体设置一种类型 (循环使用) 或为每列单独设置, 例如设置 3 列的列类型: `col_types="cnd"`
- locale: 设置区域语言环境 (时区, 编码方式, 小数标记、日期格式), 主要是用来设置所读取数据文件的编码方式, 如从默认"UTF-8" 编码改为"GBK" 编码: `locale = locale(encoding = "GBK")`

还有参数 `comment` (忽略的注释标记), `skip_empty_rows` 等。

<sup>a</sup> `read_csv()` 可选列类型: "c" (字符型), "i" (整数型), "n" (数值型), "d" (浮点型), "l" (逻辑型), "f" (因子型), "D" (日期型), "T" (日期时间型), "t" (时间型), "?" (猜测该列类型), "\_" 或 "-" (跳过该列)。

```
read_xlsx(path, sheet, range, col_names, col_types, skip, na, n_max, ...)
```

- path: 数据文件所在相对或绝对路径
- sheet: 要读取的工作表
- range: 要读取的单元格范围
- col\_names: 第一行是否作为列名
- col\_types: 设置列类型<sup>a</sup>, 可总体设置一种类型 (循环使用) 或为每列单独设置, 默认 NULL (全部猜测)

也有参数: `skip, na, n_max`。

<sup>a</sup> `read_xlsx()` 可选列类型: "skip" (跳过该列), "guess" (猜测该列), "logical", "numeric", "date", "text", "list"。

`readr` 包读取数据的函数, 默认会保守猜测各列的列类型。若在读取数据时部分列有丢失信息, 则建议先将数据以文本 (字符) 型读取进来, 再用 `dplyr` 修改列类型。

## 1. 读入单个 csv 文件

```
df = read_csv("datas/六 1 班学生成绩.csv")
df
```

```
## # A tibble: 4 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
```

## 2. 批量读取 Excel 文件

批量读取的数据文件往往具有相同的列结构（列名、列类型），读入后紧接着需要按行合并为一个数据框。批量读取并合并，道理很简单，总共分三步：

- 获取批量数据文件的路径
- 循环机制批量读取
- 合并成一个数据文件

强大的 `purrr` 包，使得后两步可以同时做，即借助

```
map_dfr(.x, .f, .id)
```

将函数 `f` 依次应用到序列 `x` 的每个元素返回数据框，再 `bind_rows` 按行合并为一个数据框，`.id` 用来增加新列描述来源。

比如，在 `read_datas` 文件夹下有 5 个 `xlsx` 文件，每个文件的列名都是相同的：



首先要得到要导入的全部 Excel 文件的完整路径，可以任意嵌套，只需将参数 `recursive` 设为 `TRUE`：

```
files = list.files("datas/read_datas", pattern = "xlsx",
                    full.names = TRUE, recursive = TRUE)
files
```

```
## [1] "datas/read_datas/六1班学生成绩.xlsx"
## [2] "datas/read_datas/六2班学生成绩.xlsx"
## [3] "datas/read_datas/六3班学生成绩.xlsx"
## [4] "datas/read_datas/六4班学生成绩.xlsx"
## [5] "datas/read_datas/六5班学生成绩.xlsx"
```



接着，用 `map_dfr()` 在该路径向量上做迭代，应用 `read_xlsx()` 到每个文件路径，再按行合并。另外，再多做一步：用 `set_names()` 将文件路径字符向量创建为命名向量，再结合参数 `.id` 将路径值作为数据来源列。

```
library(readxl)
df = map_dfr(set_names(files), read_xlsx, .id = "来源")
head(df)

## # A tibble: 6 x 7
##   来源                                班级 姓名 性别 语文 数学 英语
##   <chr>                                <chr> <chr> <chr> <dbl> <dbl> <dbl>
## 1 datas/read_datas/六1班学生成绩.xlsx~ 六1班 何娜 女      87    92    79
## 2 datas/read_datas/六1班学生成绩.xlsx~ 六1班 黄才菊~ 女      95    77    75
## 3 datas/read_datas/六1班学生成绩.xlsx~ 六1班 陈芳妹~ 女      79    87    66
## 4 datas/read_datas/六1班学生成绩.xlsx~ 六1班 陈学勤~ 男      82    79    66
## 5 datas/read_datas/六2班学生成绩.xlsx~ 六2班 黄祖娜~ 女      94    88    75
## 6 datas/read_datas/六2班学生成绩.xlsx~ 六2班 徐雅琦~ 女      92    86    72
```

函数 `read_xlsx()` 的其它控制读取的参数，可直接“作为” `map_dfr` 参数在后面添加，或改用 `purrr` 风格公式形式：

```
map_dfr(set_names(files), read_xlsx, sheet = 1, .id = "来源") # 或者
map_dfr(set_names(files), ~ read_xlsx(., sheet = 1), .id = "来源")
```

若批量 Excel 数据是来自同一 xlsx 的多个 sheet

还是上述数据，只是在“学生成绩.xlsx”的 5 个 sheet 中：

	A	B	C	D	E	F
1	班级	姓名	性别	语文	数学	英语
2	六1班	何娜	女	87	92	79
3	六1班	黄才菊	女	95	77	75
4	六1班	陈芳妹	女	79	87	66
5	六1班	陈学勤	男	82	79	66
6						
	六1班	六2班	六3班	六4班	六5班	

```
path = "datas/学生成绩.xlsx" # Excel 文件路径
df = map_dfr(set_names(excel_sheets(path)),
              ~ read_xlsx(path, sheet = .x), .id = "sheet")
head(df)
```

```
## # A tibble: 6 x 7
##   sheet 班级 姓名 性别 语文 数学 英语
##   <chr> <chr> <chr> <chr> <dbl> <dbl> <dbl>
```



## 1	六1班	六1班	何娜	女	87	92	79
## 2	六1班	六1班	黄才菊	女	95	77	75
## 3	六1班	六1班	陈芳妹	女	79	87	66
## 4	六1班	六1班	陈学勤	男	82	79	66
## 5	六2班	六2班	黄祖娜	女	94	88	75
## 6	六2班	六2班	徐雅琦	女	92	86	72

`excel_sheets()` 函数作用在该 Excel 文件上，提取各个 `sheet` 名字，得到字符向量；然后同样是实现批量读取，只是这次是在 `sheet` 名字的字符向量上循环而已。

### 3. 写出到一个 Excel 文件

用 `readr` 包中的 `write_csv()` 和 `write_rds()`，或 `writexl` 包中的 `write_xlsx()` 可以保存数据到文件。

以写出到 Excel 文件为例：

```
library(writexl)
write_xlsx(df, "datas/output_file.xlsx")
```

### 4. 批量写出到 Excel 文件

比如有多个数据框，存在一个列表中，依次将它们写入文件，需要准备好文件名；在该数据框列表和文件名上，依次应用写出函数 `write_xlsx()`，又不需要返回值，故适用 `purrr` 包中的 `walk2()` 函数：

```
df = iris %>%
  group_split(Species) # 鸢尾花按组分割，得到数据框列表
files = str_c("datas/", levels(iris$Species), ".xlsx") # 准备文件名
walk2(df, files, write_xlsx)
```

若要多个数据框分别写入一个 Excel 文件的多个 `sheet`，先将多个数据框创建为命名列表（名字将作为 `sheet` 名），再用 `write_xlsx()` 写出即可：

```
df = df %>%
  set_names(levels(iris$Species))
write_xlsx(df, "datas/iris.xlsx")
```

### 5. 保存与载入 rds 数据

除了 `save()` 和 `load()` 函数外，下面以导出数据到 `.rds` 文件为例，因为它能保存数据框及其元数据，如数据类型和分组等。

```
write_rds(iris, "my_iris.rds")
dat = read_rds("my_iris.rds")      # 导入.rds 数据
```

### 2.2.3 关于中文编码

中文乱码是让很多编程者头痛的问题。

#### 1. 什么是编码？

文字符号在计算机中是用 0 和 1 的字节序列表示的，编码就是将字节序列与所要表示的文字符号建立起映射。

要把各个国家不同的所有文字符号（字符集）正常显示和使用，需要做两件事情：

- 各个国家不同的所有文字符号一一对应地建立数字编码
- 数字编码按一定编码规则用 0-1 表示出来

第一件事情已有一种 Unicode 编码（万国码）来解决：它给全世界所有语言的所有文字符号规定了独一无二的数字编码，字符间分隔的方式是用固定长度字节数。

这样各个国家只需要做第二件事情：为自己国家的所有文字符号设计一种编码规则来表示对应的 Unicode 编码<sup>3</sup>。

从 Unicode 到各国具体编码，称为编码过程；从各国具体编码到 Unicode，称为解码过程。

再来说中国的第二件事情：汉字符号（中文）编码。历史原因产生了多种中文编码，从图来看更直观：

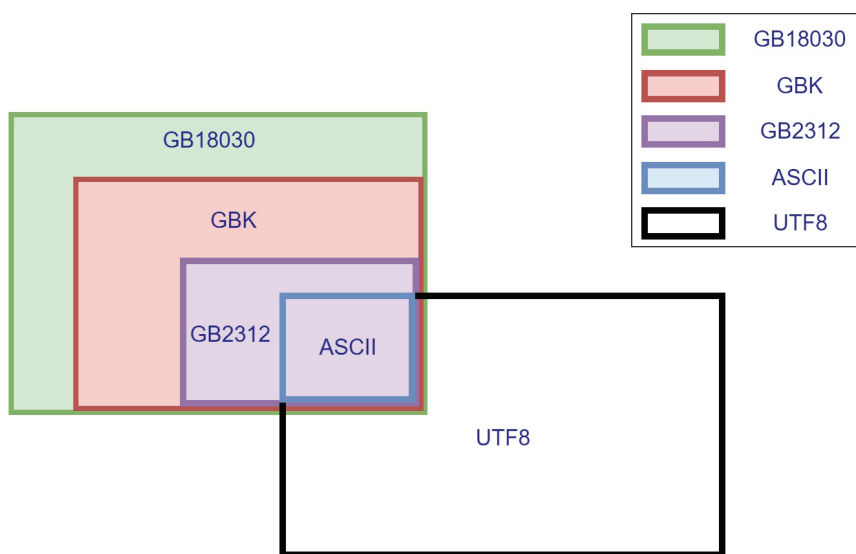


图 2.3: 几种中文编码及兼容性

<sup>3</sup>Unicode 为了表示“万国”语言，额外增大了存储开销，这第二件事也顺便节省存储开销。

所谓兼容性，可以理解为子集，同时存在也不冲突。由图 2.3 可见，ASCII（128 个字母和符号，英文够用）被所有编码兼容，而最常见的 UTF-8 与 GBK 之间除了 ASCII 部分之外没有交集。

文件采用什么编码方式，就用什么编码方式打开。只要是用不兼容的编码方式打开文件，就会出现乱码，日常最容易导致乱码场景就是：

用 UTF-8（GBK）编码方式去读取 GBK（UTF-8）编码的文字，就会出现各种乱码

GBK（国标扩展）系列，根据包含汉字符号从少到多，依次是

- GB2312: 只包含 6763 个汉字
- GBK: 包含 20902 个汉字，基本够用
- GB18030: 又分 GB1830-2000 和 GB1830-2005，包含七万多个汉字

GBK 编码的汉字基本是 2 字节，节省空间，但只适合国内中文环境。

UTF-8 编码（Unicode 转换格式），是 Unicode 的再表示，支持各个国家的文字符号，兼容性非常好。所以，目前 UTF-8 有一统天下的趋势。

UTF-8 是一种变长编码，解决字符间分隔的方式是通过二进制中最高位连续 1 的个数来决定该字是几字节编码。所有常用汉字的 Unicode 值均可用 3 字节的 UTF-8 表示出来。

UTF-8 通常不带 BOM（字节序标记 EF BB BF，位于文件的前 3 个字节）也不需要带 BOM，但 Windows 历史遗留问题又会经常遇到有 BOM UTF-8 的数据文件。

其它常见的编码：

- ANSI: 不是真正的编码，而是 Windows 系统的默认编码的统称，对于简体中文系统就是 GB2312；对于繁体中文系统就是 Big5 等
- Latin1: 又称 ISO-8859-1，欧洲人发明的编码，也是 MySQL 的默认编码
- Unicode big endian: 用 UCS-2 格式存储 Unicode 时，根据两个字节谁在前谁在后，分为 Little endian（小端）和 Big endian（大端）
- UTF-16, UTF-32: Unicode 的另两种再表示，分别用 2 字节和 4 字节。

## 2. 中文乱码的解决办法

首先，查看并确认你的 windows 系统的默认编码方式：

```
Sys.getlocale("LC_CTYPE")      # 查看系统默认字符集类型
```

```
## [1] "Chinese (Simplified)_China.936"
```

代码 936 就表明是“中国 - 简体中文（GB2312）”。

注意：不建议修改系统的默认编码方式，因为可能会导致一些软件、文件乱码。

大多数中文乱码都是 GBK 与 UTF-8 不兼容导致的，常见的有两种情形。

### R 文件中的中文乱码

在你的电脑不中文乱码的 R 脚本、Rmarkdown 等，拷贝到另一台电脑上时出现中文乱码。

解决办法：前文在配置 Rstudio 时已讲到，设置 code - saving 的 Default text encoding 为兼容性更好的 UTF-8。

### 读写数据文件中中文乱码

数据文件采用什么编码方式，就用什么编码方式打开或读取。采用了不兼容的另一种编码打开或读取，肯定出现中文乱码。

下面以最常见的中文编码 GBK、UTF-8、BOM UTF-8 来讲解。

#### R 自带函数读取 GBK 或 UTF-8

- 与所用操作系统默认编码相同的数据文件，即 GBK，R 自带的函数 `read.csv()`、`read.table()`、`readLines()` 都可以正常读取但不能直接读取 UTF-8
- 但在 `read.csv()` 和 `read.table()` 中设置参数 `fileEncoding = "UTF-8"`，可以读取 UTF-8，但无论如何不能读取 BOM UTF-8
- 在 `readLines()` 中设置参数 `encoding = "UTF-8"`，可以读取 UTF-8 和 BOM UTF-8

```
read.csv("datas/bp-gbk.csv")           # GBK, 直接读取
read.csv("datas/bp-utf8nobom.csv",     # UTF-8, 设置参数读取
         fileEncoding = "UTF-8")

readLines("datas/bp-gbk.csv")           # GBK, 直接读取
# UTF-8 和 BOM UTF-8, 设置参数读取
readLines("datas/bp-utf8nobom.csv", encoding = "UTF-8")
readLines("datas/bp-utf8bom.csv", encoding = "UTF-8")
```

#### readr 包读取 GBK 或 UTF-8

- readr 包中的 `read_csv()`、`read_table2()`、`read_lines()` 默认读取 UTF-8 和 BOM UTF-8;
- 但不能直接读取 GBK, 需要设置参数 `locale = locale(encoding="GBK")`

```
read_csv("datas/bp-utf8nobom.csv")      # UTF-8, 直接读取
read_csv("datas/bp-utf8bom.csv")        # BOM UTF-8, 直接读取
read_csv("datas/bp-gbk.csv",
        locale = locale(encoding="GBK")) # GBK, 设置参数读取
```

### 写入 GBK 或 UTF-8 文件

- R 自带的 write.csv(), writeLines() 仍是跟随操作系统默认编码，即默认写出为 GBK 文件；设置参数 fileEncoding = "UTF-8" 可写为 UTF-8
- readr 包中的 write\_csv(), write\_lines() 默认写为 UTF-8, 但不能被 Excel 软件正确打开
- readr::write\_excel\_csv() 可以写为 BOM UTF-8, Excel 软件能正确打开

```
write.csv(df, "file-GBK.csv")            # 写出为 GBK 文件
write.csv(df, "file-UTF8.csv",
        fileEncoding = "UTF-8")          # 写出为 UTF-8 文件

write_csv(df, "file-UTF8.csv")           # 写出为 UTF-8 文件
write_excel_csv(df, "file-BOM-UTF8.csv") # 写出为 BOM UTF-8 文件
```

不局限于上述编码，一个数据文件只要知道了其编码方式，就可以通过在读写时指定该编码而避免乱码。那么关键的问题就是：怎么确定一个数据文件的编码？

Notepad++是一款优秀开源的文本编辑器，用它打开数据文件，点编码，在下拉菜单黑点标记的编码方式即为该文件的编码，还可以对数据文件做编码转换：

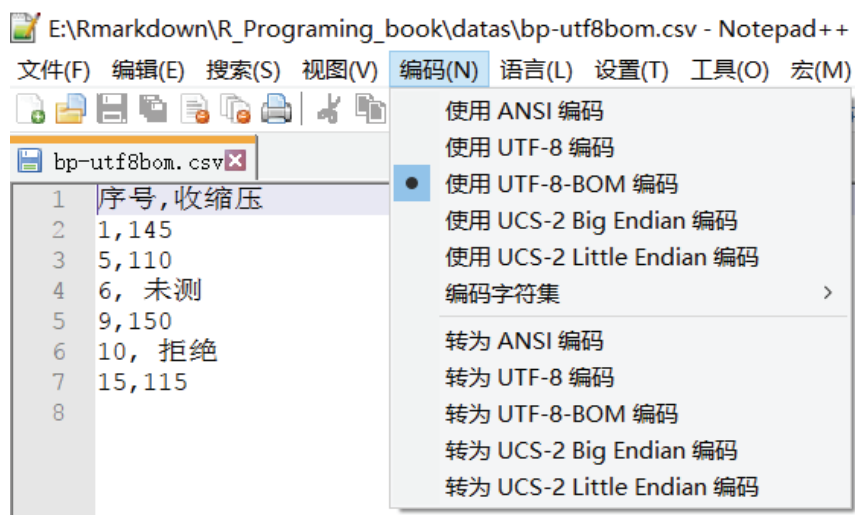


图 2.4: 用 Notepad++ 检测或转换文件编码

另外，readr 包和 rvest 包（爬虫）都提供了函数 guess\_encoding(), 可检测文本和网页的编码方式；python 有一个 chardet 库在检测文件编码方面更强大。

本节部分内容参阅 (李东风2020), (Desi Quintans 2019), 程序员必备：彻底弄懂常见的 7 种中文字符编码。

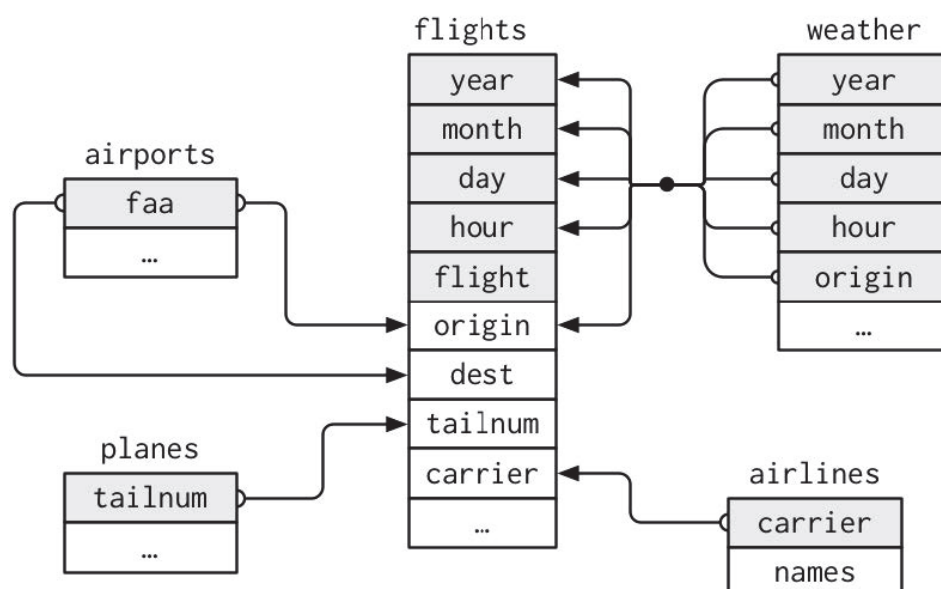
## 2.3 数据连接

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。

例如，纽约机场航班数据的关系结构：



比如，想要考察天气状况对航班的影响，就需要先将数据表 flights 和 wheater 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。要判断某（些）列是否是键列，可以先用 count() 计数，再看是否没有  $n > 1$  出现：

```
load("datas/planes.rda")
planes %>%
  count(tailnum) %>%
  filter(n > 1)

## # A tibble: 0 x 2
## # ... with 2 variables: tailnum <chr>, n <int>
```

```
load("datas/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 <int>,
## #   hour <int>, origin <chr>, n <int>
```

### 2.3.1 合并行与合并列

合并数据框最基本的方法是：

- 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）；
- 合并列：右侧拼接新列，根据位置匹配行，行数必须相同。

分别用 dplyr 包中的 bind\_rows() 和 bind\_cols() 实现。

```
bind_rows(
  sample_n(iris, 2),
  sample_n(iris, 2),
  sample_n(iris, 2)
)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
## 1         5.0         3.4         1.6         0.4     setosa
## 2         5.1         3.8         1.5         0.3     setosa
## 3         5.8         2.6         4.0         1.2 versicolor
## 4         6.3         3.3         4.7         1.6 versicolor
## 5         6.7         3.1         4.7         1.5 versicolor
## 6         5.4         3.9         1.3         0.4     setosa
```

```
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()` 函数可以在批量读入数据的同时做合并/合并列。

### 2.3.2 根据值匹配合并数据框

只介绍最常用的六种合并：左连接、右连接、全连接、内连接、半连接、反连接，前四种连接又称为修改连接，后两种连接又称为过滤连接。

这六种连接对应的六个接口一致的函数，其基本格式为：

```
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)
```

下面以两个小数据集进行演示：

```
band = tibble(name = c("Mick", "John", "Paul"),
               band = c("Stones", "Beatles", "Beatles"))
band

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

instrument = tibble(name = c("John", "Paul", "Keith"),
                    plays = c("guitar", "bass", "guitar"))
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()

左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
```

band		instrument						
name	band		name	plays		name	band	plays
Mick	Stones	+	John	guitar	=	Mick	Stones	<NA>
John	Beatles		Paul	bass		John	Beatles	guitar
Paul	Beatles		Keith	guitar		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")`。

## 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		instrument						
name	band		name	plays	name	band	plays	
Mick	Stones	+	John	guitar	=	John	Beatles	guitar
John	Beatles		Paul	bass		Paul	Beatles	bass
Paul	Beatles		Keith	guitar		Keith	<NA>	guitar

### 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
## 4 Keith <NA>    guitar
```

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

### 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		+	instrument		=			
name	band		name	plays		name	band	plays
Mick	Stones	+	John	guitar	=	John	Beatles	guitar
John	Beatles		Paul	bass		Paul	Beatles	bass
Paul	Beatles		Keith	guitar				

## 5. 半连接：semi\_join()

根据在 y 中，来筛选 x 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				instrument			
name	band			name	plays	name	band
Mick	Stones			John	guitar	John	Beatles
John	Beatles			Paul	bass	Paul	Beatles
Paul	Beatles			Keith	guitar		

## 6. 反连接：anti\_join()

根据不在 y 中，来筛选 x 中的行。


```
band %>%
  anti_join(instrument, by = "name")
```


```
## # A tibble: 1 x 2
##   name band
##   <chr> <chr>
## 1 Mick Stones
```


band				instrument			
name	band			name	plays	name	band
Mick	Stones			John	guitar	Mick	Stones
John	Beatles			Paul	bass		
Paul	Beatles			Keith	guitar		

前面讨论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 包中的 reduce() 使用即可。

比如 achieves 文件夹有 3 个 Excel 文件：

 3月.xlsx

 4月.xlsx

 5月.xlsx

A	B	C		A	B	C		A	B
人名	3月业绩			1 人名	4月业绩			1 人名	5月业绩
小明	80			2 小红	70			2 小花	100
小李	85			3 小白	60			3 小红	90
小张	90			4 小张	50			4 小李	80
				5 小王	40				

需要批量读取它们，再依次做全连接。`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，.....：

```
files = list.files("datas/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
## 4 小红         NA         70         90
## 5 小白         NA         60         NA
## 6 小王         NA         40         NA
## 7 小花         NA         NA        100
```

若还是上述数据，但是在一个工作簿的多个工作表中，批量读取并依次做全连接：

1	人名	3月业绩	
2	小明	80	
3	小李	85	
4	小张	90	
5			

```
path = "datas/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
```

### 2.3.3 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：

```
dplyr::intersect(x, y)    # 返回 x 和 y 共同包含的观测；
dplyr::union(x, y)        # 返回 x 和 y 中所有的（唯一）观测；
dplyr::setdiff(x, y)      # 返回在 x 中但不在 y 中的观测。
```

本节部分内容参阅 (Hadley Wickham 2017), (Amelia McNamara 2020), (Desi Quintans 2019).

## 2.4 数据重塑

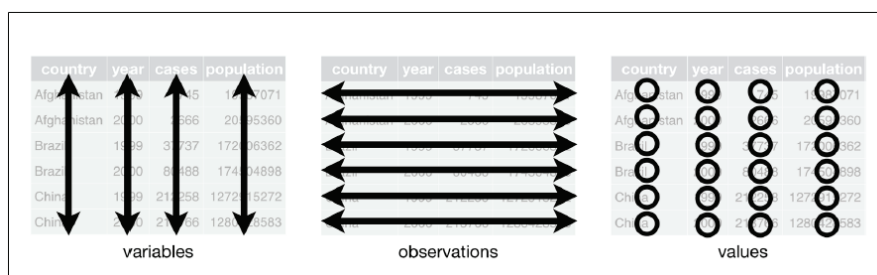
### 2.4.1 什么是整洁数据？

采用 Hadley 的表述，脏的/不整洁的数据往往具有如下特点：

- 首行（列名）是值，不是变量名
- 多个变量放在一列
- 变量既放在行也放在列
- 多种类型的观测单元在同一个单元格
- 一个观测单元放在多个表

而整洁数据具有如下特点：

- 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



tidyverse 系列包中的函数操作的都是这种整洁数据框，而不整洁数据，首先需要变成整洁数据，这就是数据重塑。

数据重塑主要包括长宽表转化、拆分/合并列。长宽表转化最初是 reshape2 包的 melt() 和 cast(); 后来又发展到早期 tidyr 包的 gather() 和 spread(), 现在 tidyr 1.0 之后提供了更加易用的 pivot\_longer() 和 pivot\_wider().

先看一个不整洁数据与整洁数据对比的例子：

```
dt = tribble(
  ~observation, ~A_count, ~B_count, ~A_dbh, ~B_dbh,
  "Richmond(Sam)", 7, 2, 100, 110,
  "Windsor(Ash)", 10, 5, 80, 87,
  "Bilpin(Jules)", 5, 8, 95, 90
)
knitr::kable(dt, align="c")
```

observation	A_count	B_count	A_dbh	B_dbh
Richmond(Sam)	7	2	100	110
Windsor(Ash)	10	5	80	87
Bilpin(Jules)	5	8	95	90

该数据框不整洁，表现在：

- observation 列有两个变量数据
- 列名中的 A/B 应是分类变量 species 的两个水平值
- 测量值列 count 和 dbh 应各占 1 列，而不是 2 列

下面借助 tidyr 包中数据重塑函数，将其变成整洁数据，读者可以学完本节内容再回头看这段代码：

```
tidy_dt = dt %>%
  pivot_longer(-observation,
    names_to = c("speices", ".value"),
    names_sep = "_") %>%
  separate(observation, into = c("site", "surveyor"))
knitr::kable(tidy_dt, align = "c")
```

site	surveyor	speices	count	dbh
Richmond	Sam	A	7	100
Richmond	Sam	B	2	110
Windsor	Ash	A	10	80
Windsor	Ash	B	5	87
Bilpin	Jules	A	5	95
Bilpin	Jules	B	8	90

这里的关键是，要学会区分哪些是变量、观测、值。



### 2.4.2 宽表变长表

宽表的特点是：表比较宽，本来该是“值”的，却出现在“变量（名）”中。

这就需要给它变到“值”中，新起个列名存为一列，这就是所谓的宽表变长表。

用 `tidyr` 包中的 `pivot_longer()` 函数来实现宽表变长表，其基本格式为：

```
pivot_longer(data, cols, names_to, values_to, values_drop_na, ...)
```

- `data`: 要重塑的数据框
- `cols`: 用选择列语法选择要变形的列
- `names_to`: 为存放变形列的列名中的“值”，指定新列名
- `values_to`: 为存放变形列中的“值”，指定新列名
- `values_drop_na`: 是否忽略变形列中的 NA

若变形列的列名除了“值”外，还包含前缀、变量名 + 分隔符、正则表达式分组捕获模式，则可以借助参数 `names_prefix`, `names_sep`, `names_pattern` 来提取出“值”。

#### 1. 每一行只有 1 个观测的情形

也是最简单的情形，以分省年度 GDP 数据为例，每一行只有一个观测，关于一个省份的信息。

```
df = read_csv("datas/分省年度 GDP.csv")
df
```

```
## # A tibble: 4 x 4
##   地区      `2019年` `2018年` `2017年`
##   <chr>      <dbl>    <dbl>    <dbl>
## 1 北京市      35371.    33106.    28015.
## 2 天津市      14104.    13363.    18549.
## 3 河北省      35105.    32495.    34016.
## 4 黑龙江省     13613.    12846.    15903.
```

- 要变形的列是除了 地区列之外的列
- 变量（名）中的 2019 年、2018 年等是年份的值，需要作为 1 列“值”来存放，新起一个列名 年份
- 2019 年、2018 年等列中的值，属于同一个变量 GDP，新起一个列名 GDP 来存放：

```
df %>%
  pivot_longer(-地区, names_to = "年份", values_to = "GDP")
```

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

```
##   地区   年份      GDP
##   <chr> <chr>   <dbl>
## 1 北京市 2019年 35371.
## 2 北京市 2018年 33106.
## 3 北京市 2017年 28015.
## 4 天津市 2019年 14104.
## 5 天津市 2018年 13363.
## 6 天津市 2017年 18549.
## # ... with 6 more rows
```

## 2. 每一行有多个观测的情形

以如下的 `family` 数据集为例，每一行有两个观测，关于 `child1` 和 `child2` 的信息。

```
load("datas/family.rda")
knitr::kable(family, align = "c")
```

family	dob_child1	dob_child2	gender_child1	gender_child2
1	1998-11-26	2000-01-29	1	2
2	1996-06-22	NA	2	NA
3	2002-07-11	2004-04-05	2	2
4	2004-10-10	2009-08-27	1	1
5	2000-12-05	2005-02-28	2	1

- 要变形的列是除了 `family` 列之外的列；
- 变形列的列名以 "\_" 分割为两部分，用 `names_to` 指定这两部分的用途：“`.value`”指定第一部分不用管将继续留作列名，而第二部分，即包含“`child1`”、“`child2`”，作为新变量 `child` 的“值”
- 忽略变形列中的缺失值

```
family %>%
  pivot_longer(-family,
               names_to = c(".value", "child"),
               names_sep = "_",
               values_drop_na = TRUE)
```

```
## # A tibble: 9 x 4
##   family child  dob      gender
##   <int> <chr> <date>    <int>
## 1     1 1 child1 1998-11-26      1
```

```
## 2      1 child2 2000-01-29      2
## 3      2 child1 1996-06-22      2
## 4      3 child1 2002-07-11      2
## 5      3 child2 2004-04-05      2
## 6      4 child1 2004-10-10      1
## # ... with 3 more rows
```

再来看一个数学建模报名信息整理的实例：每一行有 3 个观测，关于 3 名队员的信息，变成每一行只有 1 名队员的信息。用到 `names_pattern` 参数和正则表达式分组捕获。

```
df = read_csv("datas/参赛队信息.csv")
df
```

```
## # A tibble: 2 x 6
##   队员1姓名 队员1专业 队员2姓名 队员2专业 队员3姓名 队员3专业
##   <chr>      <chr>      <chr>      <chr>      <chr>      <chr>
## 1 张三      数学      李四      英语      王五      统计学
## 2 赵六      经济学    钱七      数学      孙八      计算机
```

```
df %>%
  pivot_longer(everything(),
    names_to = c("队员", ".value"),
    names_pattern = "(.*\\d)(.*)")
```

```
## # A tibble: 6 x 3
##   队员 姓名 专业
##   <chr> <chr> <chr>
## 1 队员1 张三 数学
## 2 队员2 李四 英语
## 3 队员3 王五 统计学
## 4 队员1 赵六 经济学
## 5 队员2 钱七 数学
## 6 队员3 孙八 计算机
```

### 2.4.3 长表变宽表

长表的特点是：表比较长。

有时候需要将分类变量的若干水平值，变成变量（列名）。这就是长表变宽表，它与宽表变长表正好相反（二者互逆）。

用 `tidyr` 包中的 `pivot_wider()` 函数来实现长表变宽表，其基本格式为：

```
pivot_wider(data, id_cols, names_from, values_from, values_fill, ...)
```

- `data`: 要重塑的数据框
- `id_cols`: 唯一识别观测的列，默认是除了 `names_from` 和 `values_from` 指定列之外的列
- `names_from`: 指定列名来自哪个变量列
- `values_from`: 指定列“值”来自哪个变量列
- `values_fill`: 若变宽后单元格值缺失，设置用何值填充

另外还有若干帮助修复列名的参数：`names_prefix`, `names_sep`, `names_glue`.

最简单的情形是，只有一个列名列和一个值列，比如 `animals` 数据集：

```
load("datas/animals.rda")
```

```
animals
```

```
## # A tibble: 228 x 3
##   Type      Year  Heads
##   <chr>   <dbl>  <dbl>
## 1 Sheep    2015  24943.
## 2 Cattle   1972   2189.
## 3 Camel    1985    559
## 4 Camel    1995    368.
## 5 Camel    1997    355.
## 6 Goat     1977   4411.
## # ... with 222 more rows
```

用 `names_from` 指定列名来自哪个变量；`values_from` 指定“值”来自哪个变量：

```
animals %>%
```

```
  pivot_wider(names_from = Type, values_from = Heads, values_fill = 0)
```

```
## # A tibble: 48 x 6
##   Year  Sheep Cattle Camel  Goat Horse
##   <dbl> <dbl>  <dbl> <dbl>  <dbl> <dbl>
## 1  2015 24943.  3780.  368. 23593. 3295.
## 2  1972 13716.  2189.  625. 4338. 2239.
## 3  1985 13249.  2408.  559  4299. 1971
## 4  1995      0  3317.  368. 8521. 2684.
## 5  1997 14166.  3613.  355. 10265. 2893.
## 6  1977 13430.  2388.  609  4411. 2104.
```

```
## # ... with 42 more rows
```

还可以有多个列名列或多个值列，比如 `us_rent_income` 数据集有两个值列：

```
us_rent_income
```

```
## # A tibble: 104 x 5
##   GEOID NAME      variable estimate   moe
##   <chr> <chr>    <chr>      <dbl> <dbl>
## 1 01     Alabama income    24476   136
## 2 01     Alabama rent      747     3
## 3 02     Alaska  income    32940   508
## 4 02     Alaska  rent     1200    13
## 5 04     Arizona income    27517   148
## 6 04     Arizona rent      972     4
## # ... with 98 more rows
```

```
us_rent_income %>%
```

```
  pivot_wider(names_from = variable, values_from = c(estimate, moe))
```

```
## # A tibble: 52 x 6
##   GEOID NAME      estimate_income estimate_rent moe_income moe_rent
##   <chr> <chr>          <dbl>          <dbl>      <dbl> <dbl>
## 1 01     Alabama      24476          747        136     3
## 2 02     Alaska      32940        1200        508    13
## 3 04     Arizona      27517          972        148     4
## 4 05     Arkansas      23789          709        165     5
## 5 06     California    29454        1358        109     3
## 6 08     Colorado      32401        1125        109     5
## # ... with 46 more rows
```

注意，用 `pivot_wider()` 做长变宽时，要求列名列中每个名字的各行（组内）可唯一识别，对于缺少该唯一识别列的数据框，就需要先按列名列分组，并创建唯一识别列：

```
df = read_xlsx("datas/raw_stock.xlsx")
df
```

```
## # A tibble: 60 x 2
##       vals type
##       <dbl> <chr>
## 1 1943     年份
## 2 1.3      股票A
```

```
## 3      1.23 股票B
## 4      1.15 股票C
## 5      1.26 股票指数
## 6 1944    年份
## # ... with 54 more rows
```

```
df %>%
  group_by(type) %>%
  mutate(row_id = row_number()) %>%
  pivot_wider(names_from = type, values_from = vals) %>%
  select(-row_id)
```

```
## # A tibble: 12 x 5
##   年份 股票A 股票B 股票C 股票指数
##   <dbl> <dbl> <dbl> <dbl>    <dbl>
## 1 1943 1.3    1.23 1.15    1.26
## 2 1944 1.10    1.29 1.26    1.20
## 3 1945 1.22    1.22 1.42    1.36
## 4 1946 0.954 0.728 0.922    0.919
## 5 1947 0.929 1.14 1.17    1.06
## 6 1948 1.06 1.11 0.965    1.06
## # ... with 6 more rows
```

再看一个特殊的实例：不规则通讯录整理。

```
contacts = tribble( ~field, ~value,
  " 姓名", " 张三",
  " 公司", " 百度",
  " 姓名", " 李四",
  " 公司", " 腾讯",
  "Email", "Lisi@163.com",
  " 姓名", " 王五")
contacts = contacts %>%
  mutate(ID = cumsum(field == " 姓名"))
contacts
```

```
## # A tibble: 6 x 3
##   field value      ID
##   <chr> <chr>    <int>
## 1 姓名 张三        1
## 2 公司 百度        1
```

```
## 3 姓名 李四      2
## 4 公司 腾讯      2
## 5 Email Lisi@163.com 2
## 6 姓名 王五      3
```

```
contacts %>%
```

```
  pivot_wider(names_from = field, values_from = value)
```

```
## # A tibble: 3 x 4
##       ID 姓名 公司 Email
##   <int> <chr> <chr> <chr>
## 1     1 张三 百度 <NA>
## 2     2 李四 腾讯 Lisi@163.com
## 3     3 王五 <NA> <NA>
```

#### 2.4.4 拆分列与合并列

拆分列与合并列也是正好相反（二者互逆）。

用 `separate()` 函数来拆分列，其基本语法为：

```
separate(data, col, into, sep, ...)
```

- `col`: 要拆分的列
- `into`: 拆开的新列，
- `sep`: 指定根据什么分隔符拆分

```
table3
```

```
## # A tibble: 6 x 3
##   country      year rate
## * <chr>      <int> <chr>
## 1 Afghanistan 1999 745/19987071
## 2 Afghanistan 2000 2666/20595360
## 3 Brazil      1999 37737/172006362
## 4 Brazil      2000 80488/174504898
## 5 China       1999 212258/1272915272
## 6 China       2000 213766/1280428583
```

```
table3 %>%
```

```
  separate(rate, into = c("cases", "population"), sep = "/",
            convert = TRUE) # 同时转化为数值型
```

```
## # A tibble: 6 x 4
```



```
##   country      year  cases population
##   <chr>      <int> <int>      <int>
## 1 Afghanistan 1999    745   19987071
## 2 Afghanistan 2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000   80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

还有 `separate_rows()` 函数，可对不定长的列进行分列，并按行堆叠放置：

```
df = tibble(Class = c("1 班", "2 班"),
             Name = c(" 张三, 李四, 王五", " 赵六, 钱七"))
```

```
df
```

```
## # A tibble: 2 x 2
##   Class Name
##   <chr> <chr>
## 1 1班   张三, 李四, 王五
## 2 2班   赵六, 钱七
```

```
df %>%
  separate_rows(Name, sep = ", ")
```

```
## # A tibble: 2 x 2
##   Class Name
##   <chr> <chr>
## 1 1班   张三, 李四, 王五
## 2 2班   赵六, 钱七
```

用 `unite()` 函数来合并列，其基本语法为：

```
unite(data, col, sep, ...)
```

- `col`: 要合并的列
- `sep`: 指定合并各列添加的分隔符

```
table5
```

```
## # A tibble: 6 x 4
##   country      century year  rate
## * <chr>      <chr>   <chr> <chr>
## 1 Afghanistan 19      99    745/19987071
## 2 Afghanistan 20      00    2666/20595360
```

```
## 3 Brazil      19      99      37737/172006362
## 4 Brazil      20      00      80488/174504898
## 5 China       19      99      212258/1272915272
## 6 China       20      00      213766/1280428583
```

```
table5 %>%
```

```
  unite(new, century, year, sep = "")
```

```
## # A tibble: 6 x 3
##   country      new    rate
##   <chr>        <chr> <chr>
## 1 Afghanistan 1999   745/19987071
## 2 Afghanistan 2000  2666/20595360
## 3 Brazil      1999  37737/172006362
## 4 Brazil      2000  80488/174504898
## 5 China       1999  212258/1272915272
## 6 China       2000  213766/1280428583
```

最后看一个综合示例：重塑世界银行人口数据。

```
world_bank_pop
```

```
## # A tibble: 1,056 x 20
##   country indicator `2000` `2001` `2002` `2003` `2004` `2005`
##   <chr>    <chr>      <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1 ABW     SP.URB.T~ 4.24e4 4.30e4 4.37e4 4.42e4 4.47e+4 4.49e+4
## 2 ABW     SP.URB.G~ 1.18e0 1.41e0 1.43e0 1.31e0 9.51e-1 4.91e-1
## 3 ABW     SP.POP.T~ 9.09e4 9.29e4 9.50e4 9.70e4 9.87e+4 1.00e+5
## 4 ABW     SP.POP.G~ 2.06e0 2.23e0 2.23e0 2.11e0 1.76e+0 1.30e+0
## 5 AFG     SP.URB.T~ 4.44e6 4.65e6 4.89e6 5.16e6 5.43e+6 5.69e+6
## 6 AFG     SP.URB.G~ 3.91e0 4.66e0 5.13e0 5.23e0 5.12e+0 4.77e+0
## # ... with 1,050 more rows, and 12 more variables: `2006` <dbl>,
## #   `2007` <dbl>, `2008` <dbl>, `2009` <dbl>, `2010` <dbl>,
## #   `2011` <dbl>, `2012` <dbl>, `2013` <dbl>, `2014` <dbl>,
## #   `2015` <dbl>, `2016` <dbl>, `2017` <dbl>
```

先从最显然的入手：年份跨过了多个列，应该宽表变长表：

```
pop2 = world_bank_pop %>%
```

```
  pivot_longer(`2000`:`2017`, names_to = "year", values_to = "value")
```

```
pop2
```

```
## # A tibble: 19,008 x 4
```

```
##   country indicator   year value
##   <chr>   <chr>       <chr> <dbl>
## 1 ABW     SP.URB.TOTL 2000  42444
## 2 ABW     SP.URB.TOTL 2001  43048
## 3 ABW     SP.URB.TOTL 2002  43670
## 4 ABW     SP.URB.TOTL 2003  44246
## 5 ABW     SP.URB.TOTL 2004  44669
## 6 ABW     SP.URB.TOTL 2005  44889
## # ... with 19,002 more rows
```

再来考虑 `indicator` 变量:

```
pop2 %>%
  count(indicator)
```

```
## # A tibble: 4 x 2
##   indicator      n
##   <chr>      <int>
## 1 SP.POP.GROW  4752
## 2 SP.POP.TOTL  4752
## 3 SP.URB.GROW  4752
## 4 SP.URB.TOTL  4752
```

这里, `SP.POP.GROW` 为人口增长率, `SP.POP.TOTAL` 为总人口, `SP.URB.*` 也类似, 只是城市的。将该列值拆分为两个变量: `area` (`URB, POP`) 和 `variable` (`GROW, TOTL`):

```
pop3 = pop2 %>%
  separate(indicator, c(NA, "area", "variable"))
pop3
```

```
## # A tibble: 19,008 x 5
##   country area variable year value
##   <chr>   <chr> <chr>   <chr> <dbl>
## 1 ABW    URB    TOTL    2000  42444
## 2 ABW    URB    TOTL    2001  43048
## 3 ABW    URB    TOTL    2002  43670
## 4 ABW    URB    TOTL    2003  44246
## 5 ABW    URB    TOTL    2004  44669
## 6 ABW    URB    TOTL    2005  44889
## # ... with 19,002 more rows
```

最后, 再将分类变量 `variable` 的水平值变为列名 (长表变宽表), 就完成重塑:

```
pop3 %>%
  pivot_wider(names_from = variable, values_from = value)
```

```
## # A tibble: 9,504 x 5
##   country area  year  TOTL  GROW
##   <chr>   <chr> <chr> <dbl> <dbl>
## 1 ABW     URB    2000  42444  1.18
## 2 ABW     URB    2001  43048  1.41
## 3 ABW     URB    2002  43670  1.43
## 4 ABW     URB    2003  44246  1.31
## 5 ABW     URB    2004  44669  0.951
## 6 ABW     URB    2005  44889  0.491
## # ... with 9,498 more rows
```

本节部分内容参阅 (Hadley Wickham 2017), (Desi Quintans 2019), Vignettes of tidy.

## 2.5 数据操作

用 dplyr 包实现各种数据操作，通常的数据操作无论多么复杂，往往都可以分解为若干基本数据操作步骤的组合。

共有 5 种基本数据操作：

- `select()` —— 选择列
- `filter()/slice()` —— 筛选行
- `arrange()` —— 对行排序
- `mutate()` —— 修改列/创建新列
- `summarize()` —— 汇总

这些函数都可以与

- `group_by()` —— 分组

连用，以改变数据操作的作用域：作用在整个数据框，或数据框的每个分组。

这些函数组合使用就足以完成各种数据操作，它们的相同之处是：

- 第 1 个参数是数据框，方便管道操作
- 根据列名访问数据框的列，且列名不用加引号
- 返回结果是一个新数据框，不改变原数据框

从而，可以方便地实现：“将多个简单操作，依次用管道连接，实现复杂的数据操作”。

另外，若要同时对所选择的多列应用函数，还有强大的 `across()` 函数，它支持各种选择列语法，搭配 `mutate()` 和 `summarise()` 使用，产生非常强大同时修改/汇总多列的效果。

### 2.5.1 选择列

选择列，包括对数据框做选择列、调整列序、重命名列。

下面以虚拟的学生成绩数据来演示，包含随机生成的 20 个 NA：

```
df = read_xlsx("datas/ExamDatas_NAs.xlsx")
df

## # A tibble: 50 x 8
##   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    NA     9
## 3 六1班 陈芳妹 女      79    87     66     9     10
## 4 六1班 陈学勤 男      NA    79     66     9     10
## 5 六1班 陈祝贞 女      76    79     67     8     10
## 6 六1班 何小微 女      83    73     65     8     9
## # ... with 44 more rows
```

#### 1. 选择列语法

##### (1) 用列名或索引选择列

```
df %>%
  select(name, sex, math) # 或者 select(2, 3, 5)

## # A tibble: 50 x 3
##   name  sex  math
##   <chr> <chr> <dbl>
## 1 何娜 女      92
## 2 黄才菊 女      77
## 3 陈芳妹 女      87
## 4 陈学勤 男      79
## 5 陈祝贞 女      79
## 6 何小微 女      73
## # ... with 44 more rows
```

## (2) 借助运算符选择列

- 用: 选择连续的若干列
- 用! 选择变量集合的余集（反选）
- & 和 | 选择变量集合的交或并
- c() 合并多个选择

## (3) 借助选择助手函数

- 选择指定列：
  - everything(): 选择所有列
  - last\_col(): 选择最后一列，可以带参数，如 last\_col(5) 选择倒数第 6 列
- 选择列名匹配的列：
  - starts\_with(): 以某前缀开头的列名
  - ends\_with(): 以某后缀结尾的列名
  - contains(): 包含某字符串的列名
  - matches(): 匹配正则表达式的列名
  - num\_range(): 匹配数值范围的列名，如 num\_range("x", 1:3) 匹配 x1, x2, x3
- 结合函数选择列：
  - where(): 应用一个函数到所有列，选择返回结果为 TRUE 的列，比如与 is.numeric 等函数连用

## 2. 一些选择列的示例

```
df %>%
  select(starts_with("m"))
```

```
## # A tibble: 50 x 2
##   math moral
##   <dbl> <dbl>
## 1     92     9
## 2     77    NA
## 3     87     9
## 4     79     9
## 5     79     8
## 6     73     8
## # ... with 44 more rows
```

```
df %>%
  select(ends_with("e"))
```

```
## # A tibble: 50 x 3
##   name    chinese science
##   <chr>    <dbl>    <dbl>
## 1 何娜      87      10
## 2 黄才菊    95       9
## 3 陈芳妹    79      10
## 4 陈学勤    NA      10
## 5 陈祝贞    76      10
## 6 何小薇    83       9
## # ... with 44 more rows
```

```
df %>%
  select(contains("a"))
```

```
## # A tibble: 50 x 4
##   class name    math moral
##   <chr> <chr>    <dbl> <dbl>
## 1 六1班 何娜      92      9
## 2 六1班 黄才菊    77     NA
## 3 六1班 陈芳妹    87      9
## 4 六1班 陈学勤    79      9
## 5 六1班 陈祝贞    79      8
## 6 六1班 何小薇    73      8
## # ... with 44 more rows
```

- 根据正则表达式匹配选择列：

```
df %>%
  select(matches("m.*a"))
```

```
## # A tibble: 50 x 2
##   math moral
##   <dbl> <dbl>
## 1    92      9
## 2    77     NA
## 3    87      9
## 4    79      9
## 5    79      8
```



```
## 6      73      8
## # ... with 44 more rows
```

- 根据条件（逻辑判断）选择列，例如选择所有数值型的列：

```
df %>%
  select(where(is.numeric))

## # A tibble: 50 x 5
##   chinese math english moral science
##   <dbl> <dbl>   <dbl> <dbl>   <dbl>
## 1      87   92     79     9     10
## 2      95   77     75    NA     9
## 3      79   87     66     9     10
## 4      NA   79     66     9     10
## 5      76   79     67     8     10
## 6      83   73     65     8     9
## # ... with 44 more rows
```

- 也可以自定义返回 TRUE 或 FALSE 的判断函数，支持 purrr 风格公式写法。例如，选择 列和 > 3000 的列：

```
df[, 4:8] %>%
  select(where(~ sum(.x, na.rm = TRUE) > 3000))

## # A tibble: 50 x 2
##   chinese math
##   <dbl> <dbl>
## 1      87   92
## 2      95   77
## 3      79   87
## 4      NA   79
## 5      76   79
## 6      83   73
## # ... with 44 more rows
```

再比如，结合 `n_distinct()` 选择唯一值数目 < 10 的列：

```
df %>%
  select(where(~ n_distinct(.x) < 10))

## # A tibble: 50 x 4
##   class sex   moral science
```

```
##   <chr> <chr> <dbl>   <dbl>
## 1 六1班 女           9      10
## 2 六1班 女          NA       9
## 3 六1班 女           9      10
## 4 六1班 男           9      10
## 5 六1班 女           8      10
## 6 六1班 女           8       9
## # ... with 44 more rows
```

### 3. 用“-”删除列

```
df %>%
  select(-c(name, chinese, science)) # 或者 select(-ends_with("e"))
```

```
## # A tibble: 50 x 5
##   class sex    math english moral
##   <chr> <chr> <dbl>   <dbl> <dbl>
## 1 六1班 女      92      79     9
## 2 六1班 女      77      75    NA
## 3 六1班 女      87      66     9
## 4 六1班 男      79      66     9
## 5 六1班 女      79      67     8
## 6 六1班 女      73      65     8
## # ... with 44 more rows
```

```
df %>%
  select(math, everything(), -ends_with("e"))
```

```
## # A tibble: 50 x 5
##   math class sex    english moral
##   <dbl> <chr> <chr>   <dbl> <dbl>
## 1    92 六1班 女      79     9
## 2    77 六1班 女      75    NA
## 3    87 六1班 女      66     9
## 4    79 六1班 男      66     9
## 5    79 六1班 女      67     8
## 6    73 六1班 女      65     8
## # ... with 44 more rows
```

注意：-ends\_with() 要放在 everything() 后面，否则删除的列就全回来了。

#### 4. 调整列的顺序

列是根据被选择的顺序排列：

```
df %>%
  select(ends_with("e"), math, name, class, sex)

## # A tibble: 50 x 6
##   name    chinese science  math class sex
##   <chr>    <dbl>    <dbl> <dbl> <chr> <chr>
## 1 何娜      87      10    92 六1班 女
## 2 黄才菊    95       9    77 六1班 女
## 3 陈芳妹    79      10    87 六1班 女
## 4 陈学勤    NA      10    79 六1班 男
## 5 陈祝贞    76      10    79 六1班 女
## 6 何小薇    83       9    73 六1班 女
## # ... with 44 more rows
```

everything() 返回未被选择的所有列，将某一列移到第一列时很方便：

```
df %>%
  select(math, everything())

## # A tibble: 50 x 8
##   math class name    sex  chinese english moral science
##   <dbl> <chr> <chr>  <chr>    <dbl>    <dbl> <dbl>    <dbl>
## 1    92 六1班 何娜  女      87      79      9      10
## 2    77 六1班 黄才菊 女      95      75     NA       9
## 3    87 六1班 陈芳妹 女      79      66      9      10
## 4    79 六1班 陈学勤 男      NA      66      9      10
## 5    79 六1班 陈祝贞 女      76      67      8      10
## 6    73 六1班 何小薇 女      83      65      8       9
## # ... with 44 more rows
```

用 relocate() 函数，将选择的列移到某列之前或之后，基本语法为：

```
relocate(.data, ..., .before, .after)
```

例如，将数值列移到 name 列的后面：

```
df %>%
  relocate(where(is.numeric), .after = name)

## # A tibble: 50 x 8
```

```
##   class name  chinese  math english moral science sex
##   <chr> <chr>    <dbl> <dbl>   <dbl> <dbl>   <dbl> <chr>
## 1 六1班 何娜      87    92     79    9      10 女
## 2 六1班 黄才菊    95    77     75   NA      9 女
## 3 六1班 陈芳妹    79    87     66    9     10 女
## 4 六1班 陈学勤    NA    79     66    9     10 男
## 5 六1班 陈祝贞    76    79     67    8     10 女
## 6 六1班 何小薇    83    73     65    8      9 女
## # ... with 44 more rows
```

## 5. 重命名列

`set_names()` 为所有列设置新列名:

```
df %>%
  set_names(" 班级", " 姓名", " 性别", " 语文",
            " 数学", " 英语", " 品德", " 科学")
```

```
## # A tibble: 50 x 8
##   班级 姓名 性别 语文 数学 英语 品德 科学
##   <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   NA     9
## 3 六1班 陈芳妹 女      79    87    66    9    10
## 4 六1班 陈学勤 男      NA    79    66    9    10
## 5 六1班 陈祝贞 女      76    79    67    8    10
## 6 六1班 何小薇 女      83    73    65    8     9
## # ... with 44 more rows
```

`rename()` 只修改部分列名, 格式为: 新名 = 旧名

```
df %>%
  rename(数学 = math, 科学 = science)
```

```
## # A tibble: 50 x 8
##   class name  sex  chinese 数学 english moral 科学
##   <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   NA     9
## 3 六1班 陈芳妹 女      79    87     66    9    10
## 4 六1班 陈学勤 男      NA    79     66    9    10
## 5 六1班 陈祝贞 女      76    79     67    8    10
```

```
## 6 六1班 何小微 女      83      73      65      8      9
## # ... with 44 more rows
```

### across() 函数

函数 `across()` 人如其名，让零个/一个/多个函数穿过所选择的列，即同时对所选择的多列应用若干函数，基本格式为：

```
across(.cols = everything(), .fns = NULL, ..., .names)
```

- `.cols` 为根据选择列语法选定的列范围；
- `.fns` 为应用到选定列上的函数<sup>a</sup>，它可以是：
  - `NULL`：不对列作变换；
  - 一个函数，如 `mean`；
  - 一个 `purrr` 风格的匿名函数，如 `~ .X * 10`
  - 多个函数或匿名函数构成的列表
- `.names` 用来设置输出列的列名样式，默认为 `{col}_{fn}`

`across()` 支持各种选择列语法，与 `mutate()` 和 `summarise()` 连用，产生非常强大的同时修改/（多种）汇总多列效果；

`across()` 也能与 `filter()` 连用，在限定列范围内筛选“所有值都满足某条件的行”，但不擅长筛选“存在值满足某条件的行”；

`across()` 也能与 `group_by()`, `count()` 和 `distinct()` 连用，此时 `.fns` 为 `NULL`，只起选择列的作用。

`across()` 函数的引入，使得可以弃用那些限定列范围的后缀：`_all`, `_if`, `_at`：

- `across(everything(), .fns)`：在所有列范围内，代替后缀 `_all`
- `across(where(), .fns)`：在满足条件的列范围内，代替后缀 `_if`
- `across(.cols, .fns)`：在给定的列范围内，代替后缀 `_at`

<sup>a</sup>这些函数内部可以使用 `cur_column()` 和 `cur_group()` 以访问当前列和分组键值。

## 2.5.2 筛选行

筛选行，即按行选择数据子集，包括删除行、对行切片、过滤行

先创建一个包含重复行的数据框：

```
set.seed(123)
df_dup <- df %>%
  slice_sample(n = 60, replace = TRUE)
```

### 1. 删除行

#### (1) 删除重复行

用 `dplyr` 包中的 `distinct()` 删除重复行（只保留第 1 个，删除其余）。

```
df_dup %>%
```

```
distinct()
```

```
## # A tibble: 35 x 8
##   class name sex   chinese math english moral science
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl>
## 1 六4班 周婵 女       92    94     77    10      9
## 2 六2班 杨远芸 女       93    80     68     9     10
## 3 六2班 陈华健 男       92    84     70     9     10
## 4 六1班 陈芳妹 女       79    87     66     9     10
## 5 六5班 陆曼 女       88    84     69     8     10
## 6 六5班 胡玉洁 女       74    61     52     9      6
## # ... with 29 more rows
```

也可以只根据某些列判定重复：

```
df_dup %>%
```

```
distinct(sex, math, .keep_all = TRUE) # 只根据 sex 和 math 判定重复
```

```
## # A tibble: 32 x 8
##   class name sex   chinese math english moral science
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl>
## 1 六4班 周婵 女       92    94     77    10      9
## 2 六2班 杨远芸 女       93    80     68     9     10
## 3 六2班 陈华健 男       92    84     70     9     10
## 4 六1班 陈芳妹 女       79    87     66     9     10
## 5 六5班 陆曼 女       88    84     69     8     10
## 6 六5班 胡玉洁 女       74    61     52     9      6
## # ... with 26 more rows
```

注：默认只返回选择的列，要返回所有列，需要设置参数 `.keep_all = TRUE`。

## (2) 删除包含 NA 的行

用 `tidyr` 包中的 `drop_na()` 删除所有包含 NA 的行：

```
df_dup %>%
```

```
drop_na()
```

```
## # A tibble: 38 x 8
##   class name sex   chinese math english moral science
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl>
## 1 六4班 周婵 女       92    94     77    10      9
```

```
## 2 六2班 杨远芸 女          93    80    68    9    10
## 3 六2班 陈华健 男          92    84    70    9    10
## 4 六1班 陈芳妹 女          79    87    66    9    10
## 5 六5班 陆曼 女           88    84    69    8    10
## 6 六5班 胡玉洁 女         74    61    52    9    6
## # ... with 32 more rows
```

也可以只删除某些列包含 NA 的行:

```
df_dup %>%
```

```
  drop_na(sex:math)
```

```
## # A tibble: 50 x 8
##   class name sex   chinese math english moral science
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl>
## 1 六4班 周婵 女       92    94     77    10     9
## 2 六2班 杨远芸 女       93    80     68    9    10
## 3 六2班 陈华健 男       92    84     70    9    10
## 4 六1班 陈芳妹 女       79    87     66    9    10
## 5 六5班 陆曼 女       88    84     69    8    10
## 6 六5班 胡玉洁 女       74    61     52    9    6
## # ... with 44 more rows
```

## 2. 对行切片: slice\_\*()

slice 就是对行切片的意思, 该系列函数的共同参数:

- n: 用来指定要选择的行数
- prop: 用来指定选择的行比例

<code>slice(df, 3:7)</code>	# 选择 3-7 行
<code>slice_head(df, n, prop)</code>	# 从前面开始选择若干行
<code>slice_tail(df, n, prop)</code>	# 从后面开始选择若干行
<code>slice_min(df, order_by, n, prop)</code>	# 根据 order_by 选择最小的若干行
<code>slice_max(df, order_by, n, prop)</code>	# 根据 order_by 选择最大的若干行
<code>slice_sample(df, n, prop)</code>	# 随机选择若干行

选择 math 列值中前 5 大的行:

```
df %>%
```

```
  slice_max(math, n = 5)
```

```
## # A tibble: 5 x 8
```

```
##   class name sex  chinese math english moral science
##   <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     NA     8     6
## 3 六1班 何娜 女      87    92     79     9    10
## 4 六5班 符苡榕 女      85    89     76     9    NA
## 5 六2班 黄祖娜 女      94    88     75    10    10
```

### 3. 用 `filter()` 根据值或条件筛选行

```
df_dup %>%
  filter(sex == "男", math > 80)
```

```
## # A tibble: 8 x 8
##   class name sex  chinese math english moral science
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl>
## 1 六2班 陈华健 男      92    84     70     9    10
## 2 六2班 陈华健 男      92    84     70     9    10
## 3 六4班 <NA> 男      84    85     52     9     8
## 4 六2班 陈华健 男      92    84     70     9    10
## 5 六4班 李小龄 男      90    87     69    10    10
## 6 六4班 李小龄 男      90    87     69    10    10
## # ... with 2 more rows
```

注：多个条件之间用“,” 隔开，相当于 `and`。

```
df_dup %>%
  filter(sex == "女", (is.na(english) | math > 80))
```

```
## # A tibble: 11 x 8
##   class name sex  chinese math english moral science
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl>
## 1 六4班 周婵 女      92    94     77    10     9
## 2 六1班 陈芳妹 女      79    87     66     9    10
## 3 六5班 陆曼 女      88    84     69     8    10
## 4 六5班 陆曼 女      88    84     69     8    10
## 5 六2班 徐雅琦 女      92    86     72    NA     9
## 6 六5班 陆曼 女      88    84     69     8    10
## # ... with 5 more rows
```



```
df_dup %>%
  filter(between(math, 70, 80)) # 闭区间
```

```
## # A tibble: 15 x 8
##   class name sex   chinese math english moral science
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl>
## 1 六2班 杨远芸 女       93    80     68    9      10
## 2 六5班 容唐   女       83    71     56    9       7
## 3 六4班 关小孟 男       84    78     49    8       5
## 4 六1班 陈祝贞 女       76    79     67    8      10
## 5 六1班 陈欣越 男       57    80     60    9       9
## 6 六1班 雷旺   男        NA    80     68    8       9
## # ... with 9 more rows
```

#### 4. 在限定列范围内根据条件筛选行

结合 `across()` 及选择列语法，可以在限定列范围内，根据应用函数得到的结果作为条件筛选行。

##### (1) 限定列范围内，筛选“所有值都满足某条件的行”

选出所有列范围内，所有值都 > 75 的行：

```
df[, 4:6] %>%
  filter(across(everything(), ~ .x > 75)) # 不能套 all_var
```

```
## # A tibble: 3 x 3
##   chinese math english
##   <dbl> <dbl>   <dbl>
## 1     87    92     79
## 2     92    94     77
## 3     85    89     76
```

注：`across()` 等价于 `all_var()`

选出所有列范围内，所有值都不是 NA 的行

```
df_dup %>%
  filter(across(everything(), ~ !is.na(.x)))
```

```
## # A tibble: 38 x 8
##   class name sex   chinese math english moral science
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl>
```

```
## 1 六4班 周婵 女          92    94    77    10     9
## 2 六2班 杨远芸 女        93    80    68     9    10
## 3 六2班 陈华健 男        92    84    70     9    10
## 4 六1班 陈芳妹 女        79    87    66     9    10
## 5 六5班 陆曼 女          88    84    69     8    10
## 6 六5班 胡玉洁 女        74    61    52     9     6
## # ... with 32 more rows
```

## (2) 限定列范围内，筛选“存在值满足某条件的行”

在限定的列范围内，选择“存在值满足某条件的行”，目前的支持还不好，暂时需要借助如下函数实现：

```
rowAny = function(x) rowSums(x, na.rm = TRUE) > 0
```

选出所有列范围内，存在值包含“bl”的行

```
starwars %>%
  filter(rowAny(across(everything(), ~ str_detect(.x, "bl"))))

## # A tibble: 47 x 14
##   name height mass hair_color skin_color eye_color birth_year sex
##   <chr>   <int> <dbl> <chr>      <chr>      <chr>      <dbl> <chr>
## 1 Luke~    172    77 blond      fair        blue          19 male
## 2 R2-D2     96    32 <NA>      white, bl~ red          33 none
## 3 Owen~    178   120 brown, gr~ light        blue          52 male
## 4 Beru~    165    75 brown      light        blue          47 fema~
## 5 Bigg~    183    84 black      light        brown          24 male
## 6 Obi-~    182    77 auburn, w~ fair        blue-gray      57 male
## # ... with 41 more rows, and 6 more variables: gender <chr>,
## #   homeworld <chr>, species <chr>, films <list>, vehicles <list>,
## #   starships <list>
```

选出所有列范围内，存在值 > 90 的行

```
df[, 4:6] %>%
  filter(rowAny(across(everything(), ~ .x > 90)))

## # A tibble: 8 x 3
##   chinese math english
##   <dbl> <dbl>   <dbl>
## 1     87    92     79
## 2     95    77     75
```

```
## 3      94      88      75
## 4      92      86      72
## 5      92      84      70
## 6      93      80      68
## # ... with 2 more rows
```

从数学逻辑上来说,【存在值 >90】等价于【非“所有值都 ≤ 90”】，这在 iris 上测试成功，但本例只返回 6 个观测，不知道是不是 bug。

从字符列范围内，选择包含（存在）NA 的行：

```
df_dup %>%
  filter(rowAny(across(where(is.character), any_vars(is.na(.)))))
```

```
## # A tibble: 22 x 8
##   class name sex  chinese math english moral science
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl>
## 1 六3班 洪琦希 男      NA    31     69     6     4
## 2 六3班 邹嘉伟 男      67    18     62     8    NA
## 3 六1班 黄亦婷 女      77    NA     54     8    10
## 4 六4班 <NA>   男      84    85     52     9     8
## 5 六1班 雷旺   男      NA    80     68     8     9
## 6 六1班 黄亦婷 女      77    NA     54     8    10
## # ... with 16 more rows
```

### 2.5.3 对行排序

用 dplyr 包中的 arrange() 对行排序，默认是递增。

```
df_dup %>%
  arrange(math, sex)
```

```
## # A tibble: 60 x 8
##   class name sex  chinese math english moral science
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl>
## 1 六3班 邹嘉伟 男      67    18     62     8    NA
## 2 六3班 刘虹均 男      72    23     74     3     6
## 3 六3班 刘虹均 男      72    23     74     3     6
## 4 六3班 黄凯丽 女      70    23     61     4     4
## 5 六3班 黄凯丽 女      70    23     61     4     4
## 6 六3班 黄凯丽 女      70    23     61     4     4
## # ... with 54 more rows
```

若要递减排序，套一个 `desc()`：

```
df_dup %>%
  arrange(desc(math))          # 递减排序

## # A tibble: 60 x 8
##   class name sex   chinese math english moral science
##   <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     NA     8      6
## 3 六5班 符苡榕 女       85    89     76     9     NA
## 4 六5班 符苡榕 女       85    89     76     9     NA
## 5 六1班 陈芳妹 女       79    87     66     9    10
## 6 六4班 李小龄 男       90    87     69    10    10
## # ... with 54 more rows
```

### 2.5.4 修改列

修改列，即修改数据框的列，计算新列。

#### 1. 创建新列

用 `dplyr` 包中的 `mutate()` 创建或修改列，返回原数据框并增加新列；若改用 `transmute()` 则只返回增加的新列。

若只给新列 1 个值，则循环使用得到值相同的一列：

```
df %>%
  mutate(new_col = 5)

## # A tibble: 50 x 9
##   class name sex   chinese math english moral science new_col
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl>   <dbl>
## 1 六1班 何娜 女       87    92     79     9    10      5
## 2 六1班 黄才菊 女       95    77     75    NA     9      5
## 3 六1班 陈芳妹 女       79    87     66     9    10      5
## 4 六1班 陈学勤 男       NA    79     66     9    10      5
## 5 六1班 陈祝贞 女       76    79     67     8    10      5
## 6 六1班 何小薇 女       83    73     65     8     9      5
## # ... with 44 more rows
```

正常是以长度等于行数的向量赋值：

```
df %>%
  mutate(new_col = 1:n())
```

```
## # A tibble: 50 x 9
##   class name sex   chinese math english moral science new_col
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl>   <int>
## 1 六1班 何娜 女       87    92     79     9     10     1
## 2 六1班 黄才菊 女      95    77     75    NA     9     2
## 3 六1班 陈芳妹 女      79    87     66     9    10     3
## 4 六1班 陈学勤 男      NA    79     66     9    10     4
## 5 六1班 陈祝贞 女      76    79     67     8    10     5
## 6 六1班 何小薇 女      83    73     65     8     9     6
## # ... with 44 more rows
```

注：n() 返回当前分组的样本数，未分组则为总行数。

## 2. 计算新列

用数据框的列计算新列，若修改当前列，只需要赋值给原列名。

```
df %>%
  mutate(total = chinese + math + english + moral + science)
```

```
## # A tibble: 50 x 9
##   class name sex   chinese math english moral science total
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl> <dbl>
## 1 六1班 何娜 女       87    92     79     9     10    277
## 2 六1班 黄才菊 女      95    77     75    NA     9     NA
## 3 六1班 陈芳妹 女      79    87     66     9     10    251
## 4 六1班 陈学勤 男      NA    79     66     9     10     NA
## 5 六1班 陈祝贞 女      76    79     67     8     10    240
## 6 六1班 何小薇 女      83    73     65     8     9     238
## # ... with 44 more rows
```

注意：不能用 sum()，它会将整个列的内容都加起来，类似的还有 mean()。

在同一个 mutate() 中可以同时创建或计算多个列，它们是从前往后依次计算，所以可以使用前面新创建的列，例如

- 计算 df 中 math 列的中位数
- 创建标记 math 是否大于中位数的逻辑值列
- 用 as.numeric() 将 TRUE/FALSE 转化为 1/0

```
df %>%
  mutate(med = median(math, na.rm = TRUE),
         label = math > med,
         label = as.numeric(label))
```

```
## # A tibble: 50 x 10
##   class name sex   chinese math english moral science   med label
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl> <dbl> <dbl>
## 1 六1班 何娜 女      87    92     79     9     10    73     1
## 2 六1班 黄才菊 女      95    77     75    NA     9     73     1
## 3 六1班 陈芳妹 女      79    87     66     9     10    73     1
## 4 六1班 陈学勤 男      NA    79     66     9     10    73     1
## 5 六1班 陈祝贞 女      76    79     67     8     10    73     1
## 6 六1班 何小薇 女      83    73     65     8     9     73     0
## # ... with 44 more rows
```

### 3. 修改多列

结合 `across()` 和选择列语法可以应用函数到多列，从而实现同时修改多列。

#### (1) 应用函数到所有列

将所有列转化为字符型：

```
df %>%
  mutate(across(everything(), as.character))
```

```
## # A tibble: 50 x 8
##   class name sex   chinese math english moral science
##   <chr> <chr> <chr>   <chr>   <chr>   <chr>   <chr>
## 1 六1班 何娜 女      87      92     79      9     10
## 2 六1班 黄才菊 女      95      77     75    <NA>     9
## 3 六1班 陈芳妹 女      79      87     66      9     10
## 4 六1班 陈学勤 男    <NA>      79     66      9     10
## 5 六1班 陈祝贞 女      76      79     67      8     10
## 6 六1班 何小薇 女      83      73     65      8      9
## # ... with 44 more rows
```

#### (2) 应用函数到满足条件的列

对所有数值列做归一化：

```
rescale = function(x) {
  rng = range(x, na.rm = TRUE)
  (x - rng[1]) / (rng[2] - rng[1])
}
df %>%
  mutate(across(where(is.numeric), rescale))
```

```
## # A tibble: 50 x 8
##   class name sex  chinese math english moral science
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl>  <dbl>   <dbl>
## 1 六1班 何娜 女    0.843 0.974    1    0.875    1
## 2 六1班 黄才菊 女    1    0.776    0.926 NA    0.833
## 3 六1班 陈芳妹 女    0.686 0.908    0.759 0.875    1
## 4 六1班 陈学勤 男    NA    0.803    0.759 0.875    1
## 5 六1班 陈祝贞 女    0.627 0.803    0.778 0.75    1
## 6 六1班 何小薇 女    0.765 0.724    0.741 0.75    0.833
## # ... with 44 more rows
```

### (3) 应用函数到指定的列

将 iris 中的 length 和 width 测量单位从厘米变成毫米:

```
as_tibble(iris) %>%
  mutate(across(contains("Length") | contains("Width"), ~ .x * 10))
```

```
## # A tibble: 150 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width 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
## 6           54         39         17           4 setosa
## # ... with 144 more rows
```

## 4. 替换 NA

### (1) replace\_na()

实现用某个值替换一行中的所有 NA 值，该函数接受一个命名列表，其成分为 列名 = 替换值：

```
starwars %>%
  replace_na(list(hair_color = "UNKNOWN", height = mean(. $height, na.rm = TRUE)))

## # A tibble: 87 x 14
##   name height mass hair_color skin_color eye_color birth_year sex
##   <chr> <dbl> <dbl> <chr>      <chr>      <chr>      <dbl> <chr>
## 1 Luke~    172    77 blond      fair        blue         19 male
## 2 C-3PO    167    75 UNKNOWN   gold        yellow       112 none
## 3 R2-D2     96    32 UNKNOWN   white, bl~ red         33 none
## 4 Dart~    202   136 none      white      yellow      41.9 male
## 5 Leia~    150    49 brown     light      brown        19 fema~
## 6 Owen~    178   120 brown, gr~ light      blue         52 male
## # ... with 81 more rows, and 6 more variables: gender <chr>,
## #   homeworld <chr>, species <chr>, films <list>, vehicles <list>,
## #   starships <list>
```

### (2) fill()

用前一个（或后一个）非缺失值填充 NA。有些表在记录时，会省略与上一条记录相同的内容，如下表：

```
load("datas/gap_data.rda")
knitr::kable(gap_data, align="c")
```

site	species	sample_num	bees_present
Bilpin	A. longifolia	1	TRUE
NA	NA	2	TRUE
NA	NA	3	TRUE
NA	A. elongata	1	TRUE
NA	NA	2	FALSE
NA	NA	3	TRUE
Grose Vale	A. terminalis	1	FALSE
NA	NA	2	FALSE
NA	NA	2	TRUE

tidyr 包中的 fill() 适合处理这种结构的缺失值，默认是向下填充，即用上一个



非缺失值填充:

```
gap_data %>%
  fill(site, species)

## # A tibble: 9 x 4
##   site species sample_num bees_present
##   <chr> <chr>      <dbl> <lgl>
## 1 Bilpin A. longifolia      1 TRUE
## 2 Bilpin A. longifolia      2 TRUE
## 3 Bilpin A. longifolia      3 TRUE
## 4 Bilpin A. elongata        1 TRUE
## 5 Bilpin A. elongata        2 FALSE
## 6 Bilpin A. elongata        3 TRUE
## # ... with 3 more rows
```

## 5. 重新编码

实际中，经常需要对列中的值进行重新编码。

### (1) 两类别情形：if\_else()

用 if\_else() 作 是/否 决策以确定用哪个值做重新编码：

```
df %>%
  mutate(sex = if_else(sex == "男", "M", "F"))

## # A tibble: 50 x 8
##   class name sex chinese math english moral science
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl>
## 1 六1班 何娜 F      87    92     79     9     10
## 2 六1班 黄才菊 F      95    77     75    NA     9
## 3 六1班 陈芳妹 F      79    87     66     9     10
## 4 六1班 陈学勤 M      NA    79     66     9     10
## 5 六1班 陈祝贞 F      76    79     67     8     10
## 6 六1班 何小薇 F      83    73     65     8     9
## # ... with 44 more rows
```

### (2) 多类别情形：case\_when()

用 case\_when() 做更多条件下的重新编码，避免使用很多 if\_else() 嵌套：

```
df %>%
  mutate(math = case_when(math >= 75 ~ "High",
                           math >= 60 ~ "Middle",
                           TRUE      ~ "Low"))
```

```
## # A tibble: 50 x 8
##   class name  sex  chinese math  english moral science
##   <chr> <chr> <chr>   <dbl> <chr>   <dbl> <dbl>   <dbl>
## 1 六1班 何娜 女      87 High    79     9     10
## 2 六1班 黄才菊 女      95 High    75    NA     9
## 3 六1班 陈芳妹 女      79 High    66     9     10
## 4 六1班 陈学勤 男      NA High    66     9     10
## 5 六1班 陈祝贞 女      76 High    67     8     10
## 6 六1班 何小薇 女      83 Middle   65     8     9
## # ... with 44 more rows
```

`case_when()` 中用的是公式形式,

- 左边是返回 `TRUE` 或 `FALSE` 的表达式或函数
- 右边是若左边表达式为 `TRUE`, 则重新编码的值, 也可以是表达式或函数
- 每个分支条件将从上到下的计算, 并接受第一个 `TURE` 条件
- 最后一个分支直接用 `TRUE` 表示若其它条件都不为 `TRUE` 时怎么做

### (3) 更强大的重新编码函数

基于 `tidyverse` 设计哲学, `sjmisc` 包实现了对变量做数据变换, 如重新编码、二分或分组变量、设置与替换缺失值等; `sjmisc` 包也支持标签化数据, 这对操作 `SPSS` 或 `Stata` 数据集特别有用。

重新编码函数 `rec()`, 可以将变量的旧值重新编码为新值, 基本格式为:

```
rec(x, rec, append, ...)
```

- `x`: 为数据框 (或向量);
- `append`: 默认为 `TRUE`, 则返回包含重编码新列的数据框, `FALSE` 则只返回重编码的新列;
- `rec`: 设置重编码模式, 即哪些旧值被哪些新值取代, 具体如下:
  - 重编码对: 每个重编码对用 “;” 隔开, 例如 `rec="1=1; 2=4; 3=2; 4=3"`
  - 多值: 多个旧值 (逗号分隔) 重编码为一个新值, 例如 `rec="1,2=1; 3,4=2"`
  - 值范围: 用冒号表示值范围, 例如 `rec="1:4=1; 5:8=2"`
  - 数值型值范围: 带小数部分的数值向量, 值范围内的所有值将被重新编码, 例如 `rec="1:2.5=1; 2.6:3=2"a`
  - “min” 和 “max”: 最小值和最大值分别用 `min` 和 `max` 表示, 例如 `rec = "min:4=1; 5:max=2"` (`min` 和 `max` 也可以作为新值, 如 `5:7=max`, 表示将 5~7 编码为 `max(x)`)
  - “else”: 所有未设定的其它值, 用 `else` 表示, 例如 `rec="3=1; 1=2; else=3"`
  - “copy”: `else` 可以结合 `copy` 一起使用, 表示所有未设定的其它值保持原样 (从原数值 `copy`), 例如 `rec="3=1; 1=2; else=copy"`
  - `NA`s: `NA` 既可以作为旧值, 也可以作为新值, 例如 `rec="NA=1; 3:5=NA"`
  - “rev”: 设置反转值顺序
  - 非捕获值: 不匹配的值将设置为 `NA`, 除非使用 `else` 和 `copy`.

<sup>a</sup>注意 2.55 因未包含在值范围将不被重新编码。

```
library(sjmisc)
```

```
df %>%
  rec(math, rec = "min:59= 不及格; 60:74= 中; 75:85= 良; 85:max= 优",
       append = FALSE) %>%
  frq()                                # 频率表

##
## math_r <character>
## # total N=50  valid N=50  mean=3.28  sd=1.26
##
## Value | N | Raw % | Valid % | Cum. %
## -----
## -Inf  | 3 | 6.00 | 6.00 | 6
```

```
## 不及格 | 14 | 28.00 | 28.00 | 34
## 良      | 10 | 20.00 | 20.00 | 54
## 优      | 12 | 24.00 | 24.00 | 78
## 中      | 11 | 22.00 | 22.00 | 100
## <NA>    | 0  | 0.00  | <NA>  | <NA>
```

注：新值的值标签可以在重新编码时一起设置，只需要在每个重编码对后接上中括号标签。

### 2.5.5 分组汇总

分组汇总，相当于 Excel 的透视表功能

对未分组的数据框，一些操作如 `mutate()` 是在所有行上执行——或者说，整个数据框是一个分组，所有行都属于它。

若数据框被分组，则这些操作是分别在每个分组上独立执行。可以认为是，将数据框拆分为更小的多个数据框。在每个更小的数据框上执行操作，最后再将结果合并回来。

#### 1. 创建分组

用 `group_by()` 创建分组，只是对数据框增加了分组信息（用 `group_keys()` 查看），并不是真的将数据分割为多个数据框。

```
df_grp = df %>%
  group_by(sex)
```

```
group_keys(df_grp)      # 分组键值（唯一识别分组）
group_indices(df_grp)    # 查看每一行属于哪一分组
group_rows(df_grp)       # 查看每一组包含哪些行
ungroup(df_grp)          # 解除分组
```

#### 其它分组函数

- 真正将数据框分割为多个分组： `group_split()`，返回列表，其每个成分是一个分组数据框
- 将数据框分组（`group_by`），再做嵌套（`nest`），生成嵌套数据框： `group_nest()`

```
iris %>%
  group_nest(Species)
```

```
## # A tibble: 3 x 2
##   Species      data
```

```
##   <fct>          <list<tbl_df[,4]>>
## 1 setosa         [50 x 4]
## 2 versicolor    [50 x 4]
## 3 virginica      [50 x 4]
```

- `purrr` 风格的分组迭代：将函数 `.f` 依次应用到分组数据框 `.data` 的每个分组上
  - `group_map(.data, .f, ...)`: 返回列表
  - `group_walk(.data, .f, ...)`: 不返回，只关心副作用
  - `group_modify(.data, .f, ...)`: 返回修改后的分组数据框

## 2. 分组汇总

对数据框做分组最主要的目的就是做分组汇总，汇总就是以某种方式组合行，用 `dplyr` 包中的 `summarise()` 函数实现，结果只保留分组列唯一值和新创建的汇总列。

### (1) `summarise()`

可以与很多自带或自定义的汇总函数连用，常用的汇总函数有：

- `n()`: 观测数
- `n_distinct(var)`: 变量 `var` 的唯一值数目
- `sum(var)`, `max(var)`, `min(var)`, ...
- `mean(var)`, `median(var)`, `sd(var)`, `IQR(var)`, ...

```
df %>%
  group_by(sex) %>%
  summarise(n = n(),
            math_avg = mean(math, na.rm = TRUE),
            math_med = median(math))
```

```
## # A tibble: 3 x 4
##   sex      n math_avg math_med
##   <chr> <int>   <dbl>   <dbl>
## 1 男      24    64.6      NA
## 2 女      25    70.8      NA
## 3 <NA>     1     85       85
```

函数 `summarise()`，配合 `across()` 可以对所选择的列做汇总。好处是可以借助辅助选择器或判断条件选择多列，还能在这些列上执行多个函数，只需要将它们放入一个列表。

## (2) 对某些列做汇总

```
df %>%
  group_by(class, sex) %>%
  summarise(across(contains("h"), mean, na.rm = TRUE))
```

```
## # A tibble: 12 x 5
## # Groups:   class [6]
##   class sex   chinese  math english
##   <chr> <chr>   <dbl> <dbl>   <dbl>
## 1 六1班 男       57    79.7    64.7
## 2 六1班 女      80.7    77.2    67.4
## 3 六2班 男      75.4    68.8    42.6
## 4 六2班 女      92.2    73.8    63.8
## 5 六3班 男       66    30.4    67.6
## 6 六3班 女      68.4    49.2    67.8
## # ... with 6 more rows
```

## (3) 对所有列做汇总

```
df %>%
  select(-name) %>%
  group_by(class, sex) %>%
  summarise(across(everything(), mean, na.rm = TRUE))
```

```
## # A tibble: 12 x 7
## # Groups:   class [6]
##   class sex   chinese  math english moral science
##   <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl>
## 1 六1班 男       57    79.7    64.7  8.67    9.33
## 2 六1班 女      80.7    77.2    67.4  8.33    9.57
## 3 六2班 男      75.4    68.8    42.6  8.8     9.25
## 4 六2班 女      92.2    73.8    63.8  8.33    9
## 5 六3班 男       66    30.4    67.6  4.6     4.75
## 6 六3班 女      68.4    49.2    67.8  6.25    7.2
## # ... with 6 more rows
```

## (4) 对满足条件的列做多种汇总

```
df_grp = df %>%
  group_by(class) %>%
  summarise(across(where(is.numeric),
                      list(sum=sum, mean=mean, min=min), na.rm = TRUE))
df_grp
```

```
## # A tibble: 6 x 16
##   class chinese_sum chinese_mean chinese_min math_sum math_mean
##   <chr>      <dbl>      <dbl>      <dbl>      <dbl>      <dbl>
## 1 六1班      622        77.8        57        702        78
## 2 六2班      746        82.9        66        570       71.2
## 3 六3班      606        67.3        44        349       38.8
## 4 六4班      850         85         72        771       77.1
## 5 六5班      726        72.6        58        720        72
## 6 <NA>       90         90         90         86        86
## # ... with 10 more variables: math_min <dbl>, english_sum <dbl>,
## #   english_mean <dbl>, english_min <dbl>, moral_sum <dbl>,
## #   moral_mean <dbl>, moral_min <dbl>, science_sum <dbl>,
## #   science_mean <dbl>, science_min <dbl>
```

可读性不好，再来个宽变长：

```
df_grp %>%
  pivot_longer(-class, names_to = c("Vars", ".value"), names_sep = "_")
```

```
## # A tibble: 30 x 5
##   class Vars      sum mean  min
##   <chr> <chr>   <dbl> <dbl> <dbl>
## 1 六1班 chinese  622 77.8   57
## 2 六1班 math     702 78     55
## 3 六1班 english  666 66.6   54
## 4 六1班 moral     76 8.44    8
## 5 六1班 science   95 9.5     9
## 6 六2班 chinese  746 82.9   66
## # ... with 24 more rows
```

### (5) 支持多返回值的汇总函数

`summarise()` 以前只支持一个返回值的汇总函数，如 `sum`, `mean` 等。现在也支持多返回值（返回向量值、甚至是数据框）的汇总函数，如 `range()`, `quantile()` 等。

```
qs = c(0.25, 0.5, 0.75)

df_q = df %>%
  group_by(sex) %>%
  summarise(math_qs = quantile(math, qs, na.rm = TRUE), q = qs)
df_q
```

```
## # A tibble: 9 x 3
## # Groups:   sex [3]
##   sex  math_qs    q
##   <chr>   <dbl> <dbl>
## 1 男      57.5  0.25
## 2 男      69    0.5
## 3 男      80    0.75
## 4 女      55    0.25
## 5 女      73    0.5
## 6 女     86.5  0.75
## # ... with 3 more rows
```

可读性不好，再来个长变宽：

```
df_q %>%
  pivot_wider(names_from = q, values_from = math_qs, names_prefix = "q_")

## # A tibble: 3 x 4
## # Groups:   sex [3]
##   sex  q_0.25 q_0.5 q_0.75
##   <chr>   <dbl> <dbl> <dbl>
## 1 男      57.5    69    80
## 2 女      55    73   86.5
## 3 <NA>    85    85    85
```

### 3. 分组计数

用 `count()` 按分类变量 `class` 和 `sex` 分组，并按分组大小排序：



```
df %>%
  count(class, sex, sort = TRUE)
```

```
## # A tibble: 12 x 3
##   class sex      n
##   <chr> <chr> <int>
## 1 六1班 女      7
## 2 六4班 男      6
## 3 六2班 男      5
## 4 六3班 男      5
## 5 六3班 女      5
## 6 六5班 女      5
## # ... with 6 more rows
```

对已分组的数据框，用 `tally()` 计数：

```
df %>%
  group_by(math_level = cut(math, breaks = c(0, 60, 75, 80, 100),
                             right = FALSE)) %>%
  tally()
```

```
## # A tibble: 5 x 2
##   math_level      n
##   <fct>         <int>
## 1 [0,60)         14
## 2 [60,75)        11
## 3 [75,80)         5
## 4 [80,100)       17
## 5 <NA>           3
```

`count()` 和 `tally()` 都有参数 `wt` 设置加权计数。

用 `add_count()` 和 `add_tally()` 可为数据集增加一列按分组变量分组的计数：

```
df %>%
  add_count(class, sex)
```

```
## # A tibble: 50 x 9
##   class name  sex  chinese  math english moral science      n
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl> <int>
## 1 六1班 何娜 女      87    92     79     9     10     7
## 2 六1班 黄才菊 女      95    77     75    NA     9     7
```

```
## 3 六1班 陈芳妹 女      79    87    66    9    10    7
## 4 六1班 陈学勤 男      NA    79    66    9    10    3
## 5 六1班 陈祝贞 女      76    79    67    8    10    7
## 6 六1班 何小微 女      83    73    65    8    9     7
## # ... with 44 more rows
```

本节部分内容参阅 (Hadley Wickham 2017), (Desi Quintans 2019), Vignettes of dplyr.

## 2.6 其它数据操作

### 2.6.1 按行汇总

通常的数据操作逻辑都是按列方式 (colwise), 这使得按行汇总很困难。

dplyr 包提供了 rowwise() 函数为数据框创建按行方式 (rowwise), 使用 rowwise() 后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:

```
rf = df %>%
  rowwise()
rf %>%
  mutate(total = sum(c(chinese, math, english)))
```

```
## # A tibble: 50 x 9
## # Rowwise:
##   class name sex  chinese math english moral science total
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl> <dbl>
## 1 六1班 何娜 女      87    92    79     9    10    258
## 2 六1班 黄才菊 女      95    77    75    NA     9    247
## 3 六1班 陈芳妹 女      79    87    66     9    10    232
## 4 六1班 陈学勤 男      NA    79    66     9    10     NA
## 5 六1班 陈祝贞 女      76    79    67     8    10    222
## 6 六1班 何小微 女      83    73    65     8     9    221
## # ... with 44 more rows
```

函数 c\_across() 是为按行方式 (rowwise) 在选定的列范围汇总数据而设计的, 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 total
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl> <dbl>
## 1 六1班 何娜 女      87    92     79     9      10    277
## 2 六1班 黄才菊 女     95    77     75    NA      9     NA
## 3 六1班 陈芳妹 女     79    87     66     9     10    251
## 4 六1班 陈学勤 男     NA    79     66     9     10     NA
## 5 六1班 陈祝贞 女     76    79     67     8     10    240
## 6 六1班 何小薇 女     83    73     65     8      9    238
## # ... with 44 more rows
```

若只是做按行求和或均值，直接用 `rowSums()` / `rowMeans()` 速度更快（不需要分割-汇总-合并），这里的 `rowwise` 行化后提供可以做更多的按行汇总的可能。

```
df %>%
  mutate(total = rowSums(across(where(is.numeric))))
```

```
## # A tibble: 50 x 9
##   class name sex  chinese math english moral science total
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl> <dbl>
## 1 六1班 何娜 女      87    92     79     9      10    277
## 2 六1班 黄才菊 女     95    77     75    NA      9     NA
## 3 六1班 陈芳妹 女     79    87     66     9     10    251
## 4 六1班 陈学勤 男     NA    79     66     9     10     NA
## 5 六1班 陈祝贞 女     76    79     67     8     10    240
## 6 六1班 何小薇 女     83    73     65     8      9    238
## # ... with 44 more rows
```

按行方式 (**rowwise**) 可以理解作为一种特殊的分组：每一行作为一组。为 `rowwise()` 提供行 ID，用 `summarise()` 做汇总更能体会这一点：

```
df %>%
  rowwise(name) %>%
  summarise(total = sum(c_across(where(is.numeric))))
```

```
## # A tibble: 50 x 2
## # Groups:   name [50]
##   name    total
##   <chr>   <dbl>
## 1 何娜     277
## 2 黄才菊    NA
## 3 陈芳妹   251
## 4 陈学勤    NA
```

```
## 5 陈祝贞    240
## 6 何小微    238
## # ... with 44 more rows
```

`rowwise` 行化更让人惊喜的是：它的逐行处理的逻辑 + 嵌套数据框可以更好地实现批量建模，在 `rowwise` 行化模式下，批量建模就像计算新列一样自然。批量建模[见下章]可以用“嵌套数据框 + `purrr::map_*()`”实现，但这种 `rowwise` 技术，具有异曲同工之妙。

## 2.6.2 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()`：从小到大排名(`ties.method="min"`)，若要从大到小排名需要套一个 `desc()`

```
df %>%
  mutate(ranks = min_rank(desc(math))) %>%
  arrange(ranks)
```

```
## # A tibble: 50 x 9
##   class name sex  chinese math english moral science ranks
##   <chr> <chr> <chr>   <dbl> <dbl>   <dbl> <dbl>   <dbl> <int>
## 1 六4班 周婵 女      92    94     77    10      9      1
## 2 六4班 陈丽丽 女      87    93     NA     8      6      2
## 3 六1班 何娜 女      87    92     79     9     10      3
## 4 六5班 符苡榕 女      85    89     76     9     NA      4
## 5 六2班 黄祖娜 女      94    88     75    10     10      5
## 6 六1班 陈芳妹 女      79    87     66     9     10      6
## # ... with 44 more rows
```

### 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
## 4 2019-09-05  星期四     3     30     2          1

```

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

R base 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

```

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

```

```

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

```

```

## # 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
```

### 2.6.3 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` 风格公式写法
- `...`: 用来传递 `.f` 的其它参数
- `.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”。

金融时间序列数据经常需要计算滑动平均，比如计算 `sales` 的 3 日滑动平均：

```
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
## 4 2019-09-05 星期四     3      30  2.5
```

输出每个滑动窗口更便于该 3 日滑动平均理解是如何计算的：

```
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
```

细心的读者可能发现了：上面计算的并不是真正的 3 日滑动平均，而是连续 3 个值的滑动平均。这是因为 `slide()` 函数默认是以行索引来滑动，如果日期也是连续日期这是没有问题的。但是若日期有跳跃，则结果可能不是你想要的。

那么，怎么计算真正的 3 日滑动平均呢？需要改用 `slide_index()` 函数，并提供日期索引，其基本格式为：

```
slide_index(.x, .i, .f, ...)
```

其中参数 `.i` 用来传递索引向量，实现根据“`.i` 的当前元 + 其前/后若干元”创建相应的 `.x` 的滑动窗口。

来看一下的连续 3 日滑动窗口与连续 3 值滑动窗口的区别：

```
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"
```

最后，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
## 4 2019-09-05 星期四     3     30   2.5
```

### 2.6.4 整洁计算

`tidyverse` 代码之所以这么“整洁、优雅”，访问列只需要提供列名，不需要加引号，不需要加数据框环境 `df$`，这是因为它内部采用了一套整洁计算（`tidy evaluation`）框架。

如果我们也想自定义这样的“整洁、优雅”函数，即在自定义函数中页这样“整洁、优雅”地传递参数，就需要掌握一点整洁计算的技术。

#### 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

数据屏蔽为直接使用带来了代码简洁，但作为函数参数时的间接使用，正常是环境变量，要想作为数据变量使用，则需要用两个大括号括起来 `{{var}}`：



```
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
```

若是字符向量形式，想作为数据变量，则需要在函数体中使用 `.data[[var]]`，这里 `.data` 是代替数据集的代词：

```
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的循环机制，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
```

同样地，整洁选择作为函数参数时的间接使用，也需要用两个大括号括起来 `{{vars}}`：

```
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    am  gear
##   <dbl> <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  0.727  4.09
## 2     6     7  19.7  183.  122.   3.59  3.12  18.0  0.571  0.429  3.86
## 3     8    14  15.1  353.  209.   3.23  4.00  16.8  0     0.143  3.29
## # ... with 1 more variable: carb <dbl>
```

若是字符向量形式，则需要借助函数 `all_of()` 或 `any_of()`，取决于你的选择：

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

最后，再来看使用数据屏蔽或整洁选择同时修改列名的用法：

```
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 mean_qsec
##   <dbl> <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>   <dbl>
## 1     0     4    22.9    136.    84.7     3.77     2.94    21.0
## 2     0     6    19.1    205.    115.     3.42     3.39    19.2
## 3     0     8    15.0    358.    194.     3.12     4.10    17.1
## 4     1     4    28.1     93.6    81.9     4.18     2.04    18.4
## 5     1     6    20.6    155.    132.     3.81     2.76    16.3
## 6     1     8    15.4    326.    300.     3.88     3.37    14.6
## # ... with 3 more variables: mean_vs <dbl>, mean_gear <dbl>,
## #   mean_carb <dbl>
```

## 2. 引用与反引用

创建 tidyverse 风格的整洁函数，另一种做法是使用引用与反引用机制。这需要额外的两个步骤：

- 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 = enquos(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()` 函数从引用中提取名字：

```

grouped_mean = function(data, summary_var, group_var) {
  summary_var = enquos(summary_var)
  group_var = enquos(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，我们还想让用于计算分组均值的 `group_var` 可以是任意多个，这就需要改用 `...` 参数，为了更好地应付这种传递特意将该参数放在最后一个位置。另外，将其它函数参数都增加 `.` 前缀是一个好的做法，因为可以降低与 `...` 参数的冲突风险。

```
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.
## 4     6     1 155
## 5     8     0 358.
## 6     8     1 326
```

... 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就不行了，仍需要借助引用和反引用，但是要改用 `enquos()` 和 `!!!`

```
grouped_mean = function(.data, .summary_var, ...) {
  summary_var = enquo(.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.
```

```
## 4          6          1    155
## 5          8          0    358.
## 6          8          1    326
```

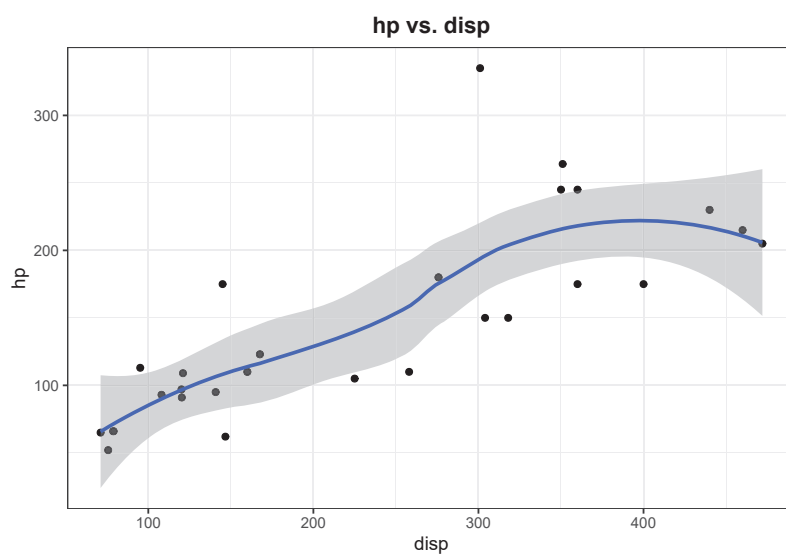
另外，参数 ... 也可以传递表达式：

```
filter_fun = function(df, ...) {
  df %>%
    filter(...)
}
mtcars %>% filter_fun(mpg > 25 & disp > 90)
```

```
##           mpg cyl  disp  hp drat   wt  qsec vs am gear carb
## Porsche 914-2 26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
## Lotus Europa 30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
```

最后，再来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)
```



本节部分内容参阅 [Vignettes of dplyr](#), [Vignettes of slider](#), (Lionel Henry 2020b), [Jesse Cambron: Practical Tidy Evaluation](#).

## 2.7 数据清洗实例