HW2_hwenjun

Wenjun Han

9/25/2020

Problem 3

In the lecture, there were two links to programming style guides. What is your takeaway from this and what specifically are *you* going to do to improve your coding style?

It is very important to have a good habit in providing neat and readable codes, not only for keeping coding history but also help others to understand what we are doing for the data. It is necessary to fully understand the rules and technics for providing easy-understandable codes, such as naming, assignment direction, explicit expression, indent, qualifying namespaces. For me, I will practice more and read more codes to improve my sense of coding.

Problem 5

```
# read into R
device <- data.frame(readRDS("HW3_data.RDS"))
```

Design a single function that can return a vector containing the requirements.

```
device_stat <- function(x=c(1:10),y=c(1:10)){
  # function to calculate mean, std, correlation between two vectors
  x_mean <- mean(x)
  y_mean <- mean(y)

  x_std <- sd(x)
  y_std <- sd(y)

  corr <- cor(x,y)

  mean <- c(x_mean, y_mean)
  std <- c(x_std, y_std)

  result_table <- data.frame(mean, std, corr, rol.names = c("dev1","dev2"))
  # Return all the statistics that we calculate
  return(result_table)
}
```

Then we use the function to run our data.

```
newdf <- c()
for(i in 1:13){
  observer_num <-device[142*i-141,1]
  observer_chunk <-device[(142*i-141):(142*i),]

  review_stat <- device_stat(observer_chunk[,2],observer_chunk[,3])
  review_data <- cbind(observer_num,review_stat)
```

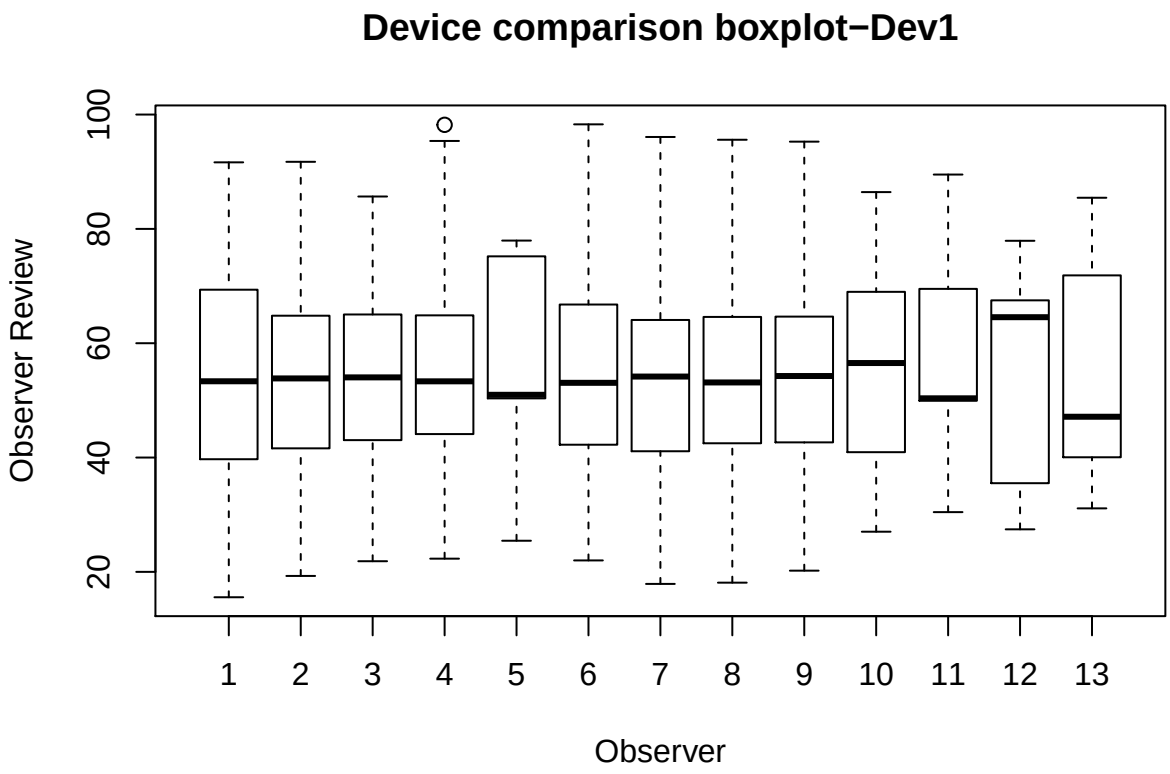
```
  newdf <- rbind(newdf,review_data)
}
newdf
```

```
##    observer_num     mean      std        corr rol.names
## 1             4 54.26327 16.76514 -0.06447185      dev1
## 2             4 47.83225 26.93540 -0.06447185      dev2
## 3             1 54.26610 16.76982 -0.06412835      dev1
## 4             1 47.83472 26.93974 -0.06412835      dev2
## 5             6 54.26144 16.76590 -0.06171484      dev1
## 6             6 47.83025 26.93988 -0.06171484      dev2
## 7            11 54.26993 16.76996 -0.06944557      dev1
## 8            11 47.83699 26.93768 -0.06944557      dev2
## 9            13 54.26015 16.76996 -0.06558334      dev1
## 10           13 47.83972 26.93000 -0.06558334      dev2
## 11           10 54.26734 16.76896 -0.06296110      dev1
## 12           10 47.83955 26.93027 -0.06296110      dev2
## 13            7 54.26881 16.76670 -0.06850422      dev1
## 14            7 47.83545 26.94000 -0.06850422      dev2
## 15            5 54.26030 16.76774 -0.06034144      dev1
## 16            5 47.83983 26.93019 -0.06034144      dev2
## 17            3 54.26732 16.76001 -0.06834336      dev1
## 18            3 47.83772 26.93004 -0.06834336      dev2
## 19            2 54.26873 16.76924 -0.06858639      dev1
## 20            2 47.83082 26.93573 -0.06858639      dev2
## 21            9 54.26588 16.76885 -0.06860921      dev1
## 22            9 47.83150 26.93861 -0.06860921      dev2
## 23            8 54.26785 16.76676 -0.06897974      dev1
## 24            8 47.83590 26.93610 -0.06897974      dev2
## 25           12 54.26692 16.77000 -0.06657523      dev1
## 26           12 47.83160 26.93790 -0.06657523      dev2
```

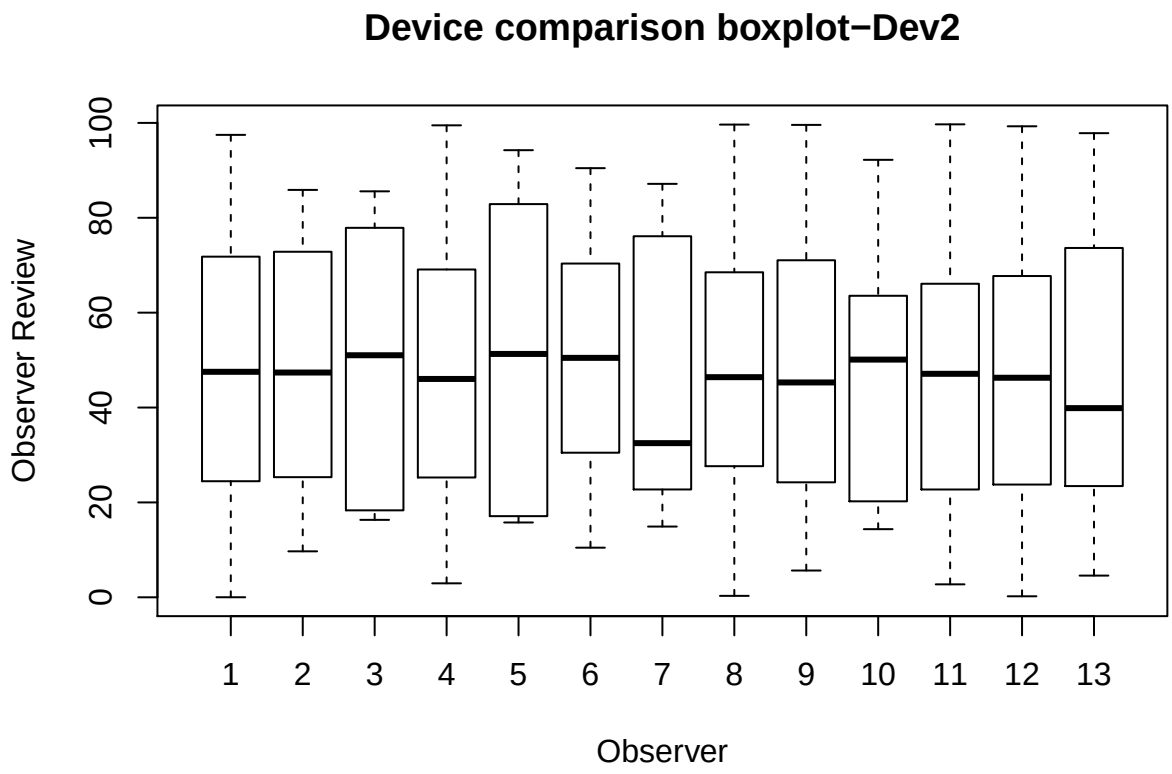
a. A single table of the means, sd, and correlation for each of the 13 Observers is created. From this table, we can conclude that the mean and std of observers look similar. The correlation between two devices is about -0.066, indicating they approximately have no relationship.

b. A box plot of dev, by Observer is created as below.

```
boxplot(device$dev1~device$Observer, xlab="Observer",ylab="Observer Review",
        main="Device comparison boxplot-Dev1")
```



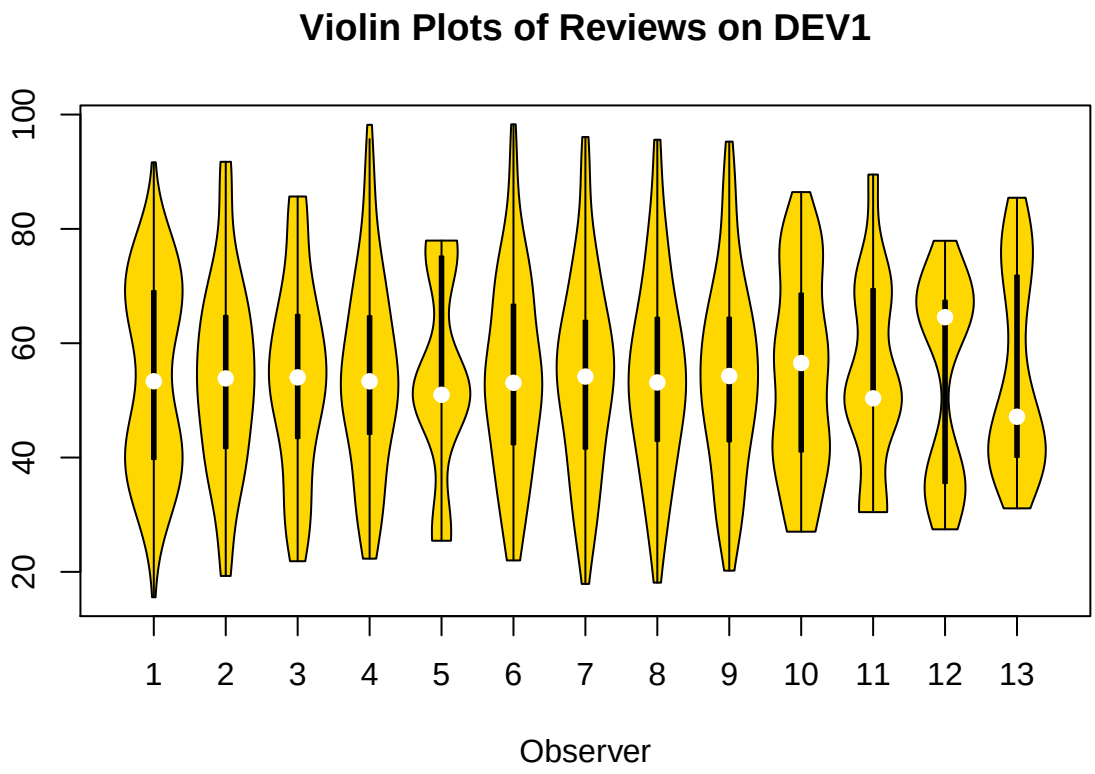

```
boxplot(device$dev2~device$Observer, xlab="Observer",ylab="Observer Review",
        main="Device comparison boxplot-Dev2")
```



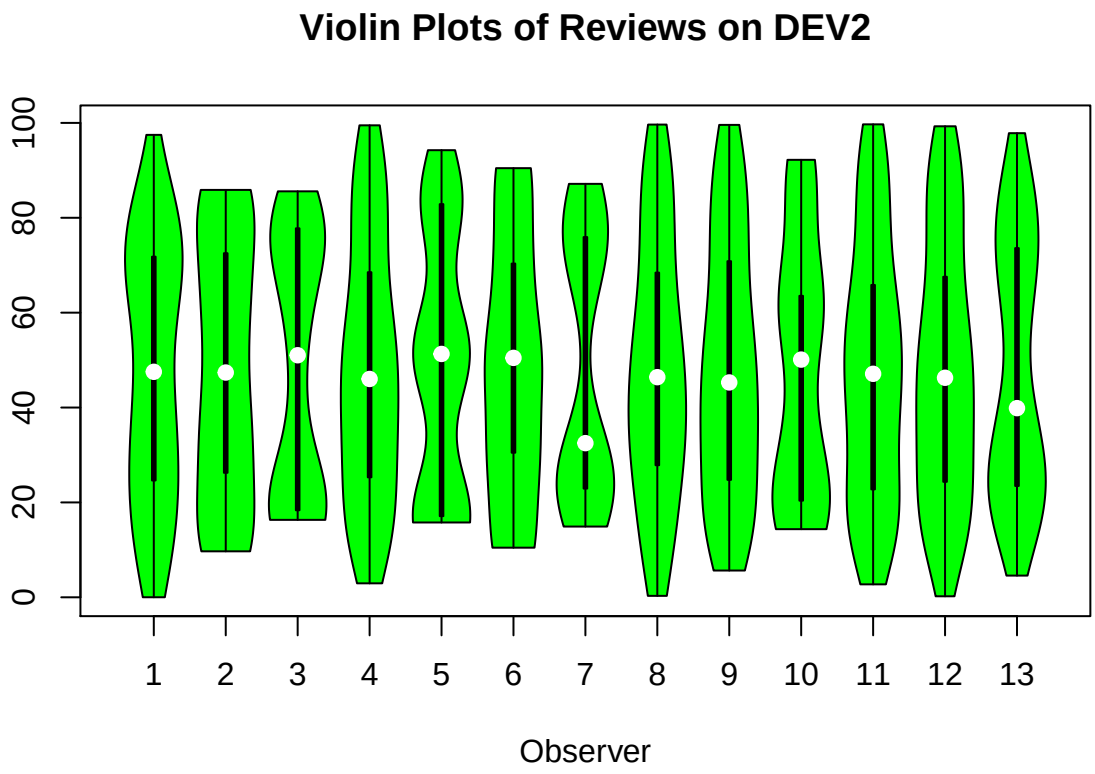
From these plots, we can learn that the range of reviews for dev2 is larger than dev1. The observers tend to provide consistent reviews to dev1.

c. Violin plots of dev by Observer.

```
o1_dev1 <- device$dev1[device$Observer==1]; o1_dev2 <- device$dev2[device$Observer==1]
o2_dev1 <- device$dev1[device$Observer==2]; o2_dev2 <- device$dev2[device$Observer==2]
o3_dev1 <- device$dev1[device$Observer==3]; o3_dev2 <- device$dev2[device$Observer==3]
o4_dev1 <- device$dev1[device$Observer==4]; o4_dev2 <- device$dev2[device$Observer==4]
o5_dev1 <- device$dev1[device$Observer==5]; o5_dev2 <- device$dev2[device$Observer==5]
o6_dev1 <- device$dev1[device$Observer==6]; o6_dev2 <- device$dev2[device$Observer==6]
o7_dev1 <- device$dev1[device$Observer==7]; o7_dev2 <- device$dev2[device$Observer==7]
o8_dev1 <- device$dev1[device$Observer==8]; o8_dev2 <- device$dev2[device$Observer==8]
o9_dev1 <- device$dev1[device$Observer==9]; o9_dev2 <- device$dev2[device$Observer==9]
o10_dev1 <- device$dev1[device$Observer==10]; o10_dev2 <- device$dev2[device$Observer==10]
o11_dev1 <- device$dev1[device$Observer==11]; o11_dev2 <- device$dev2[device$Observer==11]
o12_dev1 <- device$dev1[device$Observer==12]; o12_dev2 <- device$dev2[device$Observer==12]
o13_dev1 <- device$dev1[device$Observer==13]; o13_dev2 <- device$dev2[device$Observer==13]

vioplot(o1_dev1, o2_dev1,o3_dev1,o4_dev1,o5_dev1,
        o6_dev1,o7_dev1,o8_dev1,o9_dev1,o10_dev1,
        o11_dev1,o12_dev1,o13_dev1, col="gold",xlab = "Observer")
title("Violin Plots of Reviews on DEV1")
```




```
vioplot(o1_dev2, o2_dev2,o3_dev2,o4_dev2,o5_dev2,
        o6_dev2,o7_dev2,o8_dev2,o9_dev2,o10_dev2,
        o11_dev2,o12_dev2,o13_dev2, col="green",xlab = "Observer")
title("Violin Plots of Reviews on DEV2")
```



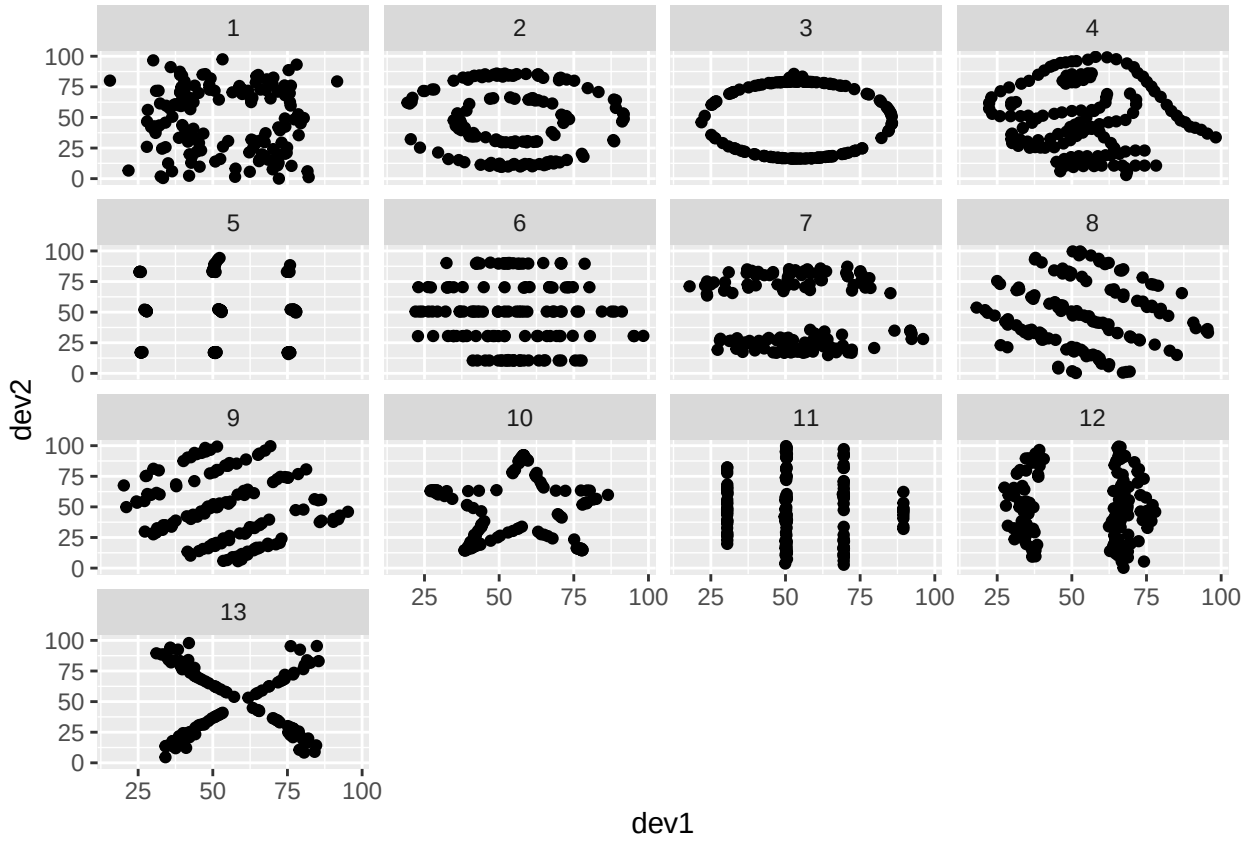
From these plots, we also can conclude the similar results as in previous questions. The reviews in Dev1 is more concentrated. But comparing to boxplot, it can directly shows the distribution of the data, which gives us more information about the data distribution. The vioplot also prove the result of the summary statistics, we can clearly see why the standard deviation of Dev2 is larger than Dev1.

d. a scatter plot of the data using ggplot, geom_points, and add facet_wrap on Observer.

```
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.6.3
```

```
ggplot(device, aes(x=dev1,y=dev2)) + geom_point() + facet_wrap(Observer~.)
```



The review points of Dev1 and Dev2 are actually related and they can be plot as some cute figures. Before we do the analysis, we should plot the scatterplot and see the data first. We need to all the basic visualizations in the "Exploratory Data Analysis" process.

Problem 6

Create a function that uses Reimann sums to approximate the integral:

$$f(x) = \int_0^1 e^{-\frac{x^2}{2}}$$

```
# Use Reimann sums for equation integration, calculate the area under curve
# First we specify the curve function
curv_func <- function(x=1){
  y <- exp(-x^2/2)
  return(y)
}

# We build up a function to achieve the goal
area_under<-function(wid = 0.1){
  x <-seq(0,1,wid)
  s <- c()
  for(i in 1:length(x)){
    s[i] <- curv_func(x[i])*wid
  }
```

```
  area <- sum(s)
  return(area)
}
```

Since we need to compare the result with analytical solution, so we need to have an analytical solution for the area under the curve.

```
# We can easily find the the curve fucntion can be
#part of the normal distribution function with mean=0, sigma=1
true_s <- sqrt(2*pi)*(pnorm(1)-pnorm(0))
```

The function should include as an argument the width of the slices used.Now use a looping construct (for or while) to loop through possible slice widths.

```
# We can build up table to compare their result, and limit the error of Riemann into 1e-6
wid <- c(0.1,0.05,0.01,10^-3,10^-4,10^-5,10^-6,10^-7)

for(i in 1:8){
  area_riemann <- area_under(wid[i])
  error_riemann <- abs(true_s - area_riemann)

  result <- c("Width:",wid[i],"Riemann Area:", area_riemann, "Error:",error_riemann)
  print(result)
}
```

```
## [1] "Width:"              "0.1"               "Riemann Area:"
## [4] "0.935445314060171"   "Error:"            "0.0798209221680221"
## [1] "Width:"              "0.05"              "Riemann Area:"
## [4] "0.895661287298931"   "Error:"            "0.0400368954067822"
## [1] "Width:"               "0.01"               "Riemann Area:"
## [4] "0.8636519907517"      "Error:"             "0.00802759885955084"
## [1] "Width:"                "0.001"               "Riemann Area:"
## [4] "0.856427606677782"     "Error:"              "0.000803214785633166"
## [1] "Width:"               "1e-04"              "Riemann Area:"
## [4] "0.855704717919692"    "Error:"             "8.0326027543598e-05"
## [1] "Width:"                "1e-05"               "Riemann Area:"
## [4] "0.855632424540393"     "Error:"              "8.03264824433825e-06"
## [1] "Width:"               "1e-06"              "Riemann Area:"
## [4] "0.855625195157428"    "Error:"             "8.0326527940322e-07"
## [1] "Width:"                "1e-07"               "Riemann Area:"
## [4] "0.855624472218681"     "Error:"              "8.03265325366453e-08"
```

The slice width necessary to obtain an answer within $1e^{-6}$ of the analytical solution is $1e^{-6}$.

Problem 7

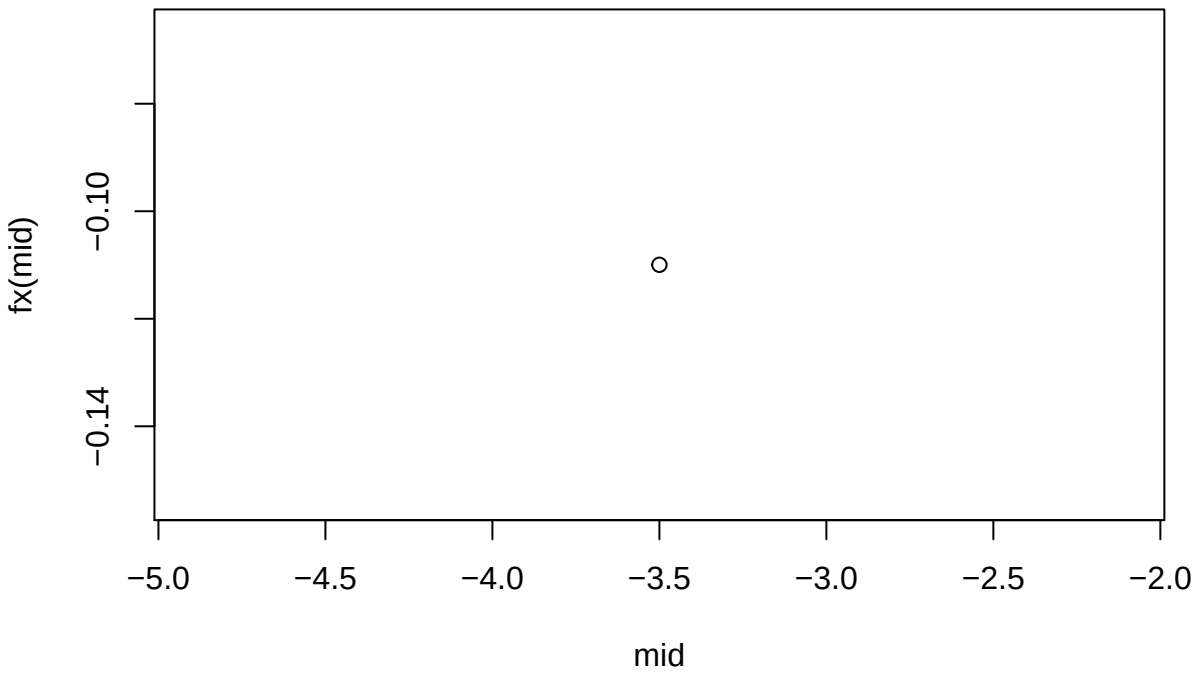
Create a function to find solutions to (1) using Newton's method.

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

```r
# We input the function

fx <- function(x=1){
  f <- 3^x-sin(x)+cos(5*x)
  return(f)
}

# find its derivative
dr <- function(x=1){
  d <- (3^x)*log(3, base = exp(1)) - cos(x) - 5*sin(5*x)
}
```

Then we build up newton function as an algorithm.

```r
newton <- function(interval=c(0,5),tolerance=1){
  # Find out the interval range and midpoint
  up_bound <- min(interval)
  low_bound <- max(interval)
  mid <- (up_bound + low_bound)/2
  plot(mid,fx(mid))
  m <- mid


  # perform newton method
  while(abs(fx(m))>tolerance){

    new_point <- m - (fx(m)/dr(m))
    m <- new_point
    points(m, fx(m))

  }

  return(m)
}

# Let's assume the interval is (-12,5), tolerance is 0.005
newton(c(-12,5),0.005)
```




```
## [1] -3.528064
```

Problem 8

In most of your classes, you will be concerned with "sums of squares" of various flavors. SST = SSR + SSE for instance. Sums of square total (SST) = sums of square regression (SSR) + sums of square error (SSE). In this problem, we want to compare use of a for loop to using matrix operations in calculating these sums of squares. We will use data simulated using:

```
X <- cbind(rep(1,100),rep.int(1:10,time=10))
beta <- c(4,5)
y <- X%*%beta + rnorm(100)
```

Without going too far into the Linear Regression material, we want to calculate SST =

$$\Sigma_{i=1}^{100}(y_i - \bar{y})^2$$

Please calculate this using:

a. accumulating values in a for loop

```
y_mean <- mean(y)
SST <- 0
```

```r
for(i in 1:100){
  SST[i] <- (y[i] - y_mean)^2
}
print(sum(SST))
```

b. matrix operations only

```r
y_mean <- mean(y)

SST <- (t(y)-y_mean)%*%(y-y_mean)
print(SST)
```

Final number is the same for two methods, but the speed of matrix operations is faster than accumulating values in a for loop.