HW3_Do_Quyen

Quyen Do

September 6, 2018

Problem 4

Some of my takeaways from Google's R programming styles and Hadley Wickham's are naming of variables and functions (use nouns for variables and verbs for function names). Commenting is an important aspect covered in both guidelines. Proper commenting helps the readers understand the code and also help the programmer trace his/her steps while looking back at the script. I particularly like Google's instructions on function documentation that needs to include a descriptive comment on the function's purpose and its required arguments and return values. All in all, the style needs to be consistent throughout the script.

I will pay more attention to the variable and function naming, spacing and indentation when creating my scripts. I will also spend more time on documenting created functions and make sure I include all necessary information. I will also utilize tools like Lintr to keep my style consistent and be aware of common mistakes.

Problem 5

Some common things I need to change in my code are:

- Putting a space after commas
- Putting spaces around all infix operators
- Keeping lines under 80 characters
- Opening curly braces should not be on their own line
- Using lower cases for variable and function names

Problem 6

a. Create a single table of the means, sd, and correlation for each of the 13 Observers in data.rds

```
data <- readRDS("../03_good_programming_R_functions/HW3_data.rds")

summarizeVectors <- function (dev1,dev2) {
  # Calculates mean, sd and correlation of 2 vectors and return the results in a data frame

  # Args:
  # dev1: One of the two vectors whose mean, sd and correlation is to be calculated
  # dev2: The other vector. dev1 and dev2 must have the same length and no missing values.

  # Returns:
  #  A data frame with 4 columns (2 columns for the mean of each vector, 2 columns for the sd of each v

  #Error handling
  n <- length(dev1)
  if (n <= 1 || n != length(dev2)) {
    stop ("Arguments dev1 and dev2 have different lengths.")
  }
```

```r
  if (TRUE %in% is.na(dev1) || TRUE %in% is.na(dev2)) {
    stop ("There is missing value within argument dev1 or dev2.")
  }

  # Calculate mean, sd and correlation of dev1 and dev2
  mean_dev1 <- mean(dev1)
  mean_dev2 <- mean(dev2)
  sd_dev1 <- sd(dev1)
  sd_dev2 <- sd(dev2)
  correlation <- cor(dev1,dev2)

  # Put the summary statistics in a data frame
  df <- data.frame(mean_dev1, mean_dev2, sd_dev1, sd_dev2, correlation)

  return(df)
}

# Create a dataframe to hold the results
Observers_summary <- data.frame(matrix(ncol = 6,nrow = 0))
names(Observers_summary) <- c("Observer", "mean_dev1", "mean_dev2",
                              "sd_dev1","sd_dev2","correlation")

# Loop through each observer's data to calculate the necessary statistics
for (obs in unique(data$Observer))
{
  current_dev1 <- data[which(data$Observer ==obs),"dev1"]
  current_dev2 <- data[which(data$Observer ==obs),"dev2"]
  result <- summarizeVectors(current_dev1,current_dev2)
  result$Observer <- obs
  Observers_summary <- rbind(Observers_summary,result)

}
# Rearrange the order of columns in the summary table and sort the row by Observer
Observers_summary <- Observers_summary[,c(6,1:5)] %>% arrange(Observer)

#Print the result
kable(Observers_summary,caption="Summary by Observer")
```

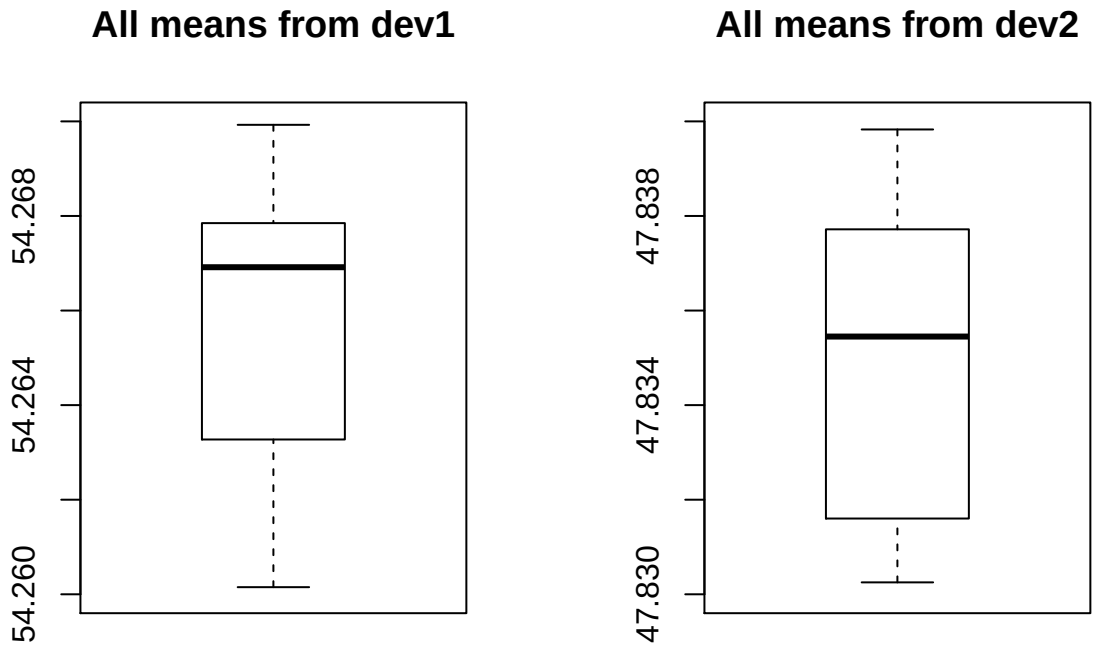
Table 1: Summary by Observer

Observer	mean_dev1	mean_dev2	sd_dev1	sd_dev2	correlation
1	54.26610	47.83472	16.76983	26.93974	-0.0641284
2	54.26873	47.83082	16.76924	26.93573	-0.0685864
3	54.26732	47.83772	16.76001	26.93004	-0.0683434
4	54.26327	47.83225	16.76514	26.93540	-0.0644719
5	54.26030	47.83983	16.76774	26.93019	-0.0603414
6	54.26144	47.83025	16.76590	26.93988	-0.0617148
7	54.26881	47.83545	16.76670	26.94000	-0.0685042
8	54.26785	47.83590	16.76676	26.93610	-0.0689797
9	54.26588	47.83150	16.76885	26.93861	-0.0686092
10	54.26734	47.83955	16.76896	26.93027	-0.0629611
11	54.26993	47.83699	16.76996	26.93768	-0.0694456
12	54.26692	47.83160	16.77000	26.93790	-0.0665752

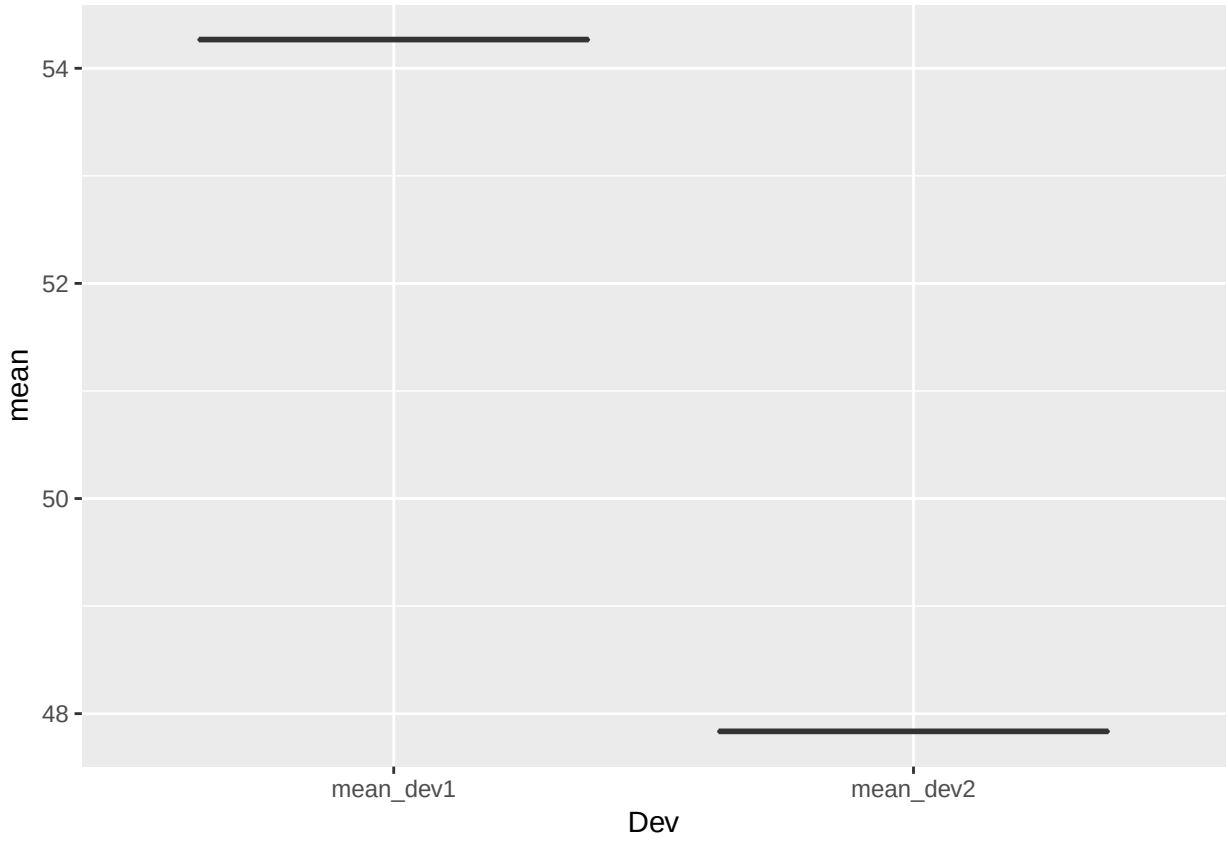
Observer	mean_dev1	mean_dev2	sd_dev1	sd_dev2	correlation
13	54.26015	47.83972	16.76996	26.93000	-0.0655833

b. Create a boxplot of all the means to compare the spread of means from dev1 to dev2

```
par(mfrow=c(1,2))
boxplot(Observers_summary$mean_dev1,main="All means from dev1",ylim=c(54.260,54.270))
boxplot(Observers_summary$mean_dev2,main="All means from dev2",ylim=c(47.830,47.840))
```



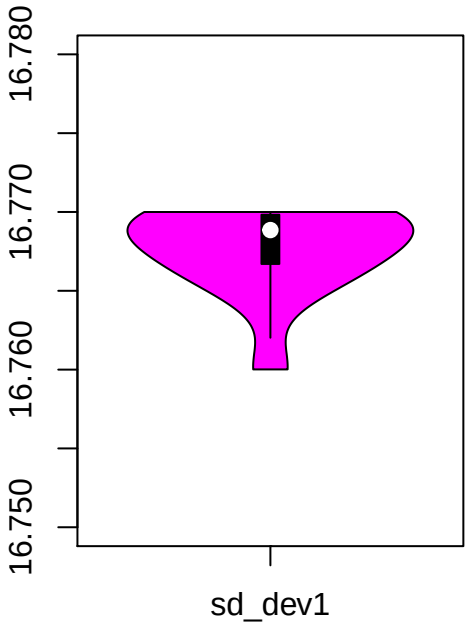

```
#Put them together on one plot
ggplot(data = gather(Observers_summary,key="Dev",value="mean","mean_dev1","mean_dev2"),aes(x=Dev,y=mean
```

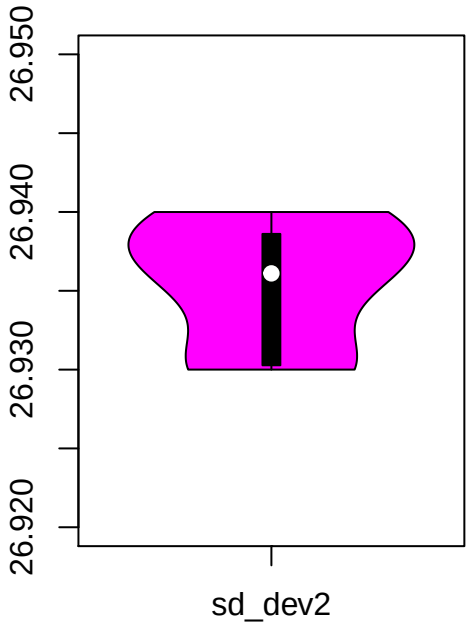


The means from device 1 and device 2 seem to have the same range, however, the distribution of all the means of device 1 skews to the left. Device 2 has a more sysmetric distribution of all the means. Putting the 2 boxplots together in the same graph does not give us much information since the placement of the two boxplots is far apart from each other on the coordinate.

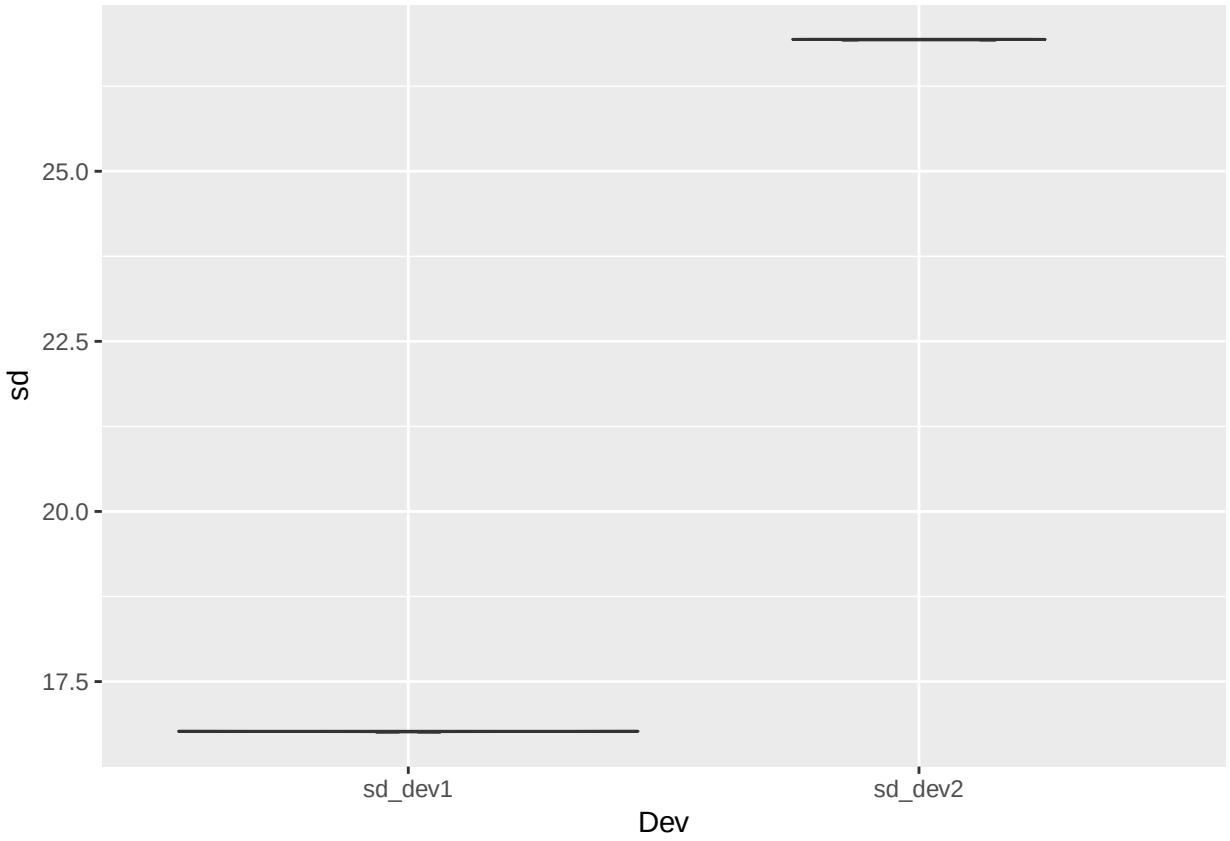
c. Create a violin plot of all means to compare the spread of sds from dev1 to dev2

```
par(mfrow=c(1,2))
vioplot(Observers_summary$sd_dev1,ylim=c(16.750,16.780),names="sd_dev1")
vioplot(Observers_summary$sd_dev2,ylim=c(26.920,26.950),names="sd_dev2")
```






```
ggplot(data = gather(Observers_summary,key="Dev",value="sd","sd_dev1","sd_dev2"),aes(x=Dev,y=sd) ) + ge
```



The distribution of the standard deviations from device 1 and device 2 are very different. Device 1's standard deviation is very right-skewed. As with the means, putting the 2 violin plots on the same graph does not convey much information because of the wide-apart displacement.

Problem 7

Summary tables

```
#Show the first 10 rows of the cleaned data
kable(head(bloodPressure_cleaned,10),caption="First 10 rows of cleaned blood pressure data")
```

Table 2: First 10 rows of cleaned blood pressure data

Day	Reading_By	Reading
1	Dev1	133.34
2	Dev1	110.94
3	Dev1	118.54
4	Dev1	137.94
5	Dev1	139.52
6	Dev1	139.23
7	Dev1	117.96
8	Dev1	119.59
9	Dev1	116.12
10	Dev1	128.38

```r
#Create a summary table
kable(summary(bloodPressure_cleaned),caption="Blood Pressure Data Summary")
```

Table 3: Blood Pressure Data Summary

Day	Reading_By	Reading
Min. : 1	Length:90	Min. :110.8
1st Qu.: 4	Class :character	1st Qu.:125.5
Median : 8	Mode :character	Median :130.4
Mean : 8	NA	Mean :129.0
3rd Qu.:12	NA	3rd Qu.:134.3
Max. :15	NA	Max. :139.6

Problem 8

Find solution to (1) using Newton's method

$$f(x) = 3^x - sin(x) + cos(5x) \tag{1}$$

```r
#Define function (1)
fun1 <- function(x) {
  3^x - sin(x) + cos(5*x)
}

get_solution_using_Newtons_method <- function(guess, FUN, tolerance) {
  # solve for the solution of a function using iterations by Newton's method

  # Args:
  #   guess: the initial solution guess to star the iteration
  #   FUN: the function to be solved
  #   tolerance: the tolerance if absolute difference between 0 and the value of a function
  #              at a point is within this tolerance, then the point is an acceptable solution

  # Returns:
  #   A list containing the solution, the tolerance, and a graph visualzing all iterations

  # Set up the max iteration
  max_iteration <- 1000
  iteration_index <- 0

  # Retrive the derivati ve of FUN using Deriv package
  FUN_deriv <- Deriv(FUN)

  # Find the value of FUN based on guess
  x_n <- guess
  f_n <- FUN(guess)

  # Set up vectors to hold iteration data
  x_vect <- c()
  y_vect <- c()
  deriv_vect <- c()
```

```
  # Set up the plot of iterations to return
  p <- ggplot(data=data.frame(x=0),mapping=aes(x=x))
  p <- p+ stat_function(fun=FUN)

  #Iterations to find the best solution within given tolerance
  while(abs(f_n) > tolerance & iteration_index <= max_iteration){

    #Save iteration data
    x_vect <- c(x_vect,x_n)
    y_vect <- c(y_vect,f_n)
    deriv_vect <- c(deriv_vect,FUN_deriv(x_n))

    #Generate new x to check for solution
    x_n <- x_n - (FUN(x_n)/FUN_deriv(x_n))
    f_n <- FUN(x_n)

    #add visualization of the iteration on the plot
    p <- p + geom_segment(x=x_vect[length(x_vect)],xend=x_n,y=y_vect[length(y_vect)],yend=0)
    p <- p + geom_segment(x=x_n,xend=x_n,y=0,yend=f_n,lty="dashed")

    #update interation_index
    iteration_index <- iteration_index + 1
  }

  # Check if going over maximum iterations
  if (iteration_index > max_iteration) {
    print ("Unable to converge: try different tolerance or initial guess")
  }

  #Loop breaks when a solution within tolerance is found
  #Save the last iteration, which is also the final solution
  x_vect <- c(x_vect,x_n)
  y_vect <- c(y_vect,f_n)


  #Add the iteration points to the plot
  p <- p + geom_point(data.frame(x_vect,y_vect),
                 mapping=aes(x=x_vect,y=y_vect), colour="red")

  p <- p + geom_point(x=x_vect[1],y=y_vect[1],color="darkgreen",size = 2) # The guess point
  p <- p + geom_point(x=x_n,y=f_n,color="blue",size = 2) #  The solution

  #Reframe the axes for the plot
  p <- p + xlim(min(x_vect)-1,max(x_vect)+1) + ggtitle("Newton's method iterations to find root")


  return(list(x_n,tolerance,p))
}
```

For my Newton's method function, I let the user decide on the tolerance so that they can get a solution such the value of the result is within the specified tolerance. I also set maximum iterations at 1000. Any call that goes beyond 1000 will get an error asking for another guess or tolerance. Returned value from the function is a list containing the solution itself, the tolerance as specified by the user and a 'ggplot' object showing the

iterations. The green point is the initial guess from the user and blue point is the final solution estimated by my Newton's method function. All the red dots are different iterations from the method.

```
get_solution_using_Newtons_method(2,fun1,10^-7)
```

```
## [[1]]
## [1] -3.930114
##
## [[2]]
## [1] 1e-07
##
## [[3]]
```