Thakur_Prabhat

Test Items

Read each question carefully and address each element. Do not print contents of vectors or data frames unless requested.

(1) (4 points) This problem deals with vector manipulations.

(1)(a) Create a vector that contains the following, in this order, and print the contents. Do not round off any values unless requested. * A sequence of integers from 0 to 5, inclusive. * The number 6 * Two repetitions of the vector c(2, -5.1, -23). * The sum of 7/42 and 3

```
#Create Vector
myvector <- c(seq(0,5),6,rep(x=c(2,-5.1,-23),times = 2),7/42+3)
#Print Vector
cat("Resulting Vector after combining all the values:\n")
```

```
## Resulting Vector after combining all the values:
```

```
myvector
```

```
##  [1]   0.000000   1.000000   2.000000   3.000000   4.000000   5.000000
##  [7]   6.000000   2.000000  -5.100000 -23.000000   2.000000  -5.100000
## [13] -23.000000   3.166667
```

(1)(b) Sort the vector created in (1)(a) in ascending order. Print this result. Determine the length of the resulting vector and denote as L. Print L. Generate a sequence starting with 1 and ending with L and add to the sorted vector. This is vector addition, not vector combination. Print the contents. Do not round off any values.

```
#Sort the vector created in (1)(a) in ascending order
myvector.sort <- sort(myvector)
#Print the sorted vector
cat("Vector after sorting values in ascending order:\n")
```

```
## Vector after sorting values in ascending order:
```

```
myvector.sort
```

```
##  [1] -23.000000 -23.000000  -5.100000  -5.100000   0.000000   1.000000
##  [7]   2.000000   2.000000   2.000000   3.000000   3.166667   4.000000
## [13]   5.000000   6.000000
```

```
#Find the length of vector.
L <- length(myvector.sort)
#Print vector length.
```

```
cat("Length L of sorted vector:", L, "\n")
```

```
## Length L of sorted vector: 14
```

```
# Add vector(1:L) to the sorted vector.
myvector.new <- myvector.sort + seq(1,L)
#Print result after vector addition
cat("Result after adding vector(1:L) to the sorted vector: \n")
```

```
## Result after adding vector(1:L) to the sorted vector:
```

```
myvector.new
```

```
##  [1] -22.00000 -21.00000  -2.10000  -1.10000   5.00000   7.00000   9.00000
##  [8]  10.00000  11.00000  13.00000  14.16667  16.00000  18.00000  20.00000
```

(1)(c) Extract the first and last elements of the vector you have created in (1)(b) to form another vector using the extracted elements. Form a third vector from the elements not extracted. Print these vectors.

```
#New vector with first and last elements of vector created in (1)(b).
myvector2 <- myvector.new[c(1,L)]
#Print the vector.
cat("New vector with first and last elements of vector created in (1)(b):\n")
```

```
## New vector with first and last elements of vector created in (1)(b):
```

```
myvector2
```

```
## [1] -22  20
```

```
#New vector without first and last elements of vector created in (1)(b).
myvector3 <- myvector.new[c(-1,-L)]
#Print the vector.
cat("New vector without first and last elements of vector created in (1)(b):\n")
```

```
## New vector without first and last elements of vector created in (1)(b):
```

```
myvector3
```

```
##  [1] -21.00000  -2.10000  -1.10000   5.00000   7.00000   9.00000  10.00000
##  [8]  11.00000  13.00000  14.16667  16.00000  18.00000
```

(1)(d) Use the vectors from (c) to reconstruct the vector in (b). Print this vector. Sum the elements and round to two decimal places.

```
```

```
#Reconstruct the vector in (1)(b) from vectores in (1)(c)
myvector4 <- c(myvector2[1],myvector3,myvector2[2])
#Print the reconstructed vector.
cat("Reconstructed vector (1)(b) from vectores in (1)(c):\n")
```

```
## Reconstructed vector (1)(b) from vectores in (1)(c):
```

```
myvector4
```

```
##  [1] -22.00000 -21.00000  -2.10000  -1.10000   5.00000   7.00000   9.00000
##  [8]  10.00000  11.00000  13.00000  14.16667  16.00000  18.00000  20.00000
```

```
#Sum the elements and round to two decimal places.
cat("Sum of vector elements rounded to two decimal places:", round(sum(myvector4),digits = 2),
 "\n")
```

```
## Sum of vector elements rounded to two decimal places: 76.97
```

4 points

(2) (5 points) The expression y = sin(x) - 2*cos(x/2) is a trigonometric function.

(2)(a) Using the trigonometric function above, write a function as defined by R in the proper format using function() that accepts values for x and returns a value for y.

```
#Function to calculate and return value of trigonometric function.
trigfuction <- function (x1) {
  y1 <- sin(x1) - 2*cos(x1/2)
  return(y1)
}
```

(2)(b) Create a vector, x, of 4001 equally-spaced values from -2 to 2, inclusive. Compute values for y using the vector x and your function in (a). **Do not print x or y.** Find the value in the vector x that corresponds to the minimum value in the vector y. Restrict attention to only the values of x and y you have computed. Round to 3 decimal places and display the value of x you find and the minimum value of y.

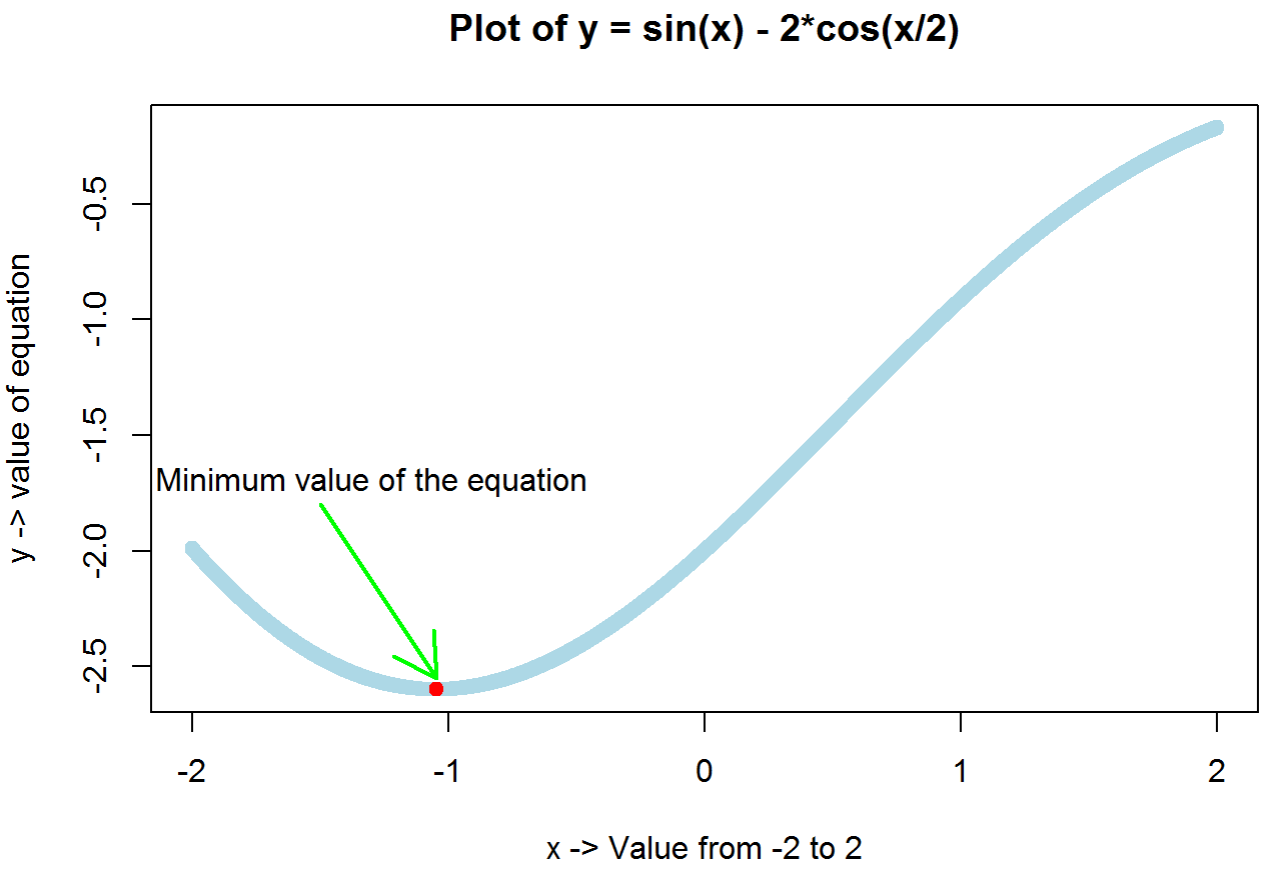
Finding the two desired values can be accomplished in as few as two lines of code. Do not use packages or programs you may find on the internet or elsewhere. Do not print the elements of the vectors x and y. Use coding methods shown in the *Quick Start Guide for R*.

```
#Create vector x,of 4001 equally-spaced values from -2 to 2,inclusive.
x <- seq(from =-2 , to =2,length.out = 4001)
#Compute the value of y for vector x using (2)(a) function.
y <- trigfuction(x)

#Find the position of minimum value in vector y and print the element from vector x in that po
sition.
y.minindex <- which.min(y)
cat("Value in vector x with respect to minimum value in vector y,rounded to 3 decimal point: "
, round(x[y.minindex],digits = 3),"\n")
```

```
## Value in vector x with respect to minimum value in vector y,rounded to 3 decimal point:  -1
.047
```

(2)(c) Plot y versus x in color, with x on the horizontal axis. Show the location of the minimum value of y determined in 2(b). Add a title and other features such as text annotations. Text annotations may be added via *text()* and *paste()*.

```
#Plot y versus x in color, with x on the horizontal axis.
plot(x,y,type ="o", main = "Plot of y = sin(x) - 2*cos(x/2)",xlab= "x -> Value from -2 to 2",
ylab = "y -> value of equation",col="lightblue")

#Show location of the minimum value of y determined in 2(b)
points(x[y.minindex],min(y),pch=19,col="red")
arrows(x0 =-1.5,y0=-1.8,x1= x[y.minindex], y1 = min(y)+0.05,col= "green",lwd=2)
text(x =-1.3,y=-1.7,labels = "Minimum value of the equation")
```



5 points

(3) (8 points) Use the “trees” dataset for the following items. This dataset has three variables (Girth, Height, Volume) on 31 trees.

(3)(a) Use *data(trees)* to load the file. Check the structure with *str()*. Use *apply()* to return the median values for the three variables in “trees.” Print these values. Using R and logicals, print the row number and the three measurements: Girth, Height and Volume, of the tree with Volume equal to median Volume.

```
#Load trees data set.
```

```
data(trees)
#Get the medians of all three variables (columns).
tree.medians <- apply(trees[1:3],2,median)
#Print the median values.
cat("Tree variables Girth, Height and Volume medians are: ",tree.medians[1],",",tree.medians[2
],",",tree.medians[3]," respectively. \n")
```

```
## Tree variables Girth, Height and Volume medians are:  12.9 , 76 , 24.2  respectively.
```

```
#Get and print the row number of median Volume
trees.volume.median.row <- which(trees[3]==tree.medians[3])
cat ("Median tree volume row number: ",trees.volume.median.row,"\n")
```

```
## Median tree volume row number:  11
```

```
#Get and print the values of all three variables for the median volume row.
trees.volume.median.row.data <- trees[trees.volume.median.row,]
cat ("Median tree volume row values, Girth =",trees[trees.volume.median.row,1], " Hight =",tre
es[trees.volume.median.row,2]," and Volume =",trees[trees.volume.median.row,3])
```

```
## Median tree volume row values, Girth = 11.3  Hight = 79  and Volume = 24.2
```

(3)(b) Girth is defined as the diameter of a tree taken at 4 feet 6 inches from the ground. Convert each diameter to a radius, r. Calculate the cross-sectional area of each tree using pi times the squared radius. Sort Radius ascending and print. Present the stem-and-leaf plot of the radius, and a histogram of the radius in color. Plot Area (y-axis) versus Radius (x-axis) in color. Label appropriately.

```
#Calculate the radious of each tree.
tree.radious <- c(trees[,3]/2)
#Cross-sectional area of each tree.
tree.area <- c(pi * (tree.radious)^2)
#Print sorted radious
cat("Sorted Radious of each tree:\n",sort(tree.radious))
```

I think you used volume not Girth. Better check.

```
## Sorted Radious of each tree:
##  5.1 5.15 5.15 7.8 8.2 9.1 9.4 9.55 9.85 9.95 10.5 10.65 10.7 11.1 11.3 12.1 12.45 12.85 13
.7 15.85 16.9 17.25 18.15 19.15 21.3 25.5 25.75 27.7 27.85 29.15 38.5
```

These values are wrong.

```
#Stem-and-leaf plot of the radius
cat("Stem-and-leaf plot of the radius.\n")
```

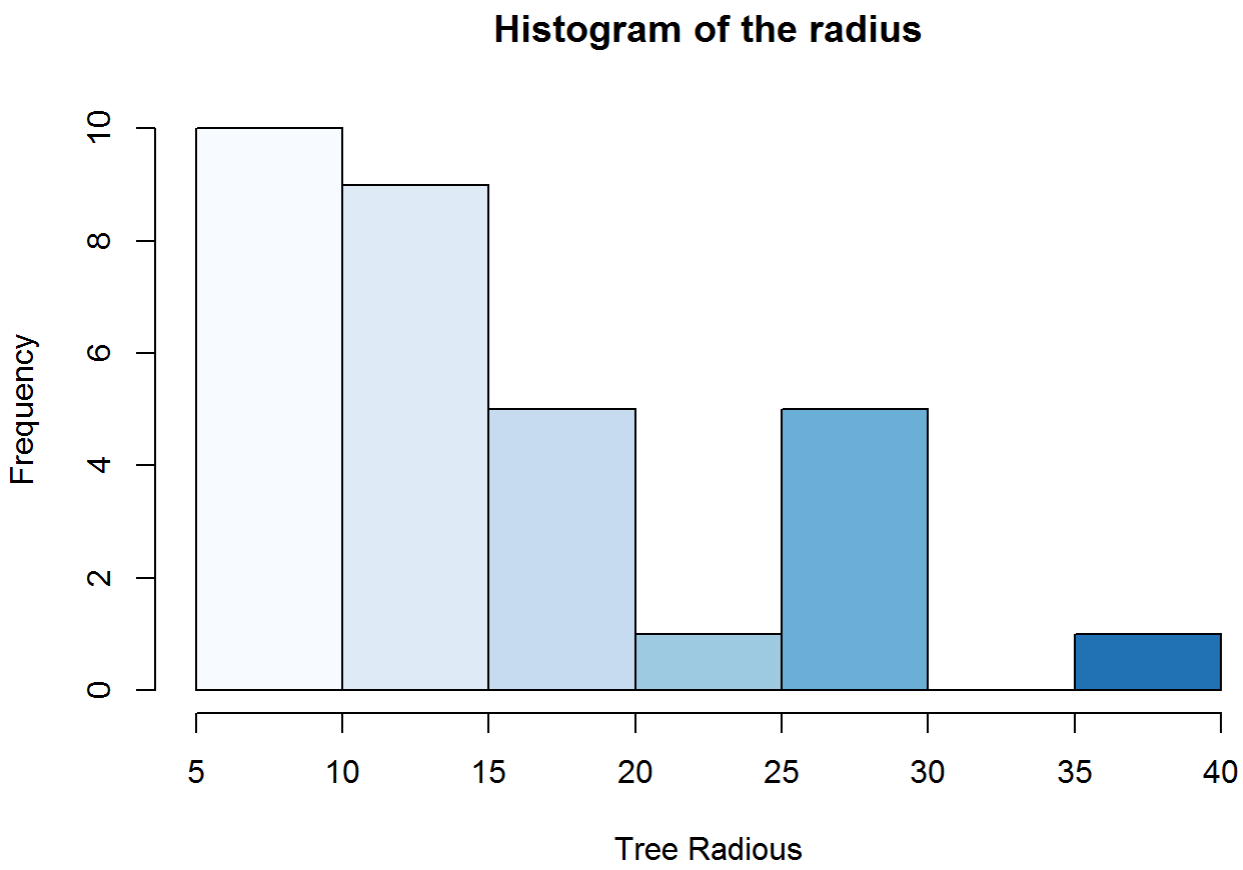
```
## Stem-and-leaf plot of the radius.
```

```
stem(tree.radious)
```

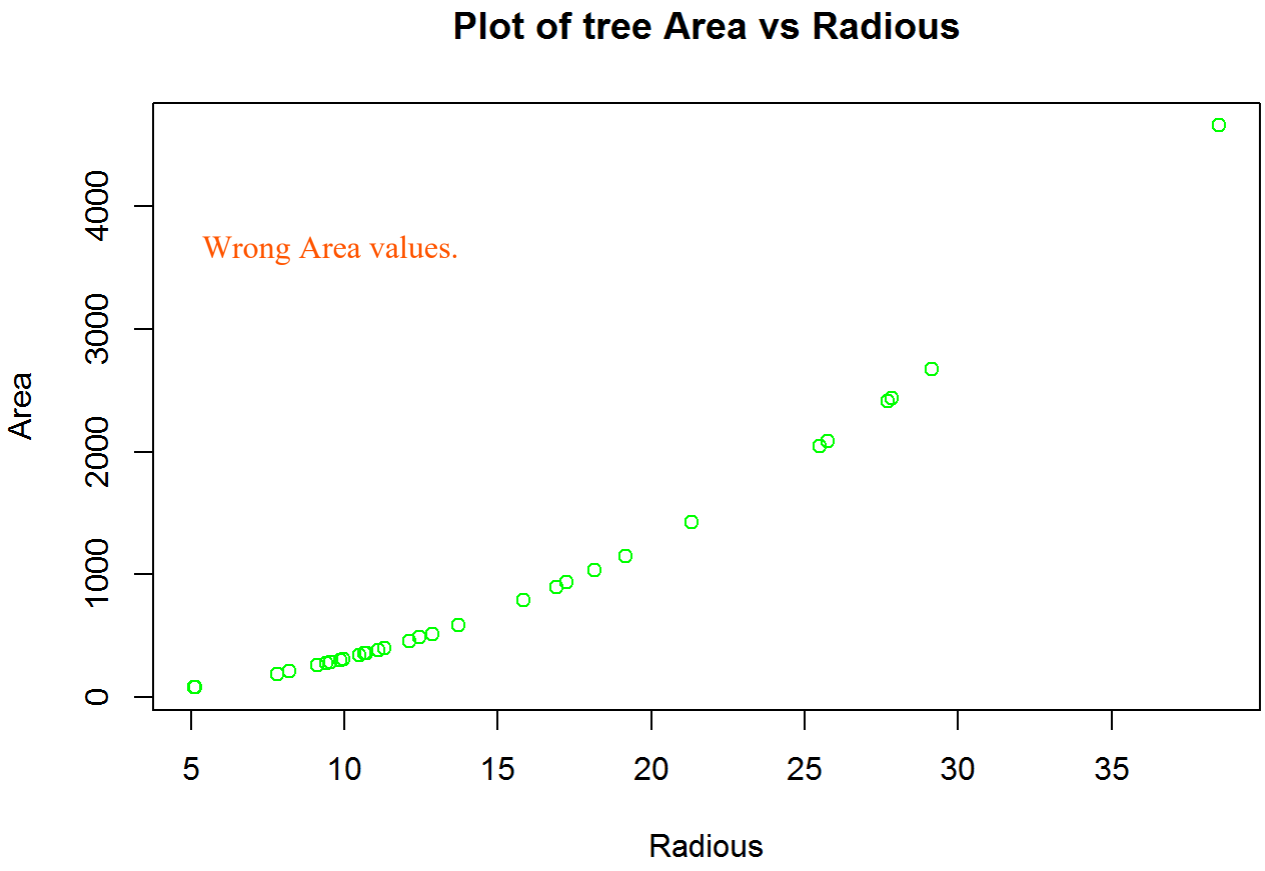
```
##
##   The decimal point is 1 digit(s) to the right of the |
```

```
##
##   0 | 5558899
##   1 | 000111112234
##   1 | 67789
##   2 | 1
##   2 | 66889
##   3 |
##   3 | 9
##   4 | 234
##   5 | 34455667779
##   6 | 055799
##   7 | 013
##   8 | 0278
##   9 | 000
##  10 | 3
```

```
#Histogram of the radius in blue color.
hist(tree.radious,col = blues9,main="Histogram of the radius", xlab = "Tree Radious")
```



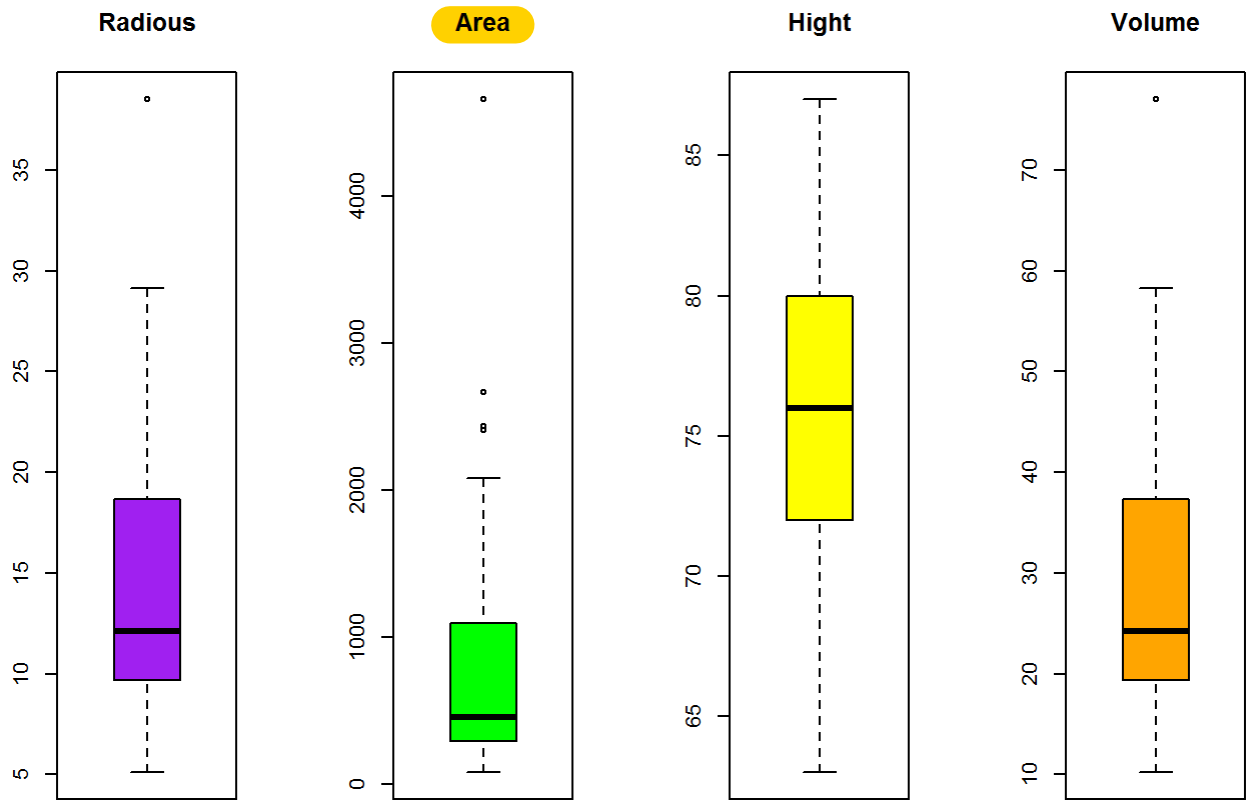

```
#Plot Area (y-axis) versus Radius (x-axis)  in color
plot(tree.radious,tree.area,type ="p", main = "Plot of tree Area vs Radious",xlab= "Radious",
ylab = "Area",col="green"  )
```



(3)(c) Use *par(mfrow = c(1, 4))* and present vertical colored boxplots of the radii and areas calculated in (b) along with Height and Volume. Label each accordingly.

```
#Set the layout to print 4 plots in one row.
par(mfrow = c(1, 4))
# boxplots of the radii
boxplot(tree.radious,main ="Radious", col = "purple")
# boxplots of the Area
boxplot(tree.area,main ="Area",col = "green")
# boxplots of the Height
boxplot(trees$Height,main ="Hight",col = "yellow")
# boxplots of the Volume
boxplot(trees$Volume,main ="Volume",col = "orange")
```

Wrong Area values.



(3)(d) Demonstrate that the outlier revealed in the boxplot of Volume is not an extreme outlier. It is possible to do this with one line of code using *boxplot.stats* or logicals.

```
#Check if extreme outlier exists in Volume from (3)(c) boxplot.
if (length(boxplot.stats(trees$Volume,coef=3.0)$out) == 0){
  cat("There are no extreme outlier in Volume data set\n")
}else {
  cat("Following are the extreme outlier in Volume data set",boxplot.stats(trees$Volume,coef=3
.0)$out,"\n" )}
```

```
## There are no extreme outlier in Volume data set
```

6 points Your work is basically correct. There is a calculation error to track down.

(4) (2 points) Use matrix operations shown in the “Quick Start Guide” to solve the following system of linear equations. Display the R script and print the numerical solutions for x, y and z. Use matrix operations with your solution to reproduce the values 1, 1, 1 as a means of checking if your solution is correct. This last demonstration can be accomplished with matrix multiplication in one line of code. Print your result.

x - y + z = 1

x + y + z = 1

x + y - z = 1

```
#Create matrix using coffcient from equations.
x <- matrix(c(1,-1,1,1,1,1,1,1,-1),nrow=3,ncol=3,byrow=TRUE)
```

```
y <- c(1,1,1)

#Below code will solve the equation and give values for all three variables.
solution <- solve(x,y)
cat ("Solution: Value of x = " , solution[1], " Value of y = " , solution[2]," and Value of z
= " , solution[3], "\n")
```

```
## Solution: Value of x =  1  Value of y =  0  and Value of z =  0
```

```
# when cofficient matrix is multiplied with solution vector, we should get the vector y.
# We can check that using below code.
if (all(y==x%*%solution)) {cat ("Solution is correct.\n")} else {cat ("Solution is incorrect.\
n")}
```

This is cumbersome. Try something like coef %*% solve(coef, v). That would suffice for me.

```
## Solution is correct.
```

2 points

(5) (6 points) The Cauchy distribution is an example of a “heavy-tailed” distribution in that it will have (more) outliers in both tails. This problem involves comparing it with a normal distribution which typically has very few outliers.

5(a) Use *set.seed(124)* and *rcauchy()* with n = 100, location = 0 and scale = 0.1 to generate a random sample designated as y. Generate a second random sample designated as x with *set.seed(127)* and *rnorm()* using n = 100, mean = 0 and sd = 0.15.

Generate a new object using *cbind(x, y)*. Do not print this object. Use *apply()* with this object to compute the inter-quartile range or IQR for each column: x and y. Use the function IQR() for this purpose. Round the results to four decimal places and present. (This exercise shows the similarity of the IQR values.)

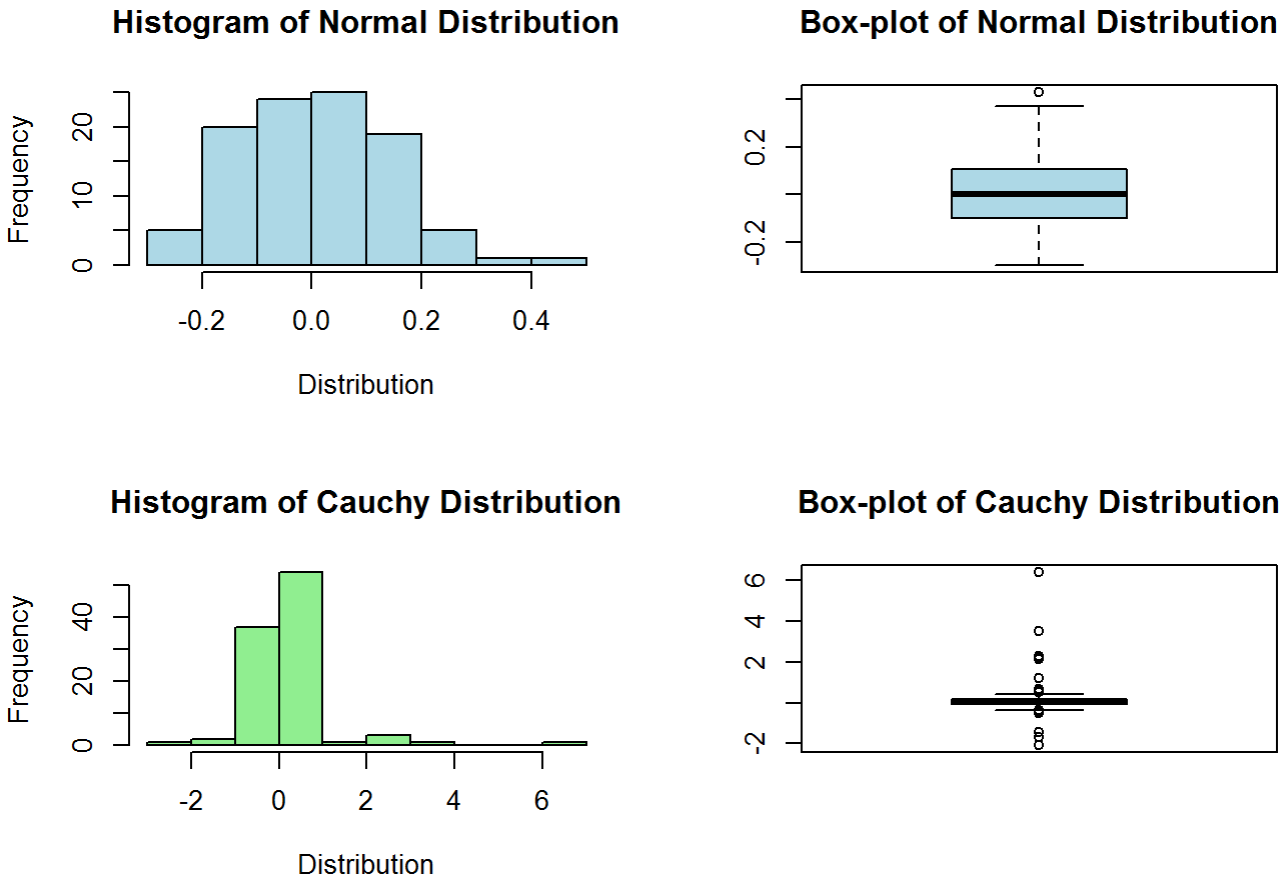
For information about rcauchy(), use help in R (?rcauchy). **Do not print x and y.**

```
#Create random sample of Cauchy distributio.
set.seed(124)
y<- rcauchy(n=100,location = 0,scale = 0.1)
#Create a random sample of Normal distribution
set.seed(127)
x<-rnorm(n=100,mean=0,sd=0.15)
#Create matirx by binding both vectores as columns.
xy <- cbind(x,y)
#Calculate IQR and round the result to 4 digits.
xy.iqr <- round(apply(xy,2,IQR),digits = 4)
cat("IQR of normal distribution (x)= ",xy.iqr[1], " and Cauchy distributio (y)= ",xy.iqr[2] )
```

```
## IQR of normal distribution (x)=  0.2041  and Cauchy distributio (y)=  0.2141
```

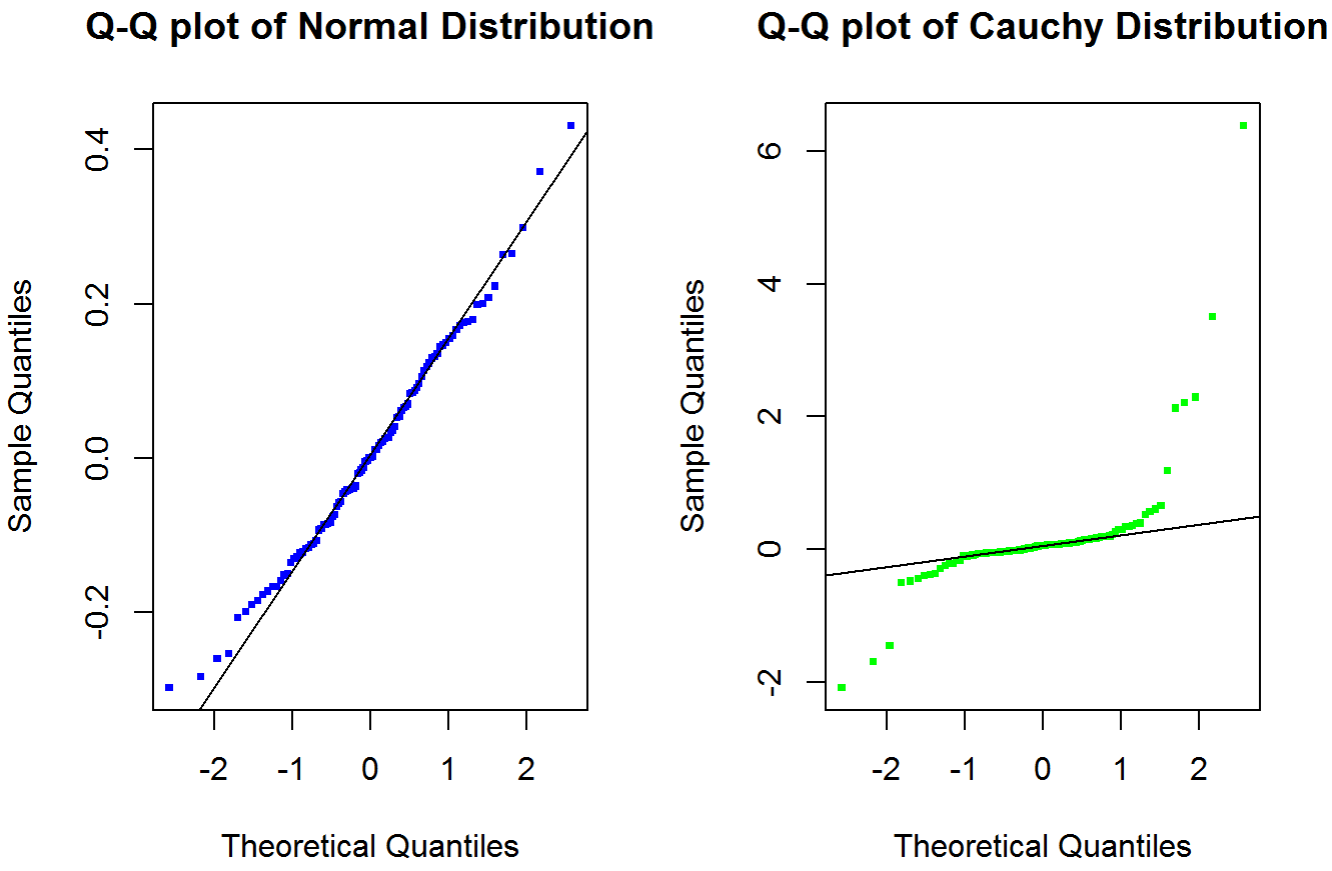
(5)(b) This item will illustrate the difference between a heavy-tailed distribution and one which does not have heavy tails. Use *par(mfrow = c(2, 2))* to generate a display with four diagrams. On the first row, For the Normal results present a histogram and a horizontal boxplot for x in color. For the Cauchy results, present a histogram and a horizontal boxplot for y in color.

```
#Set the layout to print 4 plots rowwise, 2 in each row.
par(mfrow = c(2, 2))
#Histogram of Normal Distribution
hist(x,main="Histogram of Normal Distribution",xlab = "Distribution",col="lightblue")
#Box-plot of Normal Distribution
boxplot(x,main="Box-plot of Normal Distribution",col="lightblue")
#Histogram of Cauchy Distribution
hist(y,main="Histogram of Cauchy Distribution",xlab = "Distribution",col="lightgreen")
#Box-plot of Cauchy Distribution
boxplot(y,main="Box-plot of Cauchy Distribution",col="lightgreen")
```



(5)(c) QQ plots are useful for detecting the presence of heavy-tailed distributions. Use *par(mfrow = c(1, 2))* and present side-by-side plots, one for each sample, using *qqnorm()* and *qqline()*. Add color and titles. Use cex = 0.5 to control the size of the plotted data points.

```
#Set the layout to print 2 plots in the same row.
par(mfrow = c(1, 2))
#Q-Q plot of Normal Distribution
qqnorm(x,pch=15,cex=0.5,main="Q-Q plot of Normal Distribution",col="blue")
qqline(x)
#Q-Q plot of Cauchy Distribution
qqnorm(y,pch=15,cex=0.5,main="Q-Q plot of Cauchy Distribution",col="green")
qqline(y)
```



6 points

23 points