Stat 292 - Recitation 2

Apply Family

Orçun Oltulu

24 / 3 / 2021

Exercise 1:

Part A:

Create a 20x5 matrix called A, fill with randomly generated numbers. You can generate random numbers using following command.

```
set.seed(292)
mysample <- round(rnorm(100,5,2),1)
A <- matrix(mysample, nrow = 20, dimnames = list(paste("ID.",1:20,sep=""),
                                                  paste("var.",1:5,sep="")))
```

Part B:

Find the mean of 2^{nd} and 4^{th} columns separately.

```
mean(A[,2])
```

```
## [1] 5.73
```

```
mean(A[,4])
```

```
## [1] 4.4
```

```
# Alternatively
apply(A[,c(2,4)],2,mean)
```

```
## var.2 var.4
##  5.73  4.40
```

Part C:

Find the mean of each row. Try different methods, including 'apply' function.

```r
# 1st method - easy to think but slowest
row_means <- numeric()
for(i in 1:nrow(A)){
  row_means[i] <- mean(A[i,])
}

# 2nd method - cool way doing this
rowMeans(A)
```

```
##  ID.1  ID.2  ID.3  ID.4  ID.5  ID.6  ID.7  ID.8  ID.9 ID.10 ID.11 ID.12 ID.13
##  5.08  5.78  6.32  4.78  5.48  5.88  6.28  5.60  4.00  4.96  2.40  6.02  4.28
## ID.14 ID.15 ID.16 ID.17 ID.18 ID.19 ID.20
##  3.94  5.76  4.64  7.26  3.86  6.42  4.66
```

```r
# 3rd method - cool & generalized way
apply(A, 1, mean)
```

```
##  ID.1  ID.2  ID.3  ID.4  ID.5  ID.6  ID.7  ID.8  ID.9 ID.10 ID.11 ID.12 ID.13
##  5.08  5.78  6.32  4.78  5.48  5.88  6.28  5.60  4.00  4.96  2.40  6.02  4.28
## ID.14 ID.15 ID.16 ID.17 ID.18 ID.19 ID.20
##  3.94  5.76  4.64  7.26  3.86  6.42  4.66
```

Part D:

Find the number of values that are less than the mean of each column.

```r
apply(A, 2, function(x) sum(x < mean(x)))
```

```
## var.1 var.2 var.3 var.4 var.5
##     9    13    10    11    12
```

Part E:

Calculate the variance of each column.

```r
apply(A,2, var)
```

```
##    var.1    var.2    var.3    var.4    var.5
## 3.346816 4.865368 3.377789 3.337895 3.887237
```

Part F:

Find the minimum and maximum of each column.

```r
apply(A,2, function(x){ c(min(x),max(x))})
```

```
##      var.1 var.2 var.3 var.4 var.5
## [1,]   1.1   2.3   0.5   0.1   2.8
```

```
## [2,]   8.0   9.8   7.5   7.6   9.8
```

```r
# Alternatively, use range function
apply(A,2, range)
```

```
##      var.1 var.2 var.3 var.4 var.5
## [1,]   1.1   2.3   0.5   0.1   2.8
## [2,]   8.0   9.8   7.5   7.6   9.8
```

Part G:

- First **Apply** shapiro-wilk test to test the normality of each variable. (**Hint:** you can use shapiro.test() function)

```r
apply(A, 2, shapiro.test)
```

```
## $var.1
##
##  Shapiro-Wilk normality test
##
## data:  newX[, i]
## W = 0.98146, p-value = 0.9515
##
##
## $var.2
##
##  Shapiro-Wilk normality test
##
## data:  newX[, i]
## W = 0.87791, p-value = 0.01623
##
##
## $var.3
##
##  Shapiro-Wilk normality test
##
## data:  newX[, i]
## W = 0.94844, p-value = 0.3441
##
##
## $var.4
##
##  Shapiro-Wilk normality test
##
## data:  newX[, i]
## W = 0.97526, p-value = 0.8596
```

```
##
##
## $var.5
##
##  Shapiro-Wilk normality test
##
## data:  newX[, i]
## W = 0.95556, p-value = 0.4594
```

- Now, try to get only the p-values of shapiro.test() as a vector. Make comment on the p-values.

```
apply(A, 2, function(x) shapiro.test(x)$p.val)
```

```
##      var.1      var.2      var.3      var.4      var.5
## 0.95151742 0.01623124 0.34407128 0.85960321 0.45938901
```

Part H:

Using apply() function, find the product of the rows in A matrix.

```
apply(A, 1, prod)
```

```
##        ID.1        ID.2        ID.3        ID.4        ID.5        ID.6
##  2832.09696  5990.98500  7127.24184  1931.98200  3810.73920  6312.93696
##        ID.7        ID.8        ID.9       ID.10       ID.11       ID.12
##  8716.97376  3840.63960   833.81760  2561.51000    20.99020  7716.74688
##       ID.13       ID.14       ID.15       ID.16       ID.17       ID.18
##   996.26064   546.27300  5127.59520  1341.36000 18752.81760    43.25616
##       ID.19       ID.20
##  8517.41520  2117.83880
```

Exercise 2:

Part A:

Create a list using following code block.

```
set.seed(292)
mylist <- list(item1 = seq(-6,20,2),
               item2 = round(rnorm(10),2),
               item3 = round(runif(20,-1,1),1))
```

Part B:

Using lapply(), find the length of mylist's observations.

```r
lapply(mylist, length)
```

```
## $item1
## [1] 14
##
## $item2
## [1] 10
##
## $item3
## [1] 20
```

Part C:

Using lapply(), find the sums of mylist's observations.

```r
lapply(mylist, sum)
```

```
## $item1
## [1] 98
##
## $item2
## [1] 2.37
##
## $item3
## [1] -2.9
```

Part D:

Use lapply() to find the quantiles of mylist.

```r
lapply(mylist, quantile)
```

```
## $item1
##   0%  25%  50%  75% 100%
## -6.0  0.5  7.0 13.5 20.0
##
## $item2
##     0%    25%    50%    75%   100%
## -1.100 -0.225  0.200  0.750  1.490
##
## $item3
##    0%   25%   50%   75%  100%
## -0.90 -0.60 -0.35  0.30  0.70
```

Part E:

Find the unique values in mylist.

Part F:

Create an R function using following rule;

$$
\text{f(x)} = \begin{cases} \frac{2}{|x|}, & x < 0 \\ x^2, & 0 \le x \le 1 \\ x + 2, & x > 1 \end{cases}
$$

Also, get the results as rounded to second decimals.

```
myfunction <- function(x){
  round(
    ifelse(x < 0,2/abs(x),
           ifelse(x > 1, x+2, x^2))
    ,2)
}
```

Part G:

Apply that function to your list, store the outputs in another list, called new_list.

Make sure your function is built for that purpose ! You may get some errors first, try to update your function to make it work for comparisons in vector level.

```
new_list <- lapply(mylist, myfunction)
```

Part H:

Find the range of each item in new_list.

```
lapply(new_list,range)
```

```
## $item1
## [1]  0 22
##
## $item2
## [1]   0.17 200.00
##
## $item3
## [1] 0.01 6.67
```

Part I:

Create another, using following codes, add this item to new_list.

```
set.seed(292)
new_list$item4 <- sample(letters, 30, replace = T)
```

Part J:

Find the classes of new_list's sub-variables, with lapply().

```
lapply(new_list, class)
```

```
## $item1
## [1] "numeric"
##
## $item2
## [1] "numeric"
##
## $item3
## [1] "numeric"
##
## $item4
## [1] "character"
```

Part K:

Find the duplicated values in each item in new_list.

```
lapply(new_list, function(x) x[duplicated(x)])
```

```
## $item1
## numeric(0)
##
## $item2
## numeric(0)
##
## $item3
## [1] 2.50 0.09 0.09 3.33 4.00 0.36 5.00
##
## $item4
##  [1] "l" "k" "v" "z" "v" "l" "g" "x" "n" "b" "j"
```

Exercise 3:

Part A:

Create following matrices and store them in a list.

$$X = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 \\ 1 & 0 & -1 & 0 \end{bmatrix}, \quad Y = \begin{bmatrix} 11 & -2 & -2 & 0 \\ 10 & 0 & 1 & 1 \\ -1 & 5 & 2 & -4 \\ 2 & 4 & 6 & -1 \end{bmatrix}, \quad Z = \begin{bmatrix} 5 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 8 \end{bmatrix}$$

```
X <- matrix(rep(c(1,0,-1,0),4), nrow = 4, byrow = TRUE)
Y <- matrix(c(11,-2,-2,0,10,0,1,1,-1,5,2,-4,2,4,6,-1),
            nrow = 4, byrow = TRUE)
Z <- diag(5:8)

matrix_list <- list(X,Y,Z)
```

Part B:

Using sapply() function find the maximum value of the determinants, max{|X|,|Y|,|Z|}.

```
max(sapply(matrix_list, det))
```

```
## [1] 1680
```

Part C:

Find which one has the maximum determinant value.

```
which.max(sapply(matrix_list, det))
```

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

Exercise 4:

Part A:

Load Auto data set in ISLR package, then drop ‘name’ variable. Convert cylinders and origin to factors.

```
library(ISLR)
data(Auto)
Auto$name <- NULL
Auto$cylinders <- factor(Auto$cylinders)
Auto$origin <- factor(Auto$origin)
str(Auto)
```

```
## 'data.frame':    392 obs. of  8 variables:
##  $ mpg         : num  18 15 18 16 17 15 14 14 14 15 ...
##  $ cylinders   : Factor w/ 5 levels "3","4","5","6",..: 5 5 5 5 5 5 5 5 5 5 ...
##  $ displacement: num  307 350 318 304 302 429 454 440 455 390 ...
```

```
##  $ horsepower  : num  130 165 150 150 140 198 220 215 225 190 ...
##  $ weight      : num  3504 3693 3436 3433 3449 ...
##  $ acceleration: num  12 11.5 11 12 10.5 10 9 8.5 10 8.5 ...
##  $ year        : num  70 70 70 70 70 70 70 70 70 70 ...
##  $ origin      : Factor w/ 3 levels "1","2","3": 1 1 1 1 1 1 1 1 1 1 ...
```

Part B:

Find the median MPG for each origin. (**Hint:** Use tapply() function)

```
tapply(Auto$mpg, Auto$origin, median)
```

```
##    1    2    3
## 18.5 26.0 31.6
```

Part C:

Find the range of each numeric variable using lapply() function.

```
lapply(Auto, function(x){
  if(is.numeric(x)){
    range(x)
  }else{
    "Not Numeric"
  }
})
```

```
## $mpg
## [1]  9.0 46.6
##
## $cylinders
## [1] "Not Numeric"
##
## $displacement
## [1]  68 455
##
## $horsepower
## [1]  46 230
##
## $weight
## [1] 1613 5140
##
## $acceleration
## [1]  8.0 24.8
##
## $year
```

```
## [1] 70 82
##
## $origin
## [1] "Not Numeric"
```

```
# alternatively,
#lapply(Auto[sapply(Auto, is.numeric)], range)
# alternatively,
#lapply(Filter(is.numeric, Auto), range)
```

Exercise 5:

Part A:

Import 'data.txt' file.

```
data <- read.table("data.txt", header = TRUE)
head(data)
```

```
##   representative sales territory
## 1              1    83    Area.1
## 2              2    61    Area.1
## 3              3    75    Area.1
## 4              4    54    Area.1
## 5              1    61    Area.2
## 6              2    92    Area.2
```

Part B:

Using mapply(), find the classes of each variable.

```
mapply(class, data)
```

```
## representative          sales      territory
##      "integer"      "integer"    "character"
```

Part C:

Use mapply() to paste "representative", "sales", and "territory", with the "MoreArgs=" argument of "list(sep="-")

- Example output for the first two rows; "1-83-Area.1", "2-61-Area.1"

```
mapply(paste,
       data$representative, data$sales, data$territory,
       MoreArgs=list(sep="-"))
```

```
##  [1] "1-83-Area.1" "2-61-Area.1" "3-75-Area.1" "4-54-Area.1" "1-61-Area.2"
```

```
##  [6] "2-92-Area.2" "3-92-Area.2" "4-72-Area.2" "1-69-Area.3" "2-73-Area.3"
## [11] "3-59-Area.3" "4-66-Area.3" "1-71-Area.4" "2-99-Area.4" "3-71-Area.4"
## [16] "4-93-Area.4"
```

Part D:

Find the mean Sales for each Territory.

```r
tapply(data$sales, data$territory, mean)
```

```
## Area.1 Area.2 Area.3 Area.4
##  68.25  79.25  66.75  83.50
```

Exercise 6:

Part A:

Create a sequence from 1 to 100. Then, split the sequence 10 groups.

```r
xGrid <- 1:100
seq_list <- split(xGrid, gl(10,10))
```

Part B:

Find the median of each sub-group.

```r
lapply(seq_list, median)
```

```
## $`1`
## [1] 5.5
##
## $`2`
## [1] 15.5
##
## $`3`
## [1] 25.5
##
## $`4`
## [1] 35.5
##
## $`5`
## [1] 45.5
##
## $`6`
## [1] 55.5
##
```

```
## $`7`
## [1] 65.5
##
## $`8`
## [1] 75.5
##
## $`9`
## [1] 85.5
##
## $`10`
## [1] 95.5
```

Part C:

Load iris data set from ISLR package.

```
library(ISLR)
data(iris)
```

Part D:

Split the iris data set into 3 chunks by Species.

```
spl_iris <- split(iris, iris$Species)
```

Part E:

Calculate means of each 'numeric' variable for each Species separately.

```
lapply(spl_iris, function(x) {
  colMeans(x[sapply(x,is.numeric)])
})
```

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

```
# can also be done with sapply
sapply(spl_iris, function(x) {
```

```
  colMeans(x[sapply(x,is.numeric)])
})
```

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

Bonus Exercise:

Create an R function that returns the given number of rows of Pascal Triangle without using any loop.

(**Hint:** You can use choose() function to get the sequences. You can get further information for the relationship 'Pascal Triangle' and 'Combination')

```
Pascal_Function <- function(n){
  sapply(0:n, function(x) choose(0:n, x))
}
```

Example output for n = 5

```
Pascal_Function(5)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,]    1    0    0    0    0    0
## [2,]    1    1    0    0    0    0
## [3,]    1    2    1    0    0    0
## [4,]    1    3    3    1    0    0
## [5,]    1    4    6    4    1    0
## [6,]    1    5   10   10    5    1
```