玩转数据 120 题——R 语言 tidyverse 版本 (更新版)

张敬信

2022-07-14

关于作者：

- 张敬信，哈尔滨商业大学，数学与应用数学，副教授
- 热爱学习，热爱编程，热爱 R 语言
- 我正在用最新 R 技术写一本《R 语言编程—基于 tidyverse》的书，欢迎您的阅读品鉴！

该书的 知乎交流平台, 欢迎您的留言讨论！

该书的 QQ 读者群：875664831, 交流、答疑，欢迎您的加入！

玩转数据 120 题来自刘早起的 *Pandas 进阶修炼 120 题*，涵盖了数据处理、计算、可视化等常用操作，希望通过 120 道精心挑选的习题吃透 pandas.

后来，中山大学博士陈熹提供了 R 语言版本。我[1][2]再来个更能体现 R 语言最新技术的 tidyverse 版本。

关于**更新版**：感谢 @ 鼠大米对部分解法不够 tidyverse 的题目，提供了新解法 (再加上我稍微修正)，主要是加入更好用的新函数 `slice_*()`。

先加载包：

[1]我的 Github: https://github.com/zhjx19

[2]我的知乎: https://www.zhihu.com/people/huc_zhangjingxin

```
library(tidyverse)
```

Part I 入门

题目 1（创建数据框）：将下面的字典创建为 DataFrame

data = {"grammer": ["Python","C","Java","GO",np.nan,"SQL","PHP","Python"], "score": [1,2,np.nan,4,5,6,7,10]}

难度：⋆

代码及运行结果：

```
df = tibble(
  grammer = c("Python","C","Java","GO", NA,"SQL","PHP","Python"),
  score = c(1,2,NA,4,5,6,7,10)
  )
df
```

```
## # A tibble: 8 x 2
##   grammer score
##   <chr>   <dbl>
## 1 Python      1
## 2 C           2
## 3 Java       NA
## 4 GO          4
## 5 <NA>        5
## 6 SQL         6
## # ... with 2 more rows
```

- 补充：按行录入式创建数据框

```
df = tribble(
  ~ grammer, ~ score,
  "Python",  1,
  "C",       2,
  "Java",    NA,
  "GO",      4,
  NA,        5,
  "SQL",     6,
  "PHP",     7,
```

```
  "Python",  10
)
```

问题 **2**（筛选行）：提取含有字符串”**Python**” 的行

难度：⋆

代码及运行结果：

```
df %>%
  filter(grammer == "Python")
```

```
## # A tibble: 2 x 2
##   grammer score
##   <chr>   <dbl>
## 1 Python      1
## 2 Python     10
```

题目 **3**（查看列名）：输出 **df** 的所有列名

难度：⋆

代码及运行结果：

```
names(df)
```

```
## [1] "grammer" "score"
```

题目 **4**（修改列名）：修改第 **2** 列列名为”**popularity**”

难度：⋆⋆

代码及运行结果：

```
df = df %>%
  rename(popularity = score)
df
```

```
## # A tibble: 8 x 2
##   grammer popularity
##   <chr>        <dbl>
```

```
## 1 Python           1
## 2 C                2
## 3 Java            NA
## 4 GO               4
## 5 <NA>             5
## 6 SQL              6
## # ... with 2 more rows
```

题目 5（统计频数）：统计 grammer 列中每种编程语言出现的次数

难度：★★

代码及运行结果：

```
# table(df$grammer)
# 或者
df %>%
  count(grammer)
```

```
## # A tibble: 7 x 2
##   grammer     n
##   <chr>   <int>
## 1 C           1
## 2 GO          1
## 3 Java        1
## 4 PHP         1
## 5 Python      2
## 6 SQL         1
## # ... with 1 more row
```

题目 6（缺失值处理）：将空值用上下值的平均值填充

难度：★★★

代码及运行结果：

```
df = df %>%
  mutate(popularity = zoo::na.approx(popularity))
df
```

```
## # A tibble: 8 x 2
```

```
##   grammer popularity
##   <chr>        <dbl>
## 1 Python           1
## 2 C                2
## 3 Java             3
## 4 GO               4
## 5 <NA>             5
## 6 SQL              6
## # ... with 2 more rows
```

注：tidyr 包提供了 `fill()` 函数，可以用前值或后值插补缺失值。

题目 7（筛选行）：提取 popularity 列中值大于 3 的行

难度：★★

代码及运行结果：

```
df %>%
  filter(popularity > 3)
```

```
## # A tibble: 5 x 2
##   grammer popularity
##   <chr>        <dbl>
## 1 GO               4
## 2 <NA>             5
## 3 SQL              6
## 4 PHP              7
## 5 Python          10
```

题目 8（数据去重）：按 grammer 列进行去重

难度：★★

代码及运行结果：

```
df %>%
  distinct(grammer, .keep_all = TRUE)
```

```
## # A tibble: 7 x 2
##   grammer popularity
```

```
##   <chr>        <dbl>
## 1 Python           1
## 2 C                2
## 3 Java             3
## 4 GO               4
## 5 <NA>             5
## 6 SQL              6
## # ... with 1 more row
```

题目 9（数据计算）：计算 popularity 列平均值

难度：★★

代码及运行结果：

```
df %>%
  summarise(popularity_avg = mean(popularity))
```

```
## # A tibble: 1 x 1
##   popularity_avg
##            <dbl>
## 1           4.75
```

题目 10（格式转换）：将 grammer 列转换为序列

难度：★

代码及运行结果：

```
# df$grammer

# 或者
# df %>%
#  .$grammer

# 或者
df %>%
  pull(grammer)
```

```
## [1] "Python" "C"      "Java"   "GO"     NA       "SQL"    "PHP"    "Python"
```

注：R 从数据框中提取出来就是字符向量。

题目 **11**（数据保存）：将数据框保存为 **Excel**

难度：★★

代码及运行结果：

```
writexl::write_xlsx(df, "datas/filename.xlsx")
```

题目 **12**（数据查看）：查看数据的行数列数

难度：★

代码及运行结果：

```
dim(df)
```

```
## [1] 8 2
```

题目 **13**（筛选行）：提取 **popularity** 列值大于 **3** 小于 **7** 的行

难度：★★

代码及运行结果：

```
df %>%
  filter(popularity > 3 & popularity < 7)
```

```
## # A tibble: 3 x 2
##   grammer popularity
##   <chr>        <dbl>
## 1 GO               4
## 2 <NA>             5
## 3 SQL              6
```

题目 **14**（调整列位置）：交互两列的位置

难度：★★

代码及运行结果：

```
df %>%
  select(popularity, grammer)
```

```
## # A tibble: 8 x 2
##   popularity grammer
##        <dbl> <chr>
## 1          1 Python
## 2          2 C
## 3          3 Java
## 4          4 GO
## 5          5 <NA>
## 6          6 SQL
## # ... with 2 more rows
```

注：可配合 `everything()` 放置“其余列”，更强大的调整列位置的函数是 dplyr1.0 将提供的 `relacate()`.

题目 15（筛选行）：提取 popularity 列最大值所在的行

难度：⋆⋆

代码及运行结果：

```
df %>%
  slice_max(popularity)
```

```
## # A tibble: 1 x 2
##   grammer popularity
##   <chr>        <dbl>
## 1 Python          10
```

```
# 或者
# df %>%
#  filter(popularity == max(popularity))
```

题目 16（查看数据）：查看最后几行数据

难度：⋆

代码及运行结果：

```
# tail(df)   # 默认是最后 6 行，或者
df %>%
  slice_tail(n = 6)
```

```
## # A tibble: 6 x 2
##   grammer popularity
##   <chr>        <dbl>
## 1 Java             3
## 2 GO               4
## 3 <NA>             5
## 4 SQL              6
## 5 PHP              7
## 6 Python          10
```

注：此外，dplyr 包还提供了 slice_head() 查看前 n 行或前某比例的行，slice_sample() 随机查看 n 行或某比例的行。

题目 17（修改数据）：删除最后一行数据

难度：★★

代码及运行结果：

```
df %>%
  slice(-n())
```

```
## # A tibble: 7 x 2
##   grammer popularity
##   <chr>        <dbl>
## 1 Python           1
## 2 C                2
## 3 Java             3
## 4 GO               4
## 5 <NA>             5
## 6 SQL              6
## # ... with 1 more row
```

题目 18（修改数据）：添加一行数据："Perl", 6

难度：★★

代码及运行结果：

```
df %>%
  bind_rows(tibble(grammer="Perl", popularity=6))
```

```
## # A tibble: 9 x 2
##   grammer popularity
##   <chr>        <dbl>
## 1 Python           1
## 2 C                2
## 3 Java             3
## 4 GO               4
## 5 <NA>             5
## 6 SQL              6
## # ... with 3 more rows
```

```
# 或者
# df %>%
#   add_row(grammer="Perl", popularity=6)
```

题目 19（数据整理）：对数据按 popularity 列值从到大到小排序

难度：★★

代码及运行结果：

```
df %>%
  arrange(-popularity)
```

```
## # A tibble: 8 x 2
##   grammer popularity
##   <chr>        <dbl>
## 1 Python          10
## 2 PHP              7
## 3 SQL              6
## 4 <NA>             5
## 5 GO               4
## 6 Java             3
## # ... with 2 more rows
```

```
# 或者
# df %>%
#  arrange(desc(popularity))
```

注：默认从小到大排序。

题目 20（字符统计）：统计 grammer 列每个字符串的长度

难度：★★

代码及运行结果：

```
df %>%
  mutate(strlen = str_length(grammer))
```

```
## # A tibble: 8 x 3
##   grammer popularity strlen
##   <chr>        <dbl>  <int>
## 1 Python           1      6
## 2 C                2      1
## 3 Java             3      4
## 4 GO               4      2
## 5 <NA>             5     NA
## 6 SQL              6      3
## # ... with 2 more rows
```

Part II 基础

题目 21（读取数据）：读取本地 Excel 数据

难度：★

代码及运行结果：

```
df = readxl::read_xlsx("datas/21-50 数据.xlsx")
df
```

```
## # A tibble: 135 x 3
##   createTime          education salary
##   <dttm>              <chr>     <chr>
## 1 2020-03-16 11:30:18 本科      20k-35k
## 2 2020-03-16 10:58:48 本科      20k-40k
## 3 2020-03-16 10:46:39 不限      20k-35k
## 4 2020-03-16 10:45:44 本科      13k-20k
## 5 2020-03-16 10:20:41 本科      10k-20k
## 6 2020-03-16 10:33:48 本科      10k-18k
## # ... with 129 more rows
```

题目 **22**（查看数据）：查看 **df** 数据的前几行

难度：⋆

代码及运行结果：

```
head(df)
```

```
## # A tibble: 6 x 3
##   createTime          education salary
##   <dttm>              <chr>     <chr>
## 1 2020-03-16 11:30:18 本科      20k-35k
## 2 2020-03-16 10:58:48 本科      20k-40k
## 3 2020-03-16 10:46:39 不限      20k-35k
## 4 2020-03-16 10:45:44 本科      13k-20k
## 5 2020-03-16 10:20:41 本科      10k-20k
## 6 2020-03-16 10:33:48 本科      10k-18k
```

```
# 或者
# df %>%
#  slice_head(n = 6)
```

题目 **23**（数据计算）：将 **salary** 列数据转换为最大值与最小值的平均值

难度：⋆⋆⋆

代码及运行结果：

```
df = df %>%
  separate(salary, into = c("low", "high"), sep = "-") %>%  # sep="-" 也可以省略
  mutate(salary = (parse_number(low) + parse_number(high)) * 1000 / 2) %>%
  select(-c(low, high))
df
```

```
## # A tibble: 135 x 3
##   createTime          education salary
##   <dttm>              <chr>      <dbl>
## 1 2020-03-16 11:30:18 本科       27500
## 2 2020-03-16 10:58:48 本科       30000
## 3 2020-03-16 10:46:39 不限       27500
## 4 2020-03-16 10:45:44 本科       16500
```

```
## 5 2020-03-16 10:20:41 本科        15000
## 6 2020-03-16 10:33:48 本科        14000
## # ... with 129 more rows
```

或者来个高级的，用正则表达式提取数字，定义做计算的函数，再 purrr::map_dbl 做循环计算：

```
calc = function(x) sum(as.numeric(unlist(x))) * 1000 / 2

df %>%
  mutate(salary = map_dbl(str_extract_all(salary, "\\d+"), calc))   # 结果同上 (略)
```

题目 24（分组汇总）：根据学历分组，并计算平均薪资

难度：★★★

代码及运行结果：

```
df %>%
  group_by(education) %>%
  summarise(salary_avg = mean(salary))
```

```
## # A tibble: 4 x 2
##   education salary_avg
##   <chr>          <dbl>
## 1 本科          19361.
## 2 不限          19600
## 3 大专          10000
## 4 硕士          20643.
```

题目 25（时间转换）：将 createTime 列转换为" 月-日"

难度：★★★

代码及运行结果：

```
library(lubridate)

df %>%
  mutate(createTime = str_sub(createTime, 6, 10))
```

```
## # A tibble: 135 x 3
```

```
##   createTime education salary
##   <chr>      <chr>      <dbl>
## 1 03-16      本科        27500
## 2 03-16      本科        30000
## 3 03-16      不限        27500
## 4 03-16      本科        16500
## 5 03-16      本科        15000
## 6 03-16      本科        14000
## # ... with 129 more rows
```

```
# 或者
# df %>%
#   mutate(createTime = str_extract(createTime, "(?<=-).*(?=\\s)"))

# 或者
# df %>%
#  mutate(createTime = str_c(str_pad(month(createTime), 2, pad="0"), "-", day(createTime)))
```

题目 26（查看数据）：查看数据结构信息

难度：⋆

代码及运行结果：

```
df %>%
  glimpse()       # 或者用 str()
```

```
## Rows: 135
## Columns: 3
## $ createTime <dttm> 2020-03-16 11:30:18, 2020-03-16 10:58:48, 2020-03-16 10:46~
## $ education  <chr> "本科", "本科", "不限", "本科", "本科", "本科", "硕士", "本~
## $ salary     <dbl> 27500, 30000, 27500, 16500, 15000, 14000, 23000, 12500, 700~
```

```
object.size(df)   # 查看对象占用内存
```

```
## 5112 bytes
```

题目 27（查看数据）：查看数据汇总信息

难度：⋆

代码及运行结果：

```
summary(df)
```

```
##    createTime                       education             salary
##  Min.   :2020-03-13 18:01:31.00   Length:135         Min.   : 3500
##  1st Qu.:2020-03-16 10:41:19.50   Class :character   1st Qu.:14000
##  Median :2020-03-16 11:00:27.00   Mode  :character   Median :17500
##  Mean   :2020-03-16 10:16:35.36                      Mean   :19159
##  3rd Qu.:2020-03-16 11:19:03.00                      3rd Qu.:25000
##  Max.   :2020-03-16 11:36:07.00                      Max.   :45000
```

题目 28（修改列）：新增一列将 salary 离散化为三水平值

难度：★★★★

代码及运行结果：

```
df = df %>%
  mutate(class = case_when(
    salary >= 0 & salary < 5000     ~ " 低",
    salary >= 5000 & salary < 20000 ~ " 中",
    TRUE                            ~ " 高"))  # TRUE 效果是其它
df
```

```
## # A tibble: 135 x 4
##   createTime          education salary class
##   <dttm>              <chr>      <dbl> <chr>
## 1 2020-03-16 11:30:18 本科       27500 高
## 2 2020-03-16 10:58:48 本科       30000 高
## 3 2020-03-16 10:46:39 不限       27500 高
## 4 2020-03-16 10:45:44 本科       16500 中
## 5 2020-03-16 10:20:41 本科       15000 中
## 6 2020-03-16 10:33:48 本科       14000 中
## # ... with 129 more rows
```

- 或者用 cut() 函数：

```
df %>%
  mutate(class = cut(salary,
                     breaks = c(0,5000,20000,Inf),
```

```
                    labels = c(" 低", " 中", " 高"),
                    right = FALSE))
```

- 或者用 sjmisc 包中的 rec()，和 SPSS 的重新编码一样强大。

```
df %>%
  mutate(class = sjmisc::rec(salary,
    rec = "min:5000 = 低; 5000:20000 = 中; 20000:max = 高"))
```

题目 29（数据整理）：按 salary 列对数据降序排列

难度：★★

代码及运行结果：

```
df %>%
  arrange(-salary)  # 或用 desc(salary)
```

```
## # A tibble: 135 x 4
##   createTime          education salary class
##   <dttm>              <chr>      <dbl> <chr>
## 1 2020-03-16 11:30:17 本科       45000 高
## 2 2020-03-16 11:04:00 本科       40000 高
## 3 2020-03-16 10:36:57 本科       37500 高
## 4 2020-03-16 11:01:39 本科       37500 高
## 5 2020-03-16 09:54:47 硕士       37500 高
## 6 2020-03-16 11:01:22 本科       35000 高
## # ... with 129 more rows
```

题目 30（筛选行）：提取第 33 行数据

难度：★

代码及运行结果：

```
df %>%
  slice(33)       # 或者用 df[33,]
```

```
## # A tibble: 1 x 4
##   createTime          education salary class
##   <dttm>              <chr>      <dbl> <chr>
## 1 2020-03-16 10:07:25 硕士       22500 高
```

题目 31（数据计算）：计算 salary 列的中位数

难度：⋆

代码及运行结果：

```
# median(df$salary)
# 或者
df %>%
  summarise(salary_med = median(salary))
```

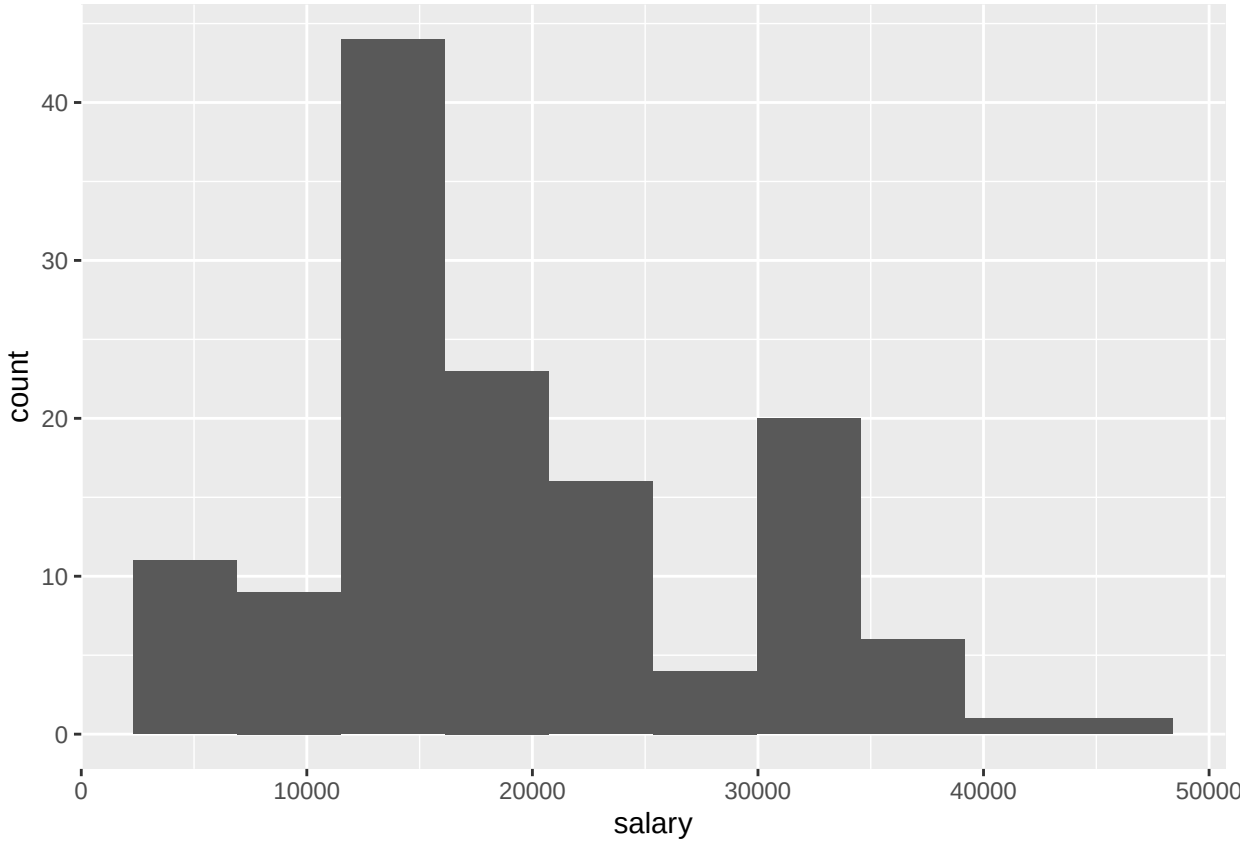
```
## # A tibble: 1 x 1
##   salary_med
##        <dbl>
## 1      17500
```

题目 32（数据可视化）：绘制 salary 的频率分布直方图

难度：⋆⋆⋆

代码及运行结果：

```
df %>%
  ggplot(aes(x = salary)) +
  geom_histogram(bins = 10)
```

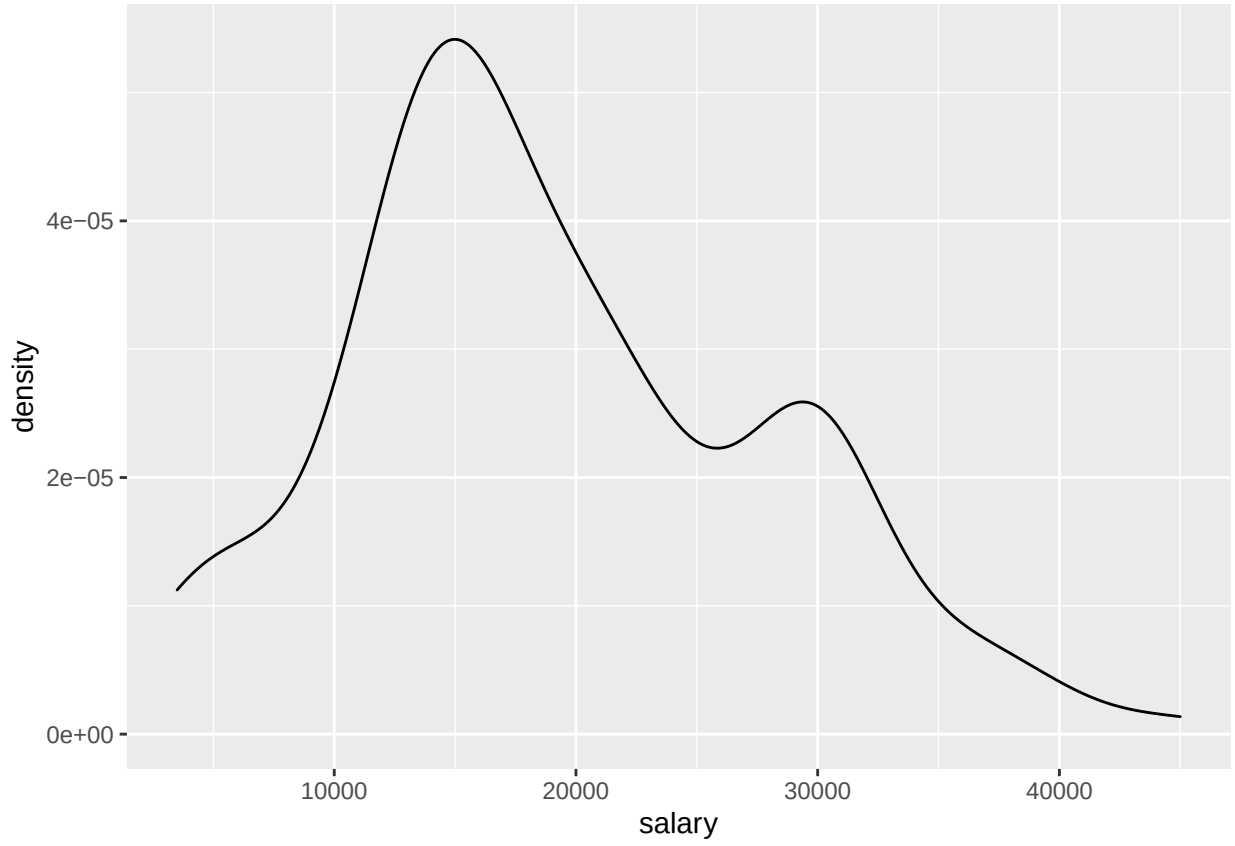


题目 **33**（数据可视化）：绘制 **salary** 的频率密度曲线图

难度：⋆⋆⋆

代码及运行结果：

```
df %>%
  ggplot(aes(x = salary)) +
  geom_density()
```



题目 34（数据删除）：删除最后一列 class

难度：⋆

代码及运行结果：

```
df %>%
  select(-class)
```

```
## # A tibble: 135 x 3
##   createTime          education salary
##   <dttm>              <chr>      <dbl>
## 1 2020-03-16 11:30:18 本科       27500
## 2 2020-03-16 10:58:48 本科       30000
## 3 2020-03-16 10:46:39 不限       27500
## 4 2020-03-16 10:45:44 本科       16500
## 5 2020-03-16 10:20:41 本科       15000
## 6 2020-03-16 10:33:48 本科       14000
## # ... with 129 more rows
```

```
# 或者
# df %>%
#  select(-last_col())  # 同 last_col(0)
```

题目 35（数据操作）：将 df 的第 1 列与第 2 列合并为新的一列

难度：★★

代码及运行结果：

```
df %>%
  unite("newcol", 1:2, sep = " ")
```

```
## # A tibble: 135 x 3
##   newcol                   salary class
##   <chr>                     <dbl> <chr>
## 1 2020-03-16 11:30:18 本科  27500 高
## 2 2020-03-16 10:58:48 本科  30000 高
## 3 2020-03-16 10:46:39 不限  27500 高
## 4 2020-03-16 10:45:44 本科  16500 中
## 5 2020-03-16 10:20:41 本科  15000 中
## 6 2020-03-16 10:33:48 本科  14000 中
## # ... with 129 more rows
```

题目 36（数据操作）：将 education 列与第 salary 列合并为新的一列

难度：★★

代码及运行结果：

```
df %>%
  unite("newcol", c(education, salary), sep = " ")
```

```
## # A tibble: 135 x 3
##   createTime          newcol     class
##   <dttm>              <chr>      <chr>
## 1 2020-03-16 11:30:18 本科 27500 高
## 2 2020-03-16 10:58:48 本科 30000 高
## 3 2020-03-16 10:46:39 不限 27500 高
## 4 2020-03-16 10:45:44 本科 16500 中
```

```
## 5 2020-03-16 10:20:41 本科 15000 中
## 6 2020-03-16 10:33:48 本科 14000 中
## # ... with 129 more rows
```

题目 37（数据计算）：计算 salary 最大值与最小值之差

难度：★★

代码及运行结果：

```
max(df$salary) - min(df$salary)
```

```
## [1] 41500
```

或者用

```
df %>%
  summarise(range = max(salary) - min(salary))
```

```
## # A tibble: 1 x 1
##   range
##   <dbl>
## 1 41500
```

题目 38（数据操作）：将第一行与最后一行拼接

难度：★★

代码及运行结果：

```
df %>%
  slice(1, n())
```

```
## # A tibble: 2 x 4
##   createTime          education salary class
##   <dttm>              <chr>      <dbl> <chr>
## 1 2020-03-16 11:30:18 本科       27500 高
## 2 2020-03-16 11:19:38 本科       30000 高
```

题目 39（数据操作）：将第 8 行添加到末尾

难度：⋆⋆

代码及运行结果：

```
df %>%
  bind_rows(slice(., 8))
```

```
## # A tibble: 136 x 4
##   createTime          education salary class
##   <dttm>              <chr>      <dbl> <chr>
## 1 2020-03-16 11:30:18 本科       27500 高
## 2 2020-03-16 10:58:48 本科       30000 高
## 3 2020-03-16 10:46:39 不限       27500 高
## 4 2020-03-16 10:45:44 本科       16500 中
## 5 2020-03-16 10:20:41 本科       15000 中
## 6 2020-03-16 10:33:48 本科       14000 中
## # ... with 130 more rows
```

题目 40（查看数据）：查看每一列的数据类型

难度：⋆

代码及运行结果：

```
df %>%
  glimpse()     # 或者用 str()
```

```
## Rows: 135
## Columns: 4
## $ createTime <dttm> 2020-03-16 11:30:18, 2020-03-16 10:58:48, 2020-03-16 10:46~
## $ education  <chr> "本科", "本科", "不限", "本科", "本科", "本科", "硕士", "本~
## $ salary     <dbl> 27500, 30000, 27500, 16500, 15000, 14000, 23000, 12500, 700~
## $ class      <chr> "高", "高", "高", "中", "中", "中", "高", "中", "中", "中",~
```

题目 41（数据操作）：将 createTime 列设置为行索引

难度：⋆⋆

代码及运行结果：

```
df %>%
  distinct(createTime, .keep_all = TRUE) %>%
  column_to_rownames("createTime") %>%
  head()
```

```
##                      education salary class
## 2020-03-16 11:30:18      本科  27500    高
## 2020-03-16 10:58:48      本科  30000    高
## 2020-03-16 10:46:39      不限  27500    高
## 2020-03-16 10:45:44      本科  16500    中
## 2020-03-16 10:20:41      本科  15000    中
## 2020-03-16 10:33:48      本科  14000    中
```

注：行索引不允许有重复，所以先做了一步去重。

题目 42（数据创建）：生成一个和 df 长度相同的随机数数据框

难度：★★

代码及运行结果：

```
df1 = tibble(rnums = sample.int(10, nrow(df), replace = TRUE))
  df1
```

```
## # A tibble: 135 x 1
##   rnums
##   <int>
## 1     9
## 2     1
## 3     5
## 4     3
## 5     3
## 6     3
## # ... with 129 more rows
```

题目 43（数据连接）：将上面生成的数据框与 df 按列合并

难度：★★

代码及运行结果：

```
df = bind_cols(df, df1)
df
```

```
## # A tibble: 135 x 5
##   createTime          education salary class rnums
##   <dttm>              <chr>      <dbl> <chr> <int>
## 1 2020-03-16 11:30:18 本科       27500 高        9
## 2 2020-03-16 10:58:48 本科       30000 高        1
## 3 2020-03-16 10:46:39 不限       27500 高        5
## 4 2020-03-16 10:45:44 本科       16500 中        3
## 5 2020-03-16 10:20:41 本科       15000 中        3
## 6 2020-03-16 10:33:48 本科       14000 中        3
## # ... with 129 more rows
```

题目 44（修改列）：生成新列 new 为 salary 列减去随机数列

难度：★★

代码及运行结果：

```
df = df %>%
  mutate(new = salary - rnums)
df
```

```
## # A tibble: 135 x 6
##   createTime          education salary class rnums   new
##   <dttm>              <chr>      <dbl> <chr> <int> <dbl>
## 1 2020-03-16 11:30:18 本科       27500 高        9 27491
## 2 2020-03-16 10:58:48 本科       30000 高        1 29999
## 3 2020-03-16 10:46:39 不限       27500 高        5 27495
## 4 2020-03-16 10:45:44 本科       16500 中        3 16497
## 5 2020-03-16 10:20:41 本科       15000 中        3 14997
## 6 2020-03-16 10:33:48 本科       14000 中        3 13997
## # ... with 129 more rows
```

题目 45（检查缺失值）：检查数据中是否含有任何缺失值

难度：★★

代码及运行结果：

```
anyNA(df)
```

```
## [1] FALSE
```

```
anyNA(df$salary)
```

```
## [1] FALSE
```

注：naniar 包提供了更强大的探索缺失值及缺失模式的函数，其中 `miss_var_summary()` 和 `miss_case_summary()` 可检查各列和各行缺失情况。

题目 46（类型转换）：将 salary 列的类型转换为浮点数

难度：★★

代码及运行结果：

```
df %>%
  mutate(rnums = as.double(rnums))
```

```
## # A tibble: 135 x 6
##   createTime          education salary class rnums   new
##   <dttm>              <chr>      <dbl> <chr> <dbl> <dbl>
## 1 2020-03-16 11:30:18 本科       27500 高        9 27491
## 2 2020-03-16 10:58:48 本科       30000 高        1 29999
## 3 2020-03-16 10:46:39 不限       27500 高        5 27495
## 4 2020-03-16 10:45:44 本科       16500 中        3 16497
## 5 2020-03-16 10:20:41 本科       15000 中        3 14997
## 6 2020-03-16 10:33:48 本科       14000 中        3 13997
## # ... with 129 more rows
```

题目 47（数据汇总）：计算 salary 列大于 10000 的次数

难度：★★★

代码及运行结果：

```
df %>%
  summarise(n = sum(salary > 10000))
```

```
## # A tibble: 1 x 1
##       n
##   <int>
## 1   119
```

或者用

```
df %>%
  count(salary > 10000)
```

```
## # A tibble: 2 x 2
##   `salary > 10000`     n
##   <lgl>            <int>
## 1 FALSE               16
## 2 TRUE               119
```

题目 48（统计频数）：查看每种学历出现的次数

难度：★★

代码及运行结果：

```
# table(df$education)
# 或者
df %>%
  count(education)
```

```
## # A tibble: 4 x 2
##   education     n
##   <chr>     <int>
## 1 本科        119
## 2 不限          5
## 3 大专          4
## 4 硕士          7
```

题目 49（数据汇总）：查看 education 列共有几种学历

难度：★★

代码及运行结果：

```
df %>%
  distinct(education)
```

```
## # A tibble: 4 x 1
##   education
##   <chr>
## 1 本科
## 2 不限
## 3 硕士
## 4 大专
```

题目 50（筛选行）：提取 salary 与 new 列之和大于 60000 的最后 3 行

难度：★★★★

代码及运行结果：

```
df %>%
  filter(salary + new > 60000) %>%
  slice_head(n = 3)
```

```
## # A tibble: 3 x 6
##   createTime          education salary class rnums   new
##   <dttm>              <chr>      <dbl> <chr> <int> <dbl>
## 1 2020-03-16 10:36:57 本科       37500 高        7 37493
## 2 2020-03-16 11:01:22 本科       35000 高        5 34995
## 3 2020-03-16 11:04:00 本科       40000 高        2 39998
```

Part III 提高

题目 51（读取数据）：使用绝对路径读取本地 Excel 数据

难度：★

代码及运行结果：

```
df = readxl::read_xls("datas/51-80 数据.xls")
df
```

```
## # A tibble: 327 x 18
##   代码      简称     日期                `前收盘价(元)` `开盘价(元)` `最高价(元)`
```

```
##   <chr>     <chr>   <dttm>                      <dbl>        <dbl>        <dbl>
## 1 600000.SH 浦发银~ 2016-01-04 00:00:00          16.1         16.1         16.1
## 2 600000.SH 浦发银~ 2016-01-05 00:00:00          15.7         15.5         16.0
## 3 600000.SH 浦发银~ 2016-01-06 00:00:00          15.9         15.8         16.0
## 4 600000.SH 浦发银~ 2016-01-07 00:00:00          16.0         15.7         15.8
## 5 600000.SH 浦发银~ 2016-01-08 00:00:00          15.5         15.7         15.8
## 6 600000.SH 浦发银~ 2016-01-11 00:00:00          15.4         15.2         15.4
## # ... with 321 more rows, and 12 more variables: `最低价(元)` <dbl>,
## #   `收盘价(元)` <dbl>, `成交量(股)` <chr>, `成交金额(元)` <chr>,
## #   `涨跌(元)` <dbl>, `涨跌幅(%)` <dbl>, `均价(元)` <chr>, `换手率(%)` <chr>,
## #   `A股流通市值(元)` <dbl>, `总市值(元)` <dbl>, `A股流通股本(股)` <dbl>,
## #   市盈率 <dbl>
```

题目 52（查看数据）：查看数据框的前 3 行

难度： ⋆

代码及运行结果：

```
head(df, 3)
```

```
## # A tibble: 3 x 18
##   代码      简称    日期                `前收盘价(元)` `开盘价(元)` `最高价(元)`
##   <chr>     <chr>   <dttm>                      <dbl>        <dbl>        <dbl>
## 1 600000.SH 浦发银~ 2016-01-04 00:00:00          16.1         16.1         16.1
## 2 600000.SH 浦发银~ 2016-01-05 00:00:00          15.7         15.5         16.0
## 3 600000.SH 浦发银~ 2016-01-06 00:00:00          15.9         15.8         16.0
## # ... with 12 more variables: `最低价(元)` <dbl>, `收盘价(元)` <dbl>,
## #   `成交量(股)` <chr>, `成交金额(元)` <chr>, `涨跌(元)` <dbl>,
## #   `涨跌幅(%)` <dbl>, `均价(元)` <chr>, `换手率(%)` <chr>,
## #   `A股流通市值(元)` <dbl>, `总市值(元)` <dbl>, `A股流通股本(股)` <dbl>,
## #   市盈率 <dbl>
```

```
# 或者
# df %>%
#  slice_head(n = 3)
```

题目 53（查看缺失值）：查看每列数据缺失值情况

难度： ⋆⋆

代码及运行结果：

```
library(naniar)

df %>%
  miss_var_summary()
```

```
## # A tibble: 18 x 3
##   variable     n_miss pct_miss
##   <chr>         <int>    <dbl>
## 1 代码              0        0
## 2 简称              0        0
## 3 日期              0        0
## 4 前收盘价(元)      0        0
## 5 开盘价(元)        0        0
## 6 最高价(元)        0        0
## # ... with 12 more rows
```

题目 54（查看缺失值）：查看日期列含有缺失值的行

难度：⋆⋆

代码及运行结果：

```
df %>%
  filter(is.na(日期))
```

```
## # A tibble: 0 x 18
## # ... with 18 variables: 代码 <chr>, 简称 <chr>, 日期 <dttm>,
## #   前收盘价(元) <dbl>, 开盘价(元) <dbl>, 最高价(元) <dbl>, 最低价(元) <dbl>,
## #   收盘价(元) <dbl>, 成交量(股) <chr>, 成交金额(元) <chr>, 涨跌(元) <dbl>,
## #   涨跌幅(%) <dbl>, 均价(元) <chr>, 换手率(%) <chr>, A股流通市值(元) <dbl>,
## #   总市值(元) <dbl>, A股流通股本(股) <dbl>, 市盈率 <dbl>
```

```
which(is.na(df$日期))   # 日期列缺失的行号
```

```
## integer(0)
```

题目 55（查看缺失值）：查看每列缺失值在哪些行

难度：⋆⋆⋆

代码及运行结果：

```
naIdx = df %>%
  where_na()          # 返回 NA 的行列索引，需要 naniar 包

split(naIdx[,1], naIdx[,2])
```

```
## named list()
```

题目 56（缺失值处理）：删除所有存在缺失值的行

难度：★★

代码及运行结果：

```
df %>%
  drop_na()
```

```
## # A tibble: 327 x 18
##   代码      简称    日期                `前收盘价(元)` `开盘价(元)` `最高价(元)`
##   <chr>     <chr>   <dttm>                       <dbl>        <dbl>        <dbl>
## 1 600000.SH 浦发银~ 2016-01-04 00:00:00           16.1         16.1         16.1
## 2 600000.SH 浦发银~ 2016-01-05 00:00:00           15.7         15.5         16.0
## 3 600000.SH 浦发银~ 2016-01-06 00:00:00           15.9         15.8         16.0
## 4 600000.SH 浦发银~ 2016-01-07 00:00:00           16.0         15.7         15.8
## 5 600000.SH 浦发银~ 2016-01-08 00:00:00           15.5         15.7         15.8
## 6 600000.SH 浦发银~ 2016-01-11 00:00:00           15.4         15.2         15.4
## # ... with 321 more rows, and 12 more variables: `最低价(元)` <dbl>,
## #   `收盘价(元)` <dbl>, `成交量(股)` <chr>, `成交金额(元)` <chr>,
## #   `涨跌(元)` <dbl>, `涨跌幅(%)` <dbl>, `均价(元)` <chr>, `换手率(%)` <chr>,
## #   `A股流通市值(元)` <dbl>, `总市值(元)` <dbl>, `A股流通股本(股)` <dbl>,
## #   市盈率 <dbl>
```

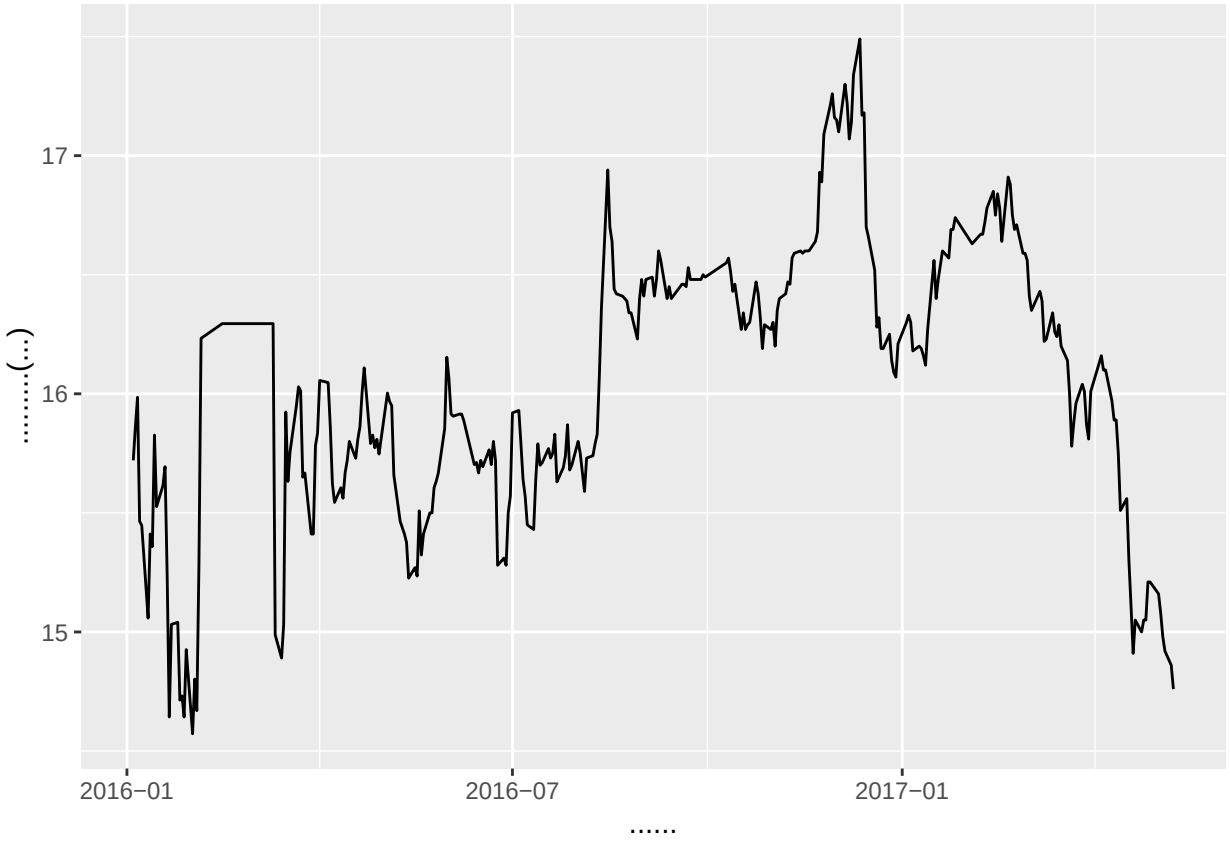
注：若要删除某些列包含缺失值的行，提供列名即可。

题目 57（数据可视化）：绘制收盘价的折线图

难度：★★

代码及运行结果：

```
df %>%
    ggplot(aes(日期, `收盘价 (元)`)) +
    geom_line()
```

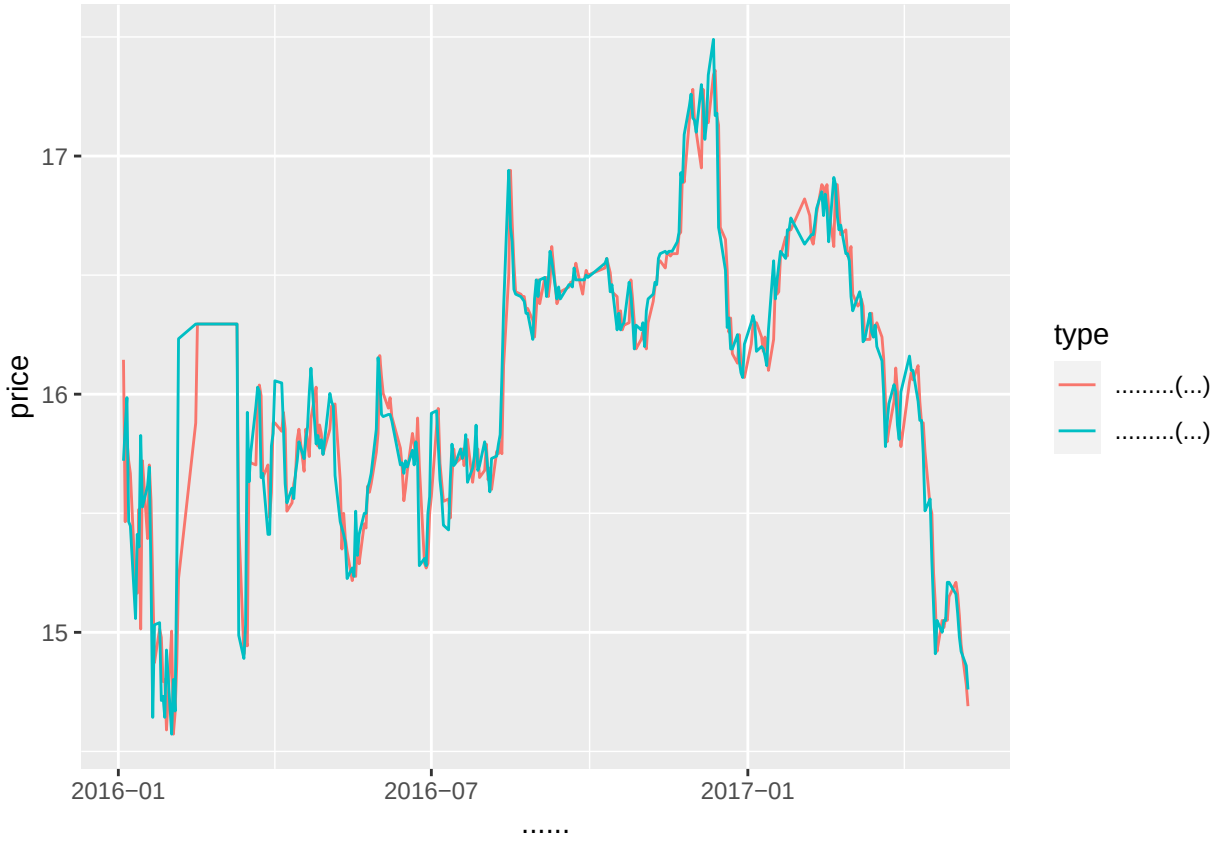


题目 58（数据可视化）：同时绘制开盘价与收盘价

难度： ⋆⋆⋆

代码及运行结果：

```
df %>%
  select(日期, `开盘价 (元)`, `收盘价 (元)`) %>%
  pivot_longer(-日期,
               names_to = "type",
               values_to = "price") %>%
  ggplot(aes(日期, price, color = type)) +
  geom_line()
```



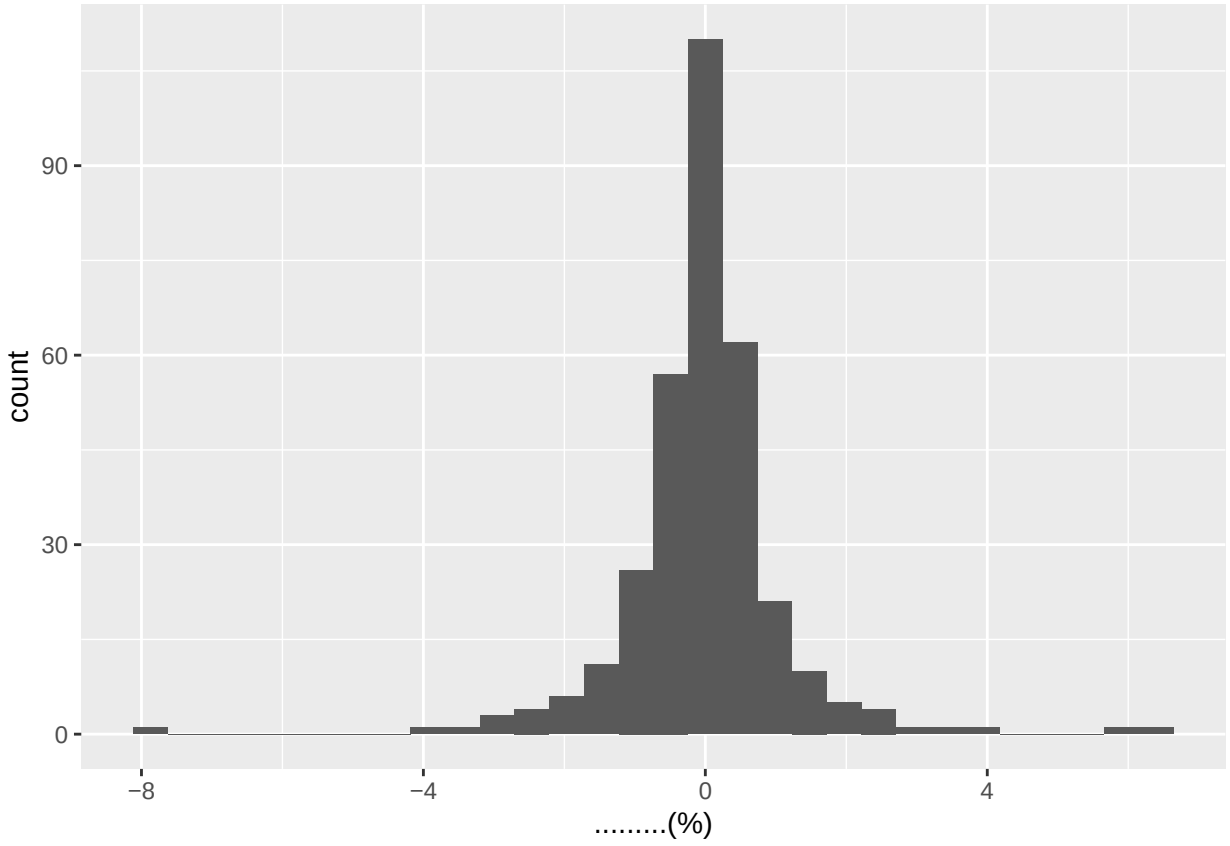
注：为了自动添加图例，先对数据做了宽变长转换。

题目 59（数据可视化）：绘制涨跌幅的直方图

难度：⋆⋆

代码及运行结果：

```
df %>%
  ggplot(aes(`涨跌幅 (%)`)) +
  geom_histogram()
```

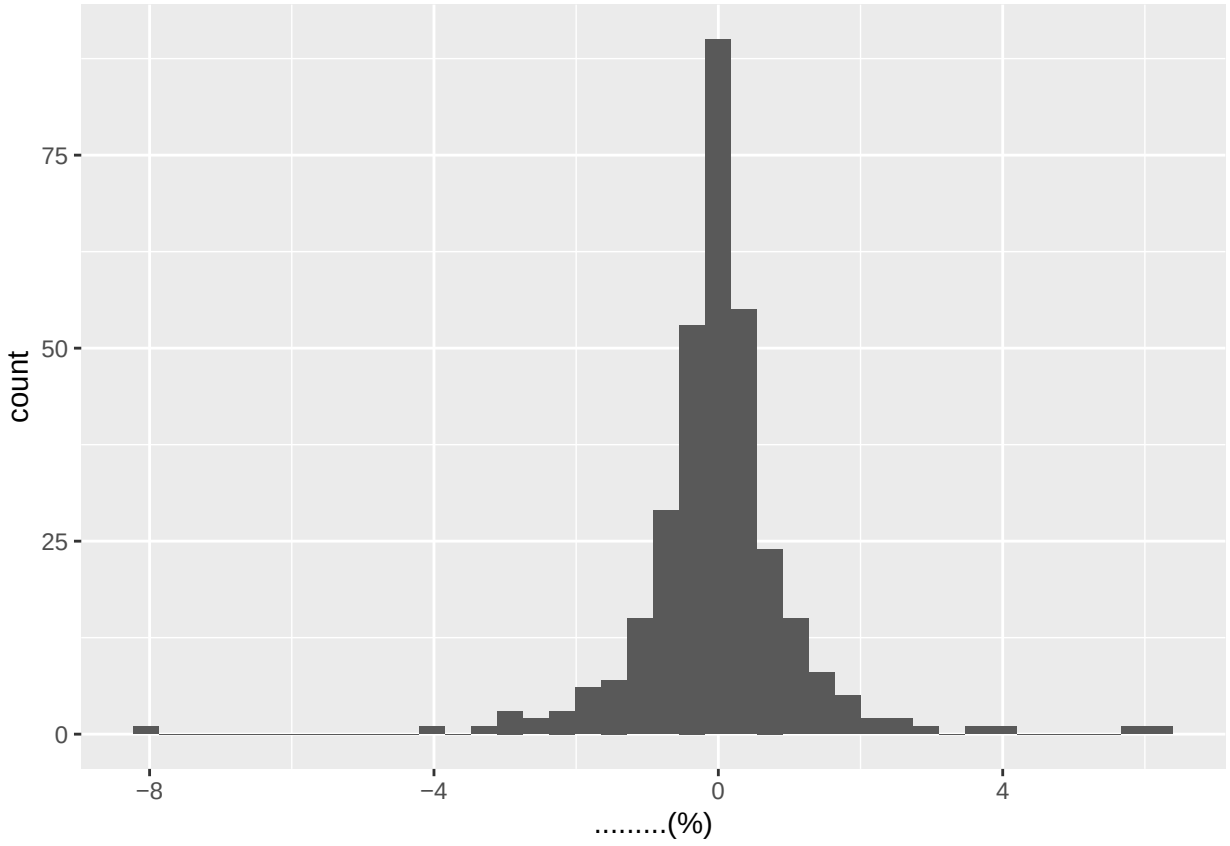


题目 60（数据可视化）：让直方图更细致

难度：★★

代码及运行结果：

```
df %>%
  ggplot(aes(`涨跌幅 (%)`)) +
  geom_histogram(bins = 40)
```



题目 61（数据创建）：用 df 的列名创建数据框

难度：⋆⋆

代码及运行结果：

```
names(df) %>%
  as_tibble()
```

```
## # A tibble: 18 x 1
##   value
##   <chr>
## 1 代码
## 2 简称
## 3 日期
## 4 前收盘价(元)
## 5 开盘价(元)
## 6 最高价(元)
## # ... with 12 more rows
```

题目 62（异常值处理）：输出所有换手率不是数字的行

难度：★★

代码及运行结果：

```
df %>%
  mutate(`换手率 (%)` = parse_number(`换手率 (%)`)) %>%
  filter(is.na(`换手率 (%)`))
```

```
## # A tibble: 18 x 18
##   代码      简称    日期                `前收盘价(元)` `开盘价(元)` `最高价(元)`
##   <chr>     <chr>   <dttm>                       <dbl>        <dbl>        <dbl>
## 1 600000.SH 浦发银~ 2016-02-16 00:00:00           16.3         16.3         16.3
## 2 600000.SH 浦发银~ 2016-02-17 00:00:00           16.3         16.3         16.3
## 3 600000.SH 浦发银~ 2016-02-18 00:00:00           16.3         16.3         16.3
## 4 600000.SH 浦发银~ 2016-02-19 00:00:00           16.3         16.3         16.3
## 5 600000.SH 浦发银~ 2016-02-22 00:00:00           16.3         16.3         16.3
## 6 600000.SH 浦发银~ 2016-02-23 00:00:00           16.3         16.3         16.3
## # ... with 12 more rows, and 12 more variables: `最低价(元)` <dbl>,
## #   `收盘价(元)` <dbl>, `成交量(股)` <chr>, `成交金额(元)` <chr>,
## #   `涨跌(元)` <dbl>, `涨跌幅(%)` <dbl>, `均价(元)` <chr>, `换手率(%)` <dbl>,
## #   `A股流通市值(元)` <dbl>, `总市值(元)` <dbl>, `A股流通股本(股)` <dbl>,
## #   市盈率 <dbl>
```

题目 63（异常值处理）：输出所有换手率为--的行

难度：★★

代码及运行结果：

```
df %>%
  filter(`换手率 (%)` == "--")
```

```
## # A tibble: 18 x 18
##   代码      简称    日期                `前收盘价(元)` `开盘价(元)` `最高价(元)`
##   <chr>     <chr>   <dttm>                       <dbl>        <dbl>        <dbl>
## 1 600000.SH 浦发银~ 2016-02-16 00:00:00           16.3         16.3         16.3
## 2 600000.SH 浦发银~ 2016-02-17 00:00:00           16.3         16.3         16.3
## 3 600000.SH 浦发银~ 2016-02-18 00:00:00           16.3         16.3         16.3
## 4 600000.SH 浦发银~ 2016-02-19 00:00:00           16.3         16.3         16.3
```

```
## 5 600000.SH 浦发银~ 2016-02-22 00:00:00           16.3         16.3         16.3
## 6 600000.SH 浦发银~ 2016-02-23 00:00:00           16.3         16.3         16.3
## # ... with 12 more rows, and 12 more variables: `最低价(元)` <dbl>,
## #   `收盘价(元)` <dbl>, `成交量(股)` <chr>, `成交金额(元)` <chr>,
## #   `涨跌(元)` <dbl>, `涨跌幅(%)` <dbl>, `均价(元)` <chr>, `换手率(%)` <chr>,
## #   `A股流通市值(元)` <dbl>, `总市值(元)` <dbl>, `A股流通股本(股)` <dbl>,
## #   市盈率 <dbl>
```

题目 64（数据操作）：重置 df 的行号

难度：★

代码及运行结果：

```
rownames(df) = NULL     # R 中无行号就是数字索引
```

题目 65（异常值处理）：删除所有换手率为非数字的行

难度：★★

代码及运行结果：

```
df %>%
  mutate(`换手率 (%)` = parse_number(`换手率 (%)`)) %>%
  filter(!is.na(`换手率 (%)`))
```

```
## # A tibble: 309 x 18
##   代码      简称    日期                `前收盘价(元)` `开盘价(元)` `最高价(元)`
##   <chr>     <chr>   <dttm>                       <dbl>        <dbl>        <dbl>
## 1 600000.SH 浦发银~ 2016-01-04 00:00:00           16.1         16.1         16.1
## 2 600000.SH 浦发银~ 2016-01-05 00:00:00           15.7         15.5         16.0
## 3 600000.SH 浦发银~ 2016-01-06 00:00:00           15.9         15.8         16.0
## 4 600000.SH 浦发银~ 2016-01-07 00:00:00           16.0         15.7         15.8
## 5 600000.SH 浦发银~ 2016-01-08 00:00:00           15.5         15.7         15.8
## 6 600000.SH 浦发银~ 2016-01-11 00:00:00           15.4         15.2         15.4
## # ... with 303 more rows, and 12 more variables: `最低价(元)` <dbl>,
## #   `收盘价(元)` <dbl>, `成交量(股)` <chr>, `成交金额(元)` <chr>,
## #   `涨跌(元)` <dbl>, `涨跌幅(%)` <dbl>, `均价(元)` <chr>, `换手率(%)` <dbl>,
## #   `A股流通市值(元)` <dbl>, `总市值(元)` <dbl>, `A股流通股本(股)` <dbl>,
## #   市盈率 <dbl>
```

补充：为了便于后续处理，做数值型转化, 并转化为 tsibble 对象

```
library(tsibble)

df = df %>%
  mutate_at(vars(4:18), as.numeric) %>%
  mutate(日期 = lubridate::as_date(日期)) %>%
  as_tsibble(index = 日期, key = c(代码, 简称))
df
```

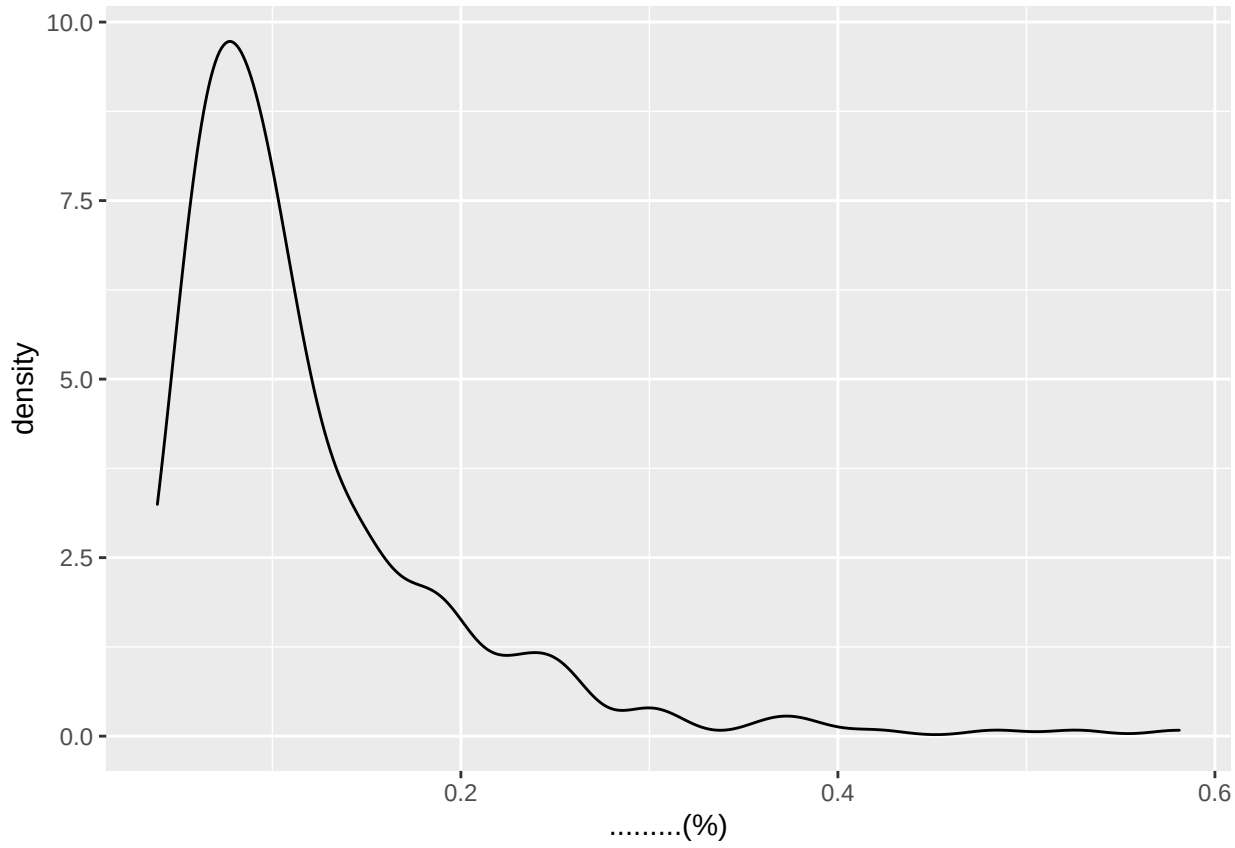
```
## # A tsibble: 327 x 18 [1D]
## # Key:       代码, 简称 [1]
##   代码    简称  日期       `前收盘价(元)` `开盘价(元)` `最高价(元)` `最低价(元)`
##   <chr>   <chr> <date>              <dbl>        <dbl>        <dbl>        <dbl>
## 1 600000~ 浦发~ 2016-01-04           16.1         16.1         16.1         15.5
## 2 600000~ 浦发~ 2016-01-05           15.7         15.5         16.0         15.4
## 3 600000~ 浦发~ 2016-01-06           15.9         15.8         16.0         15.6
## 4 600000~ 浦发~ 2016-01-07           16.0         15.7         15.8         15.4
## 5 600000~ 浦发~ 2016-01-08           15.5         15.7         15.8         14.9
## 6 600000~ 浦发~ 2016-01-11           15.4         15.2         15.4         15.0
## # ... with 321 more rows, and 11 more variables: `收盘价(元)` <dbl>,
## #   `成交量(股)` <dbl>, `成交金额(元)` <dbl>, `涨跌(元)` <dbl>,
## #   `涨跌幅(%)` <dbl>, `均价(元)` <dbl>, `换手率(%)` <dbl>,
## #   `A股流通市值(元)` <dbl>, `总市值(元)` <dbl>, `A股流通股本(股)` <dbl>,
## #   市盈率 <dbl>
```

题目 66（数据可视化）：绘制换手率的密度曲线

难度：★★

代码及运行结果：

```
df %>%
  ggplot(aes(`换手率 (%)`)) +
  geom_density()
```



题目 67（数据计算）：计算前一天与后一天收盘价的差值

难度：★★

代码及运行结果：

```
df %>%
  mutate(delta = `收盘价 (元)` - lag(`收盘价 (元)`)) %>%
  select(日期, `收盘价 (元)`, delta)
```

```
## # A tsibble: 327 x 3 [1D]
##    日期       `收盘价(元)`   delta
##    <date>            <dbl>   <dbl>
## 1 2016-01-04         15.7 NA
## 2 2016-01-05         15.9  0.141
## 3 2016-01-06         16.0  0.124
## 4 2016-01-07         15.5 -0.521
## 5 2016-01-08         15.4 -0.0177
## 6 2016-01-11         15.1 -0.389
## # ... with 321 more rows
```

题目 68（数据计算）：计算前一天与后一天收盘价的变化率

难度：★★

代码及运行结果：

```
df %>%
  mutate(change = (`收盘价 (元)` - lag(`收盘价 (元)`)) / `收盘价 (元)`) %>%
  select(日期, `收盘价 (元)`, change)
```

```
## # A tsibble: 327 x 3 [1D]
##    日期       `收盘价(元)`   change
##    <date>            <dbl>    <dbl>
## 1 2016-01-04         15.7 NA
## 2 2016-01-05         15.9  0.00891
## 3 2016-01-06         16.0  0.00774
## 4 2016-01-07         15.5 -0.0337
## 5 2016-01-08         15.4 -0.00115
## 6 2016-01-11         15.1 -0.0258
## # ... with 321 more rows
```

题目 69（数据操作）：设置日期为行索引

难度：★

代码及运行结果：

```
df %>%
  column_to_rownames(" 日期") %>%
  head()
```

```
##                  代码     简称 前收盘价(元) 开盘价(元) 最高价(元) 最低价(元)
## 2016-01-04 600000.SH 浦发银行      16.1356    16.1444    16.1444    15.4997
## 2016-01-05 600000.SH 浦发银行      15.7205    15.4644    15.9501    15.3672
## 2016-01-06 600000.SH 浦发银行      15.8618    15.8088    16.0208    15.6234
## 2016-01-07 600000.SH 浦发银行      15.9855    15.7205    15.8088    15.3672
## 2016-01-08 600000.SH 浦发银行      15.4644    15.6675    15.7912    14.9345
## 2016-01-11 600000.SH 浦发银行      15.4467    15.1994    15.4114    14.9786
##            收盘价(元) 成交量(股) 成交金额(元) 涨跌(元) 涨跌幅(%) 均价(元)
## 2016-01-04    15.7205   42240610    754425783  -0.4151   -2.5725  17.8602
## 2016-01-05    15.8618   58054793   1034181474   0.1413    0.8989  17.8139
```

```
## 2016-01-06    15.9855   46772653    838667398   0.1236    0.7795  17.9307
## 2016-01-07    15.4644   11350479    199502702  -0.5211   -3.2597  17.5766
## 2016-01-08    15.4467   71918296   1262105060  -0.0177   -0.1142  17.5492
## 2016-01-11    15.0581   90177135   1550155933  -0.3886   -2.5157  17.1901
##            换手率(%) A股流通市值(元)    总市值(元) A股流通股本(股) 市盈率
## 2016-01-04    0.2264    332031791187 332031791187     18653471415 6.5614
## 2016-01-05    0.3112    335016346613 335016346613     18653471415 6.6204
## 2016-01-06    0.2507    337627832612 337627832612     18653471415 6.6720
## 2016-01-07    0.0608    326622284477 326622284477     18653471415 6.4545
## 2016-01-08    0.3855    326249215048 326249215048     18653471415 6.4471
## 2016-01-11    0.4834    318041687626 318041687626     18653471415 6.2849
```

题目 70（数据计算）：对收盘价做步长为 5 的滑动平均

难度：★★★

代码及运行结果：

```
library(slider)

df %>%
  mutate(avg_5 = slide_dbl(`收盘价 (元)`, mean, na.rm = TRUE, .before = 2, .after = 2)) %>%
  select(日期, `收盘价 (元)`, avg_5)
```

```
## # A tsibble: 327 x 3 [1D]
##   日期       `收盘价(元)` avg_5
##   <date>            <dbl> <dbl>
## 1 2016-01-04         15.7  15.9
## 2 2016-01-05         15.9  15.8
## 3 2016-01-06         16.0  15.7
## 4 2016-01-07         15.5  15.6
## 5 2016-01-08         15.4  15.5
## 6 2016-01-11         15.1  15.3
## # ... with 321 more rows
```

题目 71（数据计算）：对收盘价做步长为 5 的滑动求和

难度：★★★

代码及运行结果：

```r
df %>%
  mutate(sum_5 = slide_dbl(`收盘价（元）`, sum, na.rm = TRUE, .before = 2, .after = 2)) %>%
  select(日期, `收盘价（元）`, sum_5)
```

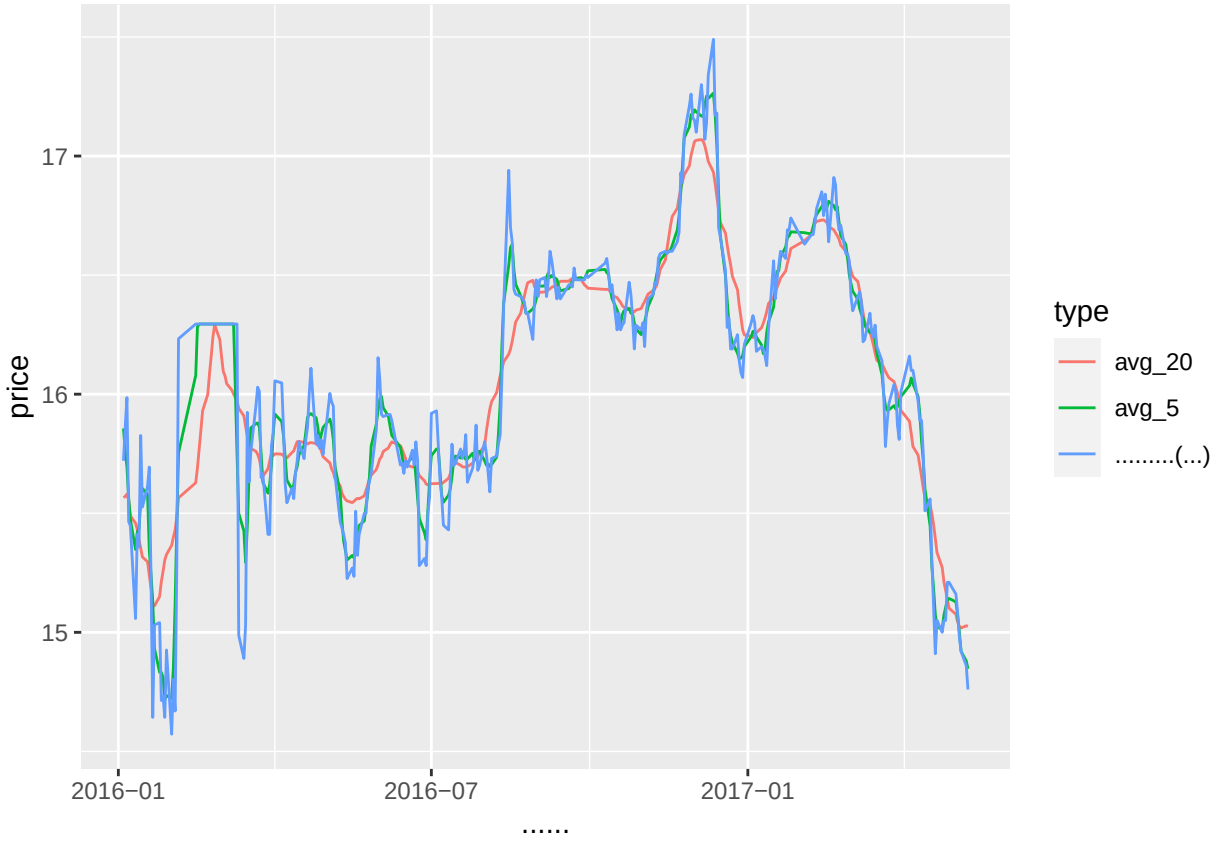
```
## # A tsibble: 327 x 3 [1D]
##    日期       `收盘价(元)` sum_5
##    <date>            <dbl> <dbl>
## 1 2016-01-04          15.7  47.6
## 2 2016-01-05          15.9  63.0
## 3 2016-01-06          16.0  78.5
## 4 2016-01-07          15.5  77.8
## 5 2016-01-08          15.4  77.4
## 6 2016-01-11          15.1  76.7
## # ... with 321 more rows
```

题目 72（数据可视化）：将收盘价及其 5 日均线、20 日均线绘制在同一个图上

难度：★★★★

代码及运行结果：

```r
df %>%
  mutate(avg_5 = slide_dbl(`收盘价（元）`, mean, na.rm = TRUE, .before = 2, .after = 2),
  avg_20 = slide_dbl(`收盘价（元）`, mean, na.rm = TRUE, .before = 10, .after = 9)) %>%
  pivot_longer(c(`收盘价（元）`, avg_5, avg_20),
                 names_to = "type",
                 values_to = "price") %>%
  ggplot(aes(日期, price, color = type)) +
    geom_line()
```



题目 **73**（数据重采样）：按周为采样规则，计算一周收盘价最大值

难度：★★★★

代码及运行结果：

```
weekmax = df %>%
  index_by(weeks = ~ yearweek(.)) %>%    # 周度汇总
  summarise(max_week = max(`收盘价 (元)`, na.rm = TRUE))
weekmax
```

```
## # A tsibble: 69 x 2 [1W]
##      weeks max_week
##     <week>    <dbl>
## 1 2016 W01     16.0
## 2 2016 W02     15.8
## 3 2016 W03     15.7
## 4 2016 W04     15.0
## 5 2016 W05     16.2
```

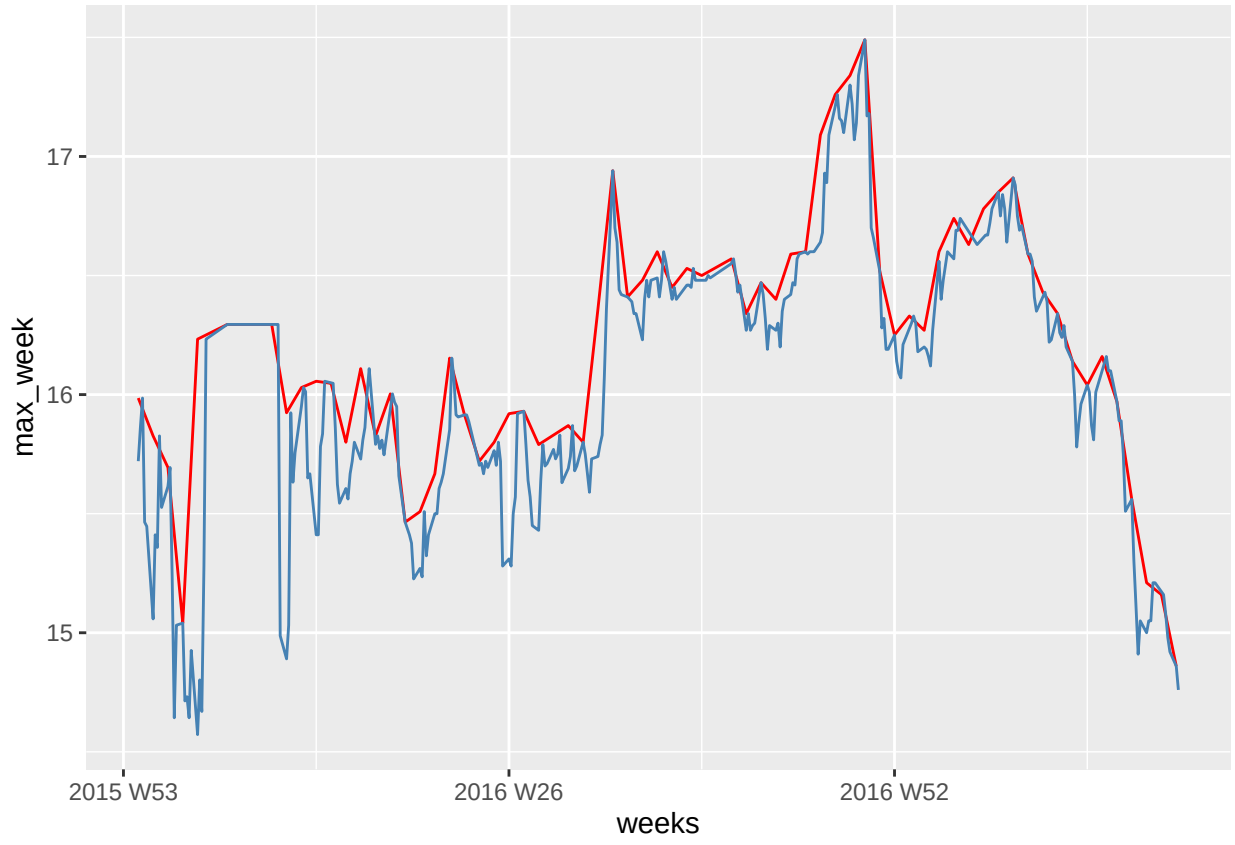
```
## 6 2016 W07     16.3
## # ... with 63 more rows
```

题目 **74**（数据可视化）：绘制重采样数据与原始数据

难度：⋆⋆⋆

代码及运行结果：

```
ggplot() +
  geom_line(data = weekmax, mapping = aes(weeks, max_week), color = "red") +
  geom_line(data = df, aes(日期, `收盘价（元）`), color = "steelblue")
```



题目 **75**（数据操作）：将数据往后移动 **5** 天

难度：⋆⋆⋆

代码及运行结果：

```
df %>%
  mutate(across(4:18, ~ lag(.x, 5)))
```

```
## # A tsibble: 327 x 18 [1D]
## # Key:       代码, 简称 [1]
##   代码     简称   日期       `前收盘价(元)` `开盘价(元)` `最高价(元)` `最低价(元)`
##   <chr>    <chr> <date>              <dbl>        <dbl>        <dbl>        <dbl>
## 1 600000~ 浦发~ 2016-01-04           NA           NA           NA           NA
## 2 600000~ 浦发~ 2016-01-05           NA           NA           NA           NA
## 3 600000~ 浦发~ 2016-01-06           NA           NA           NA           NA
## 4 600000~ 浦发~ 2016-01-07           NA           NA           NA           NA
## 5 600000~ 浦发~ 2016-01-08           NA           NA           NA           NA
## 6 600000~ 浦发~ 2016-01-11           16.1         16.1         16.1         15.5
## # ... with 321 more rows, and 11 more variables: `收盘价(元)` <dbl>,
## #   `成交量(股)` <dbl>, `成交金额(元)` <dbl>, `涨跌(元)` <dbl>,
## #   `涨跌幅(%)` <dbl>, `均价(元)` <dbl>, `换手率(%)` <dbl>,
## #   `A股流通市值(元)` <dbl>, `总市值(元)` <dbl>, `A股流通股本(股)` <dbl>,
## #   市盈率 <dbl>
```

注：这是批量做后移，单个变量做后移用 `mutate(var = lag(var, 5))` 即可。

题目 76（数据操作）：将数据往前移动 5 天

难度：★★★

代码及运行结果：

```
df %>%
  mutate(across(4:18, ~ lead(.x, 5)))
```

```
## # A tsibble: 327 x 18 [1D]
## # Key:       代码, 简称 [1]
##   代码     简称   日期       `前收盘价(元)` `开盘价(元)` `最高价(元)` `最低价(元)`
##   <chr>    <chr> <date>              <dbl>        <dbl>        <dbl>        <dbl>
## 1 600000~ 浦发~ 2016-01-04           15.4         15.2         15.4         15.0
## 2 600000~ 浦发~ 2016-01-05           15.1         15.2         15.5         15.1
## 3 600000~ 浦发~ 2016-01-06           15.4         15.5         15.8         15.3
## 4 600000~ 浦发~ 2016-01-07           15.4         15.0         15.9         14.9
## 5 600000~ 浦发~ 2016-01-08           15.8         15.7         16.0         15.5
## 6 600000~ 浦发~ 2016-01-11           15.5         15.4         15.9         15.3
```

```
## # ... with 321 more rows, and 11 more variables: `收盘价(元)` <dbl>,
## #   `成交量(股)` <dbl>, `成交金额(元)` <dbl>, `涨跌(元)` <dbl>,
## #   `涨跌幅(%)` <dbl>, `均价(元)` <dbl>, `换手率(%)` <dbl>,
## #   `A股流通市值(元)` <dbl>, `总市值(元)` <dbl>, `A股流通股本(股)` <dbl>,
## #   市盈率 <dbl>
```

题目 77（数据操作）：计算开盘价的累积平均

难度：★★★

代码及运行结果：

```
rlt = df %>%
  mutate(累积平均 = cummean(`开盘价（元）`)) %>%
  select(日期, `开盘价（元）`, 累积平均)
rlt
```

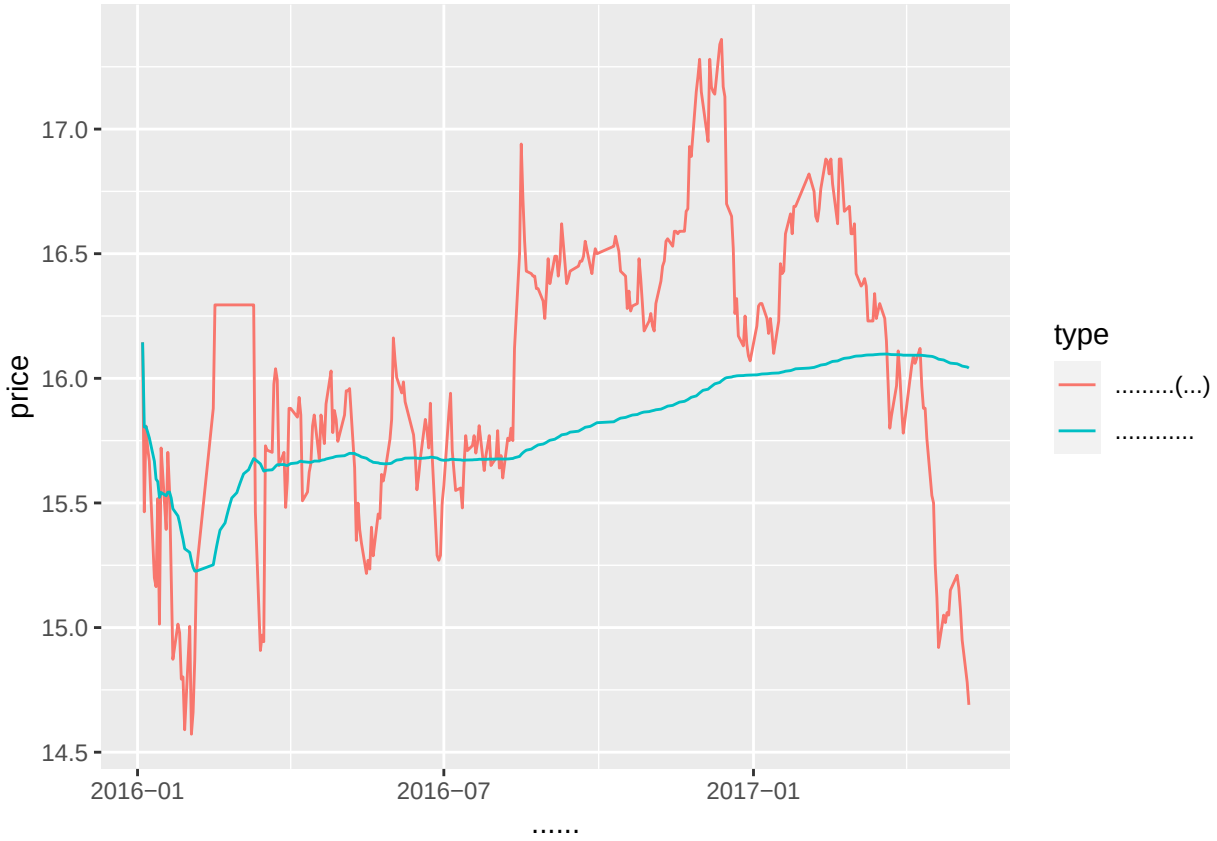
```
## # A tsibble: 327 x 3 [1D]
##   日期       `开盘价(元)` 累积平均
##   <date>            <dbl>    <dbl>
## 1 2016-01-04         16.1     16.1
## 2 2016-01-05         15.5     15.8
## 3 2016-01-06         15.8     15.8
## 4 2016-01-07         15.7     15.8
## 5 2016-01-08         15.7     15.8
## 6 2016-01-11         15.2     15.7
## # ... with 321 more rows
```

题目 78（数据计算）：绘制开盘价的累积平均与原始数据的折线图

难度：★★★

代码及运行结果：

```
rlt %>%
  pivot_longer(-日期, names_to = "type", values_to = "price") %>%
  ggplot(aes(日期, price, color = type)) +
    geom_line()
```



题目 79（数据计算）：计算布林指标

难度：★★★★

代码及运行结果：

```
boll = df %>%
  mutate(avg_20 = slide_dbl(`收盘价 (元)`, mean, na.rm = TRUE, .before = 10, .after = 9),
         sd_20 = slide_dbl(`收盘价 (元)`, sd, na.rm = TRUE, .before = 10, .after = 9),
         up = avg_20 + 2 * sd_20,
         down = avg_20 - 2 * sd_20) %>%
  select(日期, `收盘价 (元)`, avg_20, up, down)

boll %>%
  sample_n(10)
```

```
## # A tsibble: 10 x 5 [1D]
##    日期       `收盘价(元)` avg_20    up  down
##    <date>            <dbl>  <dbl> <dbl> <dbl>
```

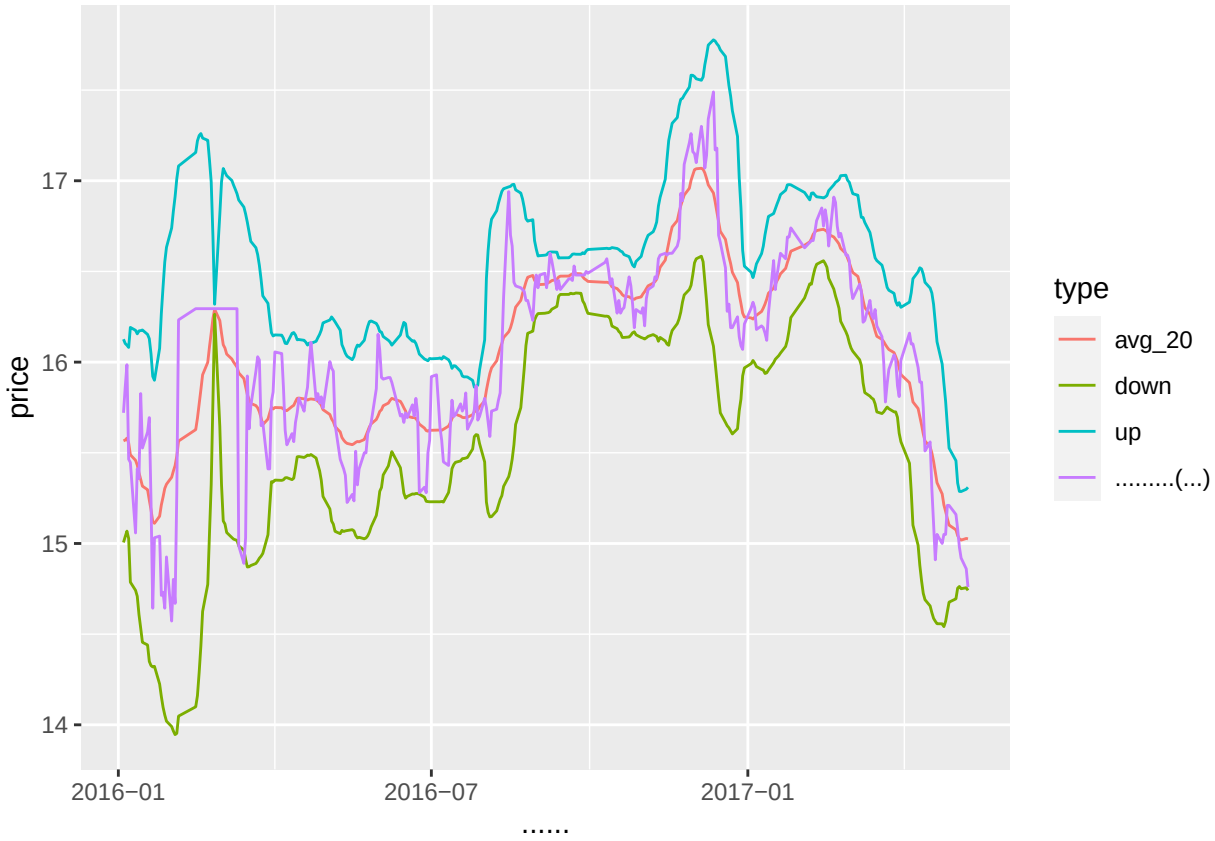
```
## 1 2016-12-08          17.1   17.0  17.7  16.3
## 2 2017-04-05          16.2   15.9  16.3  15.4
## 3 2016-12-02          17.1   17.1  17.6  16.6
## 4 2016-10-27          16.2   16.3  16.5  16.2
## 5 2016-02-18          16.3   15.8  17.3  14.4
## 6 2017-03-07          16.4   16.4  16.9  16.0
## # ... with 4 more rows
```

题目 80（数据可视化）：绘制布林曲线

难度：★★★

代码及运行结果：

```
boll %>%
  pivot_longer(-日期, names_to = "type", values_to = "price") %>%
  ggplot(aes(日期, price, color = type)) +
    geom_line()
```



Part VI 数据生成

题目 81（加载查看包）：加载并查看 tidyverse 包版本

难度：⋆

代码及运行结果：

```
library(tidyverse)
```

题目 82（生成随机数）：生成 20 个 0~100 的随机数，创建数据框

难度：⋆

代码及运行结果：

```
set.seed(123)       # 保证结果出现
df1 = tibble(nums = sample.int(100, 20))
df1
```

```
## # A tibble: 20 x 1
##    nums
##   <int>
## 1    31
## 2    79
## 3    51
## 4    14
## 5    67
## 6    42
## # ... with 14 more rows
```

题目 83（生成等差数）：生成 20 个 0~100 固定步长的数，创建数据框

难度：⋆

代码及运行结果：

```
df2 = tibble(nums = seq(0, 99, by = 5))
df2
```

```
## # A tibble: 20 x 1
##    nums
```

```
##   <dbl>
## 1     0
## 2     5
## 3    10
## 4    15
## 5    20
## 6    25
## # ... with 14 more rows
```

题目 **84**（生成指定分布随机数）：生成 **20** 个标准正态分布的随机数，创建数据框

难度：⋆

代码及运行结果：

```
df3 = tibble(nums = rnorm(20, 0, 1))
df3
```

```
## # A tibble: 20 x 1
##     nums
##    <dbl>
## 1 -1.97
## 2  0.701
## 3 -0.473
## 4 -1.07
## 5 -0.218
## 6 -1.03
## # ... with 14 more rows
```

题目 **85**（合并数据）：将 **df1, df2, df3** 按行合并为新数据框

难度：⋆

代码及运行结果：

```
bind_rows(df1, df2, df3)
```

```
## # A tibble: 60 x 1
##    nums
##   <dbl>
## 1    31
```

```
## 2    79
## 3    51
## 4    14
## 5    67
## 6    42
## # ... with 54 more rows
```

题目 86（合并数据）：将 df1, df2, df3 按列合并为新数据框

难度：⋆

代码及运行结果：

```
df = bind_cols(df1, df2, df3)
df
```

```
## # A tibble: 20 x 3
##   nums...1 nums...2 nums...3
##      <int>    <dbl>    <dbl>
## 1       31        0   -1.97
## 2       79        5    0.701
## 3       51       10   -0.473
## 4       14       15   -1.07
## 5       67       20   -0.218
## 6       42       25   -1.03
## # ... with 14 more rows
```

题目 87（查看数据）：查看 df 所有数据的最小值、25% 分位数、中位数、75% 分位数、最大值

难度：⋆⋆

代码及运行结果：

```
unlist(df) %>%
  summary()
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.9666  0.6982 25.5000 33.2497 61.2500 97.0000
```

题目 88（修改列名）：修改列名为 col1, col2, col3

难度：⋆

代码及运行结果：

```
df = df %>%
  set_names(str_c("col", 1:3))
df
```

```
## # A tibble: 20 x 3
##    col1  col2   col3
##   <int> <dbl>  <dbl>
## 1    31     0 -1.97
## 2    79     5  0.701
## 3    51    10 -0.473
## 4    14    15 -1.07
## 5    67    20 -0.218
## 6    42    25 -1.03
## # ... with 14 more rows
```

注：若只修改个别列名，用 rename(newname=oldname).

题目 89（数据操作）：提取在第 1 列中而不在第 2 列中的数

难度：⋆⋆

代码及运行结果：

```
setdiff(df$col1, df$col2)
```

```
##  [1] 31 79 51 14 67 42 43 97 69 57  9 72 26  7 87 36
```

题目 90（数据操作）：提取在第 1 列和第 2 列出现频率最高的三个数字

难度：⋆⋆⋆

代码及运行结果：

```
df %>%
  pivot_longer(1:2,
               names_to = "col",
```

```
               values_to = "value") %>%
  count(value, sort = TRUE) %>%
  slice_max(n, n = 3)
```

```
## # A tibble: 4 x 2
##   value     n
##   <dbl> <int>
## 1    25     2
## 2    50     2
## 3    90     2
## 4    95     2
```

或者用

```
c(df$col1, df$col2) %>%
  table() %>%
  sort(decreasing = TRUE) %>%
  .[1:3]
```

```
## .
## 25 50 90
##  2  2  2
```

或者用

```
rlt = tibble(nums = c(df$col1, df$col2)) %>%
  sjmisc::frq(nums, sort.frq = "desc")

rlt[[1]][1:3,]
```

```
##   val  label frq raw.prc valid.prc cum.prc
## 1  25 <none>   2       5         5       5
## 2  50 <none>   2       5         5      10
## 3  90 <none>   2       5         5      15
```

题目 91（数据操作）：提取第 1 列可以整除 5 的数的位置

难度：⋆⋆

代码及运行结果：

```r
# which(df$col1 %% 5 == 0)
# 或者
df %>%
  mutate(id = row_number()) %>%
  filter(col1 %% 5 == 0) %>%
  pull(id)
```

```
## [1]  7 10 11 18
```

- 选取满足条件的索引，通常用途还是用来选出满足条件的行，不兜圈子做法：

```r
df %>%
  filter(col1 %% 5 == 0)
```

```
## # A tibble: 4 x 3
##    col1  col2   col3
##   <int> <dbl>  <dbl>
## 1    50    30 -0.729
## 2    25    45  0.838
## 3    90    50  0.153
## 4    95    85  0.822
```

题目 92（数据计算）：计算第 1 列的 1 阶差分

难度：⋆⋆

代码及运行结果：

```r
# diff(df$col1)
# 或者
df %>%
  mutate(diff1 = col1 - lag(col1))
```

```
## # A tibble: 20 x 4
##    col1  col2   col3 diff1
##   <int> <dbl>  <dbl> <int>
## 1    31     0 -1.97     NA
## 2    79     5  0.701    48
## 3    51    10 -0.473   -28
## 4    14    15 -1.07    -37
```

```
## 5    67    20 -0.218    53
## 6    42    25 -1.03    -25
## # ... with 14 more rows
```

题目 93（数据操作）：将 col1, col2, col3 三列顺序颠倒

难度：⋆⋆

代码及运行结果：

```
df %>%
  select(rev(names(df)))
```

```
## # A tibble: 20 x 3
##     col3  col2  col1
##    <dbl> <dbl> <int>
## 1 -1.97      0    31
## 2  0.701     5    79
## 3 -0.473    10    51
## 4 -1.07     15    14
## 5 -0.218    20    67
## 6 -1.03     25    42
## # ... with 14 more rows
```

注：更灵活的调整列序，dplyr1.0 将提供 `relocate()` 函数。

题目 94（数据操作）：提取第一列位置在 1,10,15 的数

难度：⋆

代码及运行结果：

```
df[c(1,10,15),1]
```

```
## # A tibble: 3 x 1
##    col1
##   <int>
## 1    31
## 2    25
## 3    72
```

题目 95（数据操作）：查找第一列的局部最大值位置

难度：★★★★

代码及运行结果：

```
df %>%
  mutate(diff = sign(col1 - lag(col1)) + sign(col1 - lead(col1)),
         id = row_number()) %>%
  filter(diff == 2) %>%
  pull(id)
```

```
## [1]  2  5  7  9 11 15 18
```

- 不兜圈子做法：

```
df %>%
  mutate(diff = sign(col1 - lag(col1)) + sign(col1 - lead(col1))) %>%
  filter(diff == 2)
```

```
## # A tibble: 7 x 4
##    col1  col2   col3  diff
##   <int> <dbl>  <dbl> <dbl>
## 1    79     5  0.701     2
## 2    67    20 -0.218     2
## 3    50    30 -0.729     2
## 4    97    40 -1.69      2
## 5    90    50  0.153     2
## 6    72    70 -0.295     2
## # ... with 1 more row
```

题目 96（数据计算）：按行计算 df 每一行的均值

难度：★★

代码及运行结果：

```
# rowMeans(df)
# 或者
# apply(df, 1, mean)
```

```r
# 或者
df %>%
  mutate(row_avg = (col1 + col2 + col3) / 3)
```

```
## # A tibble: 20 x 4
##    col1  col2   col3 row_avg
##   <int> <dbl>  <dbl>   <dbl>
## 1    31     0 -1.97     9.68
## 2    79     5  0.701   28.2
## 3    51    10 -0.473   20.2
## 4    14    15 -1.07     9.31
## 5    67    20 -0.218   28.9
## 6    42    25 -1.03    22.0
## # ... with 14 more rows
```

```r
# 或者
# df %>%
#  rowwise() %>%
#  mutate(row_avg = mean(c_across()))

# 或者
# df %>%
#  mutate(row_avg = pmap_dbl(., ~ mean(c(...))))
```

题目 97（数据计算）：对第二列计算步长为 3 的移动平均值

* 难度：**★★★

代码及运行结果：

```r
df %>%
  mutate(avg_3 = slide_dbl(col2, mean, .before = 1, .after = 1))
```

```
## # A tibble: 20 x 4
##    col1  col2   col3 avg_3
##   <int> <dbl>  <dbl> <dbl>
## 1    31     0 -1.97    2.5
## 2    79     5  0.701   5
## 3    51    10 -0.473  10
## 4    14    15 -1.07   15
```

```
## 5    67    20 -0.218  20
## 6    42    25 -1.03   25
## # ... with 14 more rows
```

题目 **98**（数据计算）：按第三列值的大小升序排列

难度：★★

代码及运行结果：

```
df %>%
  arrange(col3)
```

```
## # A tibble: 20 x 3
##    col1  col2   col3
##   <int> <dbl>  <dbl>
## 1    31     0 -1.97
## 2    97    40 -1.69
## 3    69    55 -1.14
## 4    14    15 -1.07
## 5    42    25 -1.03
## 6    50    30 -0.729
## # ... with 14 more rows
```

题目 **99**（数据操作）：按第一列大于 **50** 的数修改为” 高”

难度：★★

代码及运行结果：

```
# df[df$col1 > 50, 1] = " 高"
# 或者
df %>%
  mutate(col1 = sjmisc::rec(col1, rec = "50:max= 高; else=copy"))
```

```
## # A tibble: 20 x 3
##   col1   col2   col3
##   <chr> <dbl>  <dbl>
## 1 31        0 -1.97
## 2 高        5  0.701
## 3 高       10 -0.473
```

```
## 4 14       15 -1.07
## 5 高       20 -0.218
## 6 42       25 -1.03
## # ... with 14 more rows
```

注：这里采用更有实用价值的重新编码。

题目 100（数据计算）：计算第一列与第二列的欧氏距离

难度：⋆⋆⋆

代码及运行结果：

```
dist(t(df[,1:2]))
```

```
##         col1
## col2 176.054
```

Part V 高级

题目 101（数据读取）：从 csv 文件中读取指定数据：读取前 10 行，positionName 和 salary 列

难度：⋆⋆

代码及运行结果：

```
read.csv("datas/数据 1_101-120 涉及.csv", nrows = 10) %>%
  select(positionName, salary)
```

```
##      positionName salary
## 1        数据分析  37500
## 2        数据建模  15000
## 3        数据分析   3500
## 4        数据分析  45000
## 5        数据分析  30000
## 6        数据分析  50000
## 7        数据分析  30000
## 8  数据建模工程师  35000
## 9    数据分析专家  60000
## 10     数据分析师  40000
```

注 1：该数据是 GBK 编码，为避免中文乱码，GBK 编码的 csv 或 txt 用 `read.csv()` 读取；UTF-8 编码的 csv 或 txt 用 `readr::read_csv()` 读取；若用 `read_csv()` 读取 GBK 编码文件，需要设置编码（见题目 101）。

注 2：R 中常规读取数据不能在读取时选择列，采用读取之后选择列。

题目 102（数据读取）：从 csv 文件中读取数据，将薪资大于 10000 的改为" 高"

难度：★★

代码及运行结果：

```
df = read_csv("datas/数据 2_101-120 涉及.csv") %>%
  mutate(薪资水平 = if_else(薪资水平 > 10000, " 高", " 低"))
```

题目 103（数据操作）：从 df 中对薪资水平每隔 20 行进行抽样

难度：★★

代码及运行结果：

```
# df[seq(1, nrow(df), 20),]
# 或者
df %>%
  slice(seq(1, n(), by = 20))
```

```
## # A tibble: 58 x 2
##   学历要求 薪资水平
##   <chr>    <chr>
## 1 本科     高
## 2 本科     高
## 3 本科     高
## 4 本科     高
## 5 本科     高
## 6 本科     高
## # ... with 52 more rows
```

题目 104（数据操作）：取消使用科学记数法

难度：★★

代码及运行结果：

```r
set.seed(123)
df = tibble(val = runif(10) ^ 10)
# 三位小数
df %>%
  mutate(val = scales::number(val, accuracy = 0.001))
```

```
## # A tibble: 10 x 1
##    val
##    <chr>
## 1 0.000
## 2 0.093
## 3 0.000
## 4 0.288
## 5 0.541
## 6 0.000
## # ... with 4 more rows
```

```r
# 科学记数法
df %>%
  mutate(val =  scales::scientific(val, 2))
```

```
## # A tibble: 10 x 1
##    val
##    <chr>
## 1 3.9e-06
## 2 9.3e-02
## 3 1.3e-04
## 4 2.9e-01
## 5 5.4e-01
## 6 3.9e-14
## # ... with 4 more rows
```

题目 105（数据操作）：将上一题的数据转换为百分数

难度：⋆⋆⋆

代码及运行结果：

```r
df %>%
  mutate(val = scales::percent(val, 0.01))
```

```
## # A tibble: 10 x 1
##   val
##   <chr>
## 1 0.00%
## 2 9.27%
## 3 0.01%
## 4 28.82%
## 5 54.13%
## 6 0.00%
## # ... with 4 more rows
```

题目 106（数据操作）：查找上一题数据中第 3 大值的行号

难度：★★★

代码及运行结果：

```r
# order(df$val, decreasing = TRUE)[3]
# 或者
df %>%
  mutate(id = row_number()) %>%
  arrange(-val) %>%
  slice(3) %>%
  pull(id)
```

```
## [1] 4
```

- 不兜圈子做法：

```r
df %>%
  arrange(-val) %>%
  slice(3)
```

```
## # A tibble: 1 x 1
##     val
##   <dbl>
## 1 0.288
```

题目 **107**（数据操作）：反转 **df** 的行

难度：★★

代码及运行结果：

```
df %>%
  slice(rev(1:n()))    # 或者用 n():1
```

```
## # A tibble: 10 x 1
##        val
##      <dbl>
## 1 3.94e- 4
## 2 2.60e- 3
## 3 3.20e- 1
## 4 1.69e- 3
## 5 3.85e-14
## 6 5.41e- 1
## # ... with 4 more rows
```

题目 **108**（数据连接：全连接）：根据多列匹配合并数据，保留 **df1** 和 **df2** 的观测

难度：★★

代码及运行结果：

```
df1 = tibble(
  key1 = c("K0","K0","K1","K2"),
  key2 = c("K0","K1","K0","K1"),
  A = str_c('A', 0:3),
  B = str_c('B', 0:3)
)

df2 = tibble(
  key1 = c("K0","K1","K1","K2"),
  key2 = str_c("K", rep(0,4)),
  C = str_c('C', 0:3),
  D = str_c('D', 0:3)
)

df1
```

```
## # A tibble: 4 x 4
##   key1  key2  A     B
##   <chr> <chr> <chr> <chr>
## 1 K0    K0    A0    B0
## 2 K0    K1    A1    B1
## 3 K1    K0    A2    B2
## 4 K2    K1    A3    B3
```

```
df2
```

```
## # A tibble: 4 x 4
##   key1  key2  C     D
##   <chr> <chr> <chr> <chr>
## 1 K0    K0    C0    D0
## 2 K1    K0    C1    D1
## 3 K1    K0    C2    D2
## 4 K2    K0    C3    D3
```

```
df1 %>%
  full_join(df2, by = c("key1", "key2"))
```

```
## # A tibble: 6 x 6
##   key1  key2  A     B     C     D
##   <chr> <chr> <chr> <chr> <chr> <chr>
## 1 K0    K0    A0    B0    C0    D0
## 2 K0    K1    A1    B1    <NA>  <NA>
## 3 K1    K0    A2    B2    C1    D1
## 4 K1    K0    A2    B2    C2    D2
## 5 K2    K1    A3    B3    <NA>  <NA>
## 6 K2    K0    <NA>  <NA>  C3    D3
```

题目 109（数据连接：左连接）：根据多列匹配合并数据，只保留 **df1** 的观测

难度：⋆⋆

代码及运行结果：

```
df1 %>%
  left_join(df2, by = c("key1", "key2"))
```

```
## # A tibble: 5 x 6
##   key1  key2  A     B     C     D
##   <chr> <chr> <chr> <chr> <chr> <chr>
## 1 K0    K0    A0    B0    C0    D0
## 2 K0    K1    A1    B1    <NA>  <NA>
## 3 K1    K0    A2    B2    C1    D1
## 4 K1    K0    A2    B2    C2    D2
## 5 K2    K1    A3    B3    <NA>  <NA>
```

注：dplyr 包还提供了右连接：right_join()，内连接：inner_join()，以及用于过滤的连接：半连接：semi_join()，反连接：anti_join().

题目 110（数据处理）：再次读取数据 1 并显示所有列

难度：⋆⋆

代码及运行结果：

```
df = read_csv("datas/数据 1_101-120 涉及.csv") %>%
glimpse()
```

```
## Rows: 105
## Columns: 53
## $ positionId           <dbl> 6802721, 5204912, 6877668, 6496141, 6467417, 688~
## $ positionName         <chr> "数据分析", "数据建模", "数据分析", "数据分析", ~
## $ companyId            <dbl> 475770, 50735, 100125, 26564, 29211, 94826, 3487~
## $ companyLogo          <chr> "i/image2/M01/B7/3E/CgoB5lwPfEaAdn8WAABWQ0Jgl5s3~
## $ companySize          <chr> "50-150人", "150-500人", "2000人以上", "500-2000~
## $ industryField        <chr> "移动互联网,电商", "电商", "移动互联网,企业服务"~
## $ financeStage         <chr> "A轮", "B轮", "上市公司", "D轮及以上", "上市公司~
## $ companyLabelList     <chr> "['绩效奖金', '带薪年假', '定期体检', '弹性工作'~
## $ firstType            <chr> "产品|需求|项目类", "开发|测试|运维类", "产品|需~
## $ secondType           <chr> "数据分析", "数据开发", "数据分析", "数据开发", ~
## $ thirdType            <chr> "数据分析", "建模", "数据分析", "数据分析", "数~
## $ skillLables          <chr> "['SQL', '数据库', '数据运营', 'BI']", "['算法',~
## $ positionLables       <chr> "['电商', '社交', 'SQL', '数据库', '数据运营', '~
## $ industryLables       <chr> "['电商', '社交', 'SQL', '数据库', '数据运营', '~
## $ createTime           <chr> "2020/3/16 11:00", "2020/3/16 11:08", "2020/3/16~
## $ formatCreateTime     <chr> "11:00发布", "11:08发布", "10:33发布", "10:10发~
## $ district             <chr> "余杭区", "滨江区", "江干区", "江干区", "余杭区"~
```

```
## $ businessZones         <chr> "['仓前']", "['西兴', '长河']", "['四季青', '钱~
## $ salary                <dbl> 37500, 15000, 3500, 45000, 30000, 50000, 30000, ~
## $ workYear              <chr> "1-3年", "3-5年", "1-3年", "3-5年", "3-5年", "1-~
## $ jobNature             <chr> "全职", "全职", "全职", "全职", "全职", "全职", ~
## $ education             <chr> "本科", "本科", "本科", "本科", "大专", "本科", ~
## $ positionAdvantage     <chr> "五险一金、弹性工作、带薪年假、年度体检", "六险~
## $ imState               <chr> "today", "disabled", "today", "threeDays", "disa~
## $ lastLogin             <chr> "2020/3/16 11:00", "2020/3/16 11:08", "2020/3/16~
## $ publisherId           <dbl> 12022406, 5491688, 5322583, 9814560, 6392394, 11~
## $ approve               <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, ~
## $ subwayline            <chr> NA, NA, "4号线", "1号线", NA, NA, NA, "2号线", N~
## $ stationname           <chr> NA, NA, "江锦路", "文泽路", NA, NA, NA, "丰潭路"~
## $ linestaion            <chr> NA, NA, "4号线_城星路;4号线_市民中心;4号线_江锦~
## $ latitude              <dbl> 30.27842, 30.18804, 30.24152, 30.29940, 30.28295~
## $ longitude             <dbl> 120.0059, 120.2012, 120.2125, 120.3503, 120.0098~
## $ hitags                <chr> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ resumeProcessRate     <dbl> 50, 23, 11, 100, 20, 16, 100, 1, 83, 1, 83, 0, 1~
## $ resumeProcessDay      <dbl> 1, 1, 4, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, ~
## $ score                 <dbl> 233, 176, 80, 68, 66, 66, 65, 47, 24, 18, 17, 17~
## $ newScore              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ matchScore            <dbl> 15.1018750, 32.5594140, 14.9723570, 12.8741530, ~
## $ matchScoreExplain     <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ query                 <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ explain               <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ isSchoolJob           <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, ~
## $ adWord                <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ plus                  <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ pcShow                <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ appShow               <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ deliver               <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ gradeDescription      <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ promotionScoreExplain <lgl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, ~
## $ isHotHire             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ count                 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ~
## $ aggregatePositionIds  <chr> "[]", "[]", "[]", "[]", "[]", "[]", "[]", "[]", ~
## $ famousCompany         <lgl> FALSE, FALSE, FALSE, TRUE, TRUE, FALSE, FALSE, T~
```

题目 111（数据操作）：查找 secondType 与 thirdType 值相等的行号

难度：⋆⋆

代码及运行结果：

```
# which(df$secondType == df$thirdType)
# 或者
df %>%
  mutate(id = row_number()) %>%
  filter(secondType == thirdType) %>%
  pull(id)
```

```
##  [1]   1   3   5   6   7  11  15  24  26  28  29  30  31  34  38  39  40  41  42
## [20]  49  50  53  54  56  58  62  66  67  68  72  74  75  76  80  81  83  86  89
## [39]  90  92  97 101
```

- 不兜圈子：

```
df %>%
  filter(secondType == thirdType)
```

```
## # A tibble: 42 x 53
##   positionId positionName companyId companyLogo        companySize industryField
##        <dbl> <chr>            <dbl> <chr>              <chr>       <chr>
## 1    6802721 数据分析        475770 i/image2/M01/B7/3~ 50-150人    移动互联网,~
## 2    6877668 数据分析        100125 image2/M00/0C/57/~ 2000人以上  移动互联网,~
## 3    6467417 数据分析         29211 i/image2/M01/77/B~ 2000人以上  物流丨运输
## 4    6882347 数据分析         94826 image2/M00/04/12/~ 50-150人    移动互联网,~
## 5    6841659 数据分析        348784 i/image2/M01/AC/0~ 50-150人    移动互联网,~
## 6    6804629 数据分析师       34132 i/image2/M01/F8/D~ 150-500人   数据服务,广~
## # ... with 36 more rows, and 47 more variables: financeStage <chr>,
## #   companyLabelList <chr>, firstType <chr>, secondType <chr>, thirdType <chr>,
## #   skillLables <chr>, positionLables <chr>, industryLables <chr>,
## #   createTime <chr>, formatCreateTime <chr>, district <chr>,
## #   businessZones <chr>, salary <dbl>, workYear <chr>, jobNature <chr>,
## #   education <chr>, positionAdvantage <chr>, imState <chr>, lastLogin <chr>,
## #   publisherId <dbl>, approve <dbl>, subwayline <chr>, stationname <chr>, ...
```

题目 **112**（数据操作）：查找薪资大于平均薪资的第三个数据

难度：⋆⋆⋆

代码及运行结果：

```
df %>%
  filter(salary > mean(salary)) %>%
  slice(3)
```

```
## # A tibble: 1 x 53
##   positionId positionName companyId companyLogo        companySize industryField
##        <dbl> <chr>            <dbl> <chr>              <chr>       <chr>
## 1    6882347 数据分析         94826 image2/M00/04/12/~ 50-150人    移动互联网,~
## # ... with 47 more variables: financeStage <chr>, companyLabelList <chr>,
## #   firstType <chr>, secondType <chr>, thirdType <chr>, skillLables <chr>,
## #   positionLables <chr>, industryLables <chr>, createTime <chr>,
## #   formatCreateTime <chr>, district <chr>, businessZones <chr>, salary <dbl>,
## #   workYear <chr>, jobNature <chr>, education <chr>, positionAdvantage <chr>,
## #   imState <chr>, lastLogin <chr>, publisherId <dbl>, approve <dbl>,
## #   subwayline <chr>, stationname <chr>, linestaion <chr>, latitude <dbl>, ...
```

题目 113（数据操作）：将上一题数据的 salary 列开根号

难度：★★

代码及运行结果：

```
df %>%
  mutate(salary_sqrt = sqrt(salary)) %>%
  select(salary, salary_sqrt)
```

```
## # A tibble: 105 x 2
##   salary salary_sqrt
##    <dbl>       <dbl>
## 1  37500       194.
## 2  15000       122.
## 3   3500        59.2
## 4  45000       212.
## 5  30000       173.
## 6  50000       224.
## # ... with 99 more rows
```

题目 114（数据操作）：将上一题数据的 linestation 列按 __ 拆分

难度：★★★

代码及运行结果：

```
df %>%
  separate(linestaion, into = c("line", "station"), sep = "_", remove = FALSE) %>%
  select(linestaion, line, station)
```

```
## # A tibble: 105 x 3
##   linestaion                                  line  station
##   <chr>                                       <chr> <chr>
## 1 <NA>                                        <NA>  <NA>
## 2 <NA>                                        <NA>  <NA>
## 3 4号线_城星路;4号线_市民中心;4号线_江锦路 4号线 城星路;4号线
## 4 1号线_文泽路                                1号线 文泽路
## 5 <NA>                                        <NA>  <NA>
## 6 <NA>                                        <NA>  <NA>
## # ... with 99 more rows
```

注：正常需要先按 “;” 分割，再分别按 “-” 分割。

题目 115（数据查看）：查看上一题数据一共有多少列

难度：⋆

代码及运行结果：

```
ncol(df)
```

```
## [1] 53
```

题目 116（数据操作）：提取 industryField 列以” 数据” 开头的行

难度：⋆⋆

代码及运行结果：

```
df %>%
  filter(str_detect(industryField, "^ 数据"))
```

```
## # A tibble: 15 x 53
##   positionId positionName   companyId companyLogo      companySize industryField
##        <dbl> <chr>              <dbl> <chr>            <chr>       <chr>
```

```
## 1    6458372 数据分析专家       34132 i/image2/M01/F8~ 150-500人   数据服务,广~
## 2    6804629 数据分析师         34132 i/image2/M01/F8~ 150-500人   数据服务,广~
## 3    6804489 资深数据分析师     34132 i/image2/M01/F8~ 150-500人   数据服务,广~
## 4    6267370 数据分析专家       31544 image1/M00/00/4~ 150-500人   数据服务
## 5    6804489 资深数据分析师     34132 i/image2/M01/F8~ 150-500人   数据服务,广~
## 6    6242470 数据分析师         31544 image1/M00/00/4~ 150-500人   数据服务
## # ... with 9 more rows, and 47 more variables: financeStage <chr>,
## #   companyLabelList <chr>, firstType <chr>, secondType <chr>, thirdType <chr>,
## #   skillLables <chr>, positionLables <chr>, industryLables <chr>,
## #   createTime <chr>, formatCreateTime <chr>, district <chr>,
## #   businessZones <chr>, salary <dbl>, workYear <chr>, jobNature <chr>,
## #   education <chr>, positionAdvantage <chr>, imState <chr>, lastLogin <chr>,
## #   publisherId <dbl>, approve <dbl>, subwayline <chr>, stationname <chr>, ...
```

题目 117（数据分组汇总）：以 salary score 和 positionID 做数据透视表

难度：⋆⋆⋆

代码及运行结果：

```
df %>%
  group_by(positionId) %>%
  summarise(salary_avg = mean(salary),
            score_avg = mean(score))
```

```
## # A tibble: 95 x 3
##   positionId salary_avg score_avg
##        <dbl>      <dbl>     <dbl>
## 1    5203054      30000         4
## 2    5204912      15000       176
## 3    5269002      37500         1
## 4    5453691      30000         4
## 5    5519962      37500        14
## 6    5520623      30000         6
## # ... with 89 more rows
```

题目 118（数据分组汇总）：同时对 salary、score 两列进行汇总计算

难度：⋆⋆⋆

代码及运行结果：

```r
df %>%
  summarise(across(c(salary, score),
                   list(sum=sum, mean=mean, min=min),
                   .names = "{.col}_{.fn}"))
```

```
## # A tibble: 1 x 6
##   salary_sum salary_mean salary_min score_sum score_mean score_min
##        <dbl>       <dbl>      <dbl>     <dbl>      <dbl>     <dbl>
## 1    3331000      31724.       3500      1335       12.7         0
```

注：若要分组再这样汇总，前面加上 group_by(var) 即可。

题目 119（数据分组汇总）：同时对不同列进行不同的汇总计算：对 salary 求平均，对 score 求和

难度：⋆⋆⋆

代码及运行结果：

```r
df %>%
  summarise(salary_avg = mean(salary),
            score_sum = sum(score))
```

```
## # A tibble: 1 x 2
##   salary_avg score_sum
##        <dbl>     <dbl>
## 1     31724.      1335
```

注：若要分组再这样汇总，前面加上 group_by(var) 即可。

题目 120（数据分组汇总）：计算并提取平均薪资最高的区

难度：⋆⋆⋆⋆

代码及运行结果：

```r
df %>%
  group_by(district) %>%
  summarise(salary_avg = mean(salary)) %>%
  slice_max(salary_avg)
```

```
## # A tibble: 1 x 2
##   district salary_avg
##   <chr>         <dbl>
## 1 萧山区        36250
```