talk08 练习与作业

目录

0.1 练习和作业说明

将相关代码填写入以 "'{r} "' 标志的代码框中，运行并看到正确的结果；

完成后，用工具栏里的"Knit" 按键生成 PDF 文档；

将 PDF 文档改为：姓名-学号-`talk08` 作业`.pdf`，并提交到老师指定的平台/钉群。

0.2 talk08 内容回顾

- for loop
- `apply` functions
- `dplyr` 的本质是遍历
- `map` functions in `purrr` package
- 遍历与并行计算

0.3 练习与作业：用户验证

请运行以下命令，验证你的用户名。

如你当前用户名不能体现你的真实姓名，请改为拼音后再运行本作业！

```
Sys.info()[["user"]]
```

```
## [1] "s56hh"
```

```
Sys.getenv("HOME")
```

```
## [1] "C:/Users/s56hh/Documents"
```

0.4 练习与作业 1：loop 初步

0.4.1 `loop` 练习（部分内容来自 r-exercises.com 网站）

1. 写一个循环，计算从 1 到 7 的平方并打印 `print`；
2. 取 `iris` 的列名，计算每个列名的长度，并打印为下面的格式：`Sepal.Length (12)`；
3. 写一个 `while` 循环，每次用 `rnorm` 取一个随机数字并打印，直到取到的数字大于 1；
4. 写一个循环，计算 Fibonacci 序列的值超过 1 百万所需的循环数；注：Fibonacci 序列的规则为：`0, 1, 1, 2, 3, 5, 8, 13, 21 ...`；

```
## 代码写这里，并运行；
for(i in 1:7){print(i*i)}
```

```
## [1] 1
## [1] 4
## [1] 9
```

```
## [1] 16
## [1] 25
## [1] 36
## [1] 49
```

```
for (n in names(iris)) {print(paste(n, " (", nchar(n), ")"))}
```

```
## [1] "Sepal.Length  ( 12 )"
## [1] "Sepal.Width  ( 11 )"
## [1] "Petal.Length  ( 12 )"
## [1] "Petal.Width  ( 11 )"
## [1] "Species  ( 7 )"
```

```
while (1) {
  x=rnorm(1)
  print(x)
  if(x>1)break
}
```

```
## [1] 0.2166882
## [1] 0.383077
## [1] 0.6668552
## [1] 0.760904
## [1] -0.08448698
## [1] -0.8235373
## [1] -1.910275
## [1] 0.9327251
## [1] -0.4205042
## [1] 0.6258796
## [1] 0.5675212
## [1] -0.4503098
## [1] 0.1145073
## [1] -0.04311397
## [1] 0.2727717
```

```
## [1] 1.008307
```

```
x=0
y=1
i=0
while(1){
  if(y>1000000){
    print(i)
    break
  }
  i=i+1
  c=y
  y=x+y
  x=c
}
```

```
## [1] 30
```

0.5 练习与作业 2：loop 进阶，系统和其它函数

0.5.1 生成一个数字 matrix，并做练习

生成一个 100 x 100 的数字 matrix：

1. 行、列平均，用 rowMeans, colMeans 函数；
2. 行、列平均，用 apply 函数
3. 行、列总和，用 rowSums, colSums 函数；
4. 行、列总和，用 apply 函数
5. 使用自定义函数，同时计算：
    - 行平均、总和、sd
    - 列平均、总和、sd

```r
## 代码写这里，并运行；
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.2 --
## v ggplot2 3.3.6      v purrr   0.3.4
## v tibble  3.1.8      v dplyr   1.0.10
## v tidyr   1.2.1      v stringr 1.4.1
## v readr   2.1.2      v forcats 0.5.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
df<-matrix(rnorm(100*100, 2, 4), nrow = 100)

rowMeans( df )
```

```
##   [1] 2.237211 2.561320 2.251888 2.683165 1.659763 1.737900 2.156323 1.914529
##   [9] 2.255690 1.981960 1.551305 2.003171 1.983989 2.149901 2.206424 2.316499
##  [17] 2.439716 2.023911 1.831881 1.590739 2.398962 1.240311 2.670237 1.761688
##  [25] 2.359972 1.505164 2.960133 1.896109 1.991247 1.924412 1.923512 1.903865
##  [33] 1.750435 3.338297 1.951837 2.580570 2.292812 1.950270 1.724061 2.341317
##  [41] 1.312544 2.168936 1.403851 1.955030 2.474856 1.754731 1.760444 1.886532
##  [49] 1.719791 1.398076 2.634272 1.995588 2.289086 1.950851 2.193313 1.861967
##  [57] 2.023289 1.573940 1.318893 2.572031 1.724108 1.952620 2.588182 2.266717
##  [65] 2.688869 2.027886 2.424315 1.647273 1.823681 2.131187 2.156419 1.928486
##  [73] 1.685024 2.280053 2.608534 2.003532 2.665841 2.112482 2.089144 1.981383
##  [81] 2.206457 2.357887 2.193453 1.517393 2.434945 1.809494 1.907338 2.446182
##  [89] 1.086443 1.739024 2.338378 1.844566 1.722431 2.671673 1.933654 2.301340
##  [97] 2.041987 2.374470 2.240776 2.499724
```

```r
colMeans( df )
```

```
##   [1] 1.506302 2.309718 2.044046 1.870102 2.045851 1.352929 1.851451 2.570470
```

```
##   [9] 2.174252 1.663822 2.657600 1.855943 1.617893 2.771453 2.182840 1.649257
##  [17] 2.463262 2.315259 2.509511 1.988113 2.095008 2.489222 1.759684 1.935487
##  [25] 1.875826 2.121479 2.763296 1.754126 2.271870 2.219549 2.914987 2.329730
##  [33] 1.971193 2.441072 1.883654 2.184502 2.494055 1.829070 2.079686 1.987856
##  [41] 2.915654 2.263425 2.554286 2.036809 2.242547 1.257043 1.538068 2.581715
##  [49] 2.356727 1.810872 2.028312 1.545861 2.125024 1.804058 2.037262 1.996856
##  [57] 2.021975 1.642865 1.914819 1.706684 2.841575 1.369619 2.398706 2.255708
##  [65] 1.953849 2.224337 1.794222 1.535959 1.731702 2.336698 1.977720 1.998956
##  [73] 1.584015 2.402400 2.145370 1.291185 2.201861 2.463810 2.684765 1.815413
##  [81] 1.697354 2.605871 1.762429 2.091668 1.767179 2.183985 2.439829 2.662391
##  [89] 2.018774 2.297991 2.363514 2.373043 1.419175 1.412508 2.100600 2.147071
##  [97] 1.900268 1.749768 1.971236 1.549056
```

```
df %>% apply( ., 1, mean )
```

```
##   [1] 2.237211 2.561320 2.251888 2.683165 1.659763 1.737900 2.156323 1.914529
##   [9] 2.255690 1.981960 1.551305 2.003171 1.983989 2.149901 2.206424 2.316499
##  [17] 2.439716 2.023911 1.831881 1.590739 2.398962 1.240311 2.670237 1.761688
##  [25] 2.359972 1.505164 2.960133 1.896109 1.991247 1.924412 1.923512 1.903865
##  [33] 1.750435 3.338297 1.951837 2.580570 2.292812 1.950270 1.724061 2.341317
##  [41] 1.312544 2.168936 1.403851 1.955030 2.474856 1.754731 1.760444 1.886532
##  [49] 1.719791 1.398076 2.634272 1.995588 2.289086 1.950851 2.193313 1.861967
##  [57] 2.023289 1.573940 1.318893 2.572031 1.724108 1.952620 2.588182 2.266717
##  [65] 2.688869 2.027886 2.424315 1.647273 1.823681 2.131187 2.156419 1.928486
##  [73] 1.685024 2.280053 2.608534 2.003532 2.665841 2.112482 2.089144 1.981383
##  [81] 2.206457 2.357887 2.193453 1.517393 2.434945 1.809494 1.907338 2.446182
##  [89] 1.086443 1.739024 2.338378 1.844566 1.722431 2.671673 1.933654 2.301340
##  [97] 2.041987 2.374470 2.240776 2.499724
```

```
df %>% apply( ., 2, mean )
```

```
##   [1] 1.506302 2.309718 2.044046 1.870102 2.045851 1.352929 1.851451 2.570470
##   [9] 2.174252 1.663822 2.657600 1.855943 1.617893 2.771453 2.182840 1.649257
##  [17] 2.463262 2.315259 2.509511 1.988113 2.095008 2.489222 1.759684 1.935487
```

```
## [25] 1.875826 2.121479 2.763296 1.754126 2.271870 2.219549 2.914987 2.329730
## [33] 1.971193 2.441072 1.883654 2.184502 2.494055 1.829070 2.079686 1.987856
## [41] 2.915654 2.263425 2.554286 2.036809 2.242547 1.257043 1.538068 2.581715
## [49] 2.356727 1.810872 2.028312 1.545861 2.125024 1.804058 2.037262 1.996856
## [57] 2.021975 1.642865 1.914819 1.706684 2.841575 1.369619 2.398706 2.255708
## [65] 1.953849 2.224337 1.794222 1.535959 1.731702 2.336698 1.977720 1.998956
## [73] 1.584015 2.402400 2.145370 1.291185 2.201861 2.463810 2.684765 1.815413
## [81] 1.697354 2.605871 1.762429 2.091668 1.767179 2.183985 2.439829 2.662391
## [89] 2.018774 2.297991 2.363514 2.373043 1.419175 1.412508 2.100600 2.147071
## [97] 1.900268 1.749768 1.971236 1.549056
```

```
rowSums( df )
```

```
##   [1] 223.7211 256.1320 225.1888 268.3165 165.9763 173.7900 215.6323 191.4529
##   [9] 225.5690 198.1960 155.1305 200.3171 198.3989 214.9901 220.6424 231.6499
##  [17] 243.9716 202.3911 183.1881 159.0739 239.8962 124.0311 267.0237 176.1688
##  [25] 235.9972 150.5164 296.0133 189.6109 199.1247 192.4412 192.3512 190.3865
##  [33] 175.0435 333.8297 195.1837 258.0570 229.2812 195.0270 172.4061 234.1317
##  [41] 131.2544 216.8936 140.3851 195.5030 247.4856 175.4731 176.0444 188.6532
##  [49] 171.9791 139.8076 263.4272 199.5588 228.9086 195.0851 219.3313 186.1967
##  [57] 202.3289 157.3940 131.8893 257.2031 172.4108 195.2620 258.8182 226.6717
##  [65] 268.8869 202.7886 242.4315 164.7273 182.3681 213.1187 215.6419 192.8486
##  [73] 168.5024 228.0053 260.8534 200.3532 266.5841 211.2482 208.9144 198.1383
##  [81] 220.6457 235.7887 219.3453 151.7393 243.4945 180.9494 190.7338 244.6182
##  [89] 108.6443 173.9024 233.8378 184.4566 172.2431 267.1673 193.3654 230.1340
##  [97] 204.1987 237.4470 224.0776 249.9724
```

```
colSums( df )
```

```
##   [1] 150.6302 230.9718 204.4046 187.0102 204.5851 135.2929 185.1451 257.0470
##   [9] 217.4252 166.3822 265.7600 185.5943 161.7893 277.1453 218.2840 164.9257
##  [17] 246.3262 231.5259 250.9511 198.8113 209.5008 248.9222 175.9684 193.5487
##  [25] 187.5826 212.1479 276.3296 175.4126 227.1870 221.9549 291.4987 232.9730
##  [33] 197.1193 244.1072 188.3654 218.4502 249.4055 182.9070 207.9686 198.7856
```

```
## [41] 291.5654 226.3425 255.4286 203.6809 224.2547 125.7043 153.8068 258.1715
## [49] 235.6727 181.0872 202.8312 154.5861 212.5024 180.4058 203.7262 199.6856
## [57] 202.1975 164.2865 191.4819 170.6684 284.1575 136.9619 239.8706 225.5708
## [65] 195.3849 222.4337 179.4222 153.5959 173.1702 233.6698 197.7720 199.8956
## [73] 158.4015 240.2400 214.5370 129.1185 220.1861 246.3810 268.4765 181.5413
## [81] 169.7354 260.5871 176.2429 209.1668 176.7179 218.3985 243.9829 266.2391
## [89] 201.8774 229.7991 236.3514 237.3043 141.9175 141.2508 210.0600 214.7071
## [97] 190.0268 174.9768 197.1236 154.9056
```

```
df %>% apply( ., 1, sum )
```

```
##  [1] 223.7211 256.1320 225.1888 268.3165 165.9763 173.7900 215.6323 191.4529
##  [9] 225.5690 198.1960 155.1305 200.3171 198.3989 214.9901 220.6424 231.6499
## [17] 243.9716 202.3911 183.1881 159.0739 239.8962 124.0311 267.0237 176.1688
## [25] 235.9972 150.5164 296.0133 189.6109 199.1247 192.4412 192.3512 190.3865
## [33] 175.0435 333.8297 195.1837 258.0570 229.2812 195.0270 172.4061 234.1317
## [41] 131.2544 216.8936 140.3851 195.5030 247.4856 175.4731 176.0444 188.6532
## [49] 171.9791 139.8076 263.4272 199.5588 228.9086 195.0851 219.3313 186.1967
## [57] 202.3289 157.3940 131.8893 257.2031 172.4108 195.2620 258.8182 226.6717
## [65] 268.8869 202.7886 242.4315 164.7273 182.3681 213.1187 215.6419 192.8486
## [73] 168.5024 228.0053 260.8534 200.3532 266.5841 211.2482 208.9144 198.1383
## [81] 220.6457 235.7887 219.3453 151.7393 243.4945 180.9494 190.7338 244.6182
## [89] 108.6443 173.9024 233.8378 184.4566 172.2431 267.1673 193.3654 230.1340
## [97] 204.1987 237.4470 224.0776 249.9724
```

```
df %>% apply( ., 2, sum )
```

```
##  [1] 150.6302 230.9718 204.4046 187.0102 204.5851 135.2929 185.1451 257.0470
##  [9] 217.4252 166.3822 265.7600 185.5943 161.7893 277.1453 218.2840 164.9257
## [17] 246.3262 231.5259 250.9511 198.8113 209.5008 248.9222 175.9684 193.5487
## [25] 187.5826 212.1479 276.3296 175.4126 227.1870 221.9549 291.4987 232.9730
## [33] 197.1193 244.1072 188.3654 218.4502 249.4055 182.9070 207.9686 198.7856
## [41] 291.5654 226.3425 255.4286 203.6809 224.2547 125.7043 153.8068 258.1715
## [49] 235.6727 181.0872 202.8312 154.5861 212.5024 180.4058 203.7262 199.6856
```

```
## [57] 202.1975 164.2865 191.4819 170.6684 284.1575 136.9619 239.8706 225.5708
## [65] 195.3849 222.4337 179.4222 153.5959 173.1702 233.6698 197.7720 199.8956
## [73] 158.4015 240.2400 214.5370 129.1185 220.1861 246.3810 268.4765 181.5413
## [81] 169.7354 260.5871 176.2429 209.1668 176.7179 218.3985 243.9829 266.2391
## [89] 201.8774 229.7991 236.3514 237.3043 141.9175 141.2508 210.0600 214.7071
## [97] 190.0268 174.9768 197.1236 154.9056
```

```
df %>% apply( ., 1, function(x) {
return( c(  mean = mean(x), sum = sum(x) , sd=sd(x)) );
} )
```

```
##            [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
## mean   2.237211   2.561320   2.251888   2.683165   1.659763   1.737900
## sum  223.721066 256.132043 225.188832 268.316524 165.976277 173.789979
## sd     3.630887   3.680503   4.497265   4.101224   3.915560   3.882969
##            [,7]       [,8]       [,9]      [,10]      [,11]      [,12]
## mean   2.156323   1.914529   2.255690   1.981960   1.551305   2.003171
## sum  215.632265 191.452913 225.568970 198.196042 155.130451 200.317074
## sd     3.809685   4.371032   3.961911   3.485458   4.240409   3.739925
##           [,13]      [,14]      [,15]      [,16]      [,17]      [,18]
## mean   1.983989   2.149901   2.206424   2.316499   2.439716   2.023911
## sum  198.398943 214.990052 220.642379 231.649895 243.971550 202.391087
## sd     4.138103   3.844233   4.124896   4.265699   3.957028   3.934486
##           [,19]      [,20]      [,21]      [,22]      [,23]      [,24]
## mean   1.831881   1.590739   2.398962   1.240311   2.670237   1.761688
## sum  183.188142 159.073863 239.896164 124.031149 267.023710 176.168791
## sd     3.868979   3.466278   4.456609   4.368117   4.291133   3.711095
##           [,25]      [,26]      [,27]      [,28]      [,29]      [,30]
## mean   2.359972   1.505164   2.960133   1.896109   1.991247   1.924412
## sum  235.997185 150.516371 296.013285 189.610936 199.124671 192.441246
## sd     4.250558   3.687070   3.598078   3.838593   4.144771   3.629384
##           [,31]      [,32]      [,33]      [,34]      [,35]      [,36]
## mean   1.923512   1.903865   1.750435   3.338297   1.951837   2.580570
## sum  192.351171 190.386511 175.043458 333.829688 195.183700 258.057009
```

```
## sd     3.695462   4.130170   4.440953   4.457127   4.511619   4.095787
##           [,37]      [,38]      [,39]      [,40]      [,41]      [,42]
## mean   2.292812   1.950270   1.724061   2.341317   1.312544   2.168936
## sum  229.281199 195.026955 172.406134 234.131670 131.254433 216.893628
## sd     3.758513   4.045385   3.987361   4.035000   3.842868   4.218277
##           [,43]      [,44]      [,45]      [,46]      [,47]      [,48]
## mean   1.403851   1.955030   2.474856   1.754731   1.760444   1.886532
## sum  140.385110 195.503005 247.485588 175.473092 176.044353 188.653240
## sd     4.534672   3.883917   3.870599   3.743872   3.931818   3.947986
##           [,49]      [,50]      [,51]      [,52]      [,53]      [,54]
## mean   1.719791   1.398076   2.634272   1.995588   2.289086   1.950851
## sum  171.979101 139.807551 263.427177 199.558785 228.908626 195.085142
## sd     4.703441   4.346694   4.020678   3.803798   3.870052   3.776024
##           [,55]      [,56]      [,57]      [,58]      [,59]      [,60]
## mean   2.193313   1.861967   2.023289   1.573940   1.318893   2.572031
## sum  219.331271 186.196672 202.328901 157.393967 131.889308 257.203072
## sd     4.056487   3.942123   4.812211   4.663019   3.941706   4.294111
##           [,61]      [,62]      [,63]      [,64]      [,65]      [,66]
## mean   1.724108   1.952620   2.588182   2.266717   2.688869   2.027886
## sum  172.410803 195.262014 258.818194 226.671722 268.886851 202.788601
## sd     4.026913   3.921206   3.487354   4.199238   3.494869   3.996626
##           [,67]      [,68]      [,69]      [,70]      [,71]      [,72]
## mean   2.424315   1.647273   1.823681   2.131187   2.156419   1.928486
## sum  242.431543 164.727254 182.368070 213.118708 215.641928 192.848646
## sd     4.318765   3.657983   4.257748   4.029817   3.906493   4.219893
##           [,73]      [,74]      [,75]      [,76]      [,77]      [,78]
## mean   1.685024   2.280053   2.608534   2.003532   2.665841   2.112482
## sum  168.502420 228.005320 260.853377 200.353157 266.584099 211.248151
## sd     3.916223   4.197617   4.002580   4.195957   3.710389   4.741790
##           [,79]      [,80]      [,81]      [,82]      [,83]      [,84]
## mean   2.089144   1.981383   2.206457   2.357887   2.193453   1.517393
## sum  208.914397 198.138267 220.645669 235.788721 219.345350 151.739342
## sd     3.672632   4.264303   4.211156   3.551260   4.199365   3.950177
```

```
##              [,85]      [,86]      [,87]      [,88]      [,89]      [,90]
## mean    2.434945   1.809494   1.907338   2.446182   1.086443   1.739024
## sum   243.494534 180.949372 190.733752 244.618191 108.644271 173.902425
## sd      4.312465   3.155313   3.957802   3.807015   3.694354   3.858992
##              [,91]      [,92]      [,93]      [,94]      [,95]      [,96]
## mean    2.338378   1.844566   1.722431   2.671673   1.933654   2.301340
## sum   233.837761 184.456631 172.243105 267.167350 193.365394 230.133987
## sd      3.913441   3.879508   3.717812   3.988352   3.874311   3.901135
##              [,97]      [,98]      [,99]     [,100]
## mean    2.041987   2.374470   2.240776   2.499724
## sum   204.198653 237.446954 224.077640 249.972373
## sd      3.408657   3.492311   4.367349   3.990085
```

```r
df %>% apply( ., 2, function(x) {
return( c(  mean = mean(x), sum = sum(x) , sd=sd(x)) );
} )
```

```
##               [,1]       [,2]       [,3]       [,4]       [,5]       [,6]
## mean    1.506302   2.309718   2.044046   1.870102   2.045851   1.352929
## sum   150.630198 230.971801 204.404590 187.010167 204.585122 135.292888
## sd      3.759408   3.608863   4.485322   3.988199   4.082145   3.738441
##               [,7]       [,8]       [,9]      [,10]      [,11]      [,12]
## mean    1.851451   2.570470   2.174252   1.663822   2.657600   1.855943
## sum   185.145059 257.046970 217.425160 166.382237 265.760005 185.594309
## sd      4.372604   4.084886   4.249401   4.081491   3.930167   3.802611
##              [,13]      [,14]      [,15]      [,16]      [,17]      [,18]
## mean    1.617893   2.771453   2.182840   1.649257   2.463262   2.315259
## sum   161.789294 277.145346 218.284038 164.925664 246.326224 231.525930
## sd      4.043780   3.782108   4.272739   4.209954   3.572875   4.173549
##              [,19]      [,20]      [,21]      [,22]      [,23]      [,24]
## mean    2.509511   1.988113   2.095008   2.489222   1.759684   1.935487
## sum   250.951082 198.811293 209.500842 248.922237 175.968350 193.548716
## sd      4.194181   3.828838   3.530424   3.766778   4.233874   3.799625
##              [,25]      [,26]      [,27]      [,28]      [,29]      [,30]
```

```
## mean   1.875826   2.121479   2.763296   1.754126   2.271870   2.219549
## sum  187.582617 212.147923 276.329607 175.412565 227.187027 221.954899
## sd     3.534523   3.813971   4.485918   3.944075   3.883618   4.145020
##           [,31]      [,32]      [,33]      [,34]      [,35]      [,36]
## mean   2.914987   2.329730   1.971193   2.441072   1.883654   2.184502
## sum  291.498699 232.972994 197.119323 244.107194 188.365401 218.450176
## sd     4.529231   4.493715   3.770104   3.677918   4.204289   3.531914
##           [,37]      [,38]      [,39]      [,40]      [,41]      [,42]
## mean   2.494055   1.829070   2.079686   1.987856   2.915654   2.263425
## sum  249.405505 182.907024 207.968603 198.785595 291.565390 226.342494
## sd     3.880416   3.745352   3.776953   4.042834   3.832568   4.366471
##           [,43]      [,44]      [,45]      [,46]      [,47]      [,48]
## mean   2.554286   2.036809   2.242547   1.257043   1.538068   2.581715
## sum  255.428612 203.680896 224.254726 125.704269 153.806849 258.171487
## sd     3.956391   3.872401   3.837012   4.103333   4.483286   3.821250
##           [,49]      [,50]      [,51]      [,52]      [,53]      [,54]
## mean   2.356727   1.810872   2.028312   1.545861   2.125024   1.804058
## sum  235.672694 181.087185 202.831165 154.586146 212.502356 180.405761
## sd     4.115683   3.987710   4.375354   3.806593   4.004989   3.661021
##           [,55]      [,56]      [,57]      [,58]      [,59]      [,60]
## mean   2.037262   1.996856   2.021975   1.642865   1.914819   1.706684
## sum  203.726189 199.685600 202.197459 164.286499 191.481918 170.668398
## sd     3.751712   3.667852   4.266503   4.237495   4.379834   4.096068
##           [,61]      [,62]      [,63]      [,64]      [,65]      [,66]
## mean   2.841575   1.369619   2.398706   2.255708   1.953849   2.224337
## sum  284.157536 136.961902 239.870645 225.570756 195.384886 222.433664
## sd     4.023824   4.400399   3.934693   4.335051   4.349678   3.699433
##           [,67]      [,68]      [,69]      [,70]      [,71]      [,72]
## mean   1.794222   1.535959   1.731702   2.336698   1.977720   1.998956
## sum  179.422192 153.595898 173.170188 233.669835 197.772047 199.895554
## sd     3.793490   4.191268   3.335507   3.283476   4.254151   4.162196
##           [,73]      [,74]      [,75]      [,76]      [,77]      [,78]
## mean   1.584015   2.402400   2.145370   1.291185   2.201861   2.463810
```

```
## sum  158.401510 240.239956 214.537016 129.118547 220.186056 246.380999
## sd     3.967972   4.148644   4.169144   4.149148   3.979157   4.237673
##          [,79]      [,80]      [,81]      [,82]      [,83]      [,84]
## mean   2.684765   1.815413   1.697354   2.605871   1.762429   2.091668
## sum  268.476534 181.541313 169.735389 260.587074 176.242866 209.166753
## sd     4.044644   3.964428   4.571538   4.037852   3.883124   4.617125
##          [,85]      [,86]      [,87]      [,88]      [,89]      [,90]
## mean   1.767179   2.183985   2.439829   2.662391   2.018774   2.297991
## sum  176.717944 218.398473 243.982902 266.239106 201.877363 229.799074
## sd     3.784048   3.899219   3.880800   3.649003   4.018943   3.914205
##          [,91]      [,92]      [,93]      [,94]    [,95]      [,96]      [,97]
## mean   2.363514   2.373043   1.419175   1.412508   2.1006   2.147071   1.900268
## sum  236.351387 237.304254 141.917462 141.250804 210.0600 214.707063 190.026767
## sd     3.701267   3.587968   3.820106   3.516562   3.6452   3.668789   4.528977
##          [,98]      [,99]     [,100]
## mean   1.749768   1.971236   1.549056
## sum  174.976758 197.123560 154.905604
## sd     4.282429   4.367797   4.453279
```

0.5.2 用 mtcars 进行练习

用 tapply 练习：

1. 用 **汽缸数**分组，计算 **油耗**的 **平均值**；
2. 用 **汽缸数**分组，计算 **wt** 的 **平均值**；

用 dplyr 的函数实现上述计算

```
## 代码写这里，并运行；
library(magrittr)
```

```
##
## 载入程辑包：'magrittr'
```

```
## The following object is masked from 'package:purrr':
##
##     set_names

## The following object is masked from 'package:tidyr':
##
##     extract
```

```
mtcars %$% tapply( mpg, cyl, mean );
```

```
##        4        6        8
## 26.66364 19.74286 15.10000
```

```
mtcars %$% tapply( wt, cyl, mean );
```

```
##        4        6        8
## 2.285727 3.117143 3.999214
```

```
mtcars %>% group_by( cyl ) %>% summarise( mean = mean( mpg ) );
```

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

```
mtcars %>% group_by( cyl ) %>% summarise( mean = mean( wt ) );
```

```
## # A tibble: 3 x 2
##     cyl  mean
##   <dbl> <dbl>
## 1     4  2.29
## 2     6  3.12
## 3     8  4.00
```

0.5.3 练习 lapply 和 sapply

1. 分别用 lapply 和 sapply 计算下面 list 里每个成员 vector 的长度：

```
list( a = 1:10, b = letters[1:5], c = LETTERS[1:8] );
```

2. 分别用 lapply 和 sapply 计算 mtcars 每列的平均值；

```
## 代码写这里，并运行；
list( a = 1:10, b = letters[1:5], c = LETTERS[1:8] ) %>%
lapply( function(x) { length(x) } );
```

```
## $a
## [1] 10
##
## $b
## [1] 5
##
## $c
## [1] 8
```

```
list( a = 1:10, b = letters[1:5], c = LETTERS[1:8] ) %>%
sapply( function(x) { length(x) } );
```

```
##  a  b  c
## 10  5  8
```

```
mtcars %>% lapply( mean );
```

```
## $mpg
```

```
## [1] 20.09062
##
## $cyl
## [1] 6.1875
##
## $disp
## [1] 230.7219
##
## $hp
## [1] 146.6875
##
## $drat
## [1] 3.596563
##
## $wt
## [1] 3.21725
##
## $qsec
## [1] 17.84875
##
## $vs
## [1] 0.4375
##
## $am
## [1] 0.40625
##
## $gear
## [1] 3.6875
##
## $carb
## [1] 2.8125
```

```
mtcars %>% sapply( mean );
```

```
##        mpg        cyl       disp         hp       drat         wt       qsec
##  20.090625   6.187500 230.721875 146.687500   3.596563   3.217250  17.848750
##         vs         am       gear       carb
##   0.437500   0.406250   3.687500   2.812500
```

0.6 练习与作业 3：loop 进阶，purr 包的函数

0.6.1 map 初步

生成一个变量：

```
df <- tibble(
  a = rnorm(10),
  b = rnorm(10),
  c = rnorm(10),
  d = rnorm(10)
)
```

用 map 计算：

- 列平均值、总和和中值

```
## 代码写这里，并运行；
df <- tibble(
  a = rnorm(10),
  b = rnorm(10),
  c = rnorm(10),
  d = rnorm(10)
)
df %>% map_dbl( mean );
```

```
##          a          b          c          d
## 0.35210282 0.12822545 0.15006180 0.07018004
```

```
df %>% map_dbl( sum );
```

```
##         a         b         c         d
## 3.5210282 1.2822545 1.5006180 0.7018004
```

```
df %>% map_dbl( median );
```

```
##           a           b           c           d
##  0.14740456  0.04763584 -0.02106441  0.07045451
```

0.6.2 map 进阶

用 map 配合 purr 包中其它函数，用 mtcars：

为每一个 **汽缸数**计算燃油效率 mpg 与重量 wt 的相关性（Pearson correlation），得到 p 值和 correlation coefficient 值。

```
## 代码写这里，并运行；

mtcars %>% split( .$cyl ) %>% map( function(df) { cor.test( df$wt, df$mpg ) } ) %>% map
```

```
##          4          6          8
## -0.7131848 -0.6815498 -0.6503580
```

0.6.3 keep 和 discard

1. 保留 iris 中有 factor 的列，并打印前 10 行；

2. 去掉 iris 中有 factor 的列，并打印前 10 行；

```
## 代码写这里，并运行；
iris1<-iris %>%
  keep(is.factor)
head(iris1,n=10)
```

```
##    Species
## 1   setosa
## 2   setosa
## 3   setosa
## 4   setosa
## 5   setosa
## 6   setosa
## 7   setosa
## 8   setosa
## 9   setosa
## 10  setosa
```

```
iris2<-iris %>%
  discard(is.factor)
head(iris2,n=10)
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1           5.1         3.5          1.4         0.2
## 2           4.9         3.0          1.4         0.2
## 3           4.7         3.2          1.3         0.2
## 4           4.6         3.1          1.5         0.2
## 5           5.0         3.6          1.4         0.2
## 6           5.4         3.9          1.7         0.4
## 7           4.6         3.4          1.4         0.3
## 8           5.0         3.4          1.5         0.2
## 9           4.4         2.9          1.4         0.2
## 10          4.9         3.1          1.5         0.1
```

0.6.4 用 reduce

用 `reduce` 得到以下三个 vector 中共有的数字：

```
c(1, 3, 5, 6, 10),
  c(1, 2, 3, 7, 8, 10),
  c(1, 2, 3, 4, 8, 9, 10)
```

```
## 代码写这里，并运行；
vs <- list(
c(1, 3, 5, 6, 10),
c(1, 2, 3, 7, 8, 10),
c(1, 2, 3, 4, 8, 9, 10)
)
vs %>% reduce(intersect)
```

```
## [1]  1  3 10
```

0.6.5 运行以下代码，观察得到的结果，并用 tidyverse 包中的 spread 等函数实现类似的结果

```
dfs <- list(
  age = tibble(name = "John", age = 30),
  sex = tibble(name = c("John", "Mary"), sex = c("M", "F")),
  trt = tibble(name = "Mary", treatment = "A")
);

dfs %>% reduce(full_join);
```

```
## 代码写这里，并运行；
dfs <- list(
  age = tibble(name = "John", age = 30),
  sex = tibble(name = c("John", "Mary"), sex = c("M", "F")),
  trt = tibble(name = "Mary", treatment = "A")
);

dfs %>% reduce(full_join);
```

```
## Joining, by = "name"
## Joining, by = "name"
```

```
## # A tibble: 2 x 4
##   name    age sex   treatment
##   <chr> <dbl> <chr> <chr>
## 1 John     30 M     <NA>
## 2 Mary     NA F     A
```

```
M<-tribble(
  ~name,~x,~y,
  "John","age","30",
  "John","sex","M",
  "Mary","sex","F",
  "Mary","treatment","A"
)
M %>% spread(x,y)
```

```
## # A tibble: 2 x 4
##   name  age   sex   treatment
##   <chr> <chr> <chr> <chr>
## 1 John  30    M     <NA>
## 2 Mary  <NA>  F     A
```

0.7 练习与作业 4：并行计算

0.7.1 安装相关包，成功运行以下代码，观察得到的结果，并回答问题

```
* parallel
* foreach
* iterators
```

```
library(parallel); ##
library(foreach);
```

```
##
## 载入程辑包：'foreach'

## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
library(iterators);

## 检测有多少个 CPU --
( cpus <- parallel::detectCores() );
```

```
## [1] 16
```

```
## 创建一个 data.frame
d <- data.frame(x=1:10000, y=rnorm(10000));

## make a cluster --
cl <- makeCluster( cpus - 1 );

## 分配任务 ...
```

```
res <- foreach( row = iter( d, by = "row" ) ) %dopar% {
  return ( row$x * row$y  );
}
```

```
## Warning: executing %dopar% sequentially: no parallel backend registered
```

```
## 注意在最后关闭创建的 cluster
stopCluster( cl );

summary(unlist(res));
```

```
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -32198.89  -2665.29     -8.56   -125.15   2525.70  30942.88
```

问：你的系统有多少个 CPU？此次任务使用了多少个？**答**：用代码打印出相应的数字即可：

```
## 代码写这里，并运行；
print(16)
```

```
## [1] 16
```

```
print(6)
```

```
## [1] 6
```