Stat 292 - Recitation 5

R-project, Pipelines, Avoid Loops

Orçun Oltulu

14 / 4 / 2021

Exercise 1:

Part A:

Create an R project for Recitation 5 on your Desktop. Then, open a new R Script.

Part B:

Check the working directory.

```
getwd()
```

```
## [1] "C:/Users/orcun/OneDrive/Masaüstü/STAT292/Recitation/Recitation5/Recitation5-Rpro
```

Part C:

Create a 10x5 matrix called A, give row and column names. Fill with random numbers from standard normal distribution. (Use set.seed(292))

```
set.seed(292)
A <- matrix(rnorm(50), nrow = 10, ncol = 5,
            dimnames = list(paste("row.", 1:10, sep=""),
                            paste("col.", 1:5, sep="")))
```

Part D:

Now, export matrix A to a MatrixA.csv file. Check if that .csv file is inside Recitation 5 folder on your Desktop.

```
write.csv(A, file = "MatrixA.cvs")
```

Part E:

Obtain mean values of each column, then export column means to a .txt file.

```
write.table(apply(A, 2, mean),
            "colmeans.txt")
```

Part F:

- Open a new R Script, Functions, in Recitation 5 on your Desktop,
- Write an R function, call myfunction, that given an object, counts the negative values and returns that count.
- Then, using source() command call the Functions.R in your main script. When you run your code, you will see that myfunction is appeared on your global environment.
- Using myfunction, find the number of negative values in A.

```
source("Functions.R")
```

Part G:

- Create some random variables; x,y,z with some random values.

```
x <- 10
y <- list(1:10,letters[1:10])
z <- c("mon","tue","wed","thu","fri","sat","sun")

B <- matrix(rnorm(80,100,2), nrow = 20, ncol = 4,
            dimnames = list(paste("row.", 1:20, sep=""),
                            paste("col.", 1:4, sep="")))
```

- Save them in mydata.RData file.

```
save(x,y,z,A,B, file = "mydata.RData")
```

- Clear your global environment, and import .Rdata file.

```
# to remove all objects
rm(list = ls())
```

```
load("mydata.RData")
```

- Use ls() command to see the objects you have in your global environment.

```
ls()
```

```
## [1] "A" "B" "x" "y" "z"
```

Exercise 2: tidyverse - magrittr

Import 'tidyverse', 'magrittr', 'dplyr' and 'ISLR' packages. Then, load iris data set from ISLR package.

Part A:

Using Pipe operator;

- Create a sequence 5 to 12.
- Take the log of that sequence.
- Round to 2 decimals.
- Convert them into characters.

```
5:12 %>% log %>% round(2) %>% as.character()
```

```
## [1] "1.61" "1.79" "1.95" "2.08" "2.2"  "2.3"  "2.4"  "2.48"
```

Part B:

Using Pipe operator;

- Find the absolute value of each element in matrix A (created in ex.1).
- Round each value to 3 decimals.
- Take transpose.
- Multiply with A.

```
A %>% abs %>% round(3) %>% t %>% '%*%' (A)
```

```
##          col.1     col.2       col.3     col.4      col.5
## col.1 3.126471 -1.257057  0.72172352  5.572422 0.76152517
## col.2 1.231602 -5.361959  2.84888843  2.581770 0.40999195
## col.3 1.829794 -2.015827  7.60385632  4.009017 0.07399466
## col.4 1.169948  2.590284  1.96570928 11.647125 1.42925091
## col.5 1.786244 -1.890020 -0.01410619  1.471697 0.49019572
```

Part C:

Assume that you want to calculate Root Mean Squared Error for a regression model;

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2}{n}}$$

- Create Y and $\hat{Y}$ as follows;

```r
set.seed(292)
Y <- sample(10:30, 30, replace = T)
Y_hat <- rnorm(30,3,2) + Y
```

- Then, calculate RMSE.

```r
Y %>% '-' (Y_hat) %>%  '^' (2) %>% mean %>% sqrt
```

```
## [1] 3.724846
```

```r
# or

Y %>% '-' (Y_hat) %>%  '^' (2) %>% sum %>% '/' (length(Y)) %>% sqrt
```

```
## [1] 3.724846
```

Part D:

Using aggregate function, find the mean of each variable for each Species.

```r
data(iris)
iris %>% aggregate(. ~ Species, ., mean)
```

```
##      Species Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     setosa        5.006       3.428        1.462       0.246
## 2 versicolor        5.936       2.770        4.260       1.326
## 3  virginica        6.588       2.974        5.552       2.026
```

Part E:

For the ones with Petal.Length > 3, find the mean of Sepal Length and Sepal Width of each Species, using aggregate function.

```r
iris %>%
  subset(Petal.Length > 3) %>%
  aggregate(cbind(Sepal.Length, Sepal.Width) ~ Species, ., mean)
```

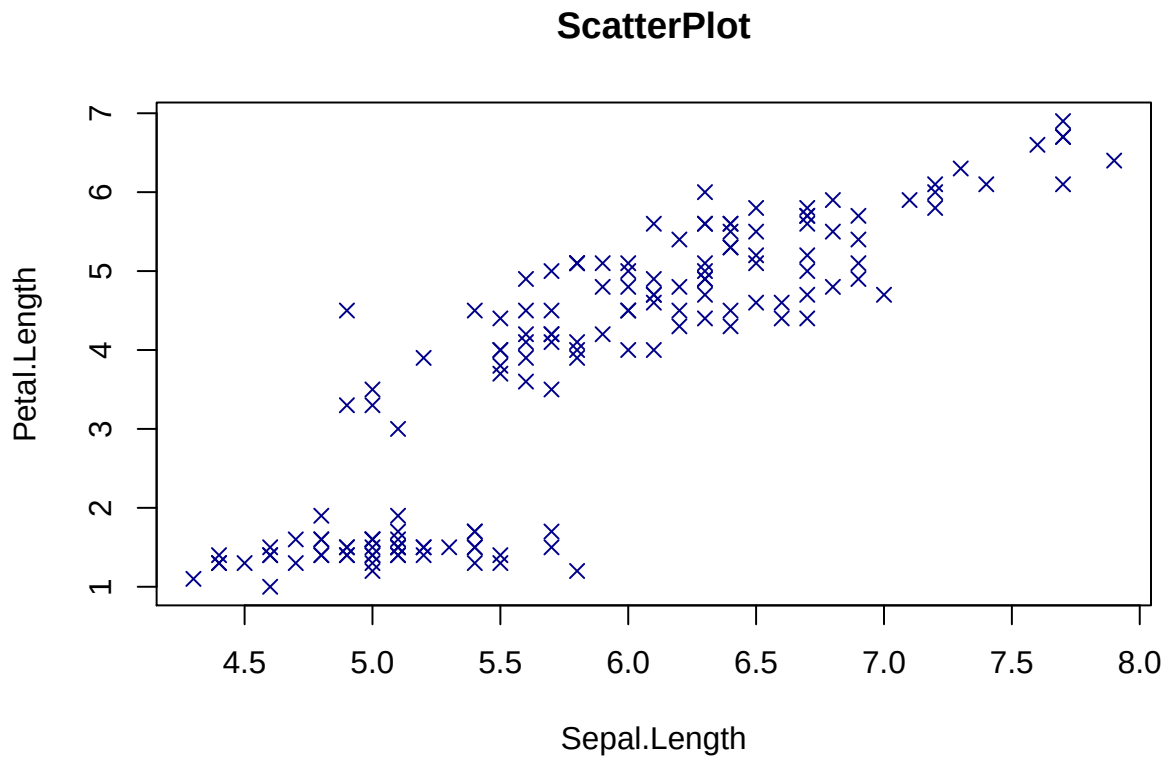
```
##      Species Sepal.Length Sepal.Width
## 1 versicolor     5.953061     2.77551
## 2  virginica     6.588000     2.97400
```

Part F:

Now using one pipe command; **select** Sepal vs Petal Lengths, plot them on a scatter plot and assign title, change color etc., then, obtain correlation between Sepal and Petal Lengths.

```r
iris %>%
  select(Sepal.Length,Petal.Length) %T>%
```

```
  plot(., main="ScatterPlot",col="Dark Blue", pch=4) %>%
  cor()
```




```
##              Sepal.Length Petal.Length
## Sepal.Length    1.0000000    0.8717538
## Petal.Length    0.8717538    1.0000000
```

Part G:

Using pipeline, obtain the frequencies of each Species.

```
iris %$%
  table(Species)
```

```
## Species
##     setosa versicolor  virginica
##         50         50         50
```

Exercise 3: dplyr

How to use basic dplyr functions

The dplyr is an R-package that is used for transformation and summarization of tabular data with rows and columns. It includes a set of functions that filter rows, select specific columns, re-order rows, adds new columns and summarizes data.

Moreover, dplyr contains a useful function to perform another common task, which is the "split-applycombine" concept. Compared to base functions in R, the functions in dplyr have an advantage in the sense that they are easier to use, more consistent in the syntax, and aim to analyze data frames instead of just vectors.

Part A:

Using the same iris data set; **select** Sepal Length and Species and print the first 10 rows of the data frame.

```
iris %>%
  select(c(Sepal.Length,Species)) %>%
  head(.,10)
```

```
##    Sepal.Length Species
## 1           5.1  setosa
## 2           4.9  setosa
## 3           4.7  setosa
## 4           4.6  setosa
## 5           5.0  setosa
## 6           5.4  setosa
## 7           4.6  setosa
## 8           5.0  setosa
## 9           4.4  setosa
## 10          4.9  setosa
```

Part B:

Select Petal.Length Petal.Width and print the first 10 rows of the data frame.

```
iris %>%
  select(starts_with("P")) %>%
  head(.,10)
```

```
##    Petal.Length Petal.Width
## 1           1.4         0.2
## 2           1.4         0.2
## 3           1.3         0.2
## 4           1.5         0.2
## 5           1.4         0.2
## 6           1.7         0.4
## 7           1.4         0.3
## 8           1.5         0.2
```

```
## 9           1.4         0.2
## 10          1.5         0.1
```

Part C:

Now, **select** all of the numeric variables and print the first 10 rows of the data frame.

```
iris %>%
  select_if(is.numeric) %>%
  head(.,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
```

Part D:

Filter out a subset of Setosa whose Sepal Length ranges 4.4 and 4.8.

```
iris %>%
  filter(between(Sepal.Length, 4.4, 4.8), Species=="setosa")
```

```
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           4.7         3.2          1.3         0.2  setosa
## 2           4.6         3.1          1.5         0.2  setosa
## 3           4.6         3.4          1.4         0.3  setosa
## 4           4.4         2.9          1.4         0.2  setosa
## 5           4.8         3.4          1.6         0.2  setosa
## 6           4.8         3.0          1.4         0.1  setosa
## 7           4.6         3.6          1.0         0.2  setosa
## 8           4.8         3.4          1.9         0.2  setosa
## 9           4.7         3.2          1.6         0.2  setosa
## 10          4.8         3.1          1.6         0.2  setosa
## 11          4.4         3.0          1.3         0.2  setosa
## 12          4.5         2.3          1.3         0.3  setosa
## 13          4.4         3.2          1.3         0.2  setosa
## 14          4.8         3.0          1.4         0.3  setosa
## 15          4.6         3.2          1.4         0.2  setosa
```

Part E:

What is the mean value of Petal Length for each Species?

```
iris %>%
  group_by(Species) %>%
  summarise(mean(Petal.Length))
```

```
## # A tibble: 3 x 2
##   Species    `mean(Petal.Length)`
##   <fct>                     <dbl>
## 1 setosa                     1.46
## 2 versicolor                 4.26
## 3 virginica                  5.55
```

Part F:

Using **mutate()** function create a new column called proportion, which is the ratio of Sepal.Length to Sepal.Width.

```
iris %<>%
  mutate(Proportion = Sepal.Length / Sepal.Width)
```

Part G:

Create another new column called Type, if proportion is less than 1.8 then Type is A, else Type is B.

```
iris %<>%
  mutate(Type = ifelse(Proportion < 1.8, "A","B"))
```

Part H:

Count how many observations are assigned Type B for each Species.

```
iris %>%
  group_by(Species) %>%
  filter(Type == "B") %>%
  count
```

```
## # A tibble: 3 x 2
## # Groups:   Species [3]
##   Species        n
##   <fct>      <int>
## 1 setosa         1
## 2 versicolor    49
## 3 virginica     50
```

Exercise 4:

1. Generate a random sample from; (use set.seed 292)

- uniform distribution between 10 and 20, sized 5000.
- normal distribution with mean 15 and standard deviation 2, sized 5000.
- standard normal distribution, sized 5000.

2. Combine those three random samples in a data.frame.
3. Using a for loop, calculate means of each column, measure run time.
4. Using apply, calculate means of each column, measure run time.
5. Using colMeans, calculate means of each column, measure run time.

```
set.seed(292)
smp1 <- runif(5000,10,20)
smp2 <- rnorm(5000,15,2)
smp3 <- rnorm(5000)

mydf <- cbind(smp1,smp2,smp3)

## For loop
system.time({
  Means <- numeric()
  for(i in 1:ncol(mydf)){
    Means[i] <- mean(mydf[,i])
  }
})
```

```
##    user  system elapsed
##       0       0       0
```

```
## apply
system.time( apply(mydf,2,mean) )
```

```
##    user  system elapsed
##       0       0       0
```

```
## colMeans

system.time( colMeans(mydf) )
```

```
##    user  system elapsed
##       0       0       0
```

Part B:

Run a simulation on a for loop by follow the three steps below.

1. Generate 100 different samples each sized 5000 from standard normal distribution.
2. Apply Shapiro-Wilk Test on each sample for normality. (**Hint:** shapiro.test)
3. Collect p-values and find the proportion of p-values are less than 0.05.
4. Measure run time.

```
system.time({
  vec <- numeric()
  vec2 <- numeric()
  for(i in 1:100){
    vec <- rnorm(5000)
    vec2[i] <- shapiro.test(vec)$p.val
  }
})
```

```
##    user  system elapsed
##    0.06    0.00    0.07
```

Part C:

Now run this simulation without using a for loop. (There are lots of way to do this, you will see only one of them in the answers, however try to come up with different ideas)

```
system.time(
  rnorm(5000*100) %>%
    matrix(.,ncol=100, byrow = TRUE) %>%
    apply(.,2,function(x) shapiro.test(x)$p.val)
)
```

```
##    user  system elapsed
##    0.07    0.00    0.06
```

Part D:

First read 'bmi_data.txt' file into R.

Then **Using a loop**, Calculate Body-mass-index for each person, and create new columns 'BMI' and 'Status' for data frame such that

$$BMI = \frac{\text{Weight}}{\text{Height}^2}$$

$$\text{Status} = \begin{cases} \text{underweight}, & BMI \leq 18.5 \\ \text{normal}, & BMI > 18.5 \quad \& \quad BMI < 24.9 \\ \text{overweight}, & BMI \geq 24.9 \end{cases}$$

Hint: Before the loop you may want to create empty numeric() and character() variables for BMI and Status so that you can store results and append them to your data frame.

```
data <- read.table("bmi_data.txt")
BMI <- numeric()
Status <- character()

system.time({
for(i in 1:nrow(data)){

  BMI[i] <- data$Weight[i] / data$Height[i]^2

  if(BMI[i] <= 18.5){
    Status[i] <- "underweight"
  } else if(BMI[i] > 18.5 & BMI[i] < 24.9){
    Status[i] <- "normal"
  } else{
    Status[i] <- "overweight"
  }

}

data <- cbind(data, BMI, Status)
})
```

```
##    user  system elapsed
##       0       0       0
```

```
data
```

```
##      Name Weight Height      BMI      Status
## 1     Ali     71   1.90 19.66759      normal
## 2   Osman     86   1.68 30.47052  overweight
## 3  Zeynep     56   1.76 18.07851 underweight
## 4   Yonca     63   1.82 19.01944      normal
## 5   Gonca     60   1.64 22.30815      normal
## 6   Halil     73   1.74 24.11151      normal
## 7    Mert     61   1.59 24.12879      normal
## 8    Cenk     55   1.55 22.89282      normal
## 9   Hasan     57   1.73 19.04507      normal
```

```
## 10   Ayse     69   1.66 25.03992  overweight
## 11  Melis     66   1.54 27.82931  overweight
## 12  Yeliz     92   1.63 34.62682  overweight
```

Part F:

Now, re-read the data, this time, do the exact operation without using any loop.

```
data <- read.table("bmi_data.txt")
system.time({
data %<>%
  mutate(Status = ifelse(Weight/Height^2 <= 18.5,
                         "underweight",
                         ifelse(Weight/Height^2 >= 24.9,
                                "overweight",
                                "normal")))
})
```

```
##    user  system elapsed
##       0       0       0
```

Additional info:

A different way to measure elapsed;

```
# install.packages("devtools")
# devtools::install_github("jabiru/tictoc")
library(tictoc)

tic("sleeping")
print("falling asleep...")
```

```
## [1] "falling asleep..."
```

```
Sys.sleep(5)
print("...waking up")
```

```
## [1] "...waking up"
```

```
toc()
```

```
## sleeping: 5.06 sec elapsed
```