最新R语言版本：玩转数据处理120题 附多重解法，要点摘要……

陈志明

2024-10-11

    - 30.1 方法1:直接用median()函数计算。
    - 30.2 方法2:使用summarise()函数实现。
- 31 绘制薪资水平频率分布直方图
- 32 绘制薪资水平密度曲线
- 33 删除最后一列categories
    - 33.1 方法1：选择列位置，列数最大的一列。前加负号，及表示不选择
    - 33.2 方法2：选择列名
    - 33.3 方法3：选择特征，比如包含，以什么开头等等…
- 34 将df的第一列与第二列合并为新的一列
- 35 将education列与salary列合并为新的一列
- 36 计算salary最大值与最小值之差
    - 36.1 方法1：使用base R 函数处理
    - 36.2 方法2：使用dplyr::select()函数处理
    - 36.3 方法3：使用dplyr::slice()函数处理
    - 36.4 方法4：使用dplyr::slice()函数直接处理
- 37 将第8行数据添加至末尾
    - 37.1 方法1：通过选择行的方法调整行顺序，使得第8行排在最后一行。
    - 37.2 方法2：数据表与第八行数据拼接，或者说第8行数据拼接至数据表尾行，然后删除第8行。
- 38 查看每列的数据类型
- 39 将createTime列设置为索引
- 40 生成一个和df长度相同的随机数dataframe
- 41 将上一题生成的dataframe与df合并
    - 41.1 方法1：cbind()函数
    - 41.2 方法2：bind_cols()函数
- 42 生成新的一列new为salary列减去之前生成随机数列
- 43 检查数据中是否含有任何缺失值
- 44 将salary列类型转换为浮点数
- 45 计算salary大于10000的次数
- 46 查看每种学历出现的次数
- 47 查看education列共有几种学历
- 48 提取salary与new列的和大于60000的最后3行
- 49 使用绝对路径读取本地Excel数据
    - 49.1 方法1：使用xlsx包函数
    - 49.2 方法2：使用readxl包包函数
- 50 查看数据前三行

- 51 查看每列数据缺失值情况
    - 51.1 对于data数据，检测不到NA值.
        - 51.1.1 方法1:colSums()函数
        - 51.1.2 方法2：用across()函数处理
    - 51.2 对于data_2 数据，tibble 类型的数据，可以检测到NA值。
        - 51.2.1 方法1:colSums()函数
        - 51.2.2 方法2：用across()函数处理
- 52 提取日期列含有空值的行
- 53 输出每列缺失值具体行数
- 54 删除所有存在缺失值的行
    - 54.1 方法1：drop.na()函数
    - 54.2 方法2：na.omit()函数
- 55 绘制收盘价的折线图
- 56 同时绘制开盘价与收盘价
    - 56.1 方法1：使用宽数据
    - 56.2 方法2：使用长数据
- 57 绘制涨跌幅的直方图
- 58 让直方图更细致
- 59 以data的列名创建一个dataframe
- 60 打印所有换手率不是数字的行
- 61 打印所有换手率为–的行
- 62 重置data的行号
- 63 删除所有换手率为非数字的行
- 64 绘制换手率的密度曲线
- 65 计算前一天与后一天收盘价的差值
- 66 计算前一天与后一天收盘价变化率
- 67 设置日期为索引
- 68 以5个数据作为一个数据滑动窗口，在这个5个数据上取均值(收盘价)
    - 68.1 方法1：用基础包和tidyverse 相关函数做函数自己计算
    - 68.2 方法2：调用相关的基础函数计算：zoo::rollmean()函数
    - 68.3 方法3：调用相关的基础函数计算：DescTools::MoveAvg()函数
    - 68.4 方法4：调用相关的基础函数计算：slider::slide_dbl()函数
- 69 以5个数据作为一个数据滑动窗口，计算这五个数据总和(收盘价)
    - 69.1 方法1：用基础包和tidyverse 相关函数做函数自己计算
    - 69.2 方法2：调用相关的基础函数计算：zoo::rollsum函数

    - 69.3 方法3：调用相关的基础函数计算：slider::slide_dbl()函数
- 70 将收盘价5日均线、20日均线与原始数据绘制在同一个图上
- 71 按周为采样规则，取一周收盘价最大值
    - 71.1 方法1：年、周两级分组求最大值；
    - 71.2 方法2：年-组合字段,group 对象作为参数选项放在函数里。
    - 71.3 方法3：年周日期字段，group对象作为参数选项放在函数里。
    - 71.4 方法4：年周日期字段,先group，再执行slice()函数。
    - 71.5 方法5：执行summarise，用max函数。
- 72 绘制重采样数据与原始数据
- 73 将数据往后移动5天
- 74 将数据向前移动5天
- 75 使用expending函数计算开盘价的移动窗口均值
- 76 绘制上一题的移动均值与原始数据折线图
    - 76.1 方法1：这种做法不可取，类似使用宽数据绘图的形式。
    - 76.2 方法2：宽数据变长数据后再作图，合理的方法
- 77 计算布林指标
    - 77.1 方法1：自行计算
    - 77.2 方法2：调包和函数计算:slider::slide_dbl()函数
    - 77.3 方法3：调包和函数计算:TTR::runMean()函数
    - 77.4 方法4：调包和函数计算:tidyquant:: tq_mutate()函数
- 78 计算布林线并绘制
- 79 查看包的版本
- 80 从数组创建DataFrame
- 81 生成20个0-100固定步长的数
- 82 将df1，df2，df3按照行合并为新DataFrame
    - 82.1 方法1:base R 包函数 rbind()
    - 82.2 方法2:bind_rows()函数
- 83 将df1，df2，df3按照列合并为新DataFrame
    - 83.1 方法1:base R 包函数 cbind()
    - 83.2 方法2:bind_cols()函数
- 84 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
- 85 修改列名为col1,col2,col3
    - 85.1 方法1：set_names()函数
    - 85.2 方法2：names()函数
    - 85.3 方法3：tibble()函数name_repair 参数

- 86 提取第一列中不在第二列出现的数字
  - 86.1 方法1 生硬的理解字面意思，第一列中不在第二列出现的数字
  - 86.2 方法2：setdiff()函数就是直接一个函数
  - 86.3 方法3：如果是数据框这种类型，还可以用anti_join()函数来实现
- 87 提取第一列和第二列出现频率最高的三个数字
- 88 提取第一列中可以整除5的数字位置
  - 88.1 方法1： which()大法
  - 88.2 方法2： filter()函数
- 89 计算第一列数字前一个与后一个的差值
- 90 将col1,col2,col3三列顺序颠倒
  - 90.1 方法1：select()函数
- 91 方法2：relocate()函数
- 92 提取第一列位置在1,10,15的数字
  - 92.1 方法1：select() + slice()
  - 92.2 方法2：rowid_to_column()+filter()+ select()
  - 92.3 方法3：filter()+row_number()+ select()
- 93 找第一列的局部最大值位置,即比它前一个与后一个数字的都大的数字
- 94 按行计算df的每一行均值
- 95 对第二列计算移动平均值,假设步长为4
- 96 将数据按照第三列值的大小升序排列
- 97 将第一列大于50的数字修改为’高’
- 98 计算第一列与第二列之间的欧式距离
- 99 从CSV文件中读取指定数据:从数据1中的前10行中读取positionName, salary两列
- 100 方法1 readr::read_csv()函数
- 101 方法2 base R read.csv()函数
- 102 从数据2中读取数据并在读取数据时将薪资大于10000的改为高
- 103 从上一题数据中，对薪资水平列每隔20行进行一次抽样
  - 103.1 方法1：slice() +n()
  - 103.2 方法2：rowid_to_column() + filter ()
  - 103.3 方法2：row_number() + filter ()
  - 103.4 方法3：order() + filter ()
- 104 将数据取消使用科学计数法
- 105 将上一题的数据转换为百分数
- 106 查找上一题数据中第3大值的行号
  - 106.1 方法1：rowid_to_column()+arrange()+slice

    - 106.2 方法2：order()+rank()
- 107 反转df的行
    - 107.1 方法1: 对行号逆排序
    - 107.2 方法2: 由后往前slice()
    - 107.3 方法3: 由后往前取子集
- 108 按照多列对数据进行合并
- 109 按照多列对数据进行合并,只保存df1的数据
- 110 再次读取数据1并显示所有的列
- 111 查找secondType与thirdType值相等的行号
    - 111.1 方法1：rowid_to_column() + filter() +select()
    - 111.2 方法2：# which(),查找
- 112 查找薪资大于平均薪资的第三个数据
- 113 将上一题数据的salary列开根号
- 114 将上一题数据的linestaion列按_拆分
- 115 查看上一题数据中一共有多少列
- 116 提取industryField列以'数据'开头的行
    - 116.1 方法1：典型的文本提取类操作，适用正则表达式
    - 116.2 扩展，本例只有1列，如果要提取多行中，以“数据”开头的行:
        - 116.2.1 再加上positionName；提取这两列里同时以“数据”开头的行；
        - 116.2.2 提取”industryField”,“positionName”这两列里只要一列以“数据”开头的行
        - 116.2.3 提取整个数据表只要一列以“数据”开头的行；
- 117 以salary score 和 positionID制作数据透视
    - 117.1 方法1：group()+summarise()
    - 117.2 方法2 ： summarise() + 参数中进行分组
- 118 同时对salary、score两列进行计算
- 119 对不同列执行不同的计算：对salary求平均，对score列求和
- 120 计算并提取平均薪资最高的区

可以自信的讲是全网最新最全的R语言版本内容了。

补充说明：

1.绝大部分函数来自tidyverse系列包；

2.对部分题目，认为有必要多种方法计算时，提供了不同的方法，其中包括自编函数或调用已有包函数；

3.个别题目进行了简要的扩展；

4.总体看来，这套题目不难，适合数据分析处理入门练手。

导入必要的包

```
library(tidyverse)
```

1 创建dataframe

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

1.1 方法1

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

```
##   grammer score
## 1  Python     1
## 2       C     2
## 3    Java    NA
## 4      GO     4
## 5    <NA>     5
## 6     SQL     6
## 7     PHP     7
## 8  Python    10
```

1.2 方法2

```
df_2 <- tibble(grammer = c("Python","C","Java","GO",NA,"SQL","PHP","Python"),
                 score = c(1,2,NA,4,5,6,7,10))
df_2
```

```
##   grammer score
##   <chr>   <dbl>
## 1 Python      1
## 2 C           2
## 3 Java       NA
## 4 GO          4
## 5 <NA>        5
## 6 SQL         6
## 7 PHP         7
## 8 Python     10
```

2 提取含有字符串”Python”的行

字面意思理解，这里的 含有Python ,应该不是简单的等于，而是字符里包含。比如换成Java，那么Javascript 的行也应该被计算在内。

2.1 方法1：理解为 等于python 的做法

```
# 理解为等于的做法
df_2 %>%
    filter(grammer =="Python")
```

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

2.2 方法2：理解为 包含python 的做法

```
df_2 %>%
    filter(str_detect(grammer,"Python"))
```

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

3 输出df的列名

```
names(df_2)
```

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

或者

```
colnames(df_2)
```

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

4 修改第二列列名为'popularity'

```
df_2 <- df_2 %>%
    rename(popularity = score)
```

5 统计grammer列中每种编程语言出现的次数

```
df_2 %>%
    count(grammer)
```

```
## # A tibble: 7 × 2
##   grammer     n
##   <chr>   <int>
## 1 C           1
## 2 GO          1
## 3 Java        1
## 4 PHP         1
## 5 Python      2
## 6 SQL         1
## 7 <NA>        1
```

6 将空值用上下值的平均值填充

```
row_na <- which(is.na(df_2$popularity))

value4na <-  mean(c(df_2$popularity[row_na+1],
                    df_2$popularity[row_na-1]))

df_2$popularity <- replace_na(df_2$popularity,
                              value4na)
df_2
```

```
## # A tibble: 8 × 2
##   grammer popularity
##   <chr>        <dbl>
## 1 Python           1
## 2 C                2
## 3 Java             3
## 4 GO               4
## 5 <NA>             5
## 6 SQL              6
## 7 PHP              7
```

7 提取popularity列中值大于3的行

```
df_2 %>%
filter(popularity > 3)
```

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

8 按照grammer列进行去除重复值

```
df_2 %>%
    dplyr::distinct(grammer)
```

```
## # A tibble: 7 × 1
##   grammer
##   <chr>
## 1 Python
## 2 C
## 3 Java
## 4 GO
## 5 <NA>
## 6 SQL
## 7 PHP
```

9 计算popularity列平均值

9.1 使用summarise函数

```
df_2 %>%
    summarise(mean = mean(popularity))
```

```
## # A tibble: 1 × 1
##    mean
##   <dbl>
## 1  4.75
```

9.2 直接使用mean()函数

```
mean(df_2$popularity)
```

```
## [1] 4.75
```

10 将grammer列转换为list

```
grammer_list = as.list(df_2$grammer)

grammer_list
```

```
## [[1]]
## [1] "Python"
##
## [[2]]
## [1] "C"
##
## [[3]]
## [1] "Java"
##
## [[4]]
```

11 将DataFrame保存为EXCEL

```
xlsx::write.xlsx(df_2,
                 file = "df_2.xlsx",
                 sheetName = "df_2" )
```

12 查看数据行列数

```
dim(df_2)
```

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

13 提取popularity列值大于3小于7的行

```
df_2 %>%
    filter(popularity > 3 & popularity < 7)
```

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

14 交换两列位置

14.1 方法1:使用dplyr::select()函数

```
df_2 %>%
    dplyr::select(2,1)
```

```
## # A tibble: 8 × 2
##   popularity grammer
##        <dbl> <chr>
## 1          1 Python
## 2          2 C
## 3          3 Java
## 4          4 GO
## 5          5 <NA>
## 6          6 SQL
## 7          7 PHP
```

14.2 方法2:使用dplyr::relocate()函数

```
df_2 %>%
    dplyr::relocate(2)
```

```
## # A tibble: 8 × 2
##   popularity grammer
##        <dbl> <chr>
## 1          1 Python
## 2          2 C
## 3          3 Java
## 4          4 GO
## 5          5 <NA>
## 6          6 SQL
## 7          7 PHP
```

15 提取popularity列最大值所在行

15.1 方法1:使用dplyr::filter()函数

```
 df_2 %>%
    dplyr::filter(popularity == max(popularity))
```

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

15.2 方法2:使用基础包 which.max()函数

```
 df[which.max(df_2$popularity),]
```

```
##   grammer score
## 8  Python    10
```

16 查看最后5行数据

```
tail(df_2,5)
```

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

17 删除最后一行数据

17.1 方法1：使用nrow()定位数据框行数值

```
df_2 %>%
    slice(1:(nrow(.)-1))
```

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

17.2 方法2：使用n()定位数据框行数值

```
df_2 %>%
    slice(1:n()-1)
```

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

18 添加一行数据

['Perl',6.6] grammer popularity

18.1 方法1：使用base R rbind()函数

```
df_2 %>% rbind(tibble(grammer = "Perl",
                      popularity = 6.6))
```

```
## # A tibble: 9 × 2
##   grammer popularity
##   <chr>        <dbl>
## 1 Python         1
## 2 C              2
## 3 Java           3
## 4 GO             4
## 5 <NA>           5
## 6 SQL            6
## 7 PHP            7
```

18.2 方法2：使用dplyr::bind_rows()函数

```
df_2 %>% dplyr::bind_rows(tibble(grammer = "Perl",
                                 popularity = 6.6))
```

```
## # A tibble: 9 × 2
##   grammer popularity
##   <chr>        <dbl>
## 1 Python         1
## 2 C              2
## 3 Java           3
## 4 GO             4
## 5 <NA>           5
## 6 SQL            6
## 7 PHP            7
```

18.3 方法3：使用tibble::add_row()函数

该函数可设置加入行的位置。比如，这里我们将加入的Perl 放在第一行

```
df_17<- df_2 %>%
    add_row(grammer = "Perl",
            popularity = 6.6,.before = 1)
df_17
```

```
## # A tibble: 9 × 2
##   grammer popularity
##   <chr>        <dbl>
## 1 Perl           6.6
## 2 Python         1
## 3 C              2
## 4 Java           3
## 5 GO             4
## 6 <NA>           5
## 7 SQL            6
```

19 对数据按照”popularity”列值的大小进行排序

19.1 方法1：arrange()函数,该函数默认从小到大排序，只需在数据列名加个负号，即可实现由小到大的排序

```
df_17 %>%
    arrange(-popularity)# 由大到小
```

```
## # A tibble: 9 × 2
##   grammer popularity
##   <chr>        <dbl>
## 1 Python        10
## 2 PHP            7
## 3 Perl           6.6
## 4 SQL            6
## 5 <NA>           5
## 6 GO             4
## 7 Java           3
```

19.2 方法2：arrange()函数,该函数默认从小到大排序，也可以使用desc()函数

```
df_17 %>%
    arrange(desc(popularity))# 由大到小
```

```
## # A tibble: 9 × 2
##   grammer popularity
##   <chr>        <dbl>
## 1 Python        10
## 2 PHP            7
## 3 Perl           6.6
## 4 SQL            6
## 5 <NA>           5
## 6 GO             4
## 7 Java           3
```

19.3 方法3：base R order()函数

```
df_17[order(-df_17$popularity),]# 由大到小
```

```
## # A tibble: 9 × 2
##   grammer popularity
##   <chr>        <dbl>
## 1 Python        10
## 2 PHP            7
## 3 Perl           6.6
## 4 SQL            6
## 5 <NA>           5
## 6 GO             4
## 7 Java           3
```

20 统计grammer列每个字符串的长度

```
df_20 <- df_17 %>%
mutate(charlen = str_length(grammer))

df_20
```

```
## # A tibble: 9 × 3
##   grammer popularity charlen
##   <chr>        <dbl>   <int>
## 1 Perl           6.6       4
## 2 Python         1         6
## 3 C              2         1
## 4 Java           3         4
## 5 GO             4         2
## 6 <NA>           5        NA
## 7 SQL            6         3
```

21 读取本地EXCEL数据

```
df <- xlsx::read.xlsx(file = "./data/pandas120.xlsx",
                      sheetName = "Sheet1")
```

21.1 查看df数据前5行

```
head(df,5)
```

```
##            createTime education  salary
## 1 2020-03-16 11:30:18      本科 20k-35k
## 2 2020-03-16 10:58:47      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
```

22 将salary列数据转换为最大值与最小值的平均值

这题的解法挺多，下面几个例子用到了tidyverse系列几个常用包的高频使用函数。

22.1 方法1:正则表达式提取数字，并数字化，然后求均值

```
df_23_1 <- df %>%
    rowwise() %>%
    mutate(salary =  str_extract_all(salary,
                                     "\\d+") %>%
    unlist() %>%
    as.numeric() %>%
    mean()*1000 %>%
        unlist()) %>%
    ungroup()
```

```
## # A tibble: 135 × 3
##    createTime          education salary
##    <dttm>              <chr>      <dbl>
##  1 2020-03-16 11:30:18 本科       27500
##  2 2020-03-16 10:58:47 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 10:33:47 本科       14000
##  7 2020-03-16 10:11:54 硕士       23000
```

##方法2:分解列，然后再解析数字求均值;

使用到 separate()函数，parse_number()函数以及rowwise()函数。

```
df_23_2 <-  df %>%
     tidyr::separate(salary,into = c("low",
                                     "high"),
                     sep = "-") %>%
    mutate(salary =1000* ((parse_number(low )+
                           parse_number( high)))/2)
df_23_2
```

```
##              createTime education low high salary
## 1   2020-03-16 11:30:18      本科 20k  35k  27500
## 2   2020-03-16 10:58:47      本科 20k  40k  30000
## 3   2020-03-16 10:46:39      不限 20k  35k  27500
## 4   2020-03-16 10:45:44      本科 13k  20k  16500
## 5   2020-03-16 10:20:41      本科 10k  20k  15000
## 6   2020-03-16 10:33:47      本科 10k  18k  14000
## 7   2020-03-16 10:11:54      硕士 16k  30k  23000
## 8   2020-03-16 09:49:11      本科 10k  15k  12500
## 9   2020-03-16 09:25:47      不限  6k   8k   7000
```

##方法3: 方法2的基础上，使用mean()函数，前面加个rowwise()函数

```
df_23_3 <-  df %>%
     tidyr::separate(salary,
                     into = c("low","high"),
                     sep = "-") %>%
       rowwise() %>%
    mutate(salary = 1000*mean(c(parse_number(low),
                           parse_number(high)
                           )
                         )
           ) %>%
```

```
## # A tibble: 135 × 5
##    createTime          education low   high  salary
##    <dttm>              <chr>     <chr> <chr>  <dbl>
##  1 2020-03-16 11:30:18 本科      20k   35k    27500
##  2 2020-03-16 10:58:47 本科      20k   40k    30000
##  3 2020-03-16 10:46:39 不限      20k   35k    27500
##  4 2020-03-16 10:45:44 本科      13k   20k    16500
##  5 2020-03-16 10:20:41 本科      10k   20k    15000
##  6 2020-03-16 10:33:47 本科      10k   18k    14000
##  7 2020-03-16 10:11:54 硕士      16k   30k    23000
```

22.2 方法4: 用map函数进行循环处理

```
df_23_4 <- df %>%
   mutate(salary = salary %>%
             map(\(x) str_extract_all(x,"\\d+") %>%
   unlist() %>%
   as.numeric() %>%
   mean()*1000) %>%
       unlist())
df_23_4
```

```
##              createTime education salary
## 1   2020-03-16 11:30:18      本科  27500
## 2   2020-03-16 10:58:47      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 10:33:47      本科  14000
## 7   2020-03-16 10:11:54      硕士  23000
## 8   2020-03-16 09:49:11      本科  12500
## 9   2020-03-16 09:25:47      不限   7000
```

23 将数据根据学历进行分组并计算平均薪资

本题数据基于上题的df_23_3。

23.1 方法1：先group_by(),再summarise()

这一方法不推荐，因为在summarise()函数中，有分组的选项，也就是方法2的做法。

```
df_23_3 %>%
   group_by(education) %>%
   summarise(average_salary = mean(salary))
```

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

23.2 方法2: 直接summarise,在该函数的参数设置中确定分组。

```
df_23_3 %>%
    summarise(average_salary = mean(salary),
              .by = education)
```

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

24 将createTime列时间转换为月-日

这题涉及到本人用得不多的时间日期型数据的处理包和函数，大家用得比较多的是lubridate包。

其实这是一道很简单的题目，就是数据类型的转换，这里为了练习数据处理常用的方法，进行拓展。

24.1 方法1：参考在excel 里常用的提取月份和日期数据，然后组合在一起的思维逻辑。

```
df_23_3 %>%
    mutate(createTime = str_c(
    month(df_23_3$createTime,label = FALSE),
    day(df_23_3$createTime),
    sep ="-")) %>%
    head(2)# 查看前两行效果
```

```
## # A tibble: 2 × 5
##   createTime education low   high  salary
##   <chr>      <chr>     <chr> <chr>  <dbl>
## 1 3-16       本科      20k   35k    27500
## 2 3-16       本科      20k   40k    30000
```

24.2 方法2：直接从数据里硬提取月和日信息，本例用正则表达式实现。

实现思路是抓取第一个“-”号后面的“dd-dd”格式内容即可。

```
df_23_2 %>%
     mutate(createTime =  createTime %>%
    str_extract_all("(?<=-)\\d{2}-\\d{2}") %>%
        unlist()
    ) %>%
    tail(3)# 查看后3行效果
```

```
##     createTime education low high salary
## 133      03-16      本科 20k  40k  30000
## 134      03-16      本科 15k  23k  19000
## 135      03-16      本科 20k  40k  30000
```

24.3 方法3：使用strftime()函数

这应该是最佳方法

```
df_23_2 %>%
     mutate(createTime =  createTime %>%
    strftime("%m-%d")) %>%
    tail(3) # 查看后3行效果
```

```
##     createTime education low high salary
## 133      03-16      本科 20k  40k  30000
## 134      03-16      本科 15k  23k  19000
## 135      03-16      本科 20k  40k  30000
```

25 查看索引、数据类型和内存信息

```
str(df)
```

```
## 'data.frame':    135 obs. of  3 variables:
##  $ createTime: POSIXct, format: "2020-03-16 11:30:18" "2020-03-16 10:58:47" ...
##  $ education : chr  "本科" "本科" "不限" "本科" ...
##  $ salary    : chr  "20k-35k" "20k-40k" "20k-35k" "13k-20k" ...
```

```
object.size(df)
```

```
## 7160 bytes
```

26 查看数值型列的汇总统计

```
df_23_3 %>%
    select(where(is.numeric))%>%
    summary()
```

```
##      salary
##  Min.   : 3500
##  1st Qu.:14000
##  Median :17500
##  Mean   :19159
##  3rd Qu.:25000
##  Max.   :45000
```

27 新增一列根据salary将数据分为三组

R语言中，常用于分组的函数有：

- case_when()
- if_else():tidyverse
- ifelse():base R
- cut
- within
- case_match 有些情形下，该函数也是很好用的，不过貌似不适合本例。

显然，本题有多种不同的做法。我常用的有下面这些，看分组数量，总体来讲，case-when() 用的比较多。这里只分三组，比较简单。就都练习一下。

27.1 方法1:ifelse() 函数实现。

ifelse()和if_else()存在一定的区别。

```
df_23_3 %>%
mutate(salary_level =if_else(salary>=25000 , "高",
                             if_else(salary<14000 ,"低","中"))) %>%
    head(3)
```

```
## # A tibble: 3 × 6
##   createTime          education low   high  salary salary_level
##   <dttm>              <chr>     <chr> <chr>  <dbl> <chr>
## 1 2020-03-16 11:30:18 本科      20k   35k    27500 高
## 2 2020-03-16 10:58:47 本科      20k   40k    30000 高
## 3 2020-03-16 10:46:39 不限      20k   35k    27500 高
```

27.2 方法2:cut() 函数实现。

```
df_23_3 %>%
mutate(salary_level = cut(salary,
                          breaks = c(3500,14000,25000,Inf),
                          labels = c("低","中","高"),
                          right = FALSE)) %>%
    head(3)
```

```
## # A tibble: 3 × 6
##   createTime          education low   high  salary salary_level
##   <dttm>              <chr>     <chr> <chr>  <dbl> <fct>
## 1 2020-03-16 11:30:18 本科      20k   35k    27500 高
## 2 2020-03-16 10:58:47 本科      20k   40k    30000 高
## 3 2020-03-16 10:46:39 不限      20k   35k    27500 高
```

27.3 方法3：within() 函数实现。

```
df_23_3 %>%
    within({salary_level <- NA
        salary_level[salary>=25000] = "高"
        salary_level[salary<14000] = "低"
        salary_level[salary<25000 & salary>=14000] = "中"
    }) %>%
    tail(3)
```

```
## # A tibble: 3 × 6
##   createTime          education low   high  salary salary_level
##   <dttm>              <chr>     <chr> <chr>  <dbl> <chr>
## 1 2020-03-16 10:48:32 本科      20k   40k    30000 高
## 2 2020-03-16 10:46:30 本科      15k   23k    19000 中
## 3 2020-03-16 11:19:38 本科      20k   40k    30000 高
```

27.4 方法4:case_when() 函数实现。

```
df_28 <- df_23_3 %>%
mutate(salary_level = case_when(salary>=25000 ~"高",
                                salary< 14000 ~"低",
                                .default =  "中"))
df_28
```

```
## # A tibble: 135 × 6
##    createTime          education low   high  salary salary_level
##    <dttm>              <chr>     <chr> <chr>  <dbl> <chr>
##  1 2020-03-16 11:30:18 本科      20k   35k    27500 高
##  2 2020-03-16 10:58:47 本科      20k   40k    30000 高
##  3 2020-03-16 10:46:39 不限      20k   35k    27500 高
##  4 2020-03-16 10:45:44 本科      13k   20k    16500 中
##  5 2020-03-16 10:20:41 本科      10k   20k    15000 中
##  6 2020-03-16 10:33:47 本科      10k   18k    14000 中
##  7 2020-03-16 10:11:54 硕士      16k   30k    23000 中
```

28 按照salary列对数据降序排列

前面已有关于排序的练习，本题不再展开。

```
df_28 %>%
    arrange(-salary) %>%
    head(5)
```

```
## # A tibble: 5 × 6
##   createTime          education low   high  salary salary_level
##   <dttm>              <chr>     <chr> <chr>  <dbl> <chr>
## 1 2020-03-16 11:30:17 本科      30k   60k    45000 高
## 2 2020-03-16 11:03:59 本科      30k   50k    40000 高
## 3 2020-03-16 10:36:57 本科      25k   50k    37500 高
## 4 2020-03-16 11:01:39 本科      30k   45k    37500 高
## 5 2020-03-16 09:54:47 硕士      25k   50k    37500 高
```

29 取出第33行数据

这属于取子集的范畴。

我们可以使用 [, [[, $ 等函数符号直接取行列子集。

当然，也可以使用tidyverse系列包中的select(),filter(),pick(),slice()等函数获取数据表行列子集。

本题比较简单。

29.1 方法1:slice()函数

```
df_28 %>%
    slice(33)
```

```
## # A tibble: 1 × 6
##   createTime          education low   high  salary salary_level
##   <dttm>              <chr>     <chr> <chr>  <dbl> <chr>
## 1 2020-03-16 10:07:25 硕士      15k   30k    22500 中
```

29.2 方法2:直接用 [提取行。

```
df_28 [33,]
```

```
## # A tibble: 1 × 6
##   createTime          education low   high  salary salary_level
##   <dttm>              <chr>     <chr> <chr>  <dbl> <chr>
## 1 2020-03-16 10:07:25 硕士      15k   30k    22500 中
```

30 计算salary列的中位数

30.1 方法1:直接用median()函数计算。

```
df_28 $salary %>%
    median()
```

```
## [1] 17500
```

30.2 方法2:使用summarise()函数实现。

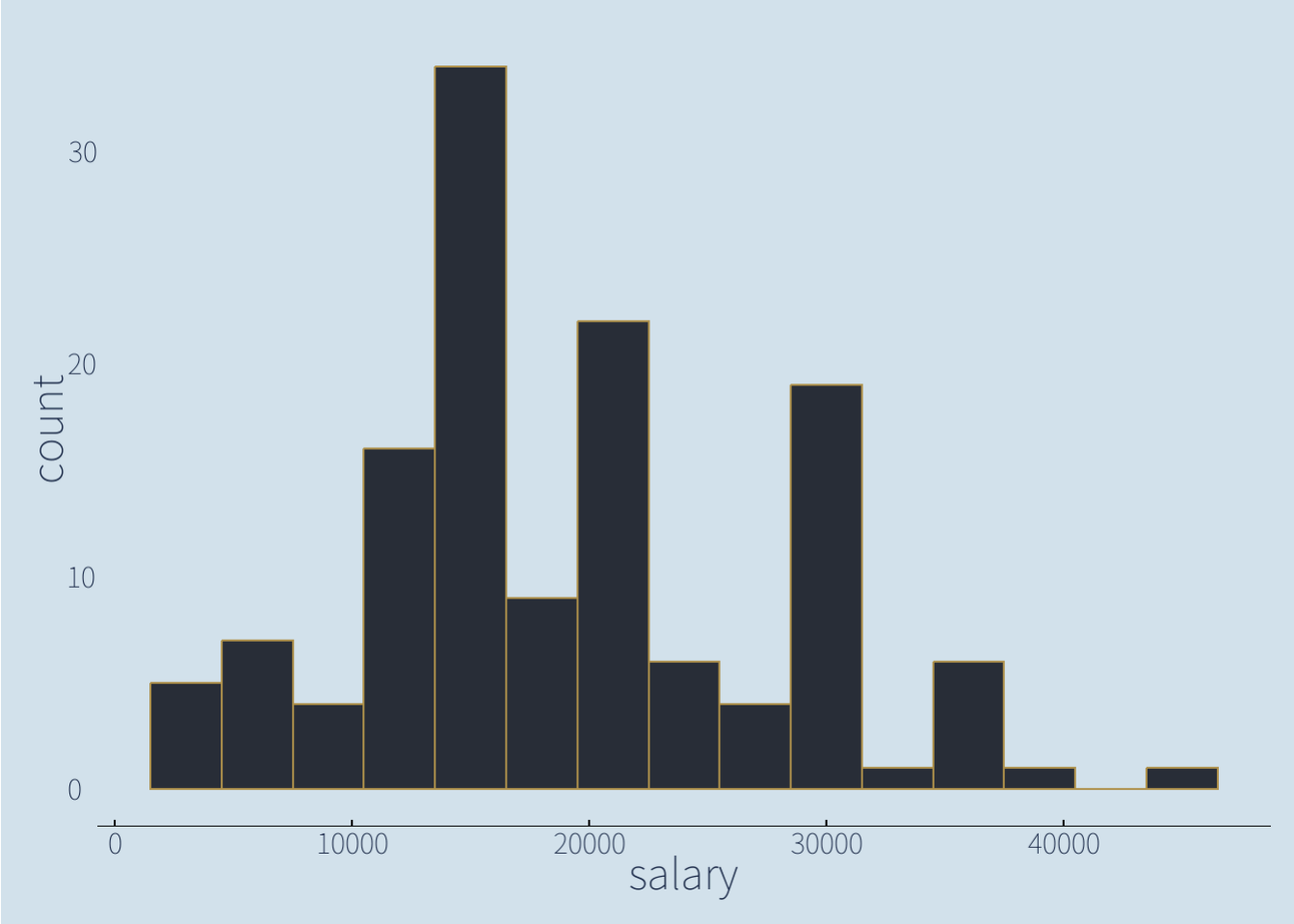
```
df_28 %>%
    summarise(salary_median =median(salary))
```

```
## # A tibble: 1 × 1
##   salary_median
##           <dbl>
## 1         17500
```

31 绘制薪资水平频率分布直方图

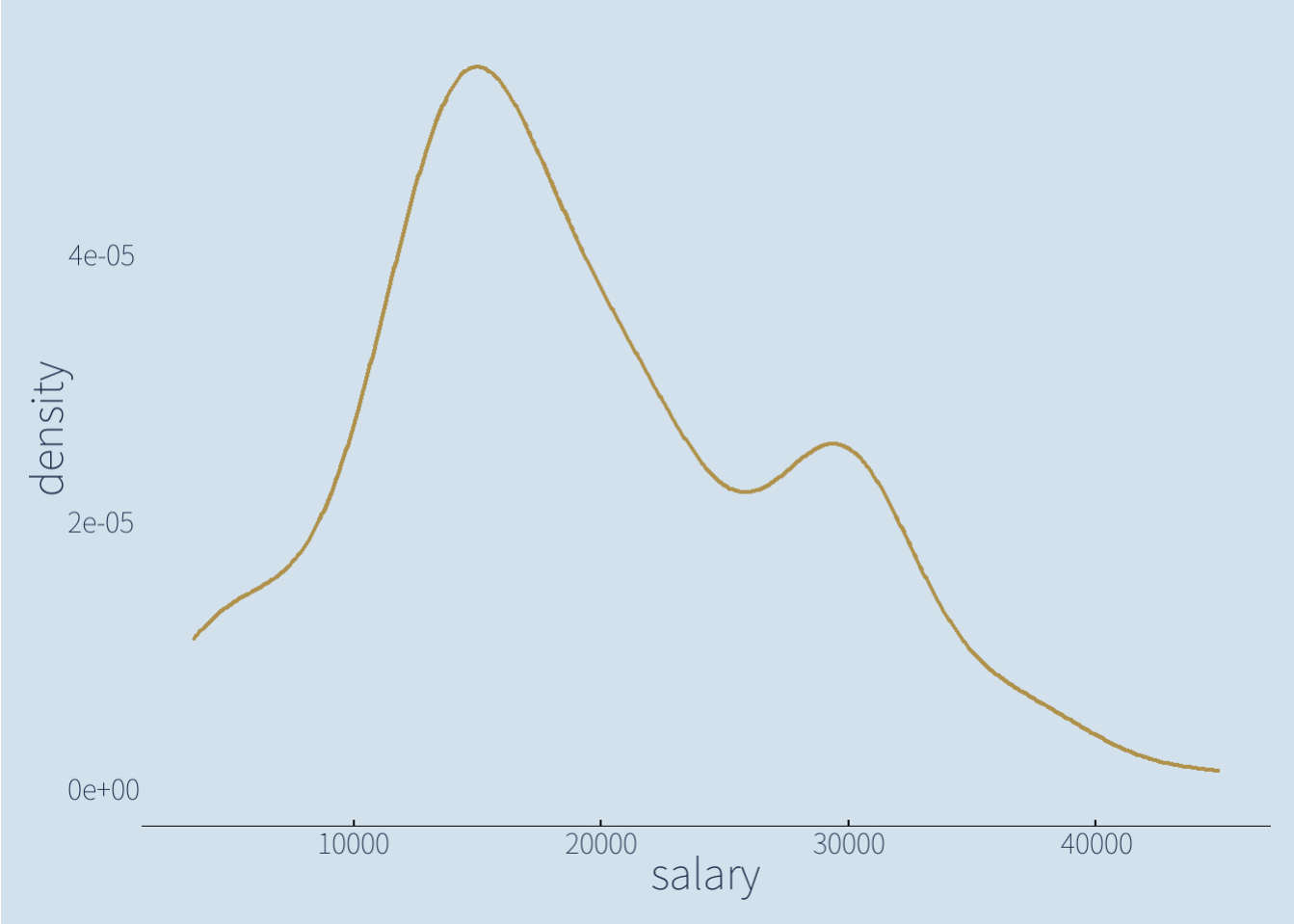
本文只用ggplot绘制图形，不展开使用其他绘制方法。

```
df_28 %>%
    ggplot(aes(x = salary))+
    geom_histogram(binwidth = 3000,
                   fill = palette[12],
                   color = palette[7])+
    theme_chen()
```



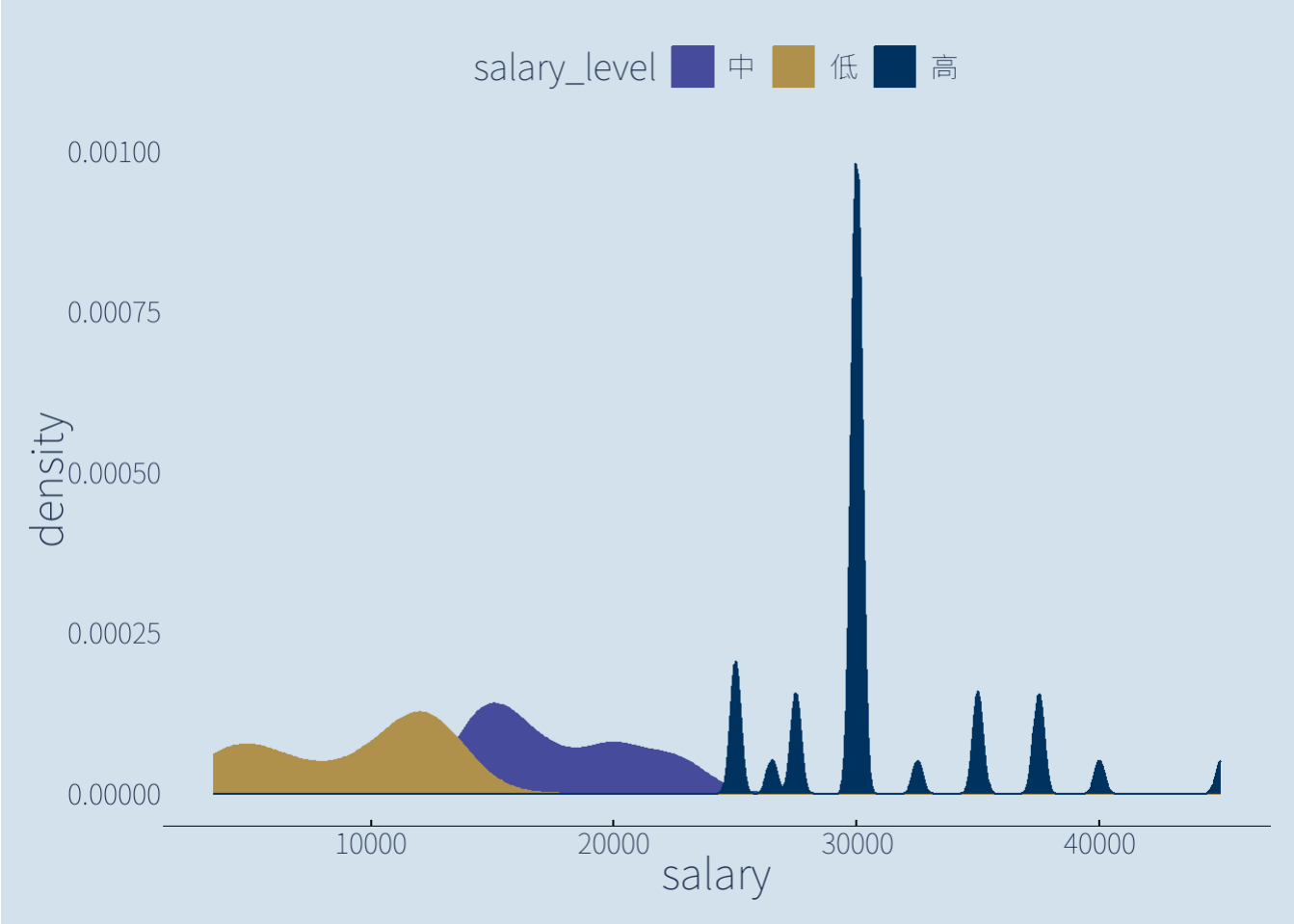
32 绘制薪资水平密度曲线

```
showtext::showtext_auto()

df_28 %>%
    ggplot(aes(x = salary))+
    geom_density(color = palette[7],
                 linewidth = 0.8)+
    theme_chen()
```



- 按等级分下，试下效果

```
df_28 %>%
    ggplot(aes(x = salary,
               group = salary_level))+
    geom_density(aes(color = salary_level,
                     fill = salary_level))+
    scale_fill_manual(values = palette[c(6,7,9)])+
    scale_color_manual(values = palette[c(6,7,9)])+
    theme_chen()
```



33 删除最后一列categories

前面文件的命名中，使用了salary_Level, 非categories.

删除列等于不选择该列。 属于列选择范畴，使用dplyr::select()函数。

dplyr::select()函数选择列的方式很多，本文不展开。

33.1 方法1：选择列位置，列数最大的一列。前加负号，及表示不选择

```
df_28 %>%
   select(-ncol(.)) %>%
   head(4)
```

```
## # A tibble: 4 × 5
##   createTime          education low   high  salary
##   <dttm>              <chr>     <chr> <chr>  <dbl>
## 1 2020-03-16 11:30:18 本科      20k   35k    27500
## 2 2020-03-16 10:58:47 本科      20k   40k    30000
## 3 2020-03-16 10:46:39 不限      20k   35k    27500
## 4 2020-03-16 10:45:44 本科      13k   20k    16500
```

33.2 方法2：选择列名

```
df_28 %>%
   select(-"salary_level") %>%
   head(4)
```

```
## # A tibble: 4 × 5
##   createTime          education low   high  salary
##   <dttm>              <chr>     <chr> <chr>  <dbl>
## 1 2020-03-16 11:30:18 本科      20k   35k    27500
## 2 2020-03-16 10:58:47 本科      20k   40k    30000
## 3 2020-03-16 10:46:39 不限      20k   35k    27500
## 4 2020-03-16 10:45:44 本科      13k   20k    16500
```

33.3 方法3：选择特征，比如包含，以什么开头等等...

```
df_28 %>%
    select(!contains("_")) %>%  #选择列名不包含“_”的列。
head(4)
```

```
## # A tibble: 4 × 5
##   createTime          education low   high  salary
##   <dttm>              <chr>     <chr> <chr>  <dbl>
## 1 2020-03-16 11:30:18 本科      20k   35k    27500
## 2 2020-03-16 10:58:47 本科      20k   40k    30000
## 3 2020-03-16 10:46:39 不限      20k   35k    27500
## 4 2020-03-16 10:45:44 本科      13k   20k    16500
```

34 将df的第一列与第二列合并为新的一列

使用tidyr::untie()函数

```
# remove 参数为FALSE
df %>%
    unite("new_col",
          1:2,
          sep = " ",
          remove = FALSE) %>%
    head(2)
```

```
##                    new_col          createTime education  salary
## 1 2020-03-16 11:30:18 本科 2020-03-16 11:30:18      本科 20k-35k
## 2 2020-03-16 10:58:48 本科 2020-03-16 10:58:47      本科 20k-40k
```

```
# remove 参数为TRUE
df %>%
    unite("new_col",
          1:2,
          sep = " ",
          remove = TRUE) %>%
    tail(2)
```

```
##                      new_col  salary
## 134 2020-03-16 10:46:31 本科 15k-23k
## 135 2020-03-16 11:19:38 本科 20k-40k
```

35 将education列与salary列合并为新的一列

方法与上题类似，上题指定了列位置，本题指定列名。

```
df %>%
    unite("new_col",
          c("education",
            "salary"),
          sep = " ",
          remove = TRUE) %>%
    head()
```

```
##            createTime      new_col
## 1 2020-03-16 11:30:18 本科 20k-35k
## 2 2020-03-16 10:58:47 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 10:33:47 本科 10k-18k
```

36 计算salary最大值与最小值之差

```
range_salary <- max(df_28$salary) - min(df_28$salary)
```

38.将第一行与最后一行拼接

这属于行操作方面的内容，可以使用dplyr::slice()或filter()函数。当然，也可以使用base R 中的数据操作函数。

36.1 方法1：使用base R 函数处理

```
rbind(df_28[1,],
      df_28[nrow(df_28),])
```

```
## # A tibble: 2 × 6
##   createTime          education low   high  salary salary_level
##   <dttm>              <chr>     <chr> <chr>  <dbl> <chr>
## 1 2020-03-16 11:30:18 本科      20k   35k    27500 高
## 2 2020-03-16 11:19:38 本科      20k   40k    30000 高
```

36.2 方法2：使用dplyr::select()函数处理

```
df_28 %>%
    filter(row_number()==1|row_number()==n())
```

```
## # A tibble: 2 × 6
##   createTime          education low   high  salary salary_level
##   <dttm>              <chr>     <chr> <chr>  <dbl> <chr>
## 1 2020-03-16 11:30:18 本科      20k   35k    27500 高
## 2 2020-03-16 11:19:38 本科      20k   40k    30000 高
```

36.3 方法3：使用dplyr::slice()函数处理

提取第一行，然后与最后一行合并，同时使用dplyr::bind_rows()函数

```
df_28 %>%
    slice(n()) %>%
    bind_rows(df_28[1,])
```

```
## # A tibble: 2 × 6
##   createTime          education low   high  salary salary_level
##   <dttm>              <chr>     <chr> <chr>  <dbl> <chr>
## 1 2020-03-16 11:19:38 本科      20k   40k    30000 高
## 2 2020-03-16 11:30:18 本科      20k   35k    27500 高
```

36.4 方法4：使用dplyr::slice()函数直接处理

直接选择对应的行，一步搞定。

```
# 这个是最直接了当的
df_28 %>%
    slice(1,n())
```

```
## # A tibble: 2 × 6
##   createTime          education low   high  salary salary_level
##   <dttm>              <chr>     <chr> <chr>  <dbl> <chr>
## 1 2020-03-16 11:30:18 本科      20k   35k    27500 高
## 2 2020-03-16 11:19:38 本科      20k   40k    30000 高
```

37 将第8行数据添加至末尾

本题与上题类似。可以理解为调整数据框行的位置。

另外，前面有做过提取行的练习。

37.1 方法1：通过选择行的方法调整行顺序，使得第8行排在最后一行。

```
  df_28[c(1:7,
          9:nrow(df_23_3),
          8),] %>%
tail() # 查看效果
```

```
## # A tibble: 6 × 6
##   createTime          education low   high  salary salary_level
##   <dttm>              <chr>     <chr> <chr>  <dbl> <chr>
## 1 2020-03-16 11:36:07 本科      10k   18k    14000 中
## 2 2020-03-16 09:54:47 硕士      25k   50k    37500 高
## 3 2020-03-16 10:48:32 本科      20k   40k    30000 高
## 4 2020-03-16 10:46:30 本科      15k   23k    19000 中
## 5 2020-03-16 11:19:38 本科      20k   40k    30000 高
## 6 2020-03-16 09:49:11 本科      10k   15k    12500 低
```

37.2 方法2：数据表与第八行数据拼接，或者说第8行数据拼接至数据表尾行，然后删除第8行。

```
df_28 %>%
    bind_rows(.,
              df_28[8,]) %>%
    slice(-8) %>%
    tail(4) #查看效果
```

```
## # A tibble: 4 × 6
##   createTime          education low   high  salary salary_level
##   <dttm>              <chr>     <chr> <chr>  <dbl> <chr>
## 1 2020-03-16 10:48:32 本科      20k   40k    30000 高
## 2 2020-03-16 10:46:30 本科      15k   23k    19000 中
## 3 2020-03-16 11:19:38 本科      20k   40k    30000 高
## 4 2020-03-16 09:49:11 本科      10k   15k    12500 低
```

38 查看每列的数据类型

```
str(df_28)
```

```
## tibble [135 × 6] (S3: tbl_df/tbl/data.frame)
##  $ createTime  : POSIXct[1:135], format: "2020-03-16 11:30:18" "2020-03-16 10:58:47" ...
##  $ education   : chr [1:135] "本科" "本科" "不限" "本科" ...
##  $ low         : chr [1:135] "20k" "20k" "20k" "13k" ...
##  $ high        : chr [1:135] "35k" "40k" "35k" "20k" ...
##  $ salary      : num [1:135] 27500 30000 27500 16500 15000 14000 23000 12500 7000 16000 ...
##  $ salary_level: chr [1:135] "高" "高" "高" "中" ...
```

39 将createTime列设置为索引

原始数据中，存在重复内容，重复内容不适宜作为索引，本题不能正常运行。

```
# 这个其实就是将第一列变成行名来着
#顺便练习一下几个相关函数

has_rownames(df_28) # 检查是否有行名

df_28 <-
    remove_rownames(df_28) # 把行名干掉

# 列名变行名，column_to_rownames()
# 不知道其他人是怎样实现的，我这里报错，说是行名不能有重复的
```

40 生成一个和df长度相同的随机数dataframe

可以使用data.frame()和tibble()实现，本题使用tibble()实现。

```
df42 <- tibble(random_num = runif(nrow(df_28),
                                  100,
                                  1000))

df42
```

```
## # A tibble: 135 × 1
##    random_num
##         <dbl>
##  1       209.
##  2       103.
##  3       747.
##  4       184.
##  5       167.
##  6       327.
##  7       590.
```

41 将上一题生成的dataframe与df合并

行数相同，列数不相同，优先使用列合并。可以使用base R 中的cbind()或dplyr::bind_cols()函数

41.1 方法1：cbind()函数

```
df_43 <- cbind(df_28,df42)
head(df_43 )
```

```
##            createTime education low high salary salary_level random_num
## 1 2020-03-16 11:30:18      本科 20k  35k  27500           高   209.1250
## 2 2020-03-16 10:58:47      本科 20k  40k  30000           高   102.8292
## 3 2020-03-16 10:46:39      不限 20k  35k  27500           高   747.3584
## 4 2020-03-16 10:45:44      本科 13k  20k  16500           中   183.7806
## 5 2020-03-16 10:20:41      本科 10k  20k  15000           中   166.5421
## 6 2020-03-16 10:33:47      本科 10k  18k  14000           中   327.3084
```

41.2 方法2：bind_cols()函数

```
df_43 <- bind_cols(df_28,df42)
head(df_43)
```

```
## # A tibble: 6 × 7
##   createTime          education low   high  salary salary_level random_num
##   <dttm>              <chr>     <chr> <chr>  <dbl> <chr>              <dbl>
## 1 2020-03-16 11:30:18 本科      20k   35k    27500 高                  209.
## 2 2020-03-16 10:58:47 本科      20k   40k    30000 高                  103.
## 3 2020-03-16 10:46:39 不限      20k   35k    27500 高                  747.
## 4 2020-03-16 10:45:44 本科      13k   20k    16500 中                  184.
## 5 2020-03-16 10:20:41 本科      10k   20k    15000 中                  167.
## 6 2020-03-16 10:33:47 本科      10k   18k    14000 中                  327.
```

42 生成新的一列new为salary列减去之前生成随机数列

在已有数据表中生成列，可以使用dplyr::mutate()函数 当然也可以直接生成数据向量，然后赋值给数据表的新增一列。

```
df_44 <- df_43 %>%
    mutate(new = salary - random_num)
head(df_44)
```

```
## # A tibble: 6 × 8
##   createTime          education low   high  salary salary_level random_num
##   <dttm>              <chr>     <chr> <chr>  <dbl> <chr>              <dbl>
## 1 2020-03-16 11:30:18 本科      20k   35k    27500 高                  209.
## 2 2020-03-16 10:58:47 本科      20k   40k    30000 高                  103.
## 3 2020-03-16 10:46:39 不限      20k   35k    27500 高                  747.
## 4 2020-03-16 10:45:44 本科      13k   20k    16500 中                  184.
## 5 2020-03-16 10:20:41 本科      10k   20k    15000 中                  167.
## 6 2020-03-16 10:33:47 本科      10k   18k    14000 中                  327.
## # ℹ 1 more variable: new <dbl>
```

43 检查数据中是否含有任何缺失值

本题使用across()函数，配合everything()执行操作。

```
df_44 %>%
   summarise(across(everything(),
                    ~sum(is.na(.x))))
```

```
## # A tibble: 1 × 8
##   createTime education   low  high salary salary_level random_num   new
##        <int>     <int> <int> <int>  <int>        <int>      <int> <int>
## 1          0         0     0     0      0            0          0     0
```

44 将salary列类型转换为浮点数

```
df_44$salary <- as.double(df_44$salary )
```

45 计算salary大于10000的次数

```
df_44 %>%
    filter(salary > 10000) %>%
    count()
```

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

46 查看每种学历出现的次数

```
df_44 %>%
    count(education)
```

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

47 查看education列共有几种学历

```
df_44 %>%
    count(education) %>%
    count()
```

```
## # A tibble: 1 × 1
##       n
##   <int>
## 1     4
```

48 提取salary与new列的和大于60000的最后3行

```
 df_44 %>%
    filter(salary + new > 60000) %>%
    tail(3)
```

```
## # A tibble: 3 × 8
##   createTime          education low   high  salary salary_level random_num
##   <dttm>              <chr>     <chr> <chr>  <dbl> <chr>              <dbl>
## 1 2020-03-16 10:41:20 本科      30k   40k    35000 高                  330.
## 2 2020-03-16 11:01:39 本科      30k   45k    37500 高                  393.
## 3 2020-03-16 09:54:47 硕士      25k   50k    37500 高                  594.
## # ℹ 1 more variable: new <dbl>
```

49 使用绝对路径读取本地Excel数据

不同的包，读过来的数据是不一样的。格式都不一样，一个是dataframe,一个是tibble,后面对于NA 值的检查等各项操作也不一样。

49.1 方法1：使用xlsx包函数

```
data <- xlsx::read.xlsx2("./data/600000.SH.xls",
                         sheetName = "600000.SH",
                         header = T,
                         na.rm =FALSE)
str(data)
```

```
## 'data.frame':    329 obs. of  19 variables:
##  $ 代码           : chr  "600000.SH" "600000.SH" "600000.SH" "600000.SH" ...
##  $ 简称           : chr  "浦发银行" "浦发银行" "浦发银行" "浦发银行" ...
##  $ 日期           : chr  "42373" "42374" "42375" "42376" ...
##  $ 前收盘价.元.   : chr  "16.1356" "15.7205" "15.8618" "15.9855" ...
##  $ 开盘价.元.     : chr  "16.1444" "15.4644" "15.8088" "15.7205" ...
##  $ 最高价.元.     : chr  "16.1444" "15.9501" "16.0208" "15.8088" ...
##  $ 最低价.元.     : chr  "15.4997" "15.3672" "15.6234" "15.3672" ...
##  $ 收盘价.元.     : chr  "15.7205" "15.8618" "15.9855" "15.4644" ...
##  $ 成交量.股.     : chr  "42240610" "58054793" "46772653" "11350479" ...
```

49.2 方法2：使用readxl包包函数

```
data_2 <- readxl::read_xls("./data/600000.SH.xls",
                           sheet = "600000.SH",
                           col_names = TRUE)

str(data_2)
```

```
## tibble [329 × 18] (S3: tbl_df/tbl/data.frame)
##  $ 代码           : chr [1:329] "600000.SH" "600000.SH" "600000.SH" "600000.SH" ...
##  $ 简称           : chr [1:329] "浦发银行" "浦发银行" "浦发银行" "浦发银行" ...
##  $ 日期           : POSIXct[1:329], format: "2016-01-04" "2016-01-05" ...
##  $ 前收盘价(元)   : num [1:329] 16.1 15.7 15.9 16 15.5 ...
##  $ 开盘价(元)     : num [1:329] 16.1 15.5 15.8 15.7 15.7 ...
##  $ 最高价(元)     : num [1:329] 16.1 16 16 15.8 15.8 ...
##  $ 最低价(元)     : num [1:329] 15.5 15.4 15.6 15.4 14.9 ...
##  $ 收盘价(元)     : num [1:329] 15.7 15.9 16 15.5 15.4 ...
##  $ 成交量(股)     : chr [1:329] "42240610" "58054793" "46772653" "11350479" ...
```

50 查看数据前三行

```
# data 和data_2两组数据首尾行数实录。
data %>%
    head(3)
```

```
##        代码     简称   日期 前收盘价.元. 开盘价.元. 最高价.元. 最低价.元.
## 1 600000.SH 浦发银行 42373      16.1356    16.1444    16.1444    15.4997
## 2 600000.SH 浦发银行 42374      15.7205    15.4644    15.9501    15.3672
## 3 600000.SH 浦发银行 42375      15.8618    15.8088    16.0208    15.6234
##   收盘价.元. 成交量.股. 成交金额.元. 涨跌.元. 涨跌幅... 均价.元. 换手率...
## 1    15.7205   42240610    754425783  -0.4151   -2.5725  17.8602    0.2264
## 2    15.8618   58054793   1034181474   0.1413    0.8989  17.8139    0.3112
## 3    15.9855   46772653    838667398   0.1236    0.7795  17.9307    0.2507
##     A股流通市值.元.        总市值.元. A股流通股本.股. 市盈率 na.rm
## 1      332031791187      332031791187     18653471415 6.5614 FALSE
```

```
data %>%
    tail(3)
```

```
##                   代码      简称   日期 前收盘价.元. 开盘价.元. 最高价.元.
## 327          600000.SH 浦发银行 42864        14.86      14.69      14.84
## 328
## 329 数据来源：Wind资讯
##     最低价.元. 收盘价.元. 成交量.股. 成交金额.元. 涨跌.元. 涨跌幅... 均价.元.
## 327      14.66      14.76   19225492    283864640     -0.1   -0.6729   14.765
## 328
## 329
##     换手率...     A股流通市值.元.          总市值.元. A股流通股本.股. 市盈率
## 327    0.0889 3.1908581164872E11 3.1908581164872E11     21618279922 6.0093
```

```
data_2 %>%
    head(3)
```

```
## # A tibble: 3 × 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
## # ℹ 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>,
```

```
data_2%>%
    tail(3)
```

```
## # A tibble: 3 × 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… 2017-05-09 00:00:00           14.9         14.7         14.8
## 2 <NA>        <NA>  NA                            NA           NA           NA
## 3 数据来源：… <NA>  NA                            NA           NA           NA
## # ℹ 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>,
```

```
# data 和data_2两组数据首尾行数实录。
```

51 查看每列数据缺失值情况

51.1 对于data数据，检测不到NA值.

51.1.1 方法1:colSums()函数

```
is.na(data) %>%
    colSums()
```

```
##            代码            简称            日期   前收盘价.元.     开盘价.元. 
##               0               0               0               0               0 
##     最高价.元.     最低价.元.     收盘价.元.     成交量.股.   成交金额.元. 
##               0               0               0               0               0 
##       涨跌.元.      涨跌幅...      均价.元.      换手率... A股流通市值.元. 
##               0               0               0               0               0 
##     总市值.元. A股流通股本.股.          市盈率          na.rm 
##               0               0               0               0
```

51.1.2 方法2：用across()函数处理

```
data %>%
   summarise(across(everything(),
                    ~sum(is.na(.x))))
```

```
##   代码 简称 日期 前收盘价.元. 开盘价.元. 最高价.元. 最低价.元. 收盘价.元.
## 1    0    0    0            0          0          0          0          0
##   成交量.股. 成交金额.元. 涨跌.元. 涨跌幅... 均价.元. 换手率... A股流通市值.元.
## 1          0           0        0         0        0         0                0
##   总市值.元. A股流通股本.股. 市盈率 na.rm
## 1          0               0      0     0
```

51.2 对于data_2 数据，tibble 类型的数据，可以检测到NA值。

51.2.1 方法1:colSums()函数

```
is.na(data_2) %>%
   colSums()
```

```
##             代码             简称             日期     前收盘价(元)       开盘价(元)
##                1                2                2                2                2
##       最高价(元)       最低价(元)       收盘价(元)       成交量(股)     成交金额(元)
##                2                2                2                2                2
##         涨跌(元)       涨跌幅(%)         均价(元)       换手率(%) A股流通市值(元)
##                2                2                2                2                2
##       总市值(元) A股流通股本(股)           市盈率
##                2                2                2
```

51.2.2 方法2：用across()函数处理

```
data_2 %>%
  summarise(across(everything(),
                   ~sum(is.na(.x))))
```

```
## # A tibble: 1 × 18
##    代码  简称  日期 `前收盘价(元)` `开盘价(元)` `最高价(元)` `最低价(元)`
##   <int> <int> <int>          <int>        <int>        <int>        <int>
## 1     1     2     2              2            2            2            2
## # ℹ 11 more variables: `收盘价(元)` <int>, `成交量(股)` <int>,
## #   `成交金额(元)` <int>, `涨跌(元)` <int>, `涨跌幅(%)` <int>,
## #   `均价(元)` <int>, `换手率(%)` <int>, `A股流通市值(元)` <int>,
## #   `总市值(元)` <int>, `A股流通股本(股)` <int>, 市盈率 <int>
```

后续各题相关数据使用data_2执行计算分析。

52 提取日期列含有空值的行

```
data_2 %>%
    filter(is.na(日期))
```

```
## # A tibble: 2 × 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 <NA>        <NA>  NA                NA           NA           NA
## 2 数据来源：… <NA>  NA                NA           NA           NA
## # ℹ 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>
```

53 输出每列缺失值具体行数

和前面题目相类似

```
  data_2 %>%
   summarise(across(everything(),
                    ~sum(is.na(.x))))
```

```
## # A tibble: 1 × 18
##    代码  简称  日期 `前收盘价(元)` `开盘价(元)` `最高价(元)` `最低价(元)`
##   <int> <int> <int>          <int>        <int>        <int>        <int>
## 1     1     2     2              2            2            2            2
## # ℹ 11 more variables: `收盘价(元)` <int>, `成交量(股)` <int>,
## #   `成交金额(元)` <int>, `涨跌(元)` <int>, `涨跌幅(%)` <int>,
## #   `均价(元)` <int>, `换手率(%)` <int>, `A股流通市值(元)` <int>,
## #   `总市值(元)` <int>, `A股流通股本(股)` <int>, 市盈率 <int>
```

54 删除所有存在缺失值的行

54.1 方法1：drop.na()函数

```
data_56 <- data_2 %>%
 drop_na()

tail(data_56) # 查看效果
```

```
## # A tibble: 6 × 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… 2017-05-02 00:00:00          15.2         15.2         15.2
## 2 600000.SH 浦发银… 2017-05-03 00:00:00          15.2         15.2         15.2
## 3 600000.SH 浦发银… 2017-05-04 00:00:00          15.1         15.1         15.1
## 4 600000.SH 浦发银… 2017-05-05 00:00:00          15.0         15.0         15.0
## 5 600000.SH 浦发银… 2017-05-08 00:00:00          14.9         14.8         14.9
## 6 600000.SH 浦发银… 2017-05-09 00:00:00          14.9         14.7         14.8
## # ℹ 12 more variables: `最低价(元)` <dbl>, `收盘价(元)` <dbl>,
```

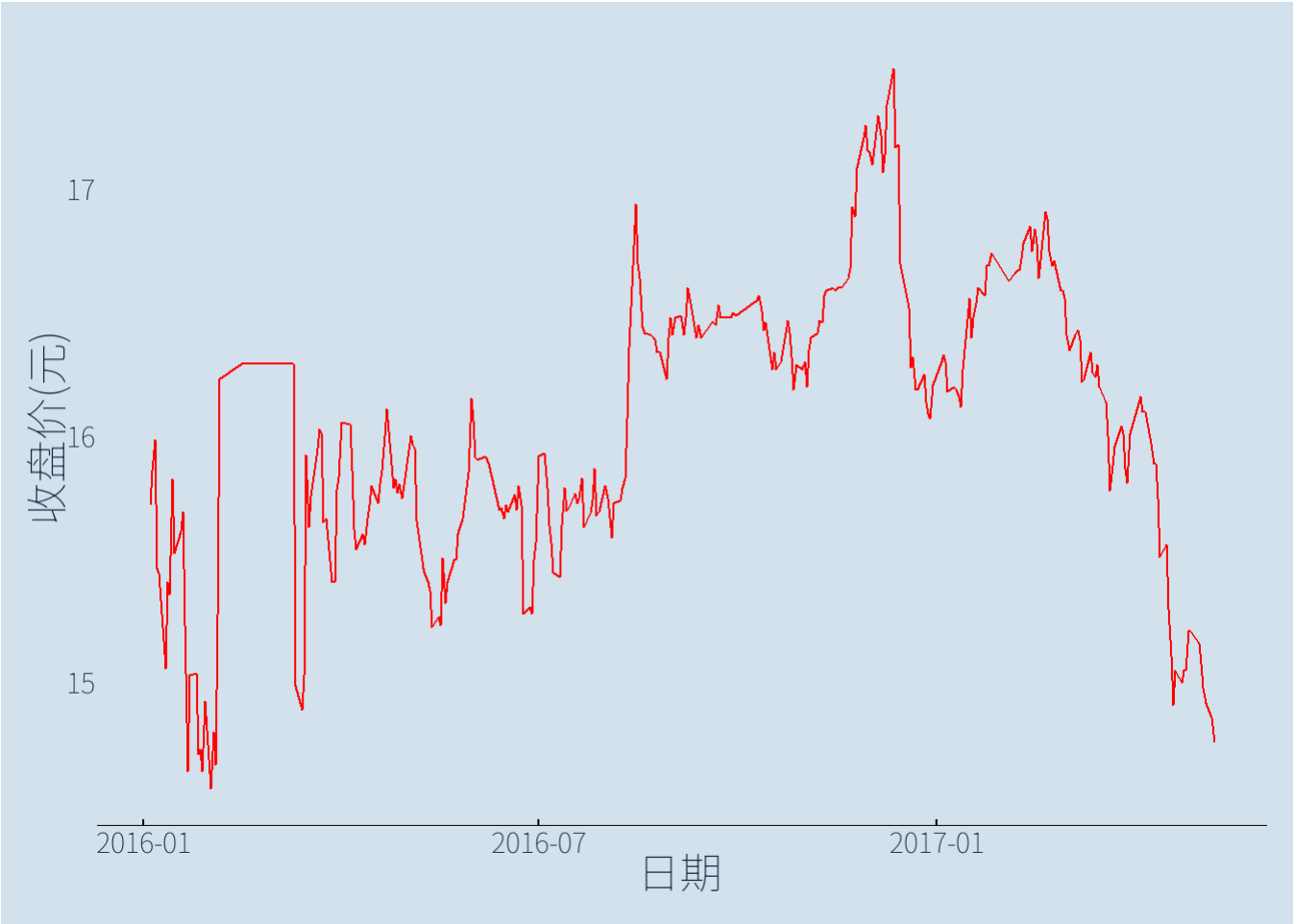
54.2 方法2：na.omit()函数

```
na.omit(data_2) %>%
      tail()  # 查看效果
```

```
## # A tibble: 6 × 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… 2017-05-02 00:00:00           15.2         15.2         15.2
## 2 600000.SH 浦发银… 2017-05-03 00:00:00           15.2         15.2         15.2
## 3 600000.SH 浦发银… 2017-05-04 00:00:00           15.1         15.1         15.1
## 4 600000.SH 浦发银… 2017-05-05 00:00:00           15.0         15.0         15.0
## 5 600000.SH 浦发银… 2017-05-08 00:00:00           14.9         14.8         14.9
## 6 600000.SH 浦发银… 2017-05-09 00:00:00           14.9         14.7         14.8
## # ℹ 12 more variables: `最低价(元)` <dbl>, `收盘价(元)` <dbl>,
```

55 绘制收盘价的折线图

```
data_56 %>%
    ggplot(aes(x = 日期,
               y = `收盘价(元)`))+
    # y值变量名有括号，用`` 把他包起来即可。
    geom_line(show.legend = TRUE,
              colour = "red")+
    theme_chen()
```



56 同时绘制开盘价与收盘价

56.1 方法1：使用宽数据

aes里没有足够的变量，show.legend为TRUE,也没有用。这种图没有图例，那是比较糟糕的。

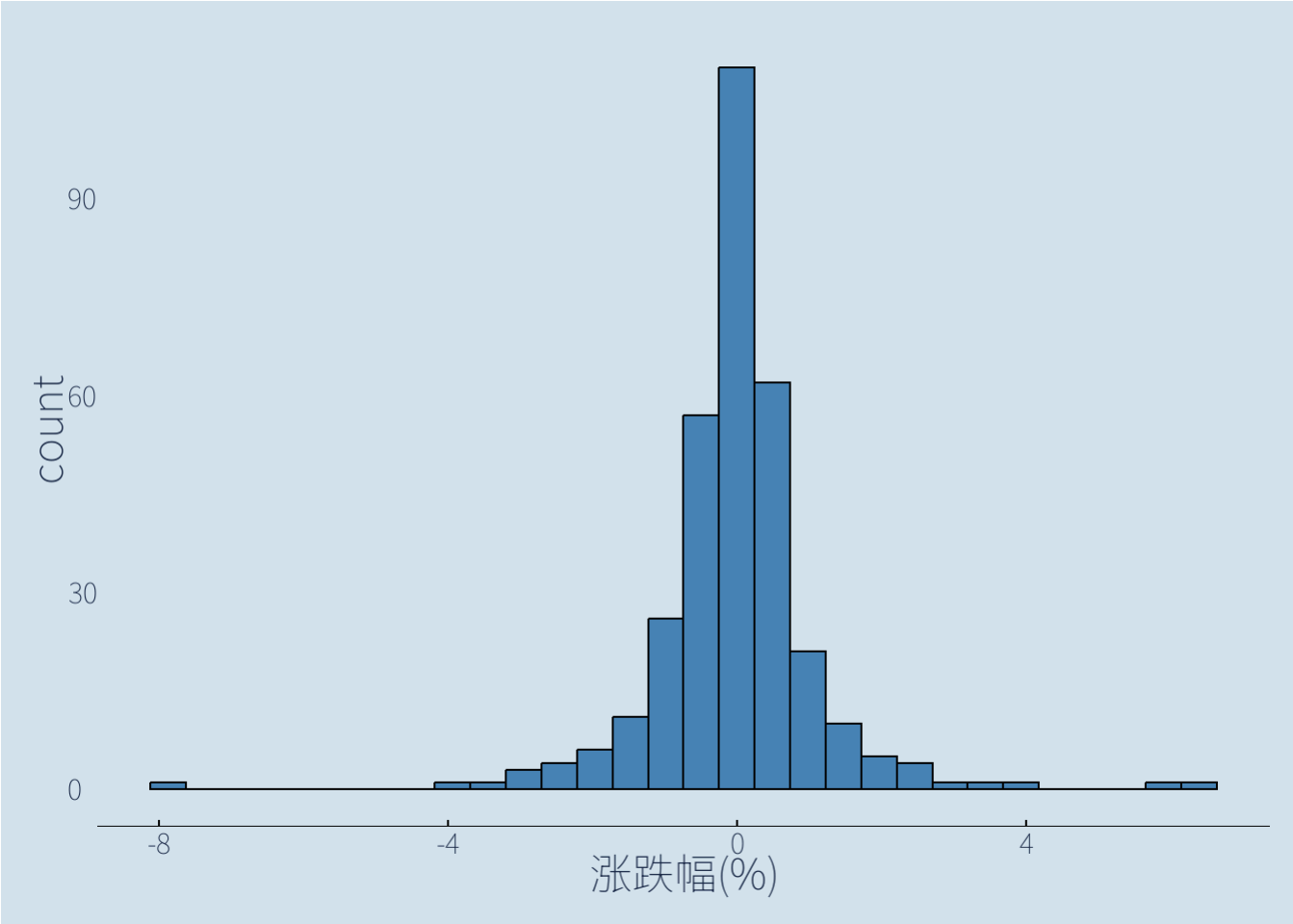
```
data_56 %>%
    ggplot(aes(x = 日期))+

    geom_line(aes(x = 日期,
                  y = `开盘价(元)`),
              linewidth = 0.4,
              color = "blue",
              show.legend = TRUE)+

    geom_line(aes(x = 日期,
```

56.2 方法2：使用长数据

```
data_56 %>%
    select(c(3,5,8)) %>%
    pivot_longer(2:3,names_to = "状态",values_to = "价格(元)") %>%
    ggplot(aes(x = 日期, y = `价格(元)`,group = 状态)) +
    geom_line(aes(color = 状态)) +
     theme_bw()+
      labs(title = "使用长数据",
          subtitle = "有图例")+
    theme_chen()
```

57 绘制涨跌幅的直方图

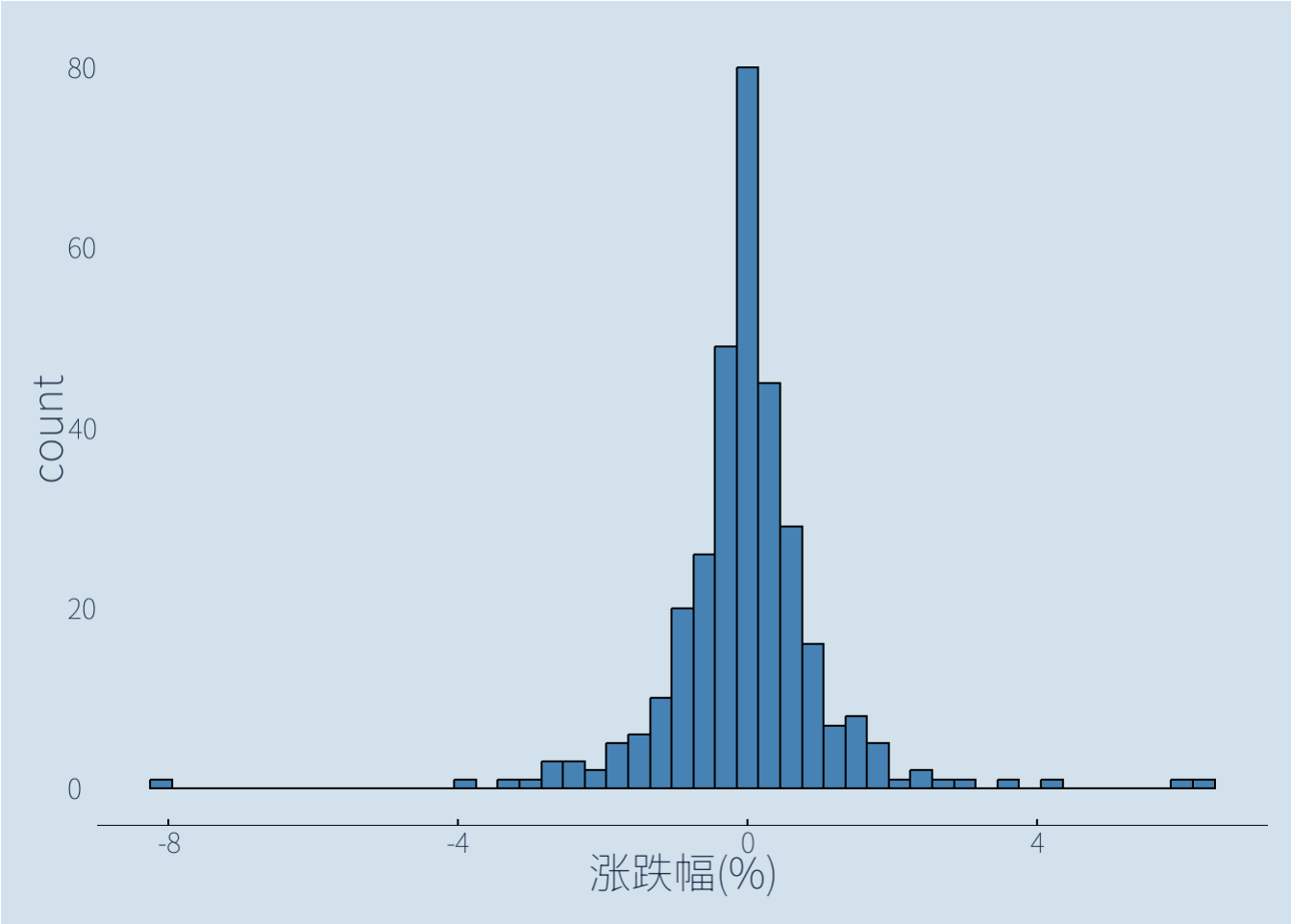
```
data_56 %>%
    ggplot(aes(`涨跌幅(%)`)) +
    geom_histogram(fill = "steelblue",color = "black")+
    theme_chen()
```



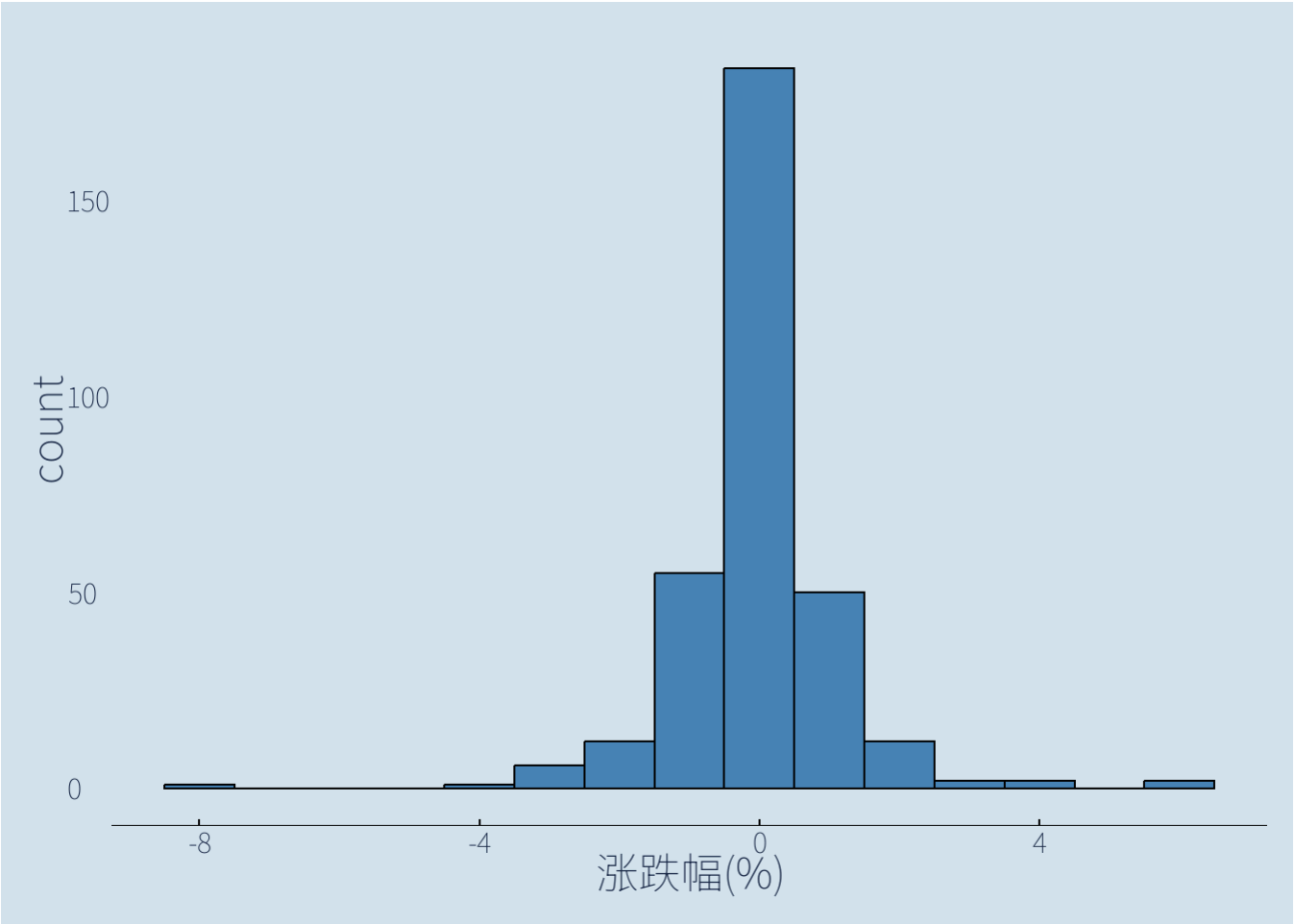
58 让直方图更细致

理解题目的意图是调节bin和，柱子宽度这方面的想法

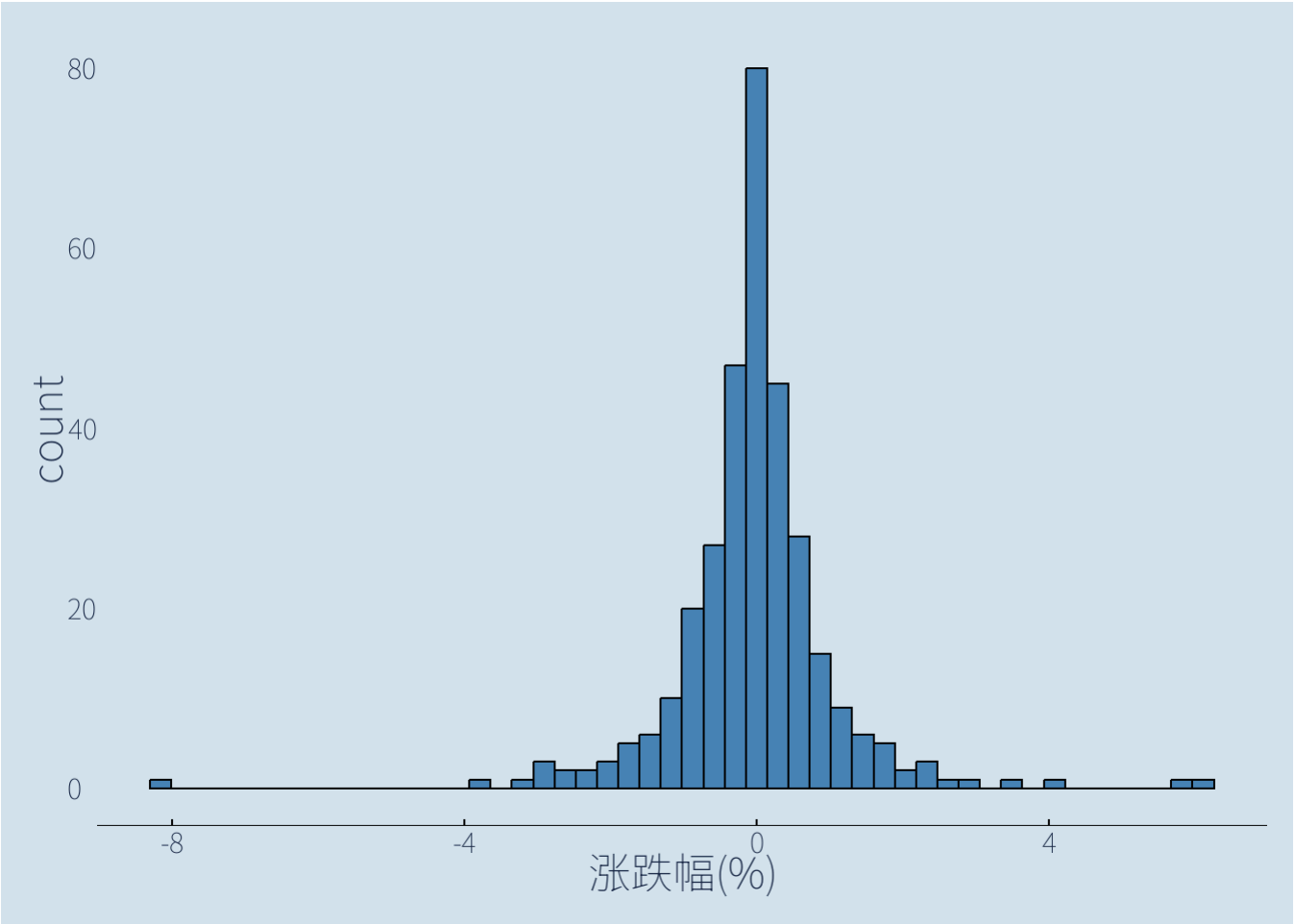
```
data_56 %>%
    ggplot(aes(`涨跌幅(%)`)) +
    geom_histogram(binwidth = 0.3,fill = "steelblue",color = "black")+
    theme_chen()
```



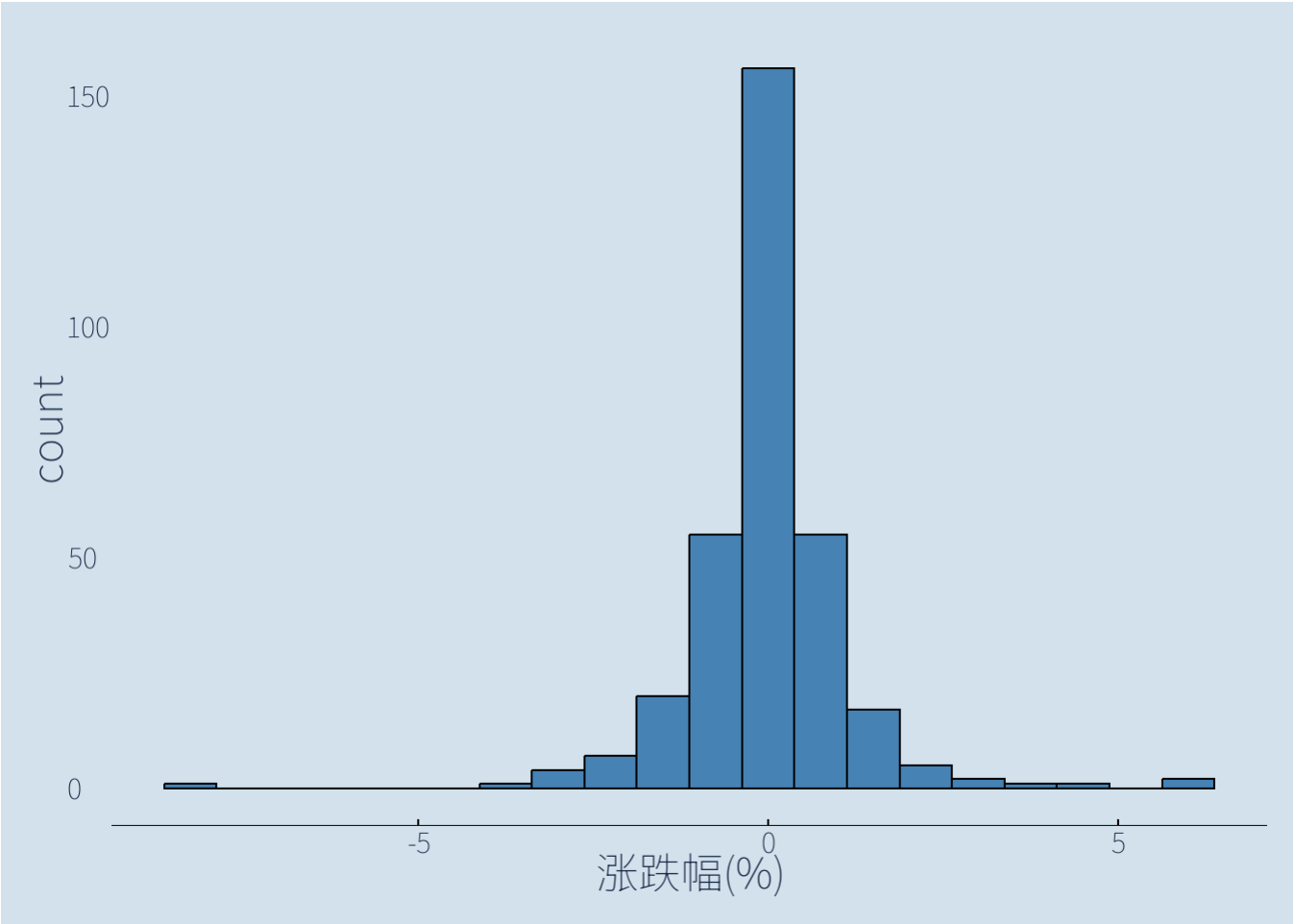

```
data_56 %>%
    ggplot(aes(`涨跌幅(%)`)) +
    geom_histogram(binwidth = 1,fill = "steelblue",color = "black")+
    theme_chen()
```




```
data_56 %>%
    ggplot(aes(`涨跌幅(%)`)) +
    geom_histogram(bins = 50,fill = "steelblue",color = "black")+
    theme_chen()
```




```
data_56 %>%
    ggplot(aes(`涨跌幅(%)`)) +
    geom_histogram(bins = 20,fill = "steelblue",color = "black")+
    theme_chen()
```



59 以data的列名创建一个dataframe

```
ndf <- names(data_56) %>%
    as_tibble()

ndf
```

```
## # A tibble: 18 × 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)
##  7 最低价(元)
```

60 打印所有换手率不是数字的行

其实原始数据读入后，该列数据全部都不是数字型，而是字符串型；

把列强制改为数字型，然后，这个过程中，非数字型的字符变成了NA;

不是数字的行，其实就变成了是NA 的行。

```
data_56 %>%
   mutate(`换手率(%)` = as.numeric(`换手率(%)`)) %>%
   filter(is.na(`换手率(%)`)) %>%
   select(`换手率(%)`,
          everything()) %>%
   head(3)
```

```
## # A tibble: 3 × 18
##   `换手率(%)` 代码      简称     日期                `前收盘价(元)` `开盘价(元)`
##         <dbl> <chr>     <chr>    <dttm>                       <dbl>        <dbl>
## 1          NA 600000.SH 浦发银行 2016-02-16 00:00:00           16.3         16.3
## 2          NA 600000.SH 浦发银行 2016-02-17 00:00:00           16.3         16.3
## 3          NA 600000.SH 浦发银行 2016-02-18 00:00:00           16.3         16.3
## # ℹ 12 more variables: `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>,
## #   `A股流通市值(元)` <dbl>, `总市值(元)` <dbl>, `A股流通股本(股)` <dbl>,
```

```
# 把换手率列前移，查看前三行
```

61 打印所有换手率为--的行

```
data_56 %>%
   filter(`换手率(%)` =="--") %>%
   select(`换手率(%)`,
          everything()) %>%
   head(3)
```

```
## # A tibble: 3 × 18
##   `换手率(%)` 代码      简称     日期                `前收盘价(元)` `开盘价(元)`
##   <chr>       <chr>     <chr>    <dttm>                       <dbl>         <dbl>
## 1 --          600000.SH 浦发银行 2016-02-16 00:00:00           16.3          16.3
## 2 --          600000.SH 浦发银行 2016-02-17 00:00:00           16.3          16.3
## 3 --          600000.SH 浦发银行 2016-02-18 00:00:00           16.3          16.3
## # ℹ 12 more variables: `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>,
## #   `A股流通市值(元)` <dbl>, `总市值(元)` <dbl>, `A股流通股本(股)` <dbl>,
```

```
# 与上题类似。
```

62 重置data的行号

重置是一个宽泛的概念，清空还是格式化？这里生成随机数来替换。

```
set.seed(1234)

data_56 %>%
    rowid_to_column() %>%
    column_to_rownames("rowid") %>%
    head(3)
```

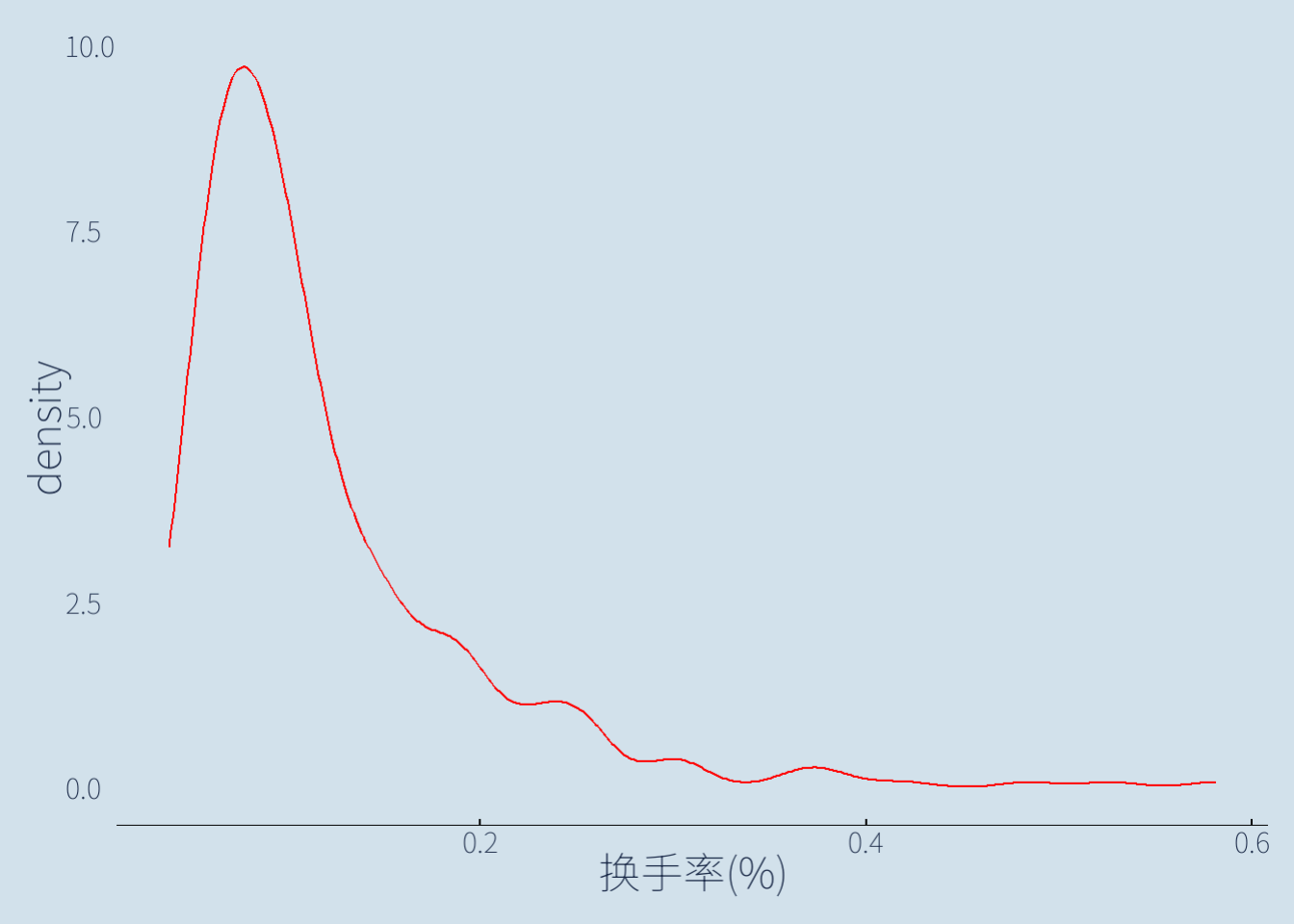
```
##        代码     简称       日期 前收盘价(元) 开盘价(元) 最高价(元) 最低价(元)
## 1 600000.SH 浦发银行 2016-01-04      16.1356    16.1444    16.1444    15.4997
## 2 600000.SH 浦发银行 2016-01-05      15.7205    15.4644    15.9501    15.3672
## 3 600000.SH 浦发银行 2016-01-06      15.8618    15.8088    16.0208    15.6234
##   收盘价(元) 成交量(股) 成交金额(元) 涨跌(元) 涨跌幅(%)           均价(元)
## 1    15.7205   42240610    754425783  -0.4151   -2.5725 17.860199999999999
## 2    15.8618   58054793   1034181474   0.1413    0.8989            17.8139
## 3    15.9855   46772653    838667398   0.1236    0.7795 17.930700000000002
##             换手率(%) A股流通市值(元)   总市值(元) A股流通股本(股) 市盈率
## 1 0.22639999999999999    332031791187 332031791187     18653471415 6.5614
```

63 删除所有换手率为非数字的行

```
data_65 <- data_56 %>%
    mutate(`换手率(%)` = as.numeric(`换手率(%)`)) %>%
    drop_na(`换手率(%)`)
```

64 绘制换手率的密度曲线

```
data_65 %>%
    ggplot(aes(`换手率(%)`)) +
    geom_density(color = "red")+
    theme_chen()
```



65 计算前一天与后一天收盘价的差值

```
data_65 %>%
mutate(gap = `收盘价(元)`- lag(n = 1,`收盘价(元)` ),
       .after = `收盘价(元)` ) %>%
    head(3)
```

```
## # A tibble: 3 × 19
##   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
## # ℹ 13 more variables: `最低价(元)` <dbl>, `收盘价(元)` <dbl>, gap <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>,
```

66 计算前一天与后一天收盘价变化率

```
data_65 %>%
mutate(gap = `收盘价(元)`- lag(n = 1,`收盘价(元)`),
       ratio = (gap/`收盘价(元)`) %>%
           scales::percent(accuracy = 0.01),
       .before = `收盘价(元)` ) %>%
    head(3)
```

```
## # A tibble: 3 × 20
##   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
## # ℹ 14 more variables: `最低价(元)` <dbl>, gap <dbl>, ratio <chr>,
## #   `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>,
```

67 设置日期为索引

```
data_65 %>%
    column_to_rownames("日期") %>%
    head(3)
```

```
##                  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
##            收盘价(元) 成交量(股) 成交金额(元) 涨跌(元) 涨跌幅(%)
## 2016-01-04    15.7205   42240610    754425783  -0.4151   -2.5725
## 2016-01-05    15.8618   58054793   1034181474   0.1413    0.8989
## 2016-01-06    15.9855   46772653    838667398   0.1236    0.7795
##                      均价(元) 换手率(%) A股流通市值(元)    总市值(元)
## 2016-01-04 17.860199999999999    0.2264    332031791187 332031791187
```

68 以5个数据作为一个数据滑动窗口，在这个5个数据上取均值(收盘价)

这就是股票里的5日均线或者说那种移动平均值的计算。实现的方法非常多，本题使用自编函数以及几种常用包及函数进行计算。

68.1 方法1：用基础包和tidyverse 相关函数做函数自己计算

```
 ma_fun <- function(data, width){
   i  = length(data)
   if(width > i){
       return("数据不够，请增加数据")
   }
   k =width:i
   ma_value =c(rep(NA,width-1),
               map_vec(k,\(x) mean(data[(x-(width-1)):x])))
return( ma_value)
 }
```

```
## # A tibble: 309 × 1
##    average_5
##        <dbl>
##  1      NA
##  2      NA
##  3      NA
##  4      NA
##  5      15.7
##  6      15.6
##  7      15.5
```

68.2 方法2：调用相关的基础函数计算：zoo::rollmean()函数

```
data_65 %>%
    transmute(average_5 = zoo::rollmean(`收盘价(元)`,5,
                                        align = "right",
                                        fill = NA))
```

```
## # A tibble: 309 × 1
##    average_5
##        <dbl>
##  1      NA
##  2      NA
##  3      NA
##  4      NA
##  5      15.7
##  6      15.6
##  7      15.5
```

68.3 方法3：调用相关的基础函数计算：DescTools::MoveAvg()函数

```
data_65 %>%
    transmute(average_5 = DescTools::MoveAvg(x = `收盘价(元)`,
                                             order = 5,
                                             align = "right",
                                             endrule = "NA" ))
```

```
## # A tibble: 309 × 1
##    average_5
##        <dbl>
##  1      NA
##  2      NA
##  3      NA
##  4      NA
##  5      15.7
##  6      15.6
##  7      15.5
```

68.4 方法4：调用相关的基础函数计算：slider::slide_dbl()函数

```
data_65 %>%
    transmute(average_5 = slider::slide_dbl( `收盘价(元)`,
                                             mean,
                                             .before = 4,
                                             .complete = TRUE))
```

```
## # A tibble: 309 × 1
##    average_5
##        <dbl>
##  1      NA
##  2      NA
##  3      NA
##  4      NA
##  5      15.7
##  6      15.6
##  7      15.5
```

69 以5个数据作为一个数据滑动窗口，计算这五个数据总和(收盘价)

与计算均值同样的逻辑。

69.1 方法1：用基础包和tidyverse 相关函数做函数自己计算

```
 ms_fun <- function(data, width){
   i  = length(data)
   if(width > i){
       return("数据不够，请增加数据")
   }
   k =width:i
   ms_value =c(rep(NA,width-1),
               map_vec(k,\(x) sum(data[(x-(width-1)):x])))
return( ms_value)
 }
```

```
## # A tibble: 309 × 1
##    sum_5
##    <dbl>
##  1  NA
##  2  NA
##  3  NA
##  4  NA
##  5  78.5
##  6  77.8
##  7  77.4
```

69.2 方法2：调用相关的基础函数计算：zoo::rollsum函数

有很多金融和时序相关的包都有类似函数，类似roll sum,move average这类函数，见诸于zoo,xts,quantmod,TTR等包，而tidyquant有个综合函数，整合引用这些函数。这里用zoo包里的函数做一下。

```
data_65 %>%
    transmute(sum_5 = zoo::rollsum(`收盘价(元)`,
                                   5,
                                   align = "right",
                                   fill = NA))
```

```
## # A tibble: 309 × 1
##    sum_5
##    <dbl>
##  1  NA
##  2  NA
##  3  NA
##  4  NA
##  5  78.5
##  6  77.8
##  7  77.4
```

69.3 方法3：调用相关的基础函数计算：slider::slide_dbl()函数

```
data_65 %>%
    transmute(sum_5 = slider::slide_dbl( `收盘价(元)`,
                                         sum,
                                         .before = 4,
                                         .complete = TRUE))
```

```
## # A tibble: 309 × 1
##    sum_5
##    <dbl>
##  1  NA
##  2  NA
##  3  NA
##  4  NA
##  5  78.5
##  6  77.8
##  7  77.4
```

70 将收盘价5日均线、20日均线与原始数据绘制在同一个图上

```
data_65 %>%
    select(日期,`收盘价(元)`) %>%
    mutate(avg_5 = ma_fun(`收盘价(元)`,5),
           avg_20 = ma_fun(`收盘价(元)`,20)) %>%
    tidyr::pivot_longer(-1,
                        names_to = "类型",
                        values_to = "价格") %>%
    ggplot(aes(x = 日期,
               y = 价格,
               group = 类型))+
```

71 按周为采样规则，取一周收盘价最大值

不能机械的使用period 滑窗取最大值，因为有些假期日期是没有数据的；

也不能简单的取日期的周期，然后分组取最大值，因为有不同的年份的考虑；

可以按年-周两级分组，求最大值；

或者机械的把他们合到一起，当作是做特征工程； 最好的办法就是找到年+周的显示方法函数(欧美公司里基本都是用年-周工作逻辑来的)

71.1 方法1：年、周两级分组求最大值；

这个逻辑还是很清晰的：先group,再执行操作

```
data_65 %>%
    select(日期,`收盘价(元)`) %>%
    mutate(year = year(日期),
        weeknum =  week(日期)) %>%
    group_by(year,weeknum) %>%
    slice_max(`收盘价(元)`)
```

```
## # A tibble: 74 × 4
## # Groups:   year, weeknum [69]
##    日期                `收盘价(元)`  year weeknum
##    <dttm>                     <dbl> <dbl>   <dbl>
##  1 2016-01-06 00:00:00         16.0  2016       1
##  2 2016-01-14 00:00:00         15.8  2016       2
##  3 2016-01-19 00:00:00         15.7  2016       3
##  4 2016-01-25 00:00:00         15.0  2016       4
##  5 2016-02-04 00:00:00         15.3  2016       5
##  6 2016-02-05 00:00:00         16.2  2016       6
```

71.2 方法2：年-组合字段,group 对象作为参数选项放在函数里。

```
data_65 %>%
    select(日期,`收盘价(元)`) %>%
    mutate(year_week = str_c(year(日期),
                             week(日期),
                             sep = "_")) %>%
    slice_max(`收盘价(元)`,by = year_week )
```

```
## # A tibble: 74 × 3
##    日期                `收盘价(元)` year_week
##    <dttm>                     <dbl> <chr>
##  1 2016-01-06 00:00:00         16.0 2016_1
##  2 2016-01-14 00:00:00         15.8 2016_2
##  3 2016-01-19 00:00:00         15.7 2016_3
##  4 2016-01-25 00:00:00         15.0 2016_4
##  5 2016-02-04 00:00:00         15.3 2016_5
##  6 2016-02-05 00:00:00         16.2 2016_6
##  7 2016-02-15 00:00:00         16.3 2016_7
```

71.3 方法3：年周日期字段，group对象作为参数选项放在函数里。

```
data_65 %>%
    select(日期,`收盘价(元)`) %>%
    mutate(年_周 = format(日期,'%Y-%W')) %>%
    slice_max(`收盘价(元)`,by = 年_周)
```

```
## # A tibble: 74 × 3
##    日期                `收盘价(元)` 年_周
##    <dttm>                     <dbl> <chr>
##  1 2016-01-06 00:00:00         16.0 2016-01
##  2 2016-01-14 00:00:00         15.8 2016-02
##  3 2016-01-19 00:00:00         15.7 2016-03
##  4 2016-01-25 00:00:00         15.0 2016-04
##  5 2016-02-05 00:00:00         16.2 2016-05
##  6 2016-02-15 00:00:00         16.3 2016-07
##  7 2016-03-11 00:00:00         15.0 2016-10
```

71.4 方法4：年周日期字段,先group，再执行slice()函数。

```
  week_max <- data_65 %>%
    select(日期,`收盘价(元)`) %>%
  group_by(年_周 = format(日期,'%Y-%W')) %>%
    slice_max(`收盘价(元)`)
week_max
```

```
## # A tibble: 74 × 3
## # Groups:   年_周 [67]
##    日期                `收盘价(元)` 年_周
##    <dttm>                     <dbl> <chr>
##  1 2016-01-06 00:00:00         16.0 2016-01
##  2 2016-01-14 00:00:00         15.8 2016-02
##  3 2016-01-19 00:00:00         15.7 2016-03
##  4 2016-01-25 00:00:00         15.0 2016-04
##  5 2016-02-05 00:00:00         16.2 2016-05
##  6 2016-02-15 00:00:00         16.3 2016-07
```

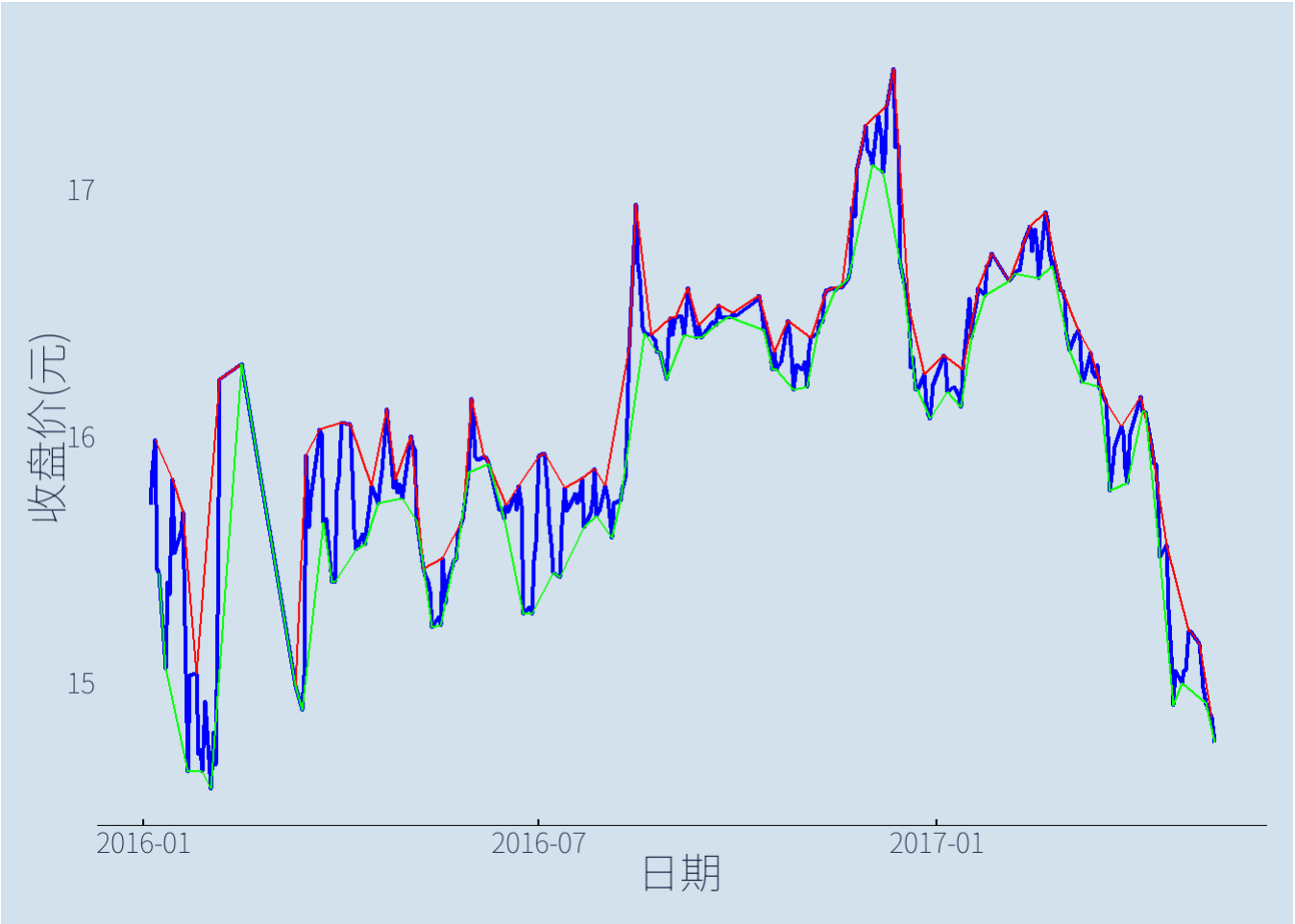
71.5 方法5：执行summarise，用max函数。

```
data_65 %>%
    select(日期,`收盘价(元)`) %>%
    mutate(年_周 = format(日期,'%Y-%W')) %>%
    summarise(week_max = max(`收盘价(元)`),
              .by = 年_周 )
```

```
## # A tibble: 67 × 2
##    年_周   week_max
##    <chr>      <dbl>
##  1 2016-01     16.0
##  2 2016-02     15.8
##  3 2016-03     15.7
##  4 2016-04     15.0
##  5 2016-05     16.2
##  6 2016-07     16.3
##  7 2016-10     15.0
```

72 绘制重采样数据与原始数据

```
ggplot()+
    geom_line(data =  data_65,
              aes(x = 日期,
                  y = `收盘价(元)`),
              color = "blue",
              linewidth = 0.8) +
    geom_line(data =  week_max,
              aes(x = 日期,
                  y = `收盘价(元)`),
              color = "red",
```



73 将数据往后移动5天

不能简单的对整个数据直接lag(5)，这样日期也同时往后移动了，没有意义。

数据长宽转变一样，确定要移动位置的数据列后，再执行lag()函数

本例中，前三列不用移动，后面全部各列均需移动。 这可以用mutate()函数结合across()函数实现

```
data_65 %>%
    mutate(across(-(1:3),
                  ~lag(.x,5)))
```

```
## # A tibble: 309 × 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… 2016-01-04 00:00:00           NA           NA           NA
##  2 600000.SH 浦发… 2016-01-05 00:00:00           NA           NA           NA
##  3 600000.SH 浦发… 2016-01-06 00:00:00           NA           NA           NA
##  4 600000.SH 浦发… 2016-01-07 00:00:00           NA           NA           NA
##  5 600000.SH 浦发… 2016-01-08 00:00:00           NA           NA           NA
##  6 600000.SH 浦发… 2016-01-11 00:00:00         16.1         16.1         16.1
##  7 600000.SH 浦发… 2016-01-12 00:00:00         15.7         15.5         16.0
```

74 将数据向前移动5天

与上题一样的逻辑，lag()函数变成lead函数即可。

```
data_65 %>%
    mutate(across(-(1:3),
                  ~lead(.x,5)))
```

```
## # A tibble: 309 × 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… 2016-01-04 00:00:00         15.4         15.2         15.4
##  2 600000.SH 浦发… 2016-01-05 00:00:00         15.1         15.2         15.5
##  3 600000.SH 浦发… 2016-01-06 00:00:00         15.4         15.5         15.8
##  4 600000.SH 浦发… 2016-01-07 00:00:00         15.4         15.0         15.9
##  5 600000.SH 浦发… 2016-01-08 00:00:00         15.8         15.7         16.0
##  6 600000.SH 浦发… 2016-01-11 00:00:00         15.5         15.4         15.9
##  7 600000.SH 浦发… 2016-01-12 00:00:00         15.6         15.7         15.8
```

75 使用expending函数计算开盘价的移动窗口均值

这个就是做累计移动平均计算，就是计算截至到本行以来的平均值。使用cummean()函数

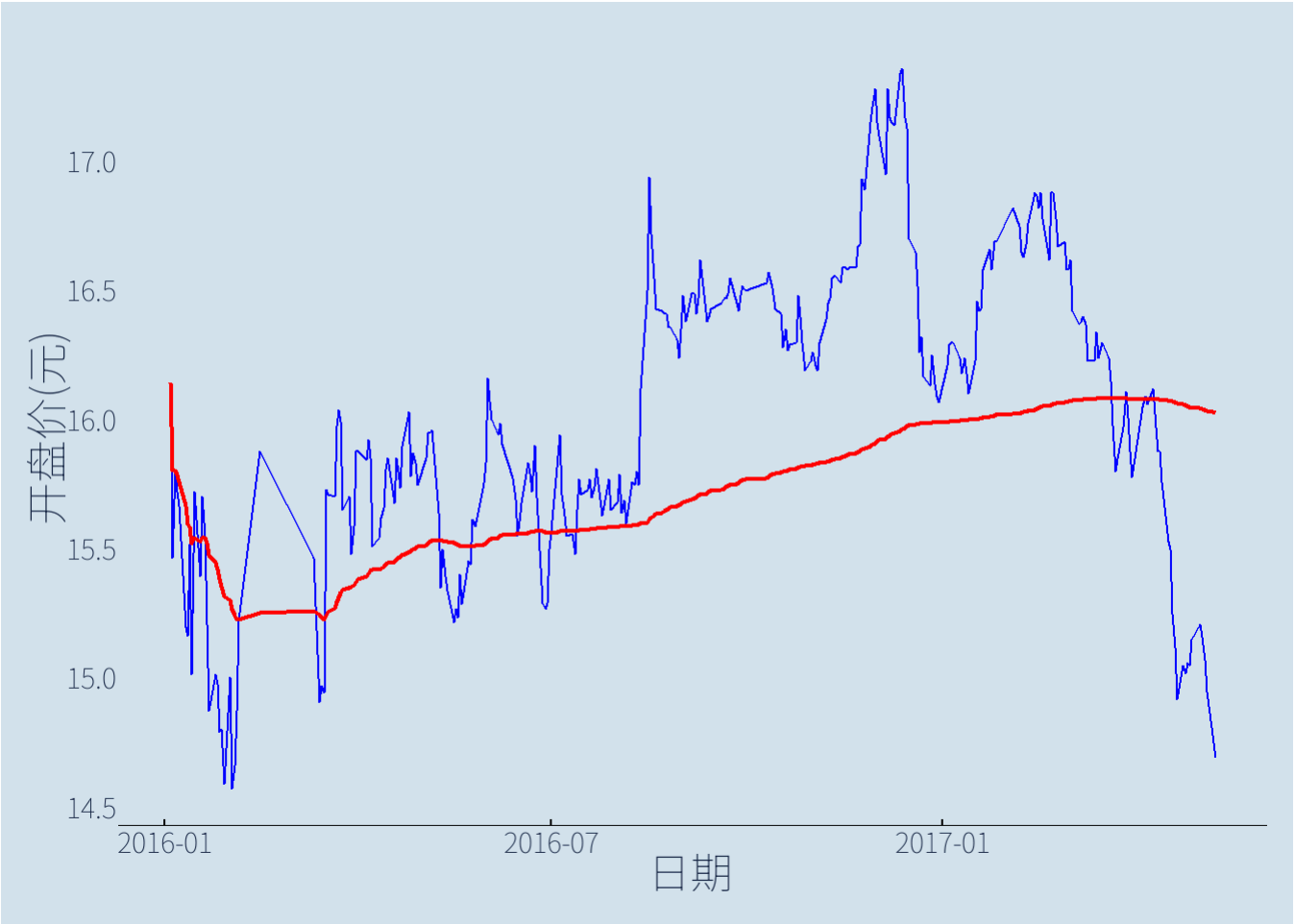
```
data_65 %>%
    mutate(mean_value =cummean(`开盘价(元)`),
           .after = `开盘价(元)` )
```

```
## # A tibble: 309 × 19
##    代码      简称     日期                `前收盘价(元)` `开盘价(元)` mean_value
##    <chr>     <chr>    <dttm>                       <dbl>        <dbl>      <dbl>
##  1 600000.SH 浦发银行 2016-01-04 00:00:00           16.1         16.1       16.1
##  2 600000.SH 浦发银行 2016-01-05 00:00:00           15.7         15.5       15.8
##  3 600000.SH 浦发银行 2016-01-06 00:00:00           15.9         15.8       15.8
##  4 600000.SH 浦发银行 2016-01-07 00:00:00           16.0         15.7       15.8
##  5 600000.SH 浦发银行 2016-01-08 00:00:00           15.5         15.7       15.8
##  6 600000.SH 浦发银行 2016-01-11 00:00:00           15.4         15.2       15.7
##  7 600000.SH 浦发银行 2016-01-12 00:00:00           15.1         15.2       15.6
```

76 绘制上一题的移动均值与原始数据折线图

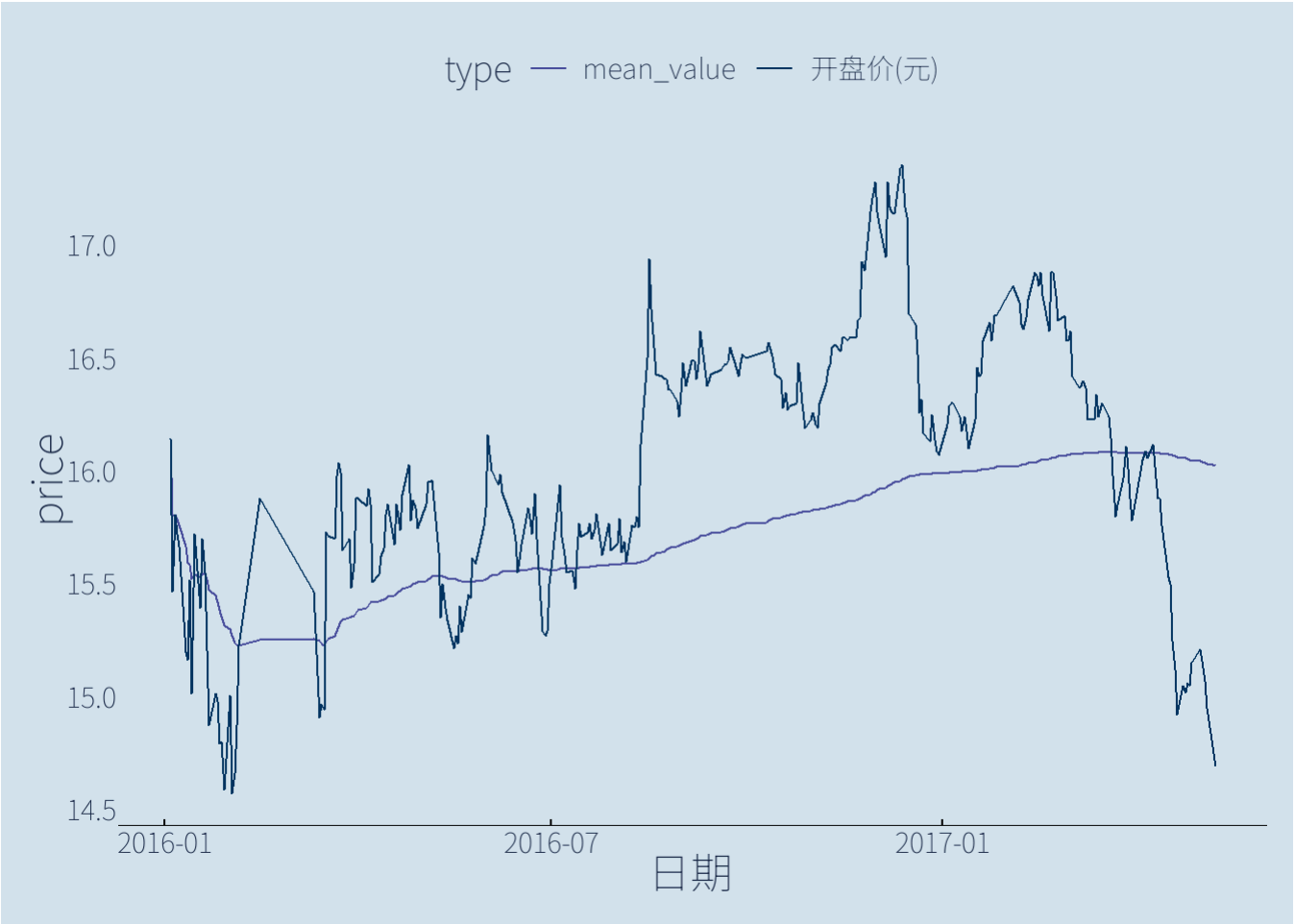
76.1 方法1：这种做法不可取，类似使用宽数据绘图的形式。

```
ggplot()+
    geom_line(data =  data_65,
              aes(x = 日期,
                  y = `开盘价(元)`),
              color = "blue",
              linewidth = 0.4) +

    geom_line(data =  data_65 %>%
    mutate(mean_value =cummean(`开盘价(元)`),
           .after = `开盘价(元)` ),
```


17.0
16.5
16.0
15.5
15.0
14.5
开盘价(元)
2016-01
2016-07
2017-01
日期

76.2 方法2：宽数据变长数据后再作图，合理的方法

```
data_65 %>%
    mutate(mean_value =cummean(`开盘价(元)`),
           .after = `开盘价(元)` ) %>%
    select(日期,
           `开盘价(元)`,
           mean_value) %>%
    pivot_longer(-1,
                 names_to = "type",
                 values_to = "price") %>%
    ggplot(aes(x = 日期,
```



77 计算布林指标

boll指标大意就是20日均值为中心线，然后上下两条线是这个中线加减一定sigma值的线。类似质量管理中SPC 管控里的上下控制限的逻辑。

77.1 方法1：自行计算

20日均线用前面自定义的函数；20日价格标准差自定义一个函数，如下

```
sd_fun <- function(data, width){
   i  = length(data)
   if(width > i){
       return("数据不够，请增加数据")
   }
   k =width:i
   sd_value =c(rep(NA,width-1),
               map_vec(k,\(x) sd(data[(x-(width-1)):x])))
return( sd_value)
 }
```

```
## # A tibble: 3 × 21
##   ma_20  b_up  b_bd 代码   简称  日期                `前收盘价(元)` `开盘价(元)`
##   <dbl> <dbl> <dbl> <chr>  <chr> <dttm>                       <dbl>        <dbl>
## 1  15.3  16.1  14.6 60000… 浦发… 2017-05-05 00:00:00          15.0         15.0
## 2  15.3  16.0  14.6 60000… 浦发… 2017-05-08 00:00:00          14.9         14.8
## 3  15.2  15.9  14.5 60000… 浦发… 2017-05-09 00:00:00          14.9         14.7
## # ℹ 13 more variables: `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(%)` <dbl>,
## #   `A股流通市值(元)` <dbl>, `总市值(元)` <dbl>, `A股流通股本(股)` <dbl>,
```

77.2 方法2：调包和函数计算:slider::slide_dbl()函数

和前面计算5日移动均值一样，可以选择各种包进行计算，比如slider包这个包。

```
data_65 %>%
    mutate(ma_20 = slider::slide_dbl( `收盘价(元)`,
                                      mean,
                                      .before = 19,
                                      .complete = TRUE),
           sd =slider::slide_dbl( `收盘价(元)`,
                                      sd,
                                      .before = 19,
                                      .complete = TRUE),
           b_up = ma_20 + 2*sd,
```

```
## # A tibble: 3 × 22
##   ma_20  b_up  b_bd 代码    简称   日期                `前收盘价(元)` `开盘价(元)`
##   <dbl> <dbl> <dbl> <chr>  <chr> <dttm>                       <dbl>        <dbl>
## 1  15.3  16.1  14.6 60000… 浦发… 2017-05-05 00:00:00           15.0         15.0
## 2  15.3  16.0  14.6 60000… 浦发… 2017-05-08 00:00:00           14.9         14.8
## 3  15.2  15.9  14.5 60000… 浦发… 2017-05-09 00:00:00           14.9         14.7
## # ℹ 14 more variables: `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(%)` <dbl>,
## #   `A股流通市值(元)` <dbl>, `总市值(元)` <dbl>, `A股流通股本(股)` <dbl>,
```

77.3 方法3：调包和函数计算:TTR::runMean()函数

搞和金融相关的事情，不用和金融相关的包，有点说不过去。

```
data_65 %>%
   mutate(ma_20 = TTR::runMean ( `收盘价(元)`, 20),
          b_up = ma_20 + 2*TTR::runSD(`收盘价(元)`,20),
          b_bd = ma_20 - 2*TTR::runSD(`收盘价(元)`,20))%>%
   select(ma_20,b_up,b_bd,everything()) %>%
   tail(3)
```

```
## # A tibble: 3 × 21
##   ma_20  b_up  b_bd 代码    简称   日期                `前收盘价(元)` `开盘价(元)`
##   <dbl> <dbl> <dbl> <chr>  <chr> <dttm>                       <dbl>        <dbl>
## 1  15.3  16.1  14.6 60000… 浦发… 2017-05-05 00:00:00           15.0         15.0
## 2  15.3  16.0  14.6 60000… 浦发… 2017-05-08 00:00:00           14.9         14.8
## 3  15.2  15.9  14.5 60000… 浦发… 2017-05-09 00:00:00           14.9         14.7
## # ℹ 13 more variables: `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(%)` <dbl>,
## #   `A股流通市值(元)` <dbl>, `总市值(元)` <dbl>, `A股流通股本(股)` <dbl>,
```

77.4 方法4：调包和函数计算:tidyquant:: tq_mutate()函数

使用tidyquant 包 tq_mutate()函数，与自编函数(把20天均线以及上下限的获取写在一个函数里，批量滚动获取即可)

```
getboll <- function(x,na.rm = TRUE){
   m  <- mean(x, na.rm = na.rm)
   s  <- sd(x, na.rm = na.rm)
   hi <- m + 2*s
   lo <- m - 2*s
   result <- c(ma_20 =m,
               b_up= hi,
               b_bd =lo)
   return(result)
}
```

```
## # A tibble: 3 × 21
##   ma_20  b_up  b_bd 代码    简称   日期                `前收盘价(元)` `开盘价(元)`
##   <dbl> <dbl> <dbl> <chr>  <chr> <dttm>                       <dbl>        <dbl>
## 1  15.3  16.1  14.6 60000… 浦发… 2017-05-05 00:00:00           15.0         15.0
## 2  15.3  16.0  14.6 60000… 浦发… 2017-05-08 00:00:00           14.9         14.8
## 3  15.2  15.9  14.5 60000… 浦发… 2017-05-09 00:00:00           14.9         14.7
## # ℹ 13 more variables: `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(%)` <dbl>,
## #   `A股流通市值(元)` <dbl>, `总市值(元)` <dbl>, `A股流通股本(股)` <dbl>,
```

78 计算布林线并绘制

```
 data_65 %>%
     tidyquant:: tq_mutate(select ="收盘价(元)",
                mutate_fun = rollapply,
                width = 20,
                align = "right",
                by.column = FALSE,
                FUN  =  getboll) %>%
   select(日期, ma_20,b_up,b_bd) %>%
   pivot_longer(-1,
                names_to = "type",
```

79 查看包的版本

map()可真是个好用的函数。咱就是说，一言不合就map。

```
c("tidyverse","zoo","DescTools") %>%
    map(packageVersion)
```

```
## [[1]]
## [1] '2.0.0'
##
## [[2]]
## [1] '1.8.12'
##
## [[3]]
## [1] '0.99.50'
```

80 从数组创建DataFrame

```
options(digits=1)
set.seed(1234)

tb1 <- sample(20:300,20) %>%
 array(dim = 20) %>%
 as_tibble( )

 head(tb1)
```

```
## # A tibble: 6 × 1
##   value
##   <int>
## 1   120
## 2   130
## 3   152
## 4   117
## 5   122
## 6   233
```

81 生成20个0-100固定步长的数

```
 tb2 <- seq(0,99,5) %>%
    as_tibble()
head(tb2)
```

```
## # A tibble: 6 × 1
##   value
##   <dbl>
## 1     0
## 2     5
## 3    10
## 4    15
## 5    20
## 6    25
```

84.生成20个指定分布(如标准正态分布)的数

```
tb3 <- rnorm(n = 20,mean = 0,sd = 1) %>%
    as_tibble()
head(tb3)
```

```
## # A tibble: 6 × 1
##    value
##    <dbl>
## 1  0.134
## 2 -0.491
## 3 -0.441
## 4  0.460
## 5 -0.694
## 6 -1.45
```

82 将df1，df2，df3按照行合并为新DataFrame

82.1 方法1:base R 包函数 rbind()

```
rbind(tb1,tb2,tb3) %>%
    head()
```

```
## # A tibble: 6 × 1
##   value
##   <dbl>
## 1   120
## 2   130
## 3   152
## 4   117
## 5   122
## 6   233
```

82.2 方法2:bind_rows()函数

```
bind_rows(tb1,tb2,tb3) %>%
    head()
```

```
## # A tibble: 6 × 1
##   value
##   <dbl>
## 1   120
## 2   130
## 3   152
## 4   117
## 5   122
## 6   233
```

83 将df1，df2，df3按照列合并为新DataFrame

83.1 方法1:base R 包函数 cbind()

```
df_combine <- cbind(tb1,tb2,tb3)
 head(df_combine)
```

```
##   value value value
## 1   120     0   0.1
## 2   130     5  -0.5
## 3   152    10  -0.4
## 4   117    15   0.5
## 5   122    20  -0.7
## 6   233    25  -1.4
```

83.2 方法2:bind_cols()函数

```
 bind_cols(tb1,tb2,tb3) %>%
    head()# tidyverse 系列包函数
```

```
## # A tibble: 6 × 3
##   value...1 value...2 value...3
##       <int>     <dbl>     <dbl>
## 1       120         0     0.134
## 2       130         5    -0.491
## 3       152        10    -0.441
## 4       117        15     0.460
## 5       122        20    -0.694
## 6       233        25    -1.45
```

84 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

使用summary()函数

```
#按列查看
df_combine %>%
    summary()
```

```
##      value          value         value       
##  Min.   : 23   Min.   : 0   Min.   :-2.2  
##  1st Qu.:106   1st Qu.:24   1st Qu.:-1.1  
##  Median :126   Median :48   Median :-0.5  
##  Mean   :144   Mean   :48   Mean   :-0.6  
##  3rd Qu.:196   3rd Qu.:71   3rd Qu.:-0.2  
##  Max.   :289   Max.   :95   Max.   : 1.1
```

```
# 查看全体数据
df_combine %>%
    unlist() %>%
    summary()
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##      -2       0      42      64     101     289
```

85 修改列名为col1,col2,col3

85.1 方法1：set_names()函数

```
df_88 <- set_names(df_combine,nm = str_c("col",1:3))
```

85.2 方法2：names()函数

```
names(df_combine) <- str_c("col",1:3)
df_combine %>%
    head(3)
```

```
##   col1 col2 col3
## 1  120    0  0.1
## 2  130    5 -0.5
## 3  152   10 -0.4
```

85.3 方法3：tibble()函数name_repair 参数

```
df_combine %>%
    as_tibble(.name_repair = ~ str_c("col",1:3))
```

```
## # A tibble: 20 × 3
##     col1  col2    col3
##    <int> <dbl>   <dbl>
##  1   120     0  0.134
##  2   130     5 -0.491
##  3   152    10 -0.441
##  4   117    15  0.460
##  5   122    20 -0.694
##  6   233    25 -1.45
##  7   109    30  0.575
```

86 提取第一列中不在第二列出现的数字

这题本质感觉是比较与匹配的逻辑。

86.1 方法1 生硬的理解字面意思，第一列中不在第二列出现的数字

```
df_combine$col1 [!df_combine$col1%in% df_combine$col2 ]
```

```
##  [1] 120 130 152 117 122 233 109  98 289 203  81  23 168  59 231 214 112 141 194
```

```
# 用简单示例理解
a = 1:10
b =5:15
# 显然，5到10在a和b里
a[!a%in%b]
```

```
## [1] 1 2 3 4
```

```
# 这里的结果就是1,2,3,4了
```

86.2 方法2：setdiff()函数就是直接一个函数

```
setdiff(a,b)
```

```
## [1] 1 2 3 4
```

```
setdiff(b,a)
```

```
## [1] 11 12 13 14 15
```

86.3 方法3：如果是数据框这种类型，还可以用anti_join()函数来实现

```
as_tibble(a) %>%
    anti_join(as_tibble(b))
```

```
## # A tibble: 4 × 1
##   value
##   <int>
## 1     1
## 2     2
## 3     3
## 4     4
```

```
tb1 %>%
    anti_join(tb2)
```

```
## # A tibble: 19 × 1
##    value
##    <int>
##  1   120
##  2   130
##  3   152
##  4   117
##  5   122
##  6   233
##  7   109
```

87 提取第一列和第二列出现频率最高的三个数字

```
df_combine %>%
    select(col1,col2) %>%
    unlist() %>%
    as_tibble() %>%
   summarise(n =n(),
             .by = value) %>%
    slice_max(n,n =3,
              with_ties = FALSE) # 并列最多的数字太多，这里选择with_ties = false, 就只出现3个，否则会全部显示。
```

```
## # A tibble: 3 × 2
##   value     n
##   <dbl> <int>
## 1    85     2
## 2   120     1
## 3   130     1
```

88 提取第一列中可以整除5的数字位置

88.1 方法1： which()大法

```
which(df_combine$col1 %% 5 == 0)
```

```
## [1]  1  2 19
```

88.2 方法2： filter()函数

```
 df_combine %>%
     rowid_to_column() %>%
     filter(col1 %%5 ==0)  %>%
     select(rowid)
```

```
##   rowid
## 1     1
## 2     2
## 3    19
```

89 计算第一列数字前一个与后一个的差值

```
df_combine %>%
    mutate(diff_1 =c(NA,
                     diff(col1,1,1)),
           .after = col1)
```

```
##    col1 diff_1 col2  col3
## 1   120     NA    0  0.13
## 2   130     10    5 -0.49
## 3   152     22   10 -0.44
## 4   117    -35   15  0.46
## 5   122      5   20 -0.69
## 6   233    111   25 -1.45
## 7   109   -124   30  0.57
## 8    98    -11   35 -1.02
## 9   289    191   40 -0.02
```

90 将col1,col2,col3三列顺序颠倒

90.1 方法1：select()函数

与列处理相关，第一时间想到select()函数

```
 df_combine %>%
    select(str_c("col",3:1))
```

```
##     col3 col2 col1
## 1   0.13    0  120
## 2  -0.49    5  130
## 3  -0.44   10  152
## 4   0.46   15  117
## 5  -0.69   20  122
## 6  -1.45   25  233
## 7   0.57   30  109
## 8  -1.02   35   98
## 9  -0.02   40  289
```

91 方法2：relocate()函数

涉及到列位置，顺序，可以考虑使用relocate()函数

```
df_combine %>%
    relocate(col2,.before = col1) %>%
    relocate(col3,.before = col2)
```

```
##     col3 col2 col1
## 1   0.13    0  120
## 2  -0.49    5  130
## 3  -0.44   10  152
## 4   0.46   15  117
## 5  -0.69   20  122
## 6  -1.45   25  233
## 7   0.57   30  109
## 8  -1.02   35   98
## 9  -0.02   40  289
```

92 提取第一列位置在1,10,15的数字

这是列和行组合应用，列：select,行：filter,slice这些

92.1 方法1：select() + slice()

```
# 方法1：slice()
df_combine %>%
    select(col1) %>%
    slice(1,10,15)
```

```
##   col1
## 1  120
## 2  203
## 3  231
```

92.2 方法2：rowid_to_column()+filter()+ select()

```
 df_combine %>%
    rowid_to_column() %>%
     filter(rowid %in% c(1,10,15))  %>%
     select(col1)
```

```
##   col1
## 1  120
## 2  203
## 3  231
```

92.3 方法3：filter()+row_number()+ select()

```
df_combine %>%
    filter(row_number()%in% c(1,10,15)) %>%
    select(col1)
```

```
##   col1
## 1  120
## 2  203
## 3  231
```

93 找第一列的局部最大值位置,即比它前一个与后一个数字的都大的数字

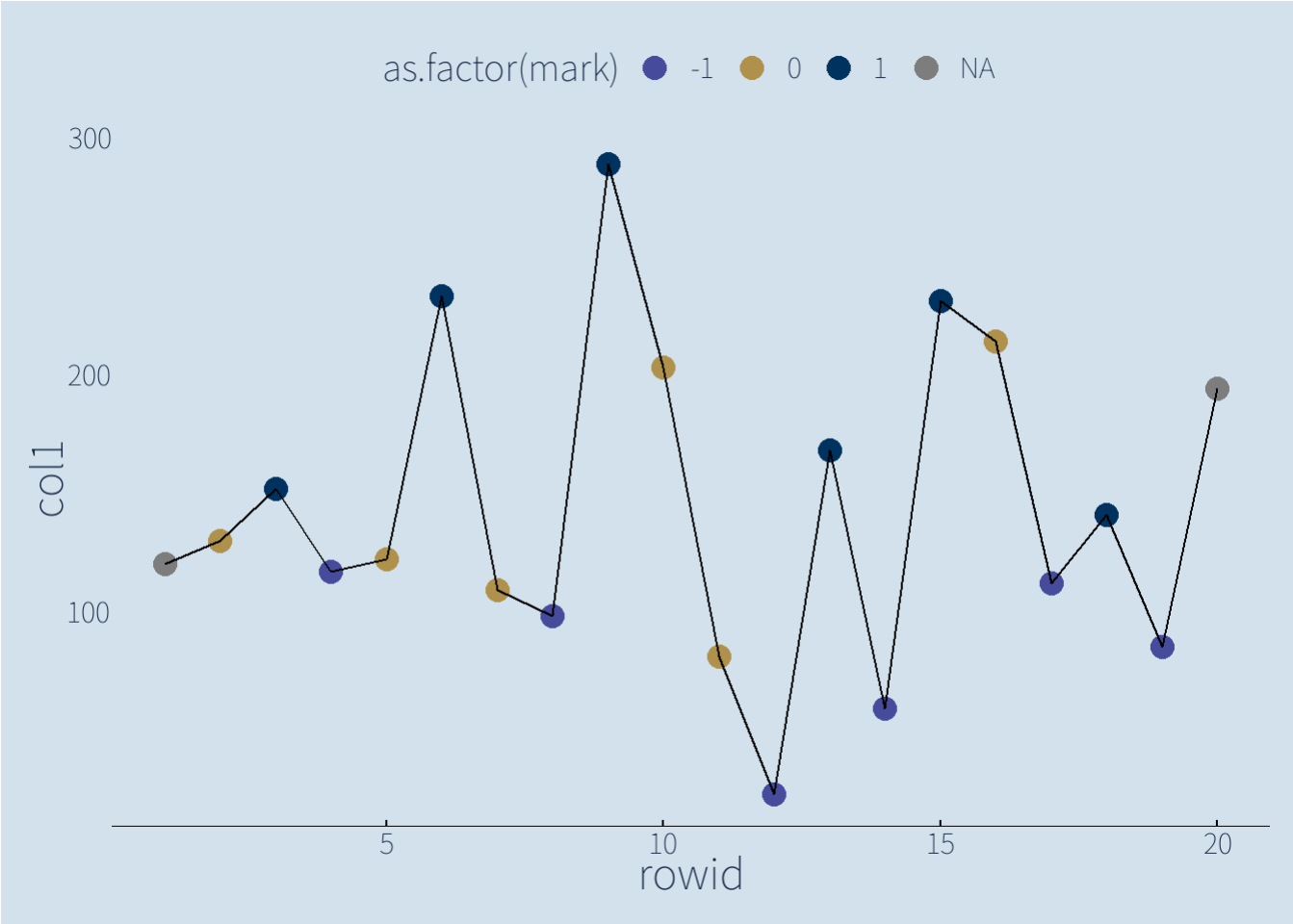
lag()和lead()函数解决这个问题

```
df_combine %>%
    mutate(max_mark = if_else((col1-lag(col1))>0,
                              if_else((col1-lead(col1))>0,
                                      1,
                                      0),
                                     0)) %>%
    filter(max_mark ==1)
```

```
##   col1 col2  col3 max_mark
## 1  152   10 -0.44        1
## 2  233   25 -1.45        1
## 3  289   40 -0.02        1
## 4  168   60 -0.71        1
## 5  231   70 -1.63        1
## 6  141   85 -1.34        1
```

- 同样的，肯定就有局部最小值,求最小值并把他们同时在一张图上显示出来

```
lag_val <- df_combine$col1-lag(df_combine$col1)
lead_val <- df_combine$col1-lead(df_combine$col1)

df_combine %>%
    mutate(mark =  if_else(lead_val*lag_val <= 0,
                           0,
                          if_else(lag_val > 0,
                                  1,
                                  -1)
                           )
```



94 按行计算df的每一行均值

使用rowwise函数

```
df_combine %>%
    rowwise() %>%
    mutate(row_mean = mean(c_across(col1:col3)
                           )
           )
```

```
## # A tibble: 20 × 4
## # Rowwise:
##     col1  col2    col3 row_mean
##    <int> <dbl>   <dbl>    <dbl>
##  1   120     0  0.134      40.0
##  2   130     5 -0.491      44.8
##  3   152    10 -0.441      53.9
##  4   117    15  0.460      44.2
##  5   122    20 -0.694      47.1
##  6   233    25 -1.45       85.5
```

```
# 或者
df_combine %>%
    rowwise() %>%
    mutate(row_mean = mean(c(col1,col2,col3)
                           )
    )
```

```
## # A tibble: 20 × 4
## # Rowwise:
##     col1  col2    col3 row_mean
##    <int> <dbl>   <dbl>    <dbl>
##  1   120     0  0.134      40.0
##  2   130     5 -0.491      44.8
##  3   152    10 -0.441      53.9
##  4   117    15  0.460      44.2
##  5   122    20 -0.694      47.1
##  6   233    25 -1.45       85.5
```

95 对第二列计算移动平均值,假设步长为4

前面的题目中已出现过，且用多种方法进行了求解。

```
df_combine %>%
   mutate(average_4 = zoo::rollmean(col2,
                                    4,
                                    align = "right",
                                    fill = NA))
```

```
##    col1 col2  col3 average_4
## 1   120    0  0.13        NA
## 2   130    5 -0.49        NA
## 3   152   10 -0.44        NA
## 4   117   15  0.46         8
## 5   122   20 -0.69        12
## 6   233   25 -1.45        18
## 7   109   30  0.57        22
## 8    98   35 -1.02        28
## 9   289   40 -0.02        32
```

96 将数据按照第三列值的大小升序排列

```
df_combine %>%
arrange(col3)
```

```
##    col1 col2  col3
## 1   112   80 -2.18
## 2   231   70 -1.63
## 3   233   25 -1.45
## 4   141   85 -1.34
## 5   214   75 -1.17
## 6    98   35 -1.02
## 7   203   45 -0.94
## 8   168   60 -0.71
## 9   122   20 -0.69
```

97 将第一列大于50的数字修改为'高'

```
df_combine %>%
    mutate(col =ifelse(col1 > 150,
                       "高",
                       col1)) # 改为大于150的
```

```
##    col1 col2  col3 col
## 1   120    0  0.13 120
## 2   130    5 -0.49 130
## 3   152   10 -0.44  高
## 4   117   15  0.46 117
## 5   122   20 -0.69 122
## 6   233   25 -1.45  高
## 7   109   30  0.57 109
## 8    98   35 -1.02  98
## 9   289   40 -0.02  高
```

还有很多种方法，比较简单的内容，不多重复。

补充一点，tidyverse系列的if_else()函数不支持类似写法，它严格要求不同情形下的内容同样的字符类型。

98 计算第一列与第二列之间的欧式距离

```
df_combine %>%
    select(col1,col2) %>%
    # as.matrix() %>%
    t() %>%
stats::dist(method = "euclidean")
```

```
##      col1
## col2  533
```

99 从CSV文件中读取指定数据:从数据1中的前10行中读取positionName, salary两列

100 方法1 readr::read_csv()函数

```
df_1 <- readr::read_csv("./data/数据1.csv",
         col_select = c( positionName , salary),
         n_max = 10,
         locale = locale(encoding = "gbk"),
         show_col_types = FALSE)
```

101 方法2 base R read.csv()函数

```
read.csv("./data/数据1.csv",
         nrows = 10,
         header = TRUE,
         fileEncoding = "gbk") %>%
    select(c(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
```

补充：read.csv(),read.csv2(),read_csv(),read_csv2(),这几个常用函数看上去简单，其实，里边的门道还是挺多的；值得花时间好好琢磨。

102 从数据2中读取数据并在读取数据时将薪资大于10000的改为高

这个分类题目在前面的题里有出现过

```
df2 <- read.csv("./data/数据2.csv",
                header = T,
                encoding = "utf-8" ) %>%
    mutate(薪资水平 = if_else(薪资水平 > 10000,
                          "高",
                          "低"))

head(df2)
```

```
##   学历要求 薪资水平
## 1     本科       高
## 2     硕士       高
## 3     本科       低
## 4     本科       高
## 5     不限       高
## 6     硕士       高
```

103 从上一题数据中，对薪资水平列每隔20行进行一次抽样

可以用filter 或者slice 函数执行。我的理解是抽取第20，40，60…行数据，直到最后。

或者抽取第1，21，31...，行? 其实都不重要，关键是实现的方法...

103.1 方法1：slice() +n()

```
df2 %>%
    select(薪资水平) %>%
    slice(seq(1,n(),20)) # 这里抽取第一行，21行，31行...
```

103.2 方法2：rowid_to_column() + filter ()

```
 df2 %>%
    rowid_to_column() %>%
     filter ( rowid %%20 ==0) %>%
     select(3)
```

103.3 方法2：row_number() + filter ()

```
  df2 %>%
     filter(row_number(薪资水平)%%20 ==0) %>%
     select(2)
```

103.4 方法3：order() + filter ()

```
df2 %>%
    filter(order(薪资水平)%%20 ==0)
```

104 将数据取消使用科学计数法

我经常用scales包里的函数来进行数据形式处理，虽然它更多用在绘图的时候。前面有一道题目，用函数确定了百分数格式(第68题) `scales::percent(accuracy = 0.01), .before =` 收盘价(元)``

科学计数法的定义是一个数字表达为一个绝对值在1-10之间的数a(如-3.23,6,4.87等)与10的n(整数)次幂相乘的形式。如3000可以表达为3*10^3

```
# format()函数

# 先生成一组科学计数法表示的数据。
 set.seed(1234)
sci_num <- runif(10,0.000001,100000) %>%
    format(scientific = TRUE,digits = 4)
sci_num # 科学计数法
```

```
##  [1] "1.137e+04" "6.223e+04" "6.093e+04" "6.234e+04" "8.609e+04" "6.403e+04"
##  [7] "9.496e+02" "2.326e+04" "6.661e+04" "5.143e+04"
```

```
as.numeric(sci_num) # 非科学计数法
```

```
##  [1] 11370 62230 60930 62340 86090 64030   950 23260 66610 51430
```

```
# 再用scales 包的相关函数鼓捣一下,以初始导入的df_1 数据为例

df_1 %>%
    mutate(salary_sci = scales:: scientific(salary,
                                  digits = 3),
           salary_num = scales:: number(salary,
                                  digits = 3,decimal.mark = ",",big.mark = "."),
           salary_pct = scales:: percent(salary,
                                  digits = 2,big.mark = ","))
```

```
## # A tibble: 10 × 5
##    positionName   salary salary_sci salary_num salary_pct
##    <chr>           <dbl> <chr>      <chr>      <chr>
##  1 数据分析        37500 4e+04      37.500     3,750,000%
##  2 数据建模        15000 2e+04      15.000     1,500,000%
##  3 数据分析         3500 3e+03      3.500      350,000%
##  4 数据分析        45000 4e+04      45.000     4,500,000%
##  5 数据分析        30000 3e+04      30.000     3,000,000%
##  6 数据分析        50000 5e+04      50.000     5,000,000%
##  7 数据分析        30000 3e+04      30.000     3,000,000%
```

105 将上一题的数据转换为百分数

已在上题中进行抢答。

本题另行记录sprintf()函数的用法，该函数来自C库,其内容比较丰富，help文件提供了大量用法和示例。

```
# 上题已实现。

num_sprintf <-  runif(10,0,0.4)

num_sprintf
```

```
##  [1] 0.28 0.22 0.11 0.37 0.12 0.33 0.11 0.11 0.07 0.09
```

```
sprintf("%.3f%%", num_sprintf*100)
```

```
##  [1] "27.744%" "21.799%" "11.309%" "36.937%" "11.693%" "33.492%" "11.449%"
##  [8] "10.673%" "7.469%"  "9.289%"
```

106 查找上一题数据中第3大值的行号

这里可以引申出一组经常容易混淆的函数：rank,order 和sort函数。

sort 是动作，就是将一组数据进行排序，其输出是排完序的数组。比如1，3，5，4，排序完会是1，3，4，5...

order 是结果，就是排完了在第几位(纯粹的物理位置)。比如上面排完后，数字4的order由4变成了3；

rank 是大小排名；

一句话，我们根据rank 来进行sort，最后决定每个元素的order.

当自己在给领导们拍照，首先要根据他们的rank，也就是职位高低来确定他们的order,整个安排他们位置的过程就是sort，也就是最终照片里的排位；然后他们的职位高低就是rank，所在的位置，比如C位…我们看很多时候领导出场也是有这个逻辑来着…

回到这个例子，第三大值，这个是rank的概念；行号是order的概念，我用到的这个arrange()动作函数，其实就是sort的概念。

106.1 方法1：rowid_to_column()+arrange()+slice

```
df_1 %>%
    rowid_to_column() %>%
    arrange(-salary) %>%
    slice(3) %>%
    select(rowid)
```

```
## # A tibble: 1 × 1
##   rowid
##   <int>
## 1     4
```

106.2 方法2：order()+rank()

```
# 这道题也可以用下面这个函数组合求解。

 order(rank(-df_1$salary))[3] #第3大的数字的行号，结果为4
```

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

```
# 再验证一下，#最小值的行号，结果为3
 order(rank(df_1$salary))[1]
```

```
## [1] 3
```

```
# 当然，不用rank()函数，结果也是一样的
 order(df_1$salary)[1]
```

```
## [1] 3
```

107 反转df的行

这个也可以理解为order顺序调整，行号最大的排在最前面。

也可以理解为重新选择行内容到固定的位置。

107.1 方法1: 对行号逆排序

```
df_1 %>%
    rowid_to_column() %>%
    arrange(-rowid) # 这样就实现了反转
```

```
## # A tibble: 10 × 3
##    rowid positionName   salary
##    <int> <chr>           <dbl>
##  1    10 数据分析师      40000
##  2     9 数据分析专家    60000
##  3     8 数据建模工程师  35000
##  4     7 数据分析        30000
##  5     6 数据分析        50000
##  6     5 数据分析        30000
##  7     4 数据分析        45000
```

107.2 方法2: 由后往前slice()

```
 df_1 %>%
     slice(n():1)# 这样也实现了反转
```

```
## # A tibble: 10 × 2
##    positionName   salary
##    <chr>           <dbl>
##  1 数据分析师      40000
##  2 数据分析专家    60000
##  3 数据建模工程师  35000
##  4 数据分析        30000
##  5 数据分析        50000
##  6 数据分析        30000
##  7 数据分析        45000
```

107.3 方法3: 由后往前取子集

```
df_1[nrow(df_1):1,]# 这样也是可以的
```

```
## # A tibble: 10 × 2
##    positionName   salary
##    <chr>           <dbl>
##  1 数据分析师      40000
##  2 数据分析专家    60000
##  3 数据建模工程师  35000
##  4 数据分析        30000
##  5 数据分析        50000
##  6 数据分析        30000
##  7 数据分析        45000
```

108 按照多列对数据进行合并

python 原题的输入数据

df1= pd.DataFrame({'key1': ['K0', 'K0', 'K1', 'K2'], 'key2': ['K0', 'K1', 'K0', 'K1'], 'A': ['A0', 'A1', 'A2', 'A3'], 'B': ['B0', 'B1', 'B2', 'B3']})

df2= pd.DataFrame({'key1': ['K0', 'K1', 'K1', 'K2'], 'key2': ['K0', 'K0', 'K0', 'K0'], 'C': ['C0', 'C1', 'C2', 'C3'], 'D': ['D0', 'D1', 'D2', 'D3']})

题目标题用的合并，理解下来应该是连接，就是join的意思,按照多列合并，我们可以用full_join()函数：相同的留一个，不同的都留下；

简单整理以下*_join()函数：

mutate join()系列，按规则添加y中符合条件的列到x中；内容包括：

```
inner join() 和

outer join(): out_join() 又分left_join(), right_join() 以及full_join() 三种类型。
```

filter join()系列，如字面意思，主要执行匹配筛选；包括：

```
anti_join() 返回在x但不在y中的内容

semi_join() 返回在x也在y中的内容

前面有道题用到了anti_join()函数；
```

cross_join() x,y 交叉匹配，返回nrow(x)*nrow(y)行数据

nest join() x 几乎保持不变，只是它添加了一个新的列表列，其中每个元素包含 y 中与 x 中相应行匹配的行。

同样用到join,但其用途和场景还是有区别的。

```
# 先按题意生成R语言dataframe

df_108_1 <- tibble(key1 = str_c("K",c(0,0,1,2)),
                   key2 = str_c("K",c(0,1,0,1)),
                   A = str_c("A",0:3),
                   B = str_c("B",0:3))

df_108_2 <- tibble(key1 = str_c("K",c(0,1,1,2)),
                   key2 = str_c("K",c(0,0,0,0)),
                   C = str_c("C",0:3),
```

```
## # A tibble: 6 × 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
```

109 按照多列对数据进行合并,只保存df1的数据

```
df_108_1 %>%
    left_join(df_108_2,
              by = c("key1","key2"))
```

```
## # A tibble: 5 × 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>
```

110 再次读取数据1并显示所有的列

```
 df_110 <- readr::read_csv("./data/数据1.csv",
          # col_select = c( positionName , salary),
          # n_max = 10,
          locale = locale(encoding = "gbk"),
          show_col_types = FALSE)

df_110 %>%
    str()
```

111 查找secondType与thirdType值相等的行号

111.1 方法1：rowid_to_column() + filter() +select()

```
df_110 %>%
    rowid_to_column() %>%
    filter(secondType == thirdType) %>%
    select(rowid) %>%
    unlist()
```

```
##  rowid1  rowid2  rowid3  rowid4  rowid5  rowid6  rowid7  rowid8  rowid9 rowid10
##       1       3       5       6       7      11      15      24      26      28
## rowid11 rowid12 rowid13 rowid14 rowid15 rowid16 rowid17 rowid18 rowid19 rowid20
##      29      30      31      34      38      39      40      41      42      49
## rowid21 rowid22 rowid23 rowid24 rowid25 rowid26 rowid27 rowid28 rowid29 rowid30
##      50      53      54      56      58      62      66      67      68      72
## rowid31 rowid32 rowid33 rowid34 rowid35 rowid36 rowid37 rowid38 rowid39 rowid40
##      74      75      76      80      81      83      86      89      90      92
## rowid41 rowid42
##      97     101
```

111.2 方法2：# which(),查找

```
which(df_110$secondType == df_110$thirdType)
```

```
##  [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
```

112 查找薪资大于平均薪资的第三个数据

```
df_110 %>%
        filter(salary > mean(salary)) %>%
    slice(3)
```

```
## # A tibble: 1 × 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,…
## # ℹ 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>,
```

113 将上一题数据的salary列开根号

```
df_110 %>%
    mutate(sqrt_salary = sqrt(salary),
           .after = positionId)
```

```
## # A tibble: 105 × 54
##    positionId sqrt_salary positionName   companyId companyLogo       companySize
##         <dbl>       <dbl> <chr>              <dbl> <chr>             <chr>
##  1    6802721       194.  数据分析          475770 i/image2/M01/B7/… 50-150人
##  2    5204912       122.  数据建模           50735 image1/M00/00/85… 150-500人
##  3    6877668        59.2 数据分析          100125 image2/M00/0C/57… 2000人以上
##  4    6496141       212.  数据分析           26564 i/image2/M01/F7/… 500-2000人
##  5    6467417       173.  数据分析           29211 i/image2/M01/77/… 2000人以上
##  6    6882347       224.  数据分析           94826 image2/M00/04/12… 50-150人
##  7    6841659       173.  数据分析          348784 i/image2/M01/AC/… 50-150人
```

114 将上一题数据的linestaion列按_拆分

这列数据比较乱，需要识别整理；

基础逻辑：

1. 明确NA 值的处理，肯定是不动的，保留即可，也就是默认操作；
2. 对每个单元格里多个linestation 信息进行处理，也就是一行数据变成多行数据；使用tidyr 包

separate_*系列函数执行操作；

3. 使用seperate_* 系列函数将该列按_拆分。这系列函数主要有：

separate_wider_*()* ::delim,regex

separate_longer_*()* ::delim,regex,position.

这系列函数 根据字面意思就可以理解，wider系列就是变宽，类似一列变两列；然后可以根据符号和正则表达式来执行拆分；longer系列就是变长，一行变多行，可以根据位置，符号，正则表达式等来进行拆分。

```
## 本例中，两个系列都需要用到。

df_114 <- df_110 %>%
    separate_longer_delim(cols = linestaion,delim = ";") %>%
    select(linestaion,everything()) %>%  # 把station放在第一列，可以看到列数已经增多，每行最多一项line 与station 的组合
   separate_wider_delim(cols = linestaion,
                        delim = "_",
                        names = c("line","station"),
                        cols_remove = FALSE)
```

```
## # A tibble: 10 × 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 城星路
##  4 4号线_市民中心 4号线 市民中心
##  5 4号线_江锦路   4号线 江锦路
##  6 1号线_文泽路   1号线 文泽路
##  7 <NA>           <NA>  <NA>
```

115 查看上一题数据中一共有多少列

```
df_114 %>%
    ncol()
```

```
## [1] 55
```

116 提取industryField列以'数据'开头的行

116.1 方法1：典型的文本提取类操作，适用正则表达式

```
df_114 %>%
    filter(str_detect(industryField, "^数据"))
```

116.2 扩展，本例只有1列，如果要提取多行中，以“数据”开头的行:

116.2.1 再加上positionName；提取这两列里同时以“数据”开头的行；

```
df_114 %>%
    filter(across(c("industryField",
                    "positionName"),
                  ~str_detect(.x, "^数据")))
```

```
df_114 %>%
    filter(if_all(c("industryField",
                    "positionName"),
                  ~str_detect(.x, "^数据")))
```

116.2.2 提取”industryField”,“positionName”这两列里只要一列以“数据”开头的行

```
df_114 %>%
    filter(if_any(c("industryField",
                    "positionName"),
                  ~str_detect(.x, "^数据")))
```

116.2.3 提取整个数据表只要一列以“数据”开头的行；

```
df_114 %>%
    filter(if_any(everything(),
                  ~str_detect(.x, "^数据")))
```

117 以salary score 和 positionID制作数据透视

117.1 方法1：group()+summarise()

数据透视，能做的事情比较多，通常的求和，求均值，计数等,和excel表里类似。

```
df_114 %>%
    group_by(positionId) %>%
    summarise(n = n(),
              salary_avg = mean(salary),
              salary_total = sum(salary),
              score_avg = mean(score),
              score_total = sum(score))
```

```
## # A tibble: 95 × 6
##    positionId     n salary_avg salary_total score_avg score_total
##         <dbl> <int>      <dbl>        <dbl>     <dbl>       <dbl>
##  1    5203054     1      30000        30000       4             4
##  2    5204912     1      15000        15000     176           176
##  3    5269002     2      37500        75000       1             2
##  4    5453691     1      30000        30000       4             4
##  5    5519962     1      37500        37500      14            14
##  6    5520623     1      30000        30000       6             6
##  7    5559894     1      30000        30000      12            12
```

117.2 方法2： summarise() + 参数中进行分组

```
# 方法2

df_114 %>%
     summarise(n = n(),
               salary_avg = mean(salary),
               salary_total = sum(salary),
               score_avg = mean(score),
               score_total = sum(score),
               .by = positionId)
```

```
## # A tibble: 95 × 6
##    positionId     n salary_avg salary_total score_avg score_total
##         <dbl> <int>      <dbl>        <dbl>     <dbl>       <dbl>
##  1    6802721     1      37500        37500       233         233
##  2    5204912     1      15000        15000       176         176
##  3    6877668     3       3500        10500        80         240
##  4    6496141     1      45000        45000        68          68
##  5    6467417     1      30000        30000        66          66
##  6    6882347     1      50000        50000        66          66
##  7    6841659     1      30000        30000        65          65
```

118 同时对salary、score两列进行计算

```
df_114 %>%
    summarise(across(c(salary,score),
                     list(mean = mean,
                          max = max,
                          min = min),
                     .names = "{.col}_{.fn}"))
```

```
## # A tibble: 1 × 6
##   salary_mean salary_max salary_min score_mean score_max score_min
##         <dbl>      <dbl>      <dbl>      <dbl>     <dbl>     <dbl>
## 1      31712.      60000       3500       10.2       233         0
```

119 对不同列执行不同的计算：对salary求平均，对score列求和

```
df_114 %>%
    summarise(salary_avg = mean(salary),
              score_sum = sum(score))
```

```
## # A tibble: 1 × 2
##   salary_avg score_sum
##        <dbl>     <dbl>
## 1     31712.      1905
```

120 计算并提取平均薪资最高的区

```
df_110 %>%
    summarise(salary_avg = mean(salary),
              .by = district) %>%
    slice_max(salary_avg)
```

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