kumar_ritesh. R Assignment #1 (50 points)

Instructions

R markdown is a plain-text file format for integrating text and R code, and creating transparent, reproducible and interactive reports. An R markdown file (.Rmd) contains metadata, markdown and R code “chunks”, and can be “knit” into numerous output types. Answer the test questions by adding R code to the fenced code areas below each item. Once completed, you will “knit” and submit the resulting .html file, as well the .Rmd file. The .html will include your R code *and* the output.

Before proceeding, look to the top of the .Rmd for the (YAML) metadata block, where the *title* and *output* are given. Please change *title* from ‘Programming with R Assignment #1’ to your name, with the format ‘lastName_firstName.’

If you encounter issues with knitting the .html, please send an email via Canvas to your TA.

Each code chunk is delineated by six (6) backticks; three (3) at the start and three (3) at the end. After the opening ticks, arguments are passed to the code chunk and in curly brackets. **Please do not add or remove backticks, or modify the arguments or values inside the curly brackets**. An example code chunk is included here:

```
# Comments are included in each code chunk, simply as prompts

...R code placed here

...R code placed here
```

You need only enter text inside the code chunks for each test item.

Depending on the problem, grading will be based on: 1) the correct result, 2) coding efficiency and 3) graphical presentation features (labeling, colors, size, legibility, etc). I will be looking for well-rendered displays. In the “knit” document, only those results specified in the problem statements should be displayed. For example, do not output - i.e. send to the Console - the contents of vectors or data frames unless requested by the problem. You should be able to code for each solution in fewer than ten lines; though code for your visualizations may exceed this.

Submit both the .Rmd and .html files for grading

Example Problem with Solution: Use *rbinom()* to generate two random samples of size 10,000 from the binomial distribution. For the first sample, use p = 0.45 and n = 10. For the second sample, use p = 0.55 and n = 10.

a. Convert the sample frequencies to sample proportions and compute the mean number of successes for each sample. Present these statistics.

```
set.seed(123)
sample.one <- table(rbinom(10000, 10, 0.45)) / 10000
sample.two <- table(rbinom(10000, 10, 0.55)) / 10000

successes <- seq(0, 10)

sum(sample.one * successes) # [1] 4.4827
```

```
## [1] 4.4827
```

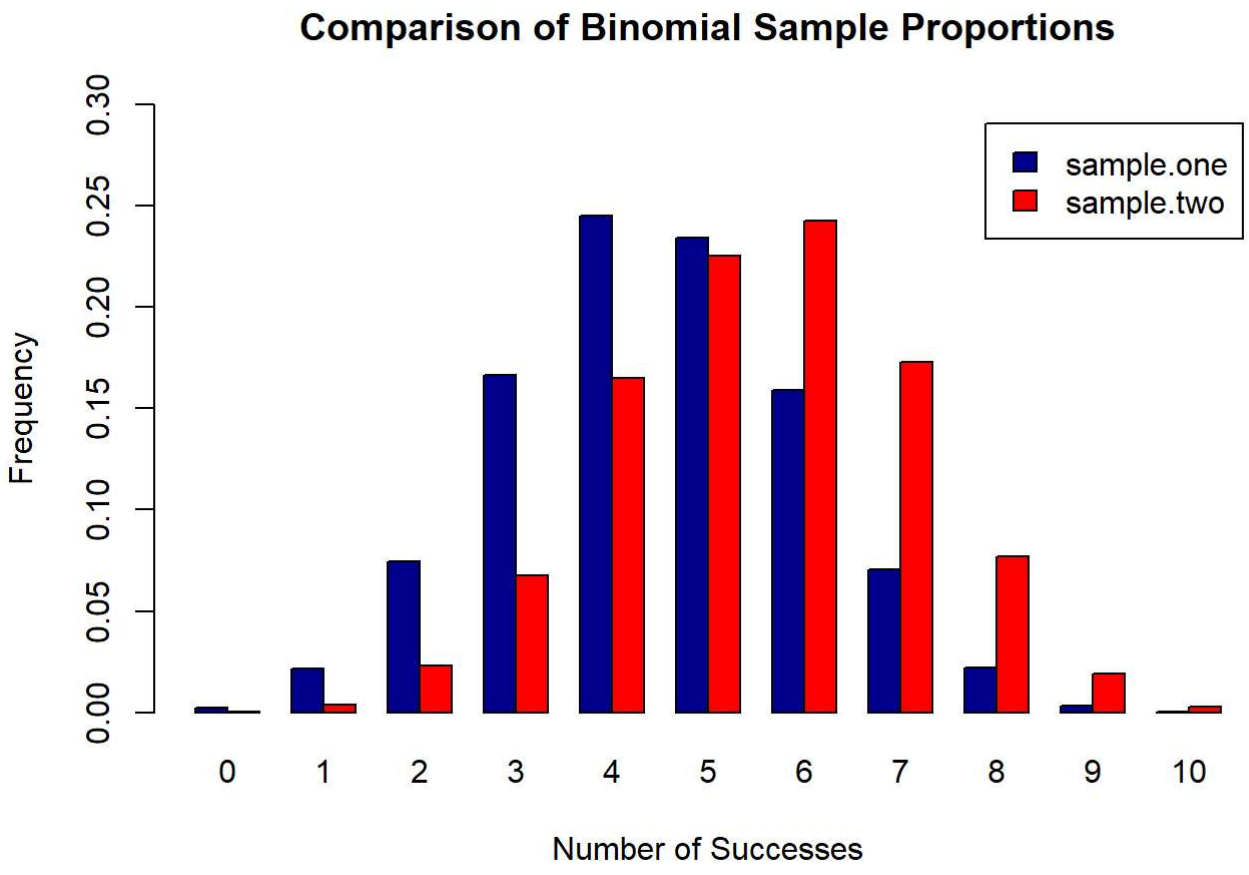
```
sum(sample.two * successes) # [1] 5.523
```

```
## [1] 5.523
```

b. Present the proportions in a vertical, side-by-side barplot color coding the two samples.

```
counts <- rbind(sample.one, sample.two)

barplot(counts, main = "Comparison of Binomial Sample Proportions",
  ylab = "Frequency", ylim = c(0,0.3),xlab = "Number of Successes",
  beside = TRUE, col = c("darkblue","red"),legend = rownames(counts),
  names.arg = c("0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "10"))
```



Please delete the Instructions and Examples shown above prior to submitting your .Rmd and .html files.

Test Items starts from here - There are 5 sections - 50 points total

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

Section 1: (8 points) This problem deals with vector manipulations.

(1)(a) Create a vector that contains the following: * The integer sequence 1 to 5, inclusive, * The square root of 2, * The product of 17 and 14, and * Three (3) repetitions of the vector c(2.5, 5, 7.5).

Assign the vector to the name q1_vector and output. You will use q1_vector for the following four (4) questions.

```
q1_vector = c(seq(1:5), 2^(0.5), 14*17, rep(c(2.5, 5, 7.5), times=3))
q1_vector
```

```
##  [1]   1.000000   2.000000   3.000000   4.000000   5.000000   1.414214
##  [7] 238.000000   2.500000   5.000000   7.500000   2.500000   5.000000
## [13]   7.500000   2.500000   5.000000   7.500000
```

(1)(b) Remove all elements of q1_vector greater than 7. Assign this new vector, with only values equal to or less than 7, to the name q1_vector_7. What is the length of q1_vector_7?

```
q1_vector_7 = q1_vector[q1_vector<=7]
q1_vector_7
```

```
##  [1] 1.000000 2.000000 3.000000 4.000000 5.000000 1.414214 2.500000 5.000000
##  [9] 2.500000 5.000000 2.500000 5.000000
```

(1)(c) Sort q1_vector in ascending order and assign the sorted vector to the name q1_vector_sorted. What is the sum of the 5th through 10th elements of q1_vector_sorted, inclusive?

```
q1_vector_sorted = sort(q1_vector)
q1_vector_sorted
```

```
##  [1]   1.000000   1.414214   2.000000   2.500000   2.500000   2.500000
##  [7]   3.000000   4.000000   5.000000   5.000000   5.000000   5.000000
## [13]   7.500000   7.500000   7.500000 238.000000
```

```
sum(q1_vector_sorted[5:10])
```

```
## [1] 22
```

(1)(d) Square each element of q1_vector and assign the new, squared value vector to the name q1_vector_sqrd. How many elements of q1_vector_sqrd are greater than 25?

```
q1_vector_sqrd = q1_vector^2
length(q1_vector_sqrd[q1_vector_sqrd>25])
```

```
## [1] 4
```

(1)(e) Remove the first and last elements of q1_vector. Assign the two (2) removed elements to the name q1_vector_short. What is the product of the elements of q1_vector_short?

```
q1_vector_short = c(q1_vector[1], tail(q1_vector,1))
q1_vector_short[1]*q1_vector_short[2]
```

```
## [1] 7.5
```

Section 2: (10 points) The expression y = sin(x/2) - cos(x/2) is a trigonometric function.

(2)(a) Create a user-defined function - via *function()* - that implements the trigonometric function above, accepts numeric values, “x,” calculates and returns values “y.”

```
func = function(x){
  y = sin(x/2) - cos(x/2)
  return(y)}
```

(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 from (2)(a). **Do not output 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; i.e. do not interpolate. Round to 3 decimal places and output both the maximum y and corresponding x value.

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 output the other elements of the vectors x and y. Relevant coding methods are given in the *Quick Start Guide for R*.

```
x = seq(from=-2, to=2, length.out=4001)
y = round(unlist(lapply(x, FUN=func)),3)
cat('Minimum Value in y is:', round(min(y),3), 'and the corresponding value in x is:',round(x[wh
ich.min(y)],3))
```

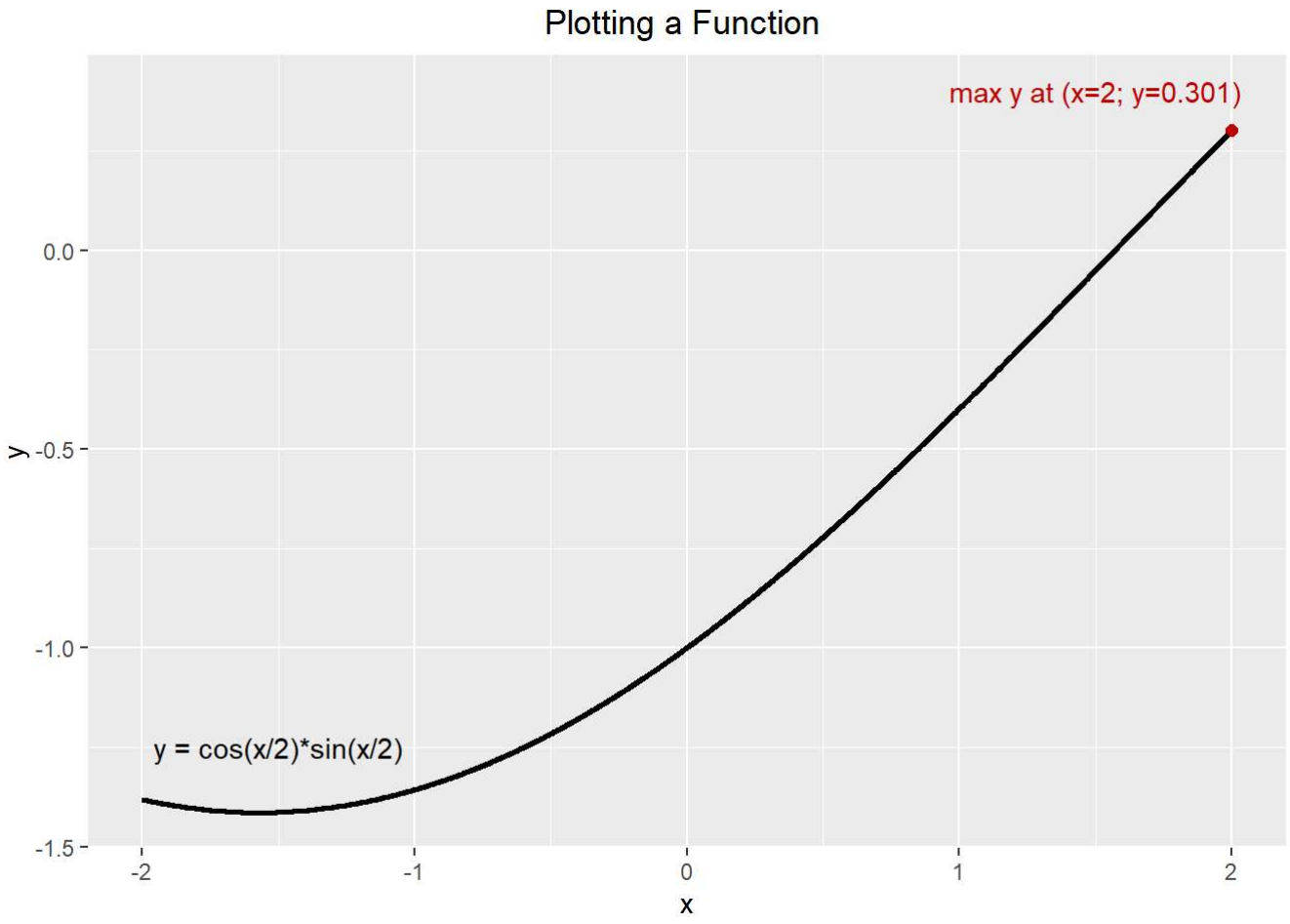
```
## Minimum Value in y is: -1.414 and the corresponding value in x is: -1.634
```

```
cat('\nMaximum Value in y is:', round(max(y),3), 'and the corresponding value in x is:',round(x
[which.max(y)],3))
```

```
##
## Maximum Value in y is: 0.301 and the corresponding value in x is: 2
```

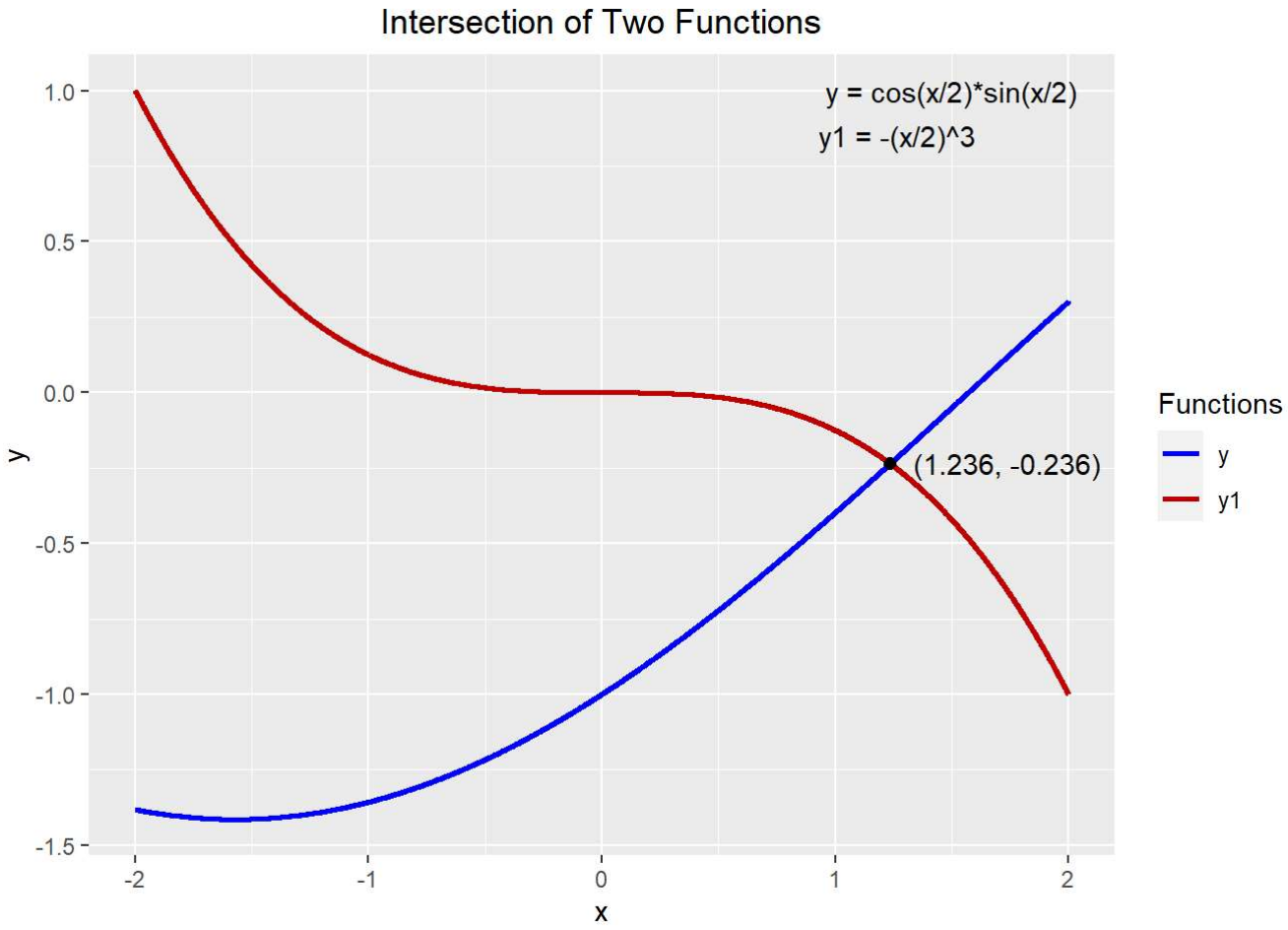
(2)(c) Plot y versus x in color, with x on the horizontal axis. Show the location of the maximum value of y determined in 2(b). Show the values of x and y corresponding to the maximum value of y in the display. Add a title and other features such as text annotations. Text annotations may be added via *text()* for base R plots and *geom_text()* or *geom_label()* for ggplots.

```
df = data.frame(x,y)
library(ggplot2)
lab = paste('max y at (x=',x[which.max(y)],'; ','y=', max(y),')', sep='')
ggplot() + geom_line(data=df, aes(x=x,y=y), linewidth=1) + annotate('text', x=-1.5, y=-1.25, lab
el='y = cos(x/2)*sin(x/2)') + ggtitle('Plotting a Function ') + theme(plot.title = element_text
(hjust = 0.5)) + geom_text(data=subset(df, y==max(y)), aes(x,y+0.05,label=lab), color='#c00000',
nudge_x=-0.5, nudge_y=0.05) + geom_point(data=subset(df,y==max(y)), aes(x=x,y=y), color='#c0000
0', size=2)
```



Section 3: (8 points) This problem requires finding the point of intersection of two functions. Using the function `y = cos(x / 2) * sin(x / 2)`, find where the curved line y = -(x/2)^3 intersects it within the range of values used in part (2) (i.e. 4001 equally-spaced values from -2 to 2). Plot both functions on the same display, and show the point of intersection. Present the coordinates of this point as text in the display.

```
y1 = round(-(x/2)^3,3)
df1 = data.frame(x,y,y1)
df2 = subset(df1,y==y1)
library(ggplot2)
lab2 = paste('(',paste(df2$x, df2$y, sep=', '),')',sep='')
ggplot() + geom_line(data=df1, aes(x=x,y=y, color='y'), linewidth=1) + geom_line(data=df1, aes(x
=x,y=y1, color='y1'), linewidth=1) + scale_color_manual(name = 'Functions', values=c('y'='blu
e','y1'='#c00000')) + annotate('text', x=1.5, y = 1, label='y = cos(x/2)*sin(x/2)') + annotate
('text', x=1.27, y=.85, label='y1 = -(x/2)^3') + ggtitle('Intersection of Two Functions') + them
e(plot.title = element_text(hjust = 0.5)) + geom_point(data=subset(df1,y==y1), aes(x=x,y=y1), co
lor='black', size=2) + geom_text(data=subset(df1,y==y1), aes(x+0.5,y,label=lab2))
```



Section 4: (12 points) Use the “trees” dataset for the following items. This dataset has three variables (Girth, Height, Volume) on 31 felled black cherry trees.

(4)(a) Use *data(trees)* to load the dataset. Check and output the structure with *str()*. Use *apply()* to return the mean values for the three variables. Output these values. Using R and logicals, determine the number of trees with Volume greater than the mean Volume; effectively, how many rows have Volume greater than the mean Volume.

```
data(trees)
str(trees)
```

```
## 'data.frame':    31 obs. of  3 variables:
##  $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
##  $ Height: num  70 65 63 72 81 83 66 75 80 75 ...
##  $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...
```

```
apply(trees, 2, mean)
```

```
##    Girth   Height   Volume
## 13.24839 76.00000 30.17097
```

```
sum(trees$Volume>mean(trees$Volume))
```

```
## [1] 12
```

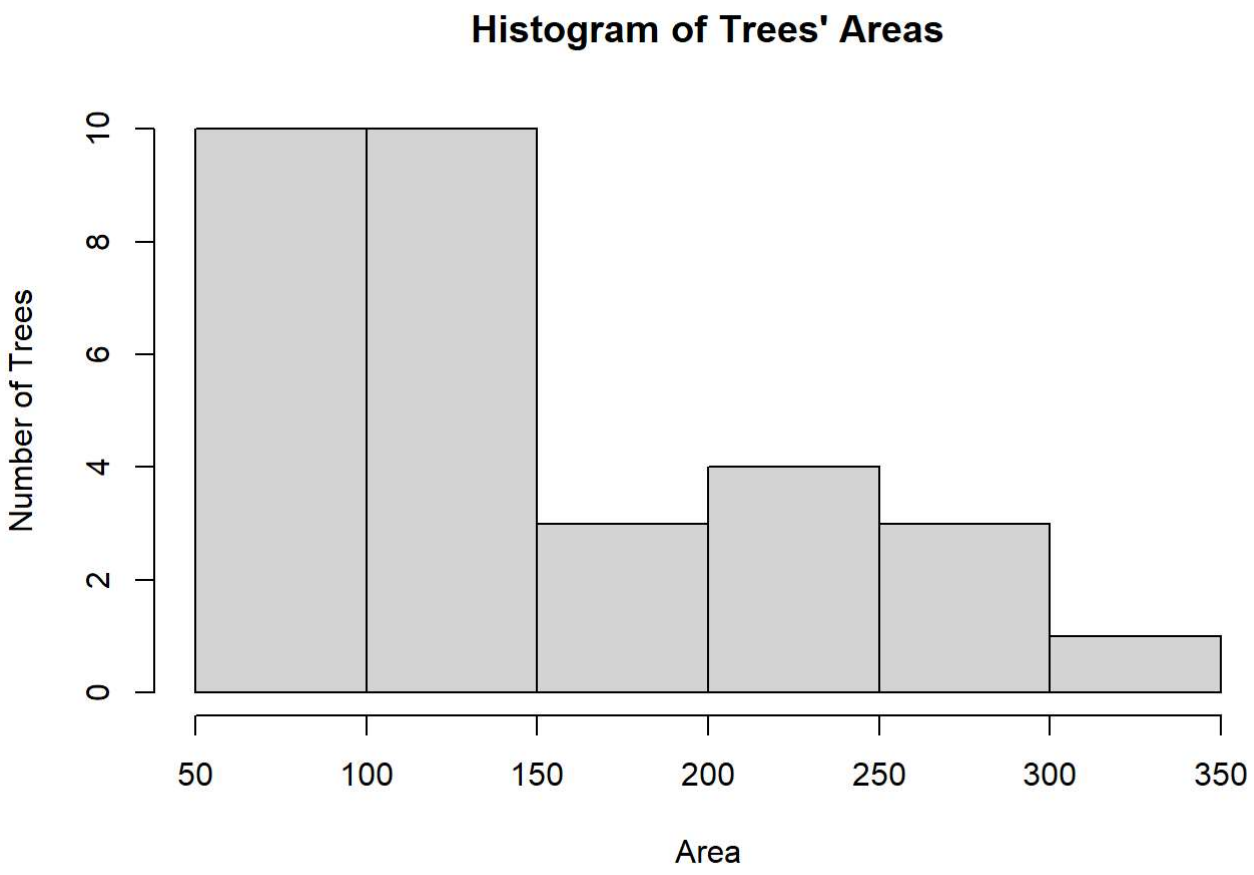
(4)(b) Girth is defined as the diameter of a tree taken at 4 feet 6 inches from the ground. Convert each diameter to a radius, r. Calculate the cross-sectional area of each tree using pi times the squared radius. What is the interquartile range (IQR) of areas?

```
trees['Radius'] = trees['Girth']/2
trees['Area'] = pi*(trees['Radius']^2)
IQR(trees$Area)
```

```
## [1] 87.1949
```

(4)(c) Create a histogram of the areas calculated in (b). Title and label the axis.

```
hist(trees$Area, main="Histogram of Trees' Areas", xlab='Area', ylab='Number of Trees')
```



(4)(d) Identify the tree with the largest area and output on one line its row number and three measurements.

```
trees[which.max(trees$Area),]
```

```
##    Girth Height Volume Radius     Area
## 31  20.6     87     77   10.3 333.2916
```

Section 5: (12 points) The Student's t distribution is an example of a symmetric, bell-shaped distribution but with 'heavier' tails than a normal distribution. This problem involves comparing the two.

5(a) Use *set.seed(9999)* and *rt()* with n = 100, df = 10 to generate a random sample designated as y. Generate a second random sample designated as x with *set.seed(123)* and *rnorm()* using n = 100, mean = 0 and sd = 1.25.

Generate a new object using *cbind(x, y)*. Do not output this object; instead, assign it to a new name. Pass this object to *apply()* and compute the inter-quartile range (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.).

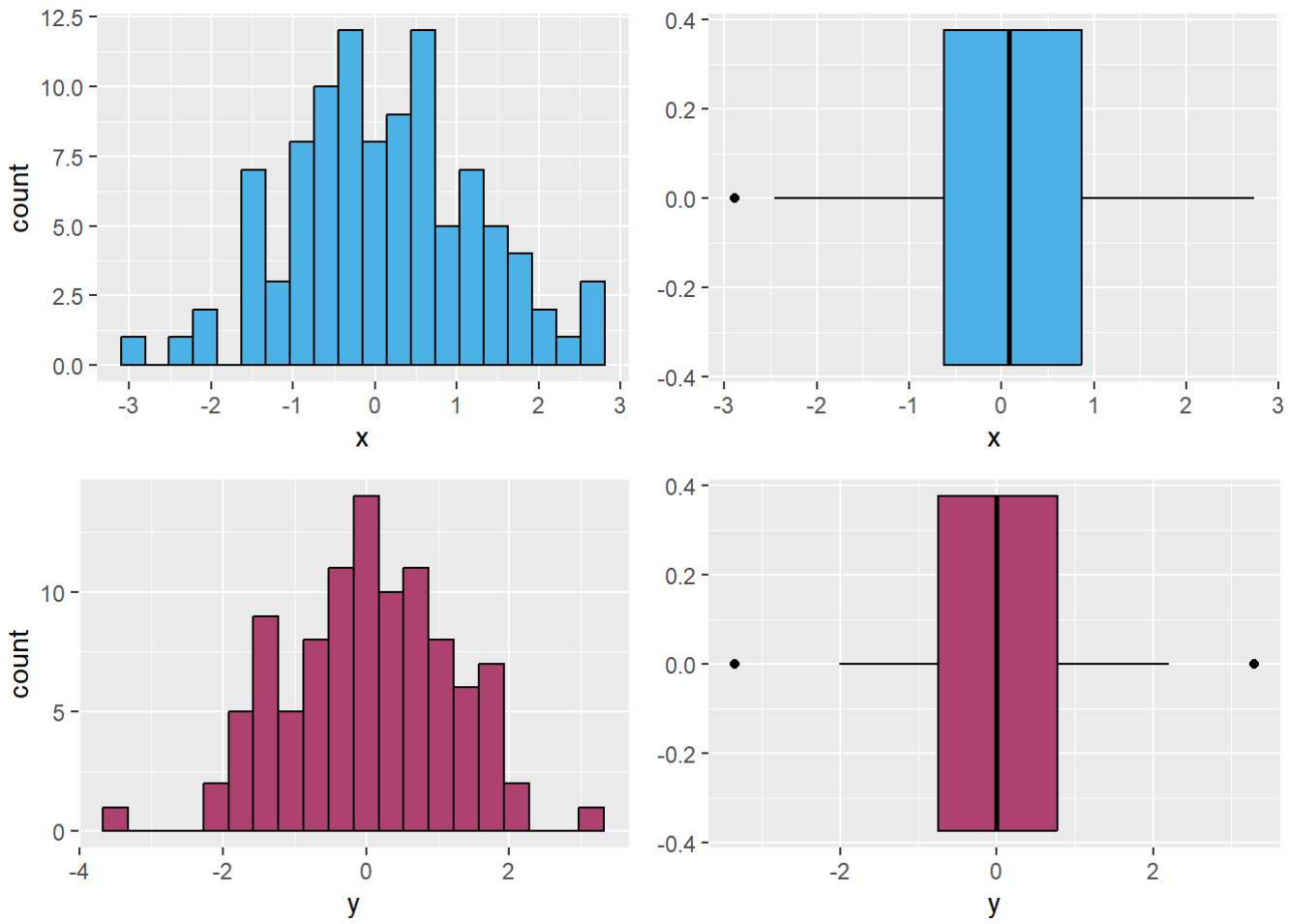
For information about *rt()*, use *help(rt)* or *?rt()*. **Do not output x or y.**

```
set.seed(9999)
y = rt(n=100, df=10)
set.seed(123)
x = rnorm(n=100, mean=0, sd=1.25)
new_name = cbind(x, y)
round(apply(new_name, 2, IQR), 4)
```

```
##      x      y
## 1.4821 1.5242
```

(5)(b) This item will illustrate the difference between a normal and heavy-tailed distribution. For base R plots, use *par(mfrow = c(2, 2))* to generate a display with four diagrams; *grid.arrange()* for ggplots. On the first row, for the normal results, present a histogram and a horizontal boxplot for x in color. For the t results, present a histogram and a horizontal boxplot for y in color.

```
df = data.frame(x,y)
library(gridExtra)
p1 = ggplot(data = df, aes(x = x)) + geom_histogram(color = "black", fill = "#4DB3E6", bins=20)
p2 = ggplot(data=df, aes(x=x)) + geom_boxplot(color='black', fill='#4DB3E6')
p3 = ggplot(data = df, aes(x = y)) + geom_histogram(color = "black", fill = "#AE4371", bins=20)
p4 = ggplot(data=df, aes(x=y)) + geom_boxplot(color='black', fill='#AE4371')
grid.arrange(p1,p2,p3,p4, nrow=2)
```



(5)(c) QQ plots are useful for detecting the presence of heavy-tailed distributions. Present side-by-side QQ plots, one for each sample, using *qqnorm()* and *qqline()*. Add color and titles. In base R plots, “cex” can be used to control the size of the plotted data points and text; “size” for ggplot2 figures. Lastly, determine if there are any extreme outliers in either sample.Remember extreme outliers are based on 3 multiplied by the IQR in the box plot. R uses a default value of 1.5 times the IQR to define outliers (not extreme) in both boxplot and boxplot stats.

```
par(mfrow=c(1,2))
qqnorm(df$x, cex=1, pch=1, col='cyan', frame=FALSE, main='x', xlim=c(-3,4), ylim=c(-3,4))
qqline(df$x, col="darkblue", lwd=2)
qqnorm(df$y, ces=1, pch=1, col='pink', frame=FALSE, main='y', xlim=c(-3,4), ylim=c(-3,4))
```

```
## Warning in plot.window(...): "ces" is not a graphical parameter
```

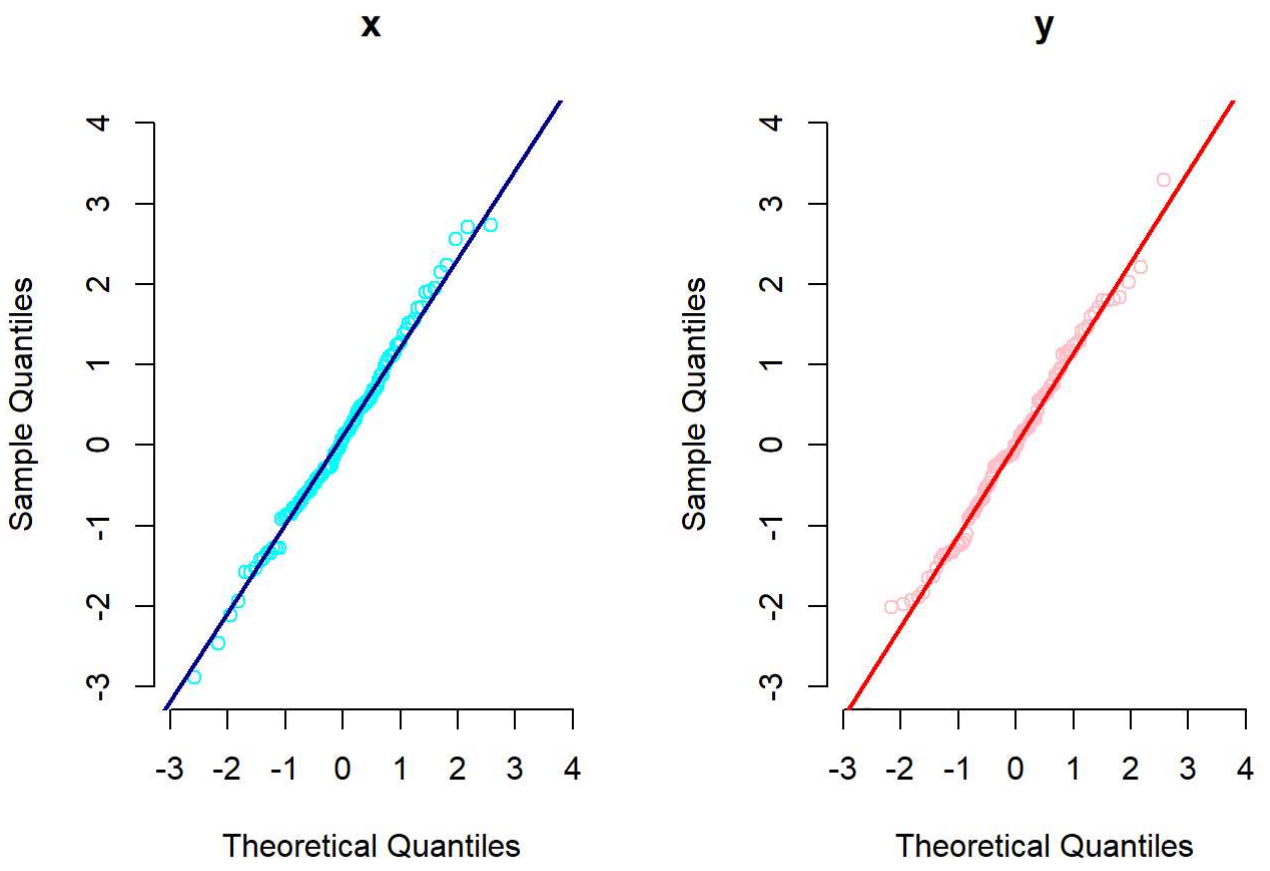
```
## Warning in plot.xy(xy, type, ...): "ces" is not a graphical parameter
```

```
## Warning in axis(side = side, at = at, labels = labels, ...): "ces" is not a
## graphical parameter

## Warning in axis(side = side, at = at, labels = labels, ...): "ces" is not a
## graphical parameter
```

```
## Warning in title(...): "ces" is not a graphical parameter
```

```
qqline(df$y, col="red", lwd=2)
```




```
cat('\nNumber of Extreme Outliers in x:', sum(x>(as.double(quantile(x)[4])+3*IQR(x)))+sum(x<as.d
ouble(quantile(x)[2])-3*IQR(x)))
```

```
##
## Number of Extreme Outliers in x: 0
```

```
cat('\nNumber of Extreme Outliers in y:', sum(y>(as.double(quantile(y)[4])+3*IQR(y)))+sum(x<as.d
ouble(quantile(y)[2])-3*IQR(y)))
```

```
##
## Number of Extreme Outliers in y: 0
```