Homework 4

Hwasoo Shin

2019 9 24

Problem 4

A takeaway from this links is that I can use some pacakge in R to clean up data more efficiently and effectively. I used to use only the base R package to clean up data, but from now on I will be able to try other handy packages to process data munging in better ways. Most of all, I learned pipe operators for the first time in this class. I think this will enhance my data importing and summarising speed and skills.

Problem 5

```
lint('C:/Users/pc/Desktop/HWASOO/STUDY/StatPackage/Homework3/HW3_Hwasoo_Shin(2).Rmd')
```

Through this syntax, we can lint the file. The modified file is attached in the file.
The major things I changed from my code was that I put spaces around infix operators and changed single quotes into double quotes.

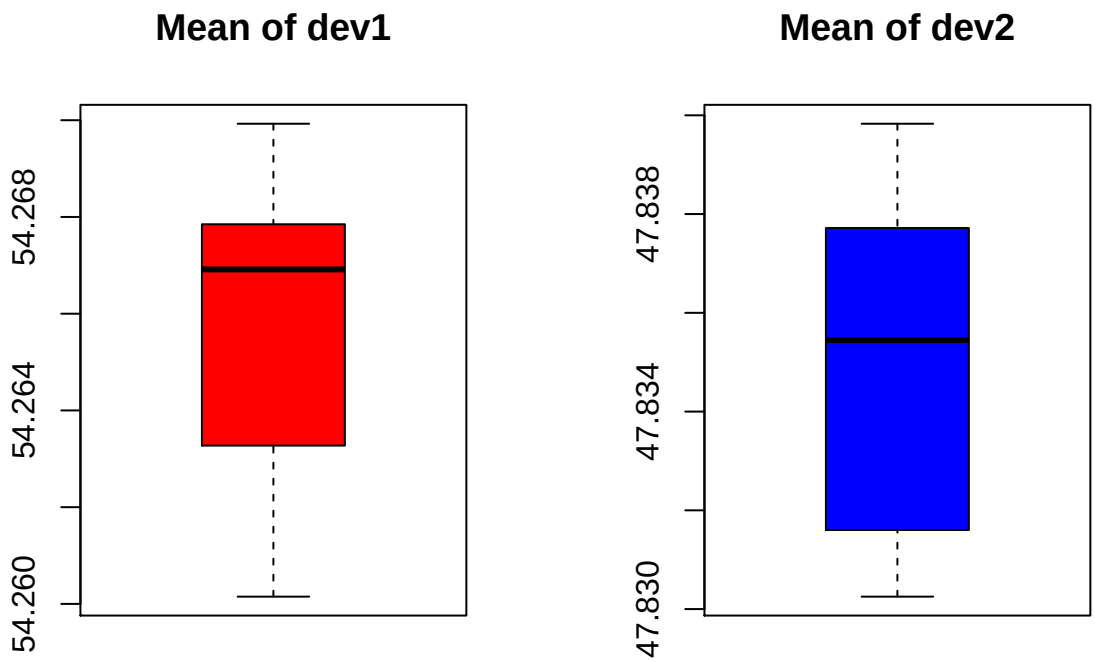
Problem 6

```
RDS<-readRDS("C:/Users/pc/Desktop/HWASOO/STUDY/StatPackage/Homework4/HW4_data.rds")
#Reading the file using readRDS function
RDSf<-function(){ #Start making a function
  RDSM<-matrix(0,5,13) #Make an empty matrix
  for(i in 1:13){ #Make a loop that runs 13 times
    k<-RDS %>% filter(Observer==i) #Get values of a specific observer
    RDSM[,i]=c(mean(k[,2]),mean(k[,3]),sd(k[,2]),sd(k[,3]),cor(k[,2],k[,3]))
    #Get means, standard deviations, and correlation
  }
  RDSM<-data.frame(RDSM) #Change the form into data frame
  colnames(RDSM)<-paste('Observer',1:13,sep="") #Put column names for the data. It will be observers
  rownames(RDSM)<-c(paste('mean',c('dev1','dev2')),paste('sd',c('dev1','dev2')),'cor')
  #Put rownames of the data.
  return(RDSM) #Return the data frame
}
#Making the function that gets makes the data frame.

RDSf() #We can see the information of each variable with Observers.
```

```
##             Observer1   Observer2   Observer3   Observer4   Observer5
## mean dev1 54.26609978 54.26873002 54.26731971 54.26327324 54.26030345
## mean dev2 47.83472062 47.83082316 47.83771727 47.83225282 47.83982921
## sd dev1   16.76982495 16.76923949 16.76001266 16.76514204 16.76773549
## sd dev2   26.93974342 26.93572669 26.93003609 26.93540349 26.93019152
## cor       -0.06412835 -0.06858639 -0.06834336 -0.06447185 -0.06034144
```

```
##             Observer6   Observer7   Observer8   Observer9 Observer10
## mean dev1 54.26144178 54.26880528 54.26784882 54.26588179 54.2673411
## mean dev2 47.83025191 47.83545020 47.83589633 47.83149565 47.8395452
## sd dev1   16.76589790 16.76670402 16.76675895 16.76885267 16.7689592
## sd dev2   26.93987622 26.93999796 26.93610493 26.93860807 26.9302747
## cor       -0.06171484 -0.06850422 -0.06897974 -0.06860921 -0.0629611
##            Observer11  Observer12  Observer13
## mean dev1 54.26992723 54.26691630 54.26015033
## mean dev2 47.83698799 47.83160199 47.83971728
## sd dev1   16.76995861 16.76999962 16.76995770
## sd dev2   26.93768381 26.93790193 26.93000169
## cor       -0.06944557 -0.06657523 -0.06558334
```

```
rd<-RDSf() #Assign the data frame
layout(rbind(c(1,2))) #Put two plots in same window.
boxplot(as.numeric(rd[1,]),col='red',main='Mean of dev1')
boxplot(as.numeric(rd[2,]),col='blue',main='Mean of dev2')
```




```
#Boxplots of mean of dev1 and dev2

rd1<-rd[3,] ; rd1<-data.frame(t(rd1),v=rep(1,13))
rd2<-rd[4,] ; rd2<-data.frame(t(rd2),v=rep(1,13))
#Prepare data frames for standard deviation values
```

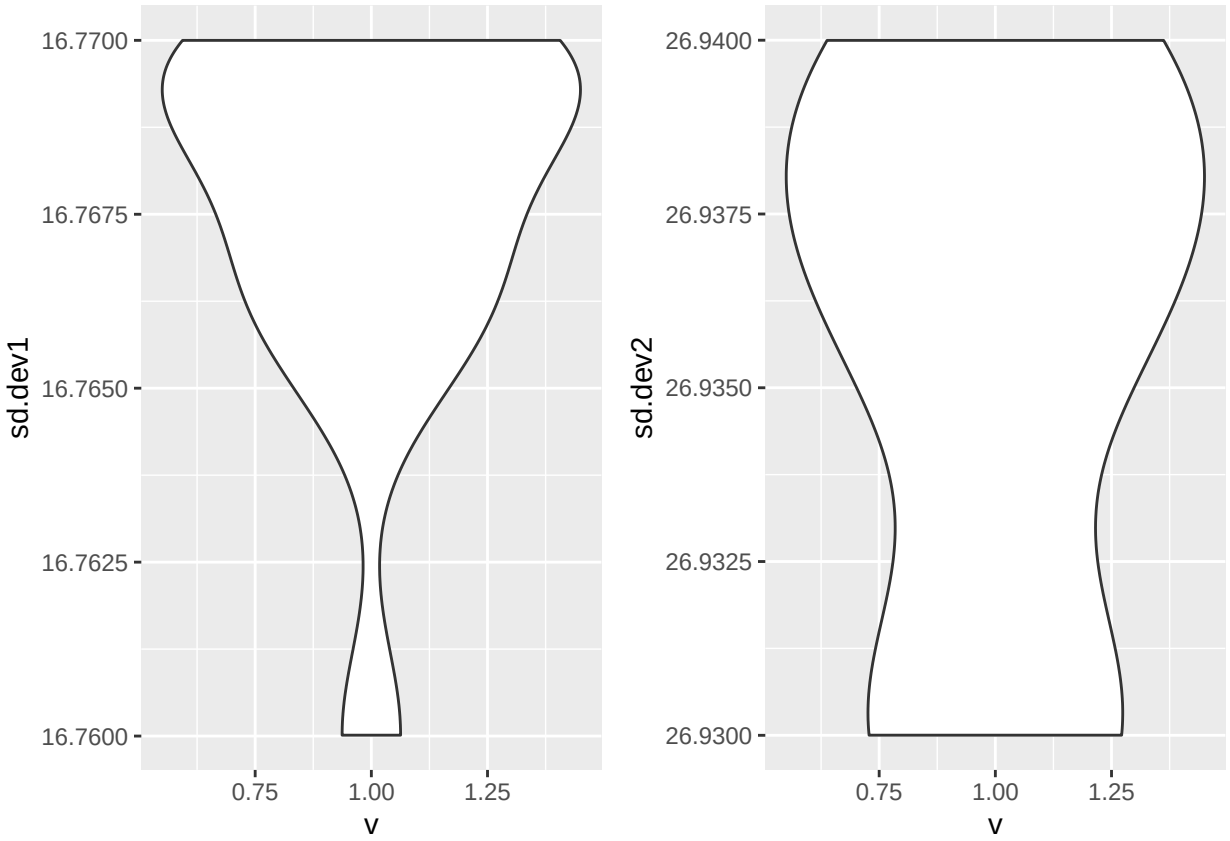
```
library(ggplot2) #A package to make a violin plot
library(gridExtra) #A package to put multiple ggplots in one window
```

```
## Warning: package 'gridExtra' was built under R version 3.6.1
```

```
##
## Attaching package: 'gridExtra'
```

```
## The following object is masked from 'package:dplyr':
##
##     combine
```

```
a1<-ggplot(rd1,aes(x=v,y=sd.dev1))+geom_violin() #Make a violin plot for sd(dev1)
a2<-ggplot(rd2,aes(x=v,y=sd.dev2))+geom_violin() #Make a violin plot for sd(dev2)
grid.arrange(a1,a2,nrow=1) #Draw the plots in one window
```



Problem 7

The function problem7 will have only argument, which indicates the number of blocks that will be used for Riemann sums. In order to integrate a specific function, we will also make a $e^{-x^2/2}$ function, which will be fx.

```r
fx<-function(x){exp((-x^2)/2)} #The function we will be integrating

problem7<-function(x){
  xseq<-seq(0+1/x,1,length=x)
  delta<-1/x
  k<-sum(fx(xseq)*delta)
  return(k)
}
#Riemann sum function

blocks<-c(10,100,1000,10000,100000) #Number of slices
for(i in blocks){
  cat('Riemann sum is ',problem7(i),' when n = ',i,'\n')
}
```

```
## Riemann sum is  0.8354453  when n =  10
## Riemann sum is  0.853652  when n =  100
## Riemann sum is  0.8554276  when n =  1000
## Riemann sum is  0.8556047  when n =  10000
## Riemann sum is  0.8556224  when n =  1e+05
```

```r
integrate(fx,0,1) #Integrate the function from 0 to 1
```

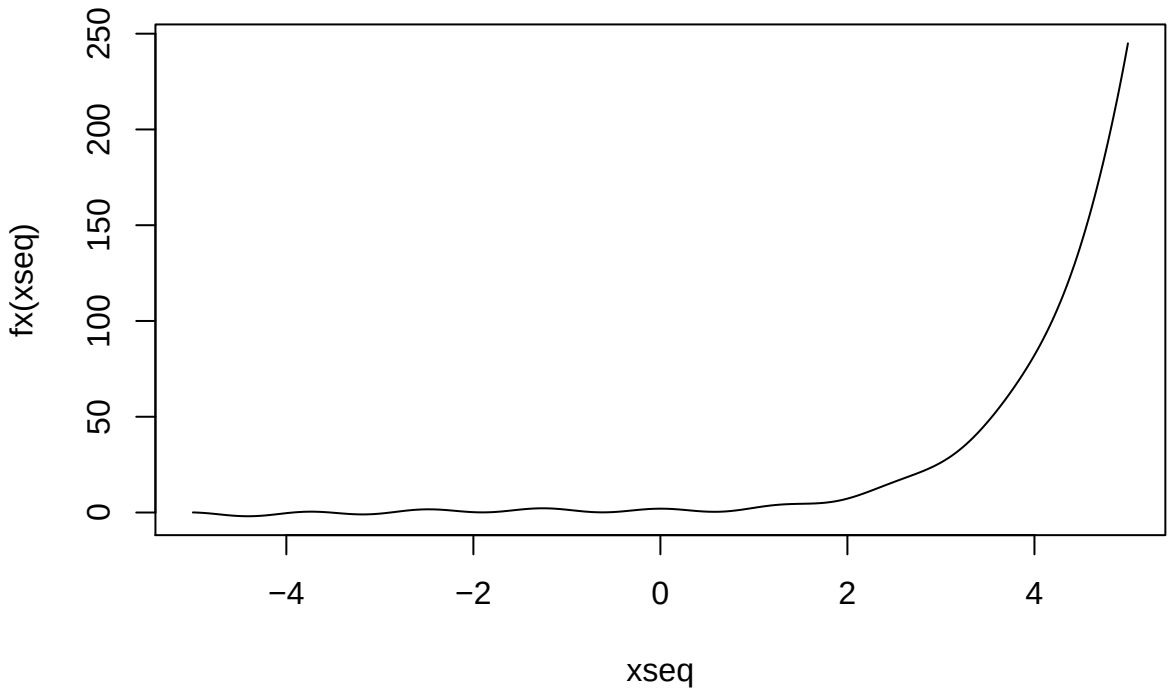
```
## 0.8556244 with absolute error < 9.5e-15
```

Therefore, we can see that the Riemman sum gets close to the true integrated value as the slices of widths increase.

Problem 8

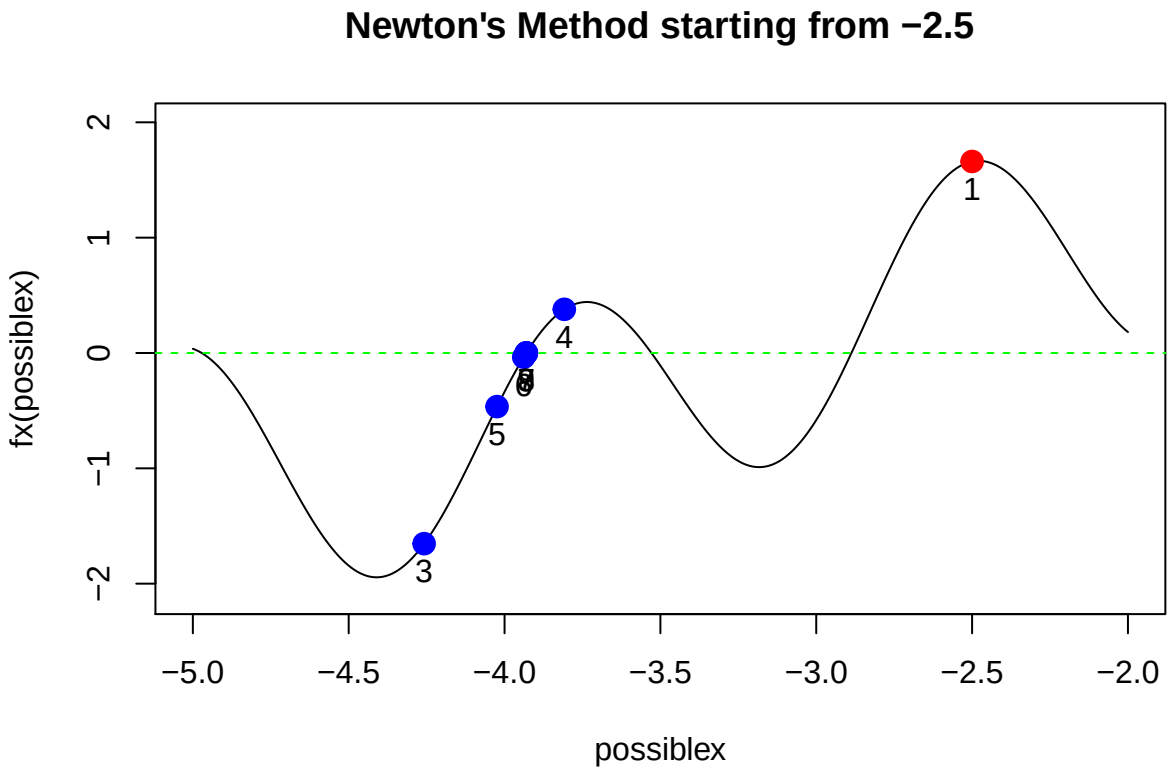
We will first make a function for $f(x) = 3^x - sin(x) + cos(5x)$, which will be fx. Also, we will make function called 'problem8', which will be a Newton Method function.

```r
fx<-function(x){3^(x)-sin(x)+cos(5*x)} #The original function for the problem
xseq<-seq(-5,5,by=0.01)
plot(xseq,fx(xseq),type='l')
```

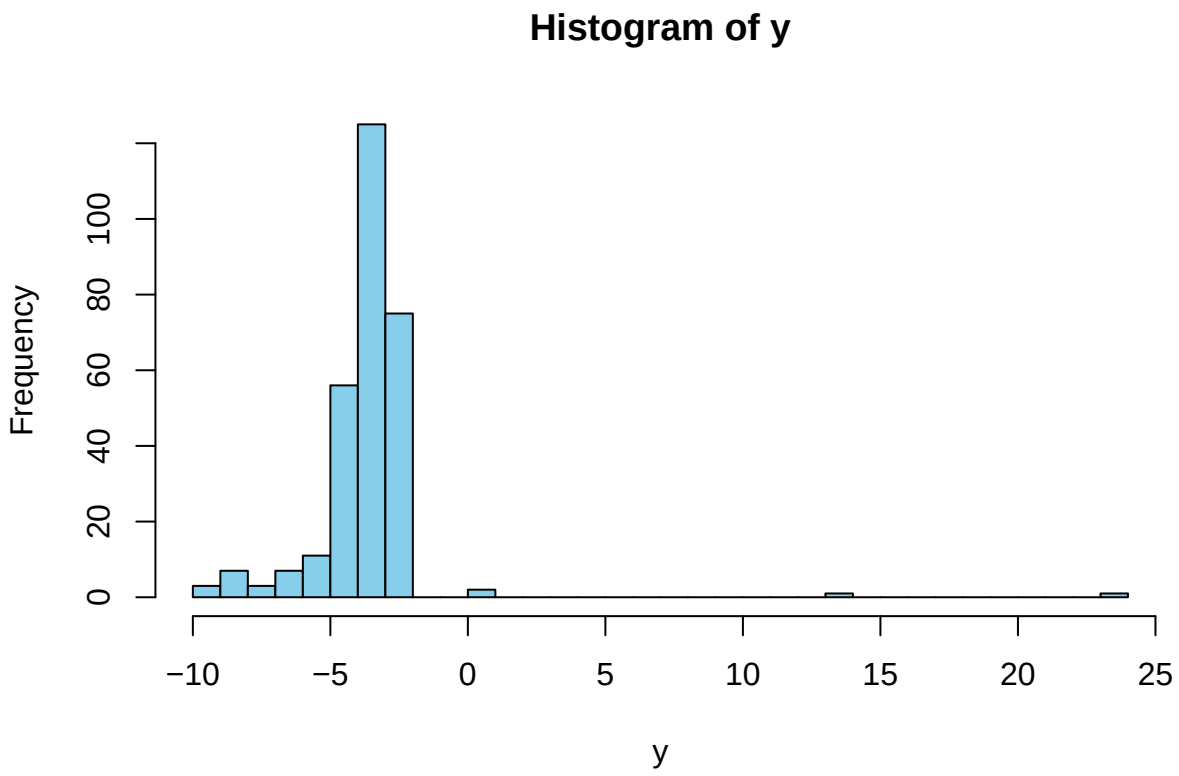



```r
#This is the graph of f(x). We can assume that it will have solutions when x<0. Therefore,
#we can set the interval from -5 to -2.

possiblex<-seq(-5,-2,by=0.01)
#Values we will use as initial values

problem8<-function(k){
  for(i in 1:100){
    k<-k-(3^(k)-sin(k)+cos(5*k))/(log(3)*3^(k)-cos(k)-5*sin(5*k))
  }
return(k)
} #Using Newton Method to get the solution.
#We will loop 100 times to find the solutions.

#For example, we can check how the Newton's Method works when the initial value is -2.5.

init=-2.5 #Initial point
plot(possiblex,fx(possiblex),type='l',ylim=c(-2.1,2),main="Newton's Method starting from -2.5")
#Plot of f(x) from -5 to -2
abline(h=0,lty=2,col='green') #Horizontal line x=0
points(init,fx(init),cex=1.5,pch=19,col='red') #Plot the initial point
text(init,fx(init),'1',pos=1) #Put a text for number of loops
for(i in 1:9){
  trial<-2:9;trial<-as.character(trial)
  init<-init-(3^(init)-sin(init)+cos(5*init))/(log(3)*3^(init)-cos(init)-5*sin(5*init))
  points(init,fx(init),col='blue',pch=19,cex=1.5)
```

```
  text(init,fx(init),labels=trial[i],pos=1)
}
```




```
#Therefore, we can check that the value will converge around -3.8, which is the root of this function.

x<-problem8(possiblex) #We put values between -4 and -1 to get the solution
y<-x[x>-10&x<50] #Among the results, we will get values that are bigger than -10
hist(y,breaks=40,col='skyblue')
```




```
#Distribution of solutions. The values are usually centered around -3.5.
```

From the previous steps, we can see that the Newton's Method can find the solution of the function as the number of loops increase.