# HW3_IS457_127

## HW 3 - Due Monday, Oct 1, 2018 in moodle and hardcopy in class

(1)Please upload R code and report to Moodle with filename: HW3_IS457_YourClassID. (2)Turn in hard copy of your report in class without your name but only your class ID

### ClassID: 127

## Part 1. Start with "apply" function (9pts)

### (1) Create a new matrix by the following codes, briefly but completely explain what each line is doing. (3pts)

```
set.seed(457)
one_num <- sample(1:9,25,replace=TRUE)#random chose 25 numbers from [1,9] and store it to one_num
one_matrix <- matrix(one_num,ncol=5)
print(one_matrix)
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    2    2    8    3    1
## [2,]    2    3    7    9    1
## [3,]    1    9    9    7    9
## [4,]    5    2    2    4    4
## [5,]    9    7    2    3    3
```

`set.seed(457)` Set a seed which can make us have the same result when generate random numbers.

`one_num <- sample(1:9,25,replace=TRUE)` randomly choses 25 numbers from [1,9] and store it to one_num.

`one_matrix <- matrix(one_num,ncol=5)`Store the integers in one_num in one_matrix by column, and column is 5.

### Use the "apply" function on one_matrix to answer questions(2)(3)(4).

### (2) Calculate the mean of each row. (1pt) Calculate the sum of each column. (1pt)

```
rowmeans <- apply(one_matrix,1,mean)
print(rowmeans)
```

```
## [1] 3.2 4.4 7.0 3.4 4.8
```

```
colmeans <- apply(one_matrix,2,mean)
print(colmeans)
```

```
## [1] 3.8 4.6 5.6 5.2 3.6
```

### (3) Find the difference between the biggest and the smallest number for each row. (2pts)

```
difference<- function(x)
{
  difference = max(x)-min(x)
}
diff<- apply(one_matrix,1,difference)
print(diff)
```

```
## [1] 7 8 8 3 7
```

### (4) Calculate the sum of all numbers smaller than 5 for each column. (2pts)

```
subsumcal<-function(x)
{
if (x<=5)
    x=x
  else
    x=0
}
one_matrix1 = matrix(0,5,5) #tansform the matrix
for(i in 1:5){
  for(j in 1:5)
one_matrix1[i,j] = subsumcal(one_matrix[i,j])
}
subsum<- apply(one_matrix1,2,sum)
print(subsum)
```

```
## [1] 10  7  4 10  9
```

## Part 2. Get familiar with "sapply" and "lapply" functions (6pts)

### (5) Let's play with the "iris" dataset. Here's the command to load it:

```
data(iris)
```

Tell me the data types of each column(variable) by using "sapply" function (1pt)

```
sapply(iris,class)
```

```
## Sepal.Length  Sepal.Width Petal.Length  Petal.Width      Species
##    "numeric"    "numeric"    "numeric"    "numeric"     "factor"
```

### (6) Do question(5) again and get the same result (check the data type), but use "lapply" function. (1pt)

```
lapply(iris,class)
```

```
## $Sepal.Length
## [1] "numeric"
##
## $Sepal.Width
## [1] "numeric"
##
## $Petal.Length
## [1] "numeric"
##
## $Petal.Width
## [1] "numeric"
##
```

```
## $Species
## [1] "factor"
```

**Based on the output format, briefly explain the difference between "sapply" and "lapply" functions. (2pts)**

First, sapply will try to simplify as much as it can. Second, lapply returns a list while sapply by default returning a vector, matrix or an array if appropriate.

**(7) Now create a new list from one_matrix by using the following codes.**

```
list_1 <- list(a=one_matrix[,1],b=c(one_matrix[,2],one_matrix[,3]))
```

Take natural log of this list using the following code: `log(list_1)` #####Please briefly explain why the code above doesn't work. (1pt) The data type of argument to mathematical function should be numeric.

Now write code that takes the natural log of list_1 (using the apply family of functions). (1pt)

```
log_list<-list(loga = log(list_1[[1]]),logb =log(list_1[[2]]))
print(log_list)
```

```
## $loga
## [1] 0.6931472 0.6931472 0.0000000 1.6094379 2.1972246
##
## $logb
##  [1] 0.6931472 1.0986123 2.1972246 0.6931472 1.9459101 2.0794415 1.9459101
##  [8] 2.1972246 0.6931472 0.6931472
```

## Part 3. Try "tapply" function and its equivalents. (12pts)

we will use the data set "mtcars"; familiarize yourself with it first. Here is the code to load the data:

```
data(mtcars)
```

**(8) Subset 4 columns "mpg", "hp", "wt" and "am" to a new data frame, named df_car. (1pt) In df_car, convert "am" to a factor variable with two levels: "automatic" and "manual" (2pts) (Hint: read help documentation of mtcars)**

```
df_car<-data.frame(mpg=mtcars$mpg,hp=mtcars$hp,wt=mtcars$wt,am =mtcars$am)
df_car$am = ifelse(df_car$am==0,"automatic","manual")
head(df_car)
```

```
##    mpg  hp    wt        am
## 1 21.0 110 2.620    manual
## 2 21.0 110 2.875    manual
## 3 22.8  93 2.320    manual
## 4 21.4 110 3.215 automatic
## 5 18.7 175 3.440 automatic
## 6 18.1 105 3.460 automatic
```

**(9) Use "tapply" function on df_car to find the mean of mpg by different am levels. (2pts)**

```
mean_mp<-tapply(df_car$mpg,df_car$am=="manual",mean)
names(mean_mp)<-c("automatic","manual")
print(mean_mp)
```

```
## automatic    manual
##  17.14737  24.39231
```

**(10) Look up the function "by". Obtain the mean of mpg, hp and wt, by different am levels, using only one function call. (2pts)**

```
by(df_car[,1:3],df_car[,4],colMeans)
```

```
## df_car[, 4]: automatic
##        mpg         hp         wt
##  17.147368 160.263158   3.768895
## ------------------------------------------------------------
## df_car[, 4]: manual
##       mpg        hp        wt
##  24.39231 126.84615   2.41100
```

**(11) Do question(10) again by using the "aggregate" function. You may need to look this up as well. (2pts)**

```
aggregate(df_car[,1:3], list(df_car[,4]),mean)
```

```
##     Group.1      mpg       hp       wt
## 1 automatic 17.14737 160.2632 3.768895
## 2    manual 24.39231 126.8462 2.411000
```

**(13) Same as question(10), but this time use the combination of "apply" and "tapply" functions to get the same results. (3pts)**

**Your code here**

```
sort<-function(x){
  tapply(x,df_car$am=="manual",mean)
}
result<-apply(df_car[,1:3], 2, sort)
rownames(result)<-c("automatic","manual")
print(result)
```

```
##                mpg       hp       wt
## automatic 17.14737 160.2632 3.768895
## manual    24.39231 126.8462 2.411000
```

## Part 4. More functions in "apply" family (4pts)

**(14) "mapply" function is a multivariate version of "sapply".**

Create list_2 by following code:

```
list_2 <- list(a=one_matrix[,2],b=c(one_matrix[,3],one_matrix[,4]))
```

Add up corresponding elements in list_1 and list_2, then take natural log of it. (2pts)

```
list3<-c(a=list(rep(0,5)),b = list(rep(0,10)))
add<-function(x){
  list3[[x]] <- list_1[[x]]+list_2[[x]]
}
result14<-mapply(add,1:length(list_2))
mapply(log,result14)
```

```
## [[1]]
## [1] 1.386294 1.609438 2.302585 1.945910 2.772589
##
## [[2]]
##  [1] 2.302585 2.302585 2.890372 1.386294 2.197225 2.397895 2.772589
##  [8] 2.772589 1.791759 1.609438
```

### (15) "rapply" function is used to apply a function to all elements of a list recursively.

Create list_3 by following code:

```
list_3 <- list(aa=one_matrix[,1],b=c("this","is","character"))
```

Calculate the natural log of all integer in list_3. (2pts) Do not remove characters in the list or subsetting.

```
logint<-function(x){
if (class(x) == 'integer')
    y = log(x)
  else
    y =x
}
rapply(list_3,logint)
```

```
##                 aa1                 aa2                 aa3
## "0.693147180559945" "0.693147180559945"                 "0"
##                 aa4                 aa5                  b1
##   "1.6094379124341"  "2.19722457733622"              "this"
##                  b2                  b3
##                "is"         "character"
```

## Part 5. Linear regression (9pts)

### (16) Look back in the "iris" dataset.

Fit a simple linear regression model using lm() to predict Petal.Length from Petal.Width. (2pts) How do you interpret the result of regression? Hint: interpret the two coefficients from the output of lm(). (2pts)

```
data(iris)
lm(Petal.Length~Petal.Width, data = iris)
```

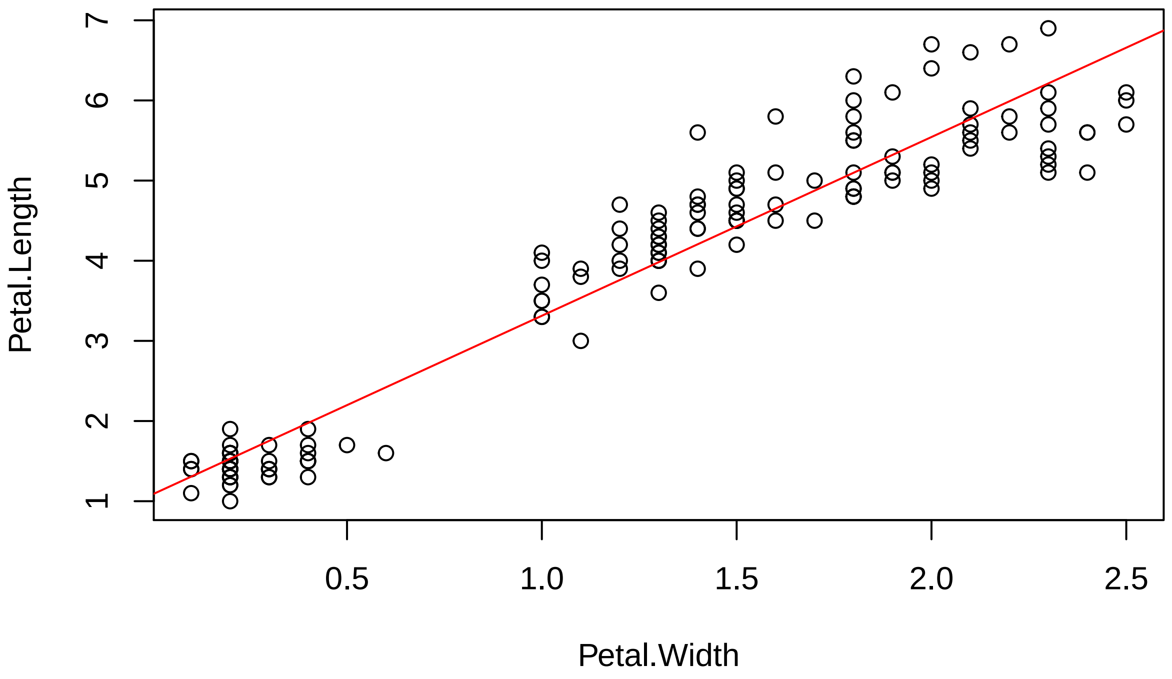
```
##
## Call:
## lm(formula = Petal.Length ~ Petal.Width, data = iris)
##
## Coefficients:
## (Intercept)  Petal.Width
##       1.084        2.230
```

1.084 is intercept, which means when `Petal.Length` equals to 0 the `Petal.Width` estimated by this model is 1.084, which means when `Petal.Width` equals to 0,`Petal.Length` equals to 1.084.
2.230 : Each 1 unit increases in `Petal.Length` will increase `Petal.Width` by 2.230, which means there exist a positive linear relationship between `Petal.Width` and `Petal.Width`.

### (17) Create a scatterplot with x-axis of Petal.Width and y-axis of Petal.Length. (2pt)

Add the linear regression line you found above to the scatterplot. (1pt) Provide an interpretation for your plot. (2pts)

```
plot(x=iris$Petal.Width,y=iris$Petal.Length,xlab = "Petal.Width",ylab="Petal.Length")
abline(lm(Petal.Length~Petal.Width, data = iris),col = "red")
```



As we can see from the graph above, basically, the regression line fits the actual value very well and for actual value, there exists a positive relationship between `Petal.Length` and `Petal.Width`. And, according to the graph above, the intercept for regression line is about 1.1, while the slope is about 2.