Homework 2: STAT5400

Dev Narayan Baiju

09/12/2024

Submit your solutions as an .Rmd file and accompanying .pdf file. Include all the **relevant** R code and output. With the `echo` option in R markdown, you opt to hide some R code that is not very relevant (but still run it to generate the document).

Always comment on your result whenever it is necessary. For example, in problem 3, display and also comment on the Q-Q plot that you produce

Reading assignments.

Read Chapters 2-4, 7 of *Using R for Data Analysis and Graphics*. https://cran.r-project.org/doc/contrib/usingR.pdf.

Problems

1. Use the `system.time` function in R to time the performance of the same task in two different ways:

- Generate a vector of 500,000 random variables from a Normal (0, 1) density and use the `sum` function to calculate their sum.
- Create a variable called `answer` and initialize it to 0. Then, using a `for` loop, do the following steps 500,000 times: generate a single Normal (0, 1) value and add it to sum contained in `answer`.
- Besides including the R code and output, compare on the relevant timings for both methods and state which one is more efficient.

```
longvec <- rnorm(500000, 0, 1)
mysum <- function(){
  answer = 0
  for(i in 1:500000){
    answer <- answer + rnorm(1, 0, 1)
    }
  return(answer)
}
system.time(sum(longvec))
```

```
##    user  system elapsed
##   0.002   0.000   0.002
```

```
system.time(mysum())
```

```
##    user  system elapsed
##   0.575   0.013   0.593
```

We can see that generating 500000 numbers and finding their sum at once takes lesser time than using a for loop for generating normal values 500000 times and finding their sum.

2. Use R to do the following:

- Create a matrix called M with the following entires:

$$\begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \\ 3 & 6 & 9 \end{bmatrix}.$$

- Create a vector called v with the following entries: `17 46 181`.
- Compute and display the product Mv produced by matrix multiplication.
- Compute and display the transpose of M.
- Display only those elements of v that have values less than 50.

Matrix M and vector v are created.

```
M <- matrix(c(1:9), 3, 3, byrow = FALSE)
v <- c(17,46,181)
```

Matrix multiplication of M and v is executed.

```
print(M %*% v)
```

```
##      [,1]
## [1,] 1468
## [2,] 1712
## [3,] 1956
```

The transpose of M is computed and displayed.

```
print(t(M))
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    4    5    6
## [3,]    7    8    9
```

Values less than 50 within vector v are displayed.

```
print(v[which(v<50)])
```

```
## [1] 17 46
```

3.

- Use either `help.search` function or just google it to locate a package that contains a function to compute the skewness of a vector of numbers. Make sure that it uses the standard definition of skewness. What is the name of the function, and which package is it in? *We can find the skewness() function within the 'moments' library/package.*
- Locate an R function that computes the five-number summary of a vector of numbers. What is the name of the function, and which package is it in? *The in-built function fivenum() can be used to get the five-number summary of a vector of numbers.*
- Write an R function that does the following:
  - Accepts one argument: a vector.
  - Checks whether the vector is numeric.
  - If not, displays the message `Vector must be numeric` and exist.
  - If yes, computes the skewness of the values (after removing any missing values).
    - If the absolute value of skewness is less than 1, return a list containing two objects: skewness in an object named `skewness`; a vector consisting of the mean and standard deviation in an object named `dsescstats`.

    - Otherwise, returns a list containing two objects: skewness in an object named `skewness`; a vector consisting of the five-number summary in an object named `descstats`.
  - Run your function in R three times, using the following vectors as arguments:
    - `c("stat", "actuarial", "2022")`
    - `rnorm(100)`
    - `rexp(5)`
  - In the document that you submit for homework, show both the R code and output for the three calls to it.

The function for the above problem is as follows:

```
library(moments)

info_func <- function(myvec){
  if(is.numeric(myvec) == TRUE){
    myvec <- myvec[!is.na(myvec)]
    myvec_skew <- skewness(myvec)
    if(myvec_skew < 1){
      skewness <- myvec_skew
      dsescstats <- c(mean(myvec), sd(myvec))
      return(list(skewness, dsescstats))
    }
    else{
      skewness <- myvec_skew
      descstats <- fivenum(myvec)
      return(list(skewness, descstats))
    }
  }
  else{
    stop("Vector must be numeric")
  }
}
```

Now let us run the three vectors through the function. Trying first with the vector c("stat", "actuarial", "2022")

```
#let us store the given vectors in variables
trialvec1 <- c("stat", "actuarial", "2022")
print(info_func(trialvec1))
```

```
## Error in info_func(trialvec1): Vector must be numeric
```

Now with rnorm(100) vector

```
trialvec2 <- rnorm(100)
print(info_func(trialvec2))
```

```
## [[1]]
## [1] -0.36771
##
## [[2]]
## [1] 0.1122804 0.9844465
```

Now with rexp(5)

```
trialvec3 <- rexp(5)
print(info_func(trialvec3))
```

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

```
## [1] 0.001299269
##
## [[2]]
## [1] 1.2791166 0.8494596
```

4. Review the example on two-sample t-test on slide 45 of S2P1.pdf. If you are not familiar with t-test, you may google some online tutorials.

- Imagine we have run 1000 experiments (rows) and each of which collects data on 50 individuals (columns). The first 25 individuals in each experiment are assigned to group 1 and the rest to group 2.
- You may imagine in practice we only have 50 observations, 25 of which belongs to group 1. With simulations, we have luxury to generate 1000 samples, each of which has 50 observations, to explore the distribution of t-statistics.
  - We first generate some random data to represent this problem.

```
set.seed(1)
m <- 1000
n <- 50
X <- matrix(rnorm(m * n, mean=10, sd=3), nrow=m)
grp <- rep(1:2, each=n/2)
```

- For the first sample (i.e., the first row), compute the t-statistic manually. Compare your result with the output from the following code.

```
t.test(X[1, grp==1], X[1, grp==2])$stat
```

```
##          t
## -0.5284632
```

Manually computed t-statistic for the first row is as follows

```
x1 = X[1, grp == 1]
x2 = X[1, grp == 2]
manual_tstat <- (mean(x1) - mean(x2))/sqrt((sd(x1)*sd(x1)/25) + (sd(x2)*sd(x2)/25))
print(manual_tstat)
```

```
## [1] -0.5284632
```

- Use R to compute the t-statistics for all 1000 samples. Try to optimize your code to make it efficient. *The code containing the function to compute t-statistics for all 1000 samples is as follows*

```
myfunc_tstat <- function(X, grp){
  tstat_1000 = rep(NA, times <- 1000)
  for (i in 1:1000) {
    x1 <- X[i, grp == 1]
    x2 <- X[i, grp == 2]
    tstat_1000[i] <- (mean(x1) - mean(x2))/sqrt((sd(x1)*sd(x1)/25) + (sd(x2)*sd(x2)/25))
  }
  return( tstat_1000)
}
```

- Check the R code in the end of section 24.7 on https://adv-r.hadley.nz/perf-improve.html#t-test. Use `system.time` to compare the computing time of the R function provided on the webpage with your own function. Comment on why your code is slower or faster. Don't check their code before you finish writing your own code.

```
rowtstat <- function(X, grp){
  t_stat <- function(X) {
```

```
    m <- rowMeans(X)
    n <- ncol(X)
    var <- rowSums((X - m) ^ 2) / (n - 1)

    list(m = m, n = n, var = var)
  }

  g1 <- t_stat(X[, grp == 1])
  g2 <- t_stat(X[, grp == 2])

  se_total <- sqrt(g1$var / g1$n + g2$var / g2$n)
  (g1$m - g2$m) / se_total
}

system.time(t3 <- rowtstat(X, grp))
```

```
##    user  system elapsed
##   0.015   0.001   0.015
```

```
system.time(t4 <- myfunc_tstat(X, grp))
```

```
##    user  system elapsed
##   0.073   0.002   0.076
```

My function for t statistics is slower than the function from the Advanced R website probably because I used the mean and variance functions built in within R to compute my results instead of using rowMeans function for mean and rowSums function for computing variance

Also, it is to be noted that the Advanced R function was checked before my function was fully completed and the arguments within my function were edited subsequently.