Gesheva_Mariana

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, in this order, and output the final, resulting vector. Do not round any values, unless requested. * A sequence of integers from 0 to 4, inclusive. * The number 13 * Three repetitions of the vector c(2, -5.1, -23). * The arithmetic sum of 7/42, 3 and 35/42

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(ggplot2)

vector <- c(0, 1, 2, 3, 4, 13, rep(c(2, -5.1, -23), times = 3), sum((7/42), 3, (3
5/42)))
dput(vector)
```

```
## c(0, 1, 2, 3, 4, 13, 2, -5.1, -23, 2, -5.1, -23, 2, -5.1, -23,
## 4)
```

(1)(b) Sort the vector created in (1)(a) in ascending order. Output this result. Determine the length of the resulting vector and assign to “L”. Output L. Generate a descending sequence starting with L and ending with 1. Add this descending sequence arithmetically the sorted vector. This is vector addition, not vector combination. Output the contents. Do not round any values.

```
vector_sorted <- sort(vector, decreasing = FALSE)
dput(vector_sorted)
```

```
## c(-23, -23, -23, -5.1, -5.1, -5.1, 0, 1, 2, 2, 2, 2, 3, 4, 4,
## 13)
```

```
vector_sorted
```

```
##  [1] -23.0 -23.0 -23.0  -5.1  -5.1  -5.1   0.0   1.0   2.0   2.0   2.0
## [12]   2.0   3.0   4.0   4.0  13.0
```

```
l <- length(vector_sorted)
l
```

```
## [1] 16
```

```
#

l:1
```

```
##  [1] 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
```

```
descending_sequence <- l:1

vector_sorted
```

```
##  [1] -23.0 -23.0 -23.0  -5.1  -5.1  -5.1   0.0   1.0   2.0   2.0   2.0
## [12]   2.0   3.0   4.0   4.0  13.0
```

```
vector_addition <- vector_sorted + descending_sequence

vector_addition
```

```
##  [1] -7.0 -8.0 -9.0  7.9  6.9  5.9 10.0 10.0 10.0  9.0  8.0  7.0  7.0  7.0
## [15]  6.0 14.0
```

```
dput(vector_addition)
```

```
## c(-7, -8, -9, 7.9, 6.9, 5.9, 10, 10, 10, 9, 8, 7, 7, 7, 6, 14
## )
```

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

```
vector_addition[1]
```

```
## [1] -7
```

```
vector_addition[length(vector_addition)]
```

```
## [1] 14
```

```
first_and_last_vector <- c(vector_addition[1], vector_addition[length(vector_addi
tion)])
first_and_last_vector
```

```
## [1] -7 14
```

```
not_first_and_last_vector <- c(vector_addition[3:length(vector_addition) - 1])
not_first_and_last_vector
```

```
##  [1] -8.0 -9.0  7.9  6.9  5.9 10.0 10.0 10.0  9.0  8.0  7.0  7.0  7.0  6.0
```

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

```
sum(first_and_last_vector)
```

```
## [1] 7
```

```
sum(not_first_and_last_vector)
```

```
## [1] 77.7
```

```
reconstructed_vector = c(first_and_last_vector, not_first_and_last_vector)
sum(reconstructed_vector)
```

```
## [1] 84.7
```

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.”

```
trig_function <- function(x) {
  y <- sin(x/2) + cos(x/2)
  return(y)
}

trig_function(2)
```

```
## [1] 1.381773
```

(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 maximum 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.

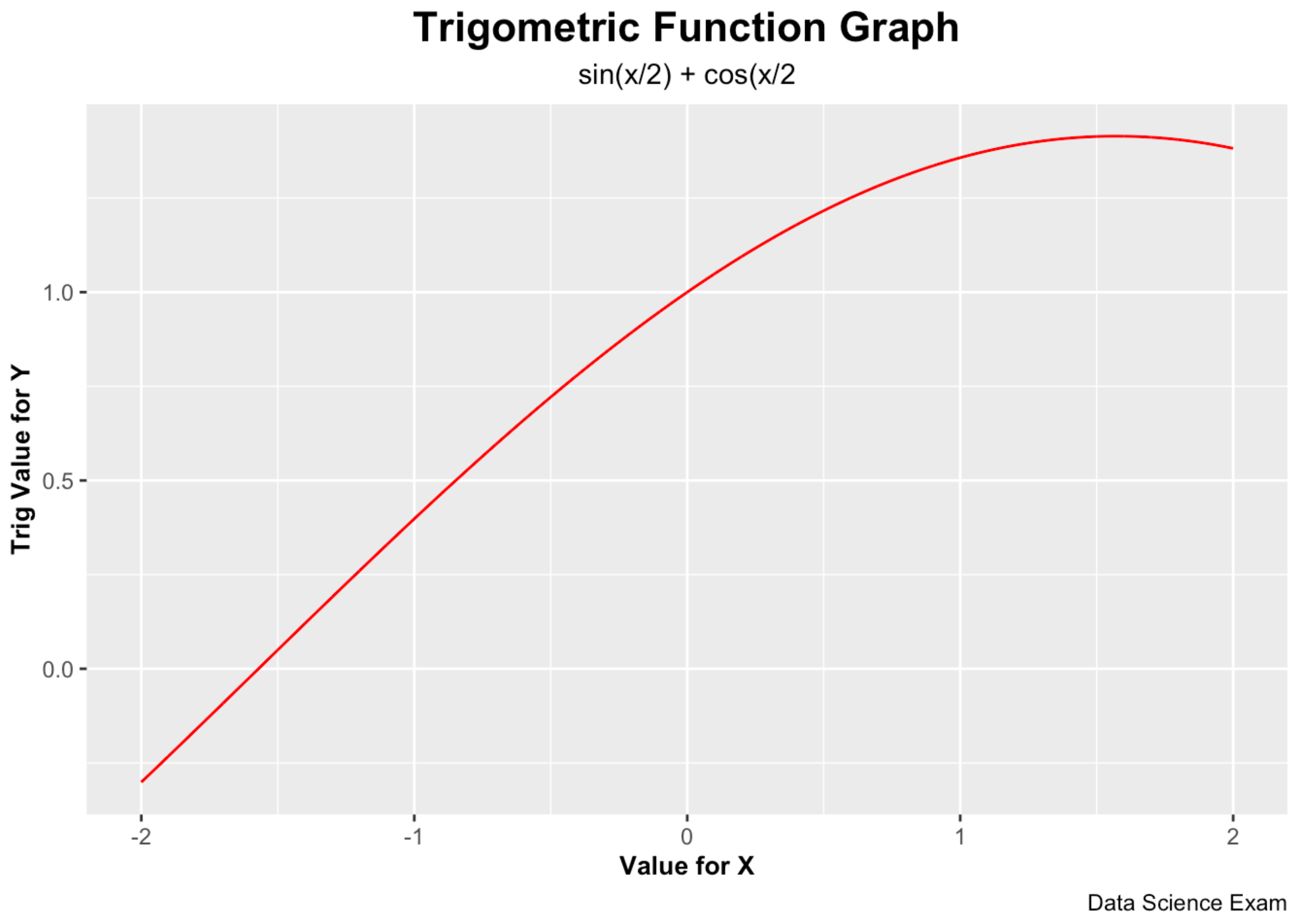
Finding the two desired values can be accomplished in as few as two lines of code. Do not use packages or programs you may find on the internet or elsewhere. Do not output the other elements of the vectors x and y. Relevant coding methods are given in the *Quick Start Guide for R*.

```
x <- seq(from = -2, to = 2, length.out = 4001)



trig_function_for_x <- trig_function(x)
trig_function_for_x <- as.data.frame(trig_function_for_x)

max_value_for_y_in_trig_function_for_x <- round(max(trig_function(x)),3)


x <- as.data.frame(x)

x <- x %>%
  mutate(corresponding_trig_function_value_y = sin(x/2) + cos(x/2))



x_hightest_value <- x %>%
  arrange(desc(corresponding_trig_function_value_y)) %>%
  slice(1) %>%
  select(x)

dput(x_hightest_value)
```

```
## structure(list(x = 1.571), row.names = c(NA, -1L), class = "data.frame")
```

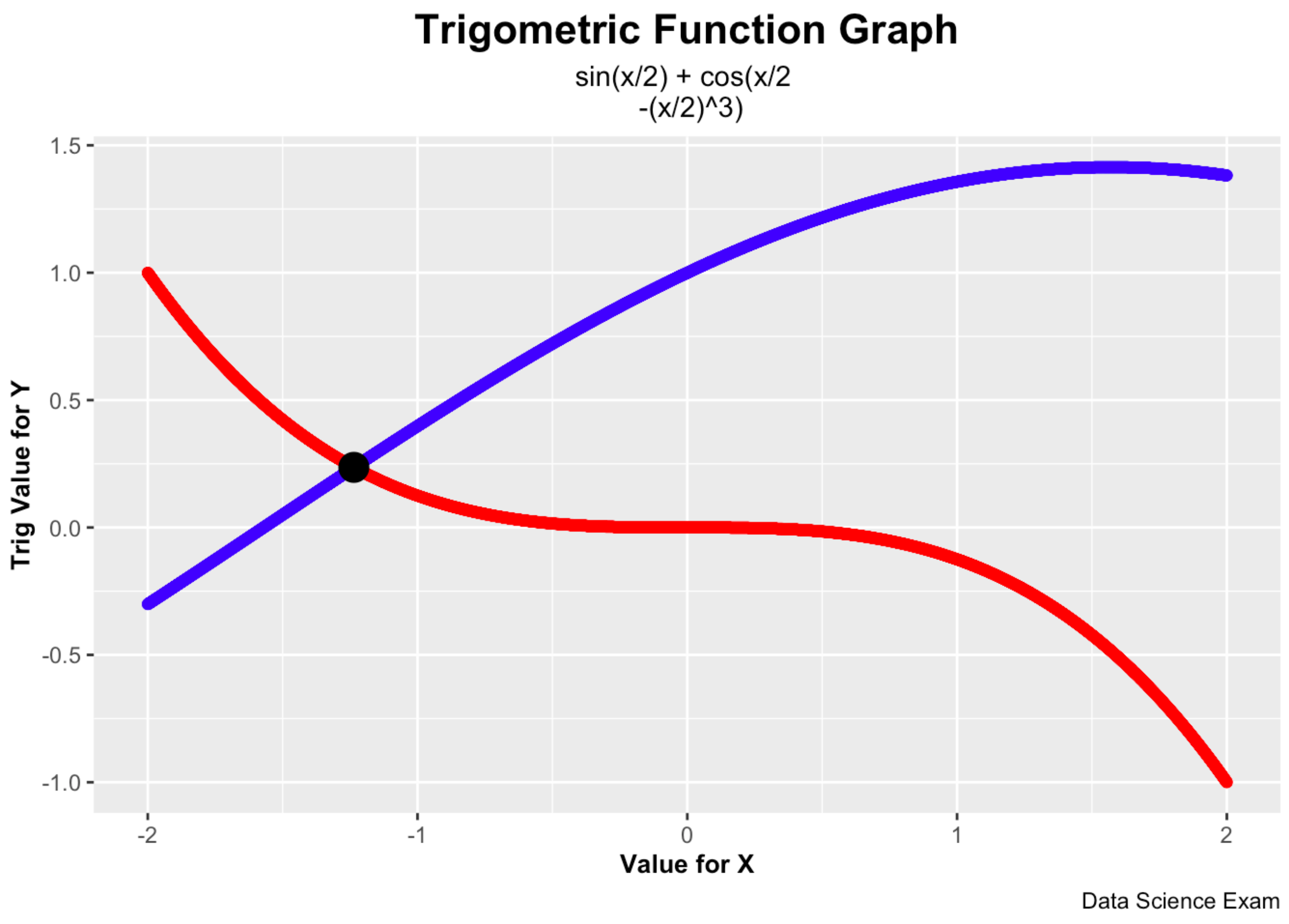
(2)(c) Plot y versus x in color, with x on the horizontal axis. Show the location of the maximum value of y determined in 2(b). Show the values of x and y corresponding to the maximum value of y in the display. Add a title and other features such as text annotations. Text annotations may be added via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

```
ggplot(data = x, aes(x, corresponding_trig_function_value_y)) +
  geom_line(color = "red") +
  labs(title = "Trigometric Function Graph", subtitle = "sin(x/2) + cos(x/2") +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme(plot.subtitle = element_text(hjust = 0.5)) +
  theme(plot.title = (element_text(size = 16, face = "bold"))) +
  theme(axis.title = (element_text(size = 10, face = "bold"))) +
  labs(caption = "Data Science Exam") +
  xlab('Value for X') +
  ylab('Trig Value for Y')
```



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.

```
function.one <- function(x) {
  y <- sin(x/2) + cos(x/2)
  round(y,3)
  return(y)
}

function.two <- function(x) {
  y <- -(x/2)^3
  round(y,3)
  return(y)
}

 x <- seq(from = -2, to = 2, length.out = 4001)
#
 x <- as.data.frame(x)

x_0 <- x %>%
  mutate(function.one = sin(x/2) + cos(x/2)) %>%
  mutate(function.two = -(x/2)^3) %>%
  mutate(function.one = round(function.one, 3)) %>%
  mutate(function.two = round(function.two, 3)) %>%
  mutate(where_y_equals_y = function.two - function.one) %>%
  filter(where_y_equals_y == 0) %>%
  select(x, function.one)

x_0 <- as.data.frame(x_0)

x_0 <- x_0 %>%
  mutate(equation = "")

colnames(x_0) <- c('x_values', 'y_values', 'equation')

x_1 <- x %>%
  mutate(function.one = sin(x/2) + cos(x/2)) %>%
  mutate(function.one = round(function.one, 3))

x_1 <- x_1 %>%
  mutate(equation = "sin(x/2) + cos(x/2)")

colnames(x_1) <- c('x_values', 'y_values', 'equation')

x_1 <- as.data.frame(x_1)

x_2 <- x %>%
  mutate(function.two = -(x/2)^3) %>%
  mutate(function.two = round(function.two, 3))

x_2 <- x_2 %>%
  mutate(equation = "-(x/2)^3)")

colnames(x_2) <- c('x_values', 'y_values', 'equation')

x_2 <- as.data.frame(x_2)

ggplot(x_1, aes(x = x_values, y = y_values)) +
  geom_point(color = "blue") +
  geom_point(data = x_2, aes(x = x_values, y = y_values), color = "red") +
  geom_point(data = x_0, aes(x = x_values, y = y_values), size = 5) +
  labs(title = "Trigometric Function Graph", subtitle = "sin(x/2) + cos(x/2 \n
-(x/2)^3)") +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme(plot.subtitle = element_text(hjust = 0.5)) +
  theme(plot.title = (element_text(size = 16, face = "bold"))) +
  theme(axis.title = (element_text(size = 10, face = "bold"))) +
  labs(caption = "Data Science Exam") +
  xlab('Value for X') +
  ylab('Trig Value for Y')
```



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 median values for the three variables. Output these values. Using R and logicals, output the row number and the three measurements - Girth, Height and Volume - of any trees with Girth equal to median Girth. It is possible to accomplish this last request with one line of code.

```
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, FUN = median)
```

```
##  Girth Height Volume
##   12.9   76.0   24.2
```

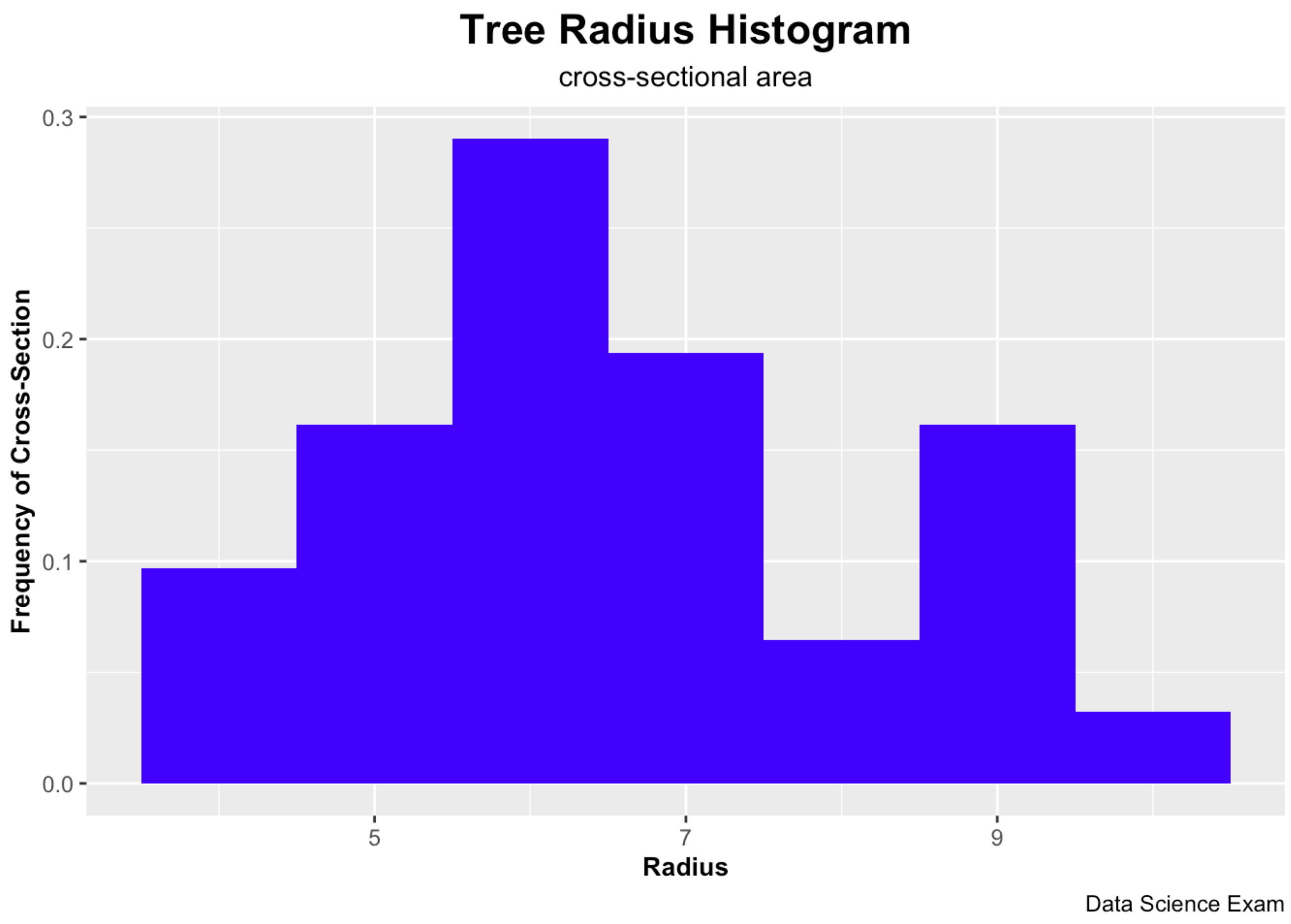
```
# median_girth[1] <- apply(trees, 2, FUN = median)
# median_girth[1]

trees_with_median_girth <- trees %>%
  mutate(row_number = row_number()) %>%
  filter(Girth == 12.9)
```

(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. Present a stem-and-leaf plot of the radii, and a histogram of the radii in color. Plot Area (y-axis) versus Radius (x-axis) in color showing the individual data points. Label appropriately.

```
data(trees)
trees <- trees %>%
  mutate(row_number = row_number()) %>%
  mutate(radius = Girth / 2) %>%
  mutate(cross_sectional = radius ^ 2 *3.14)

stem_and_leaf_cross_sectional <- stem(trees$cross_sectional)
```

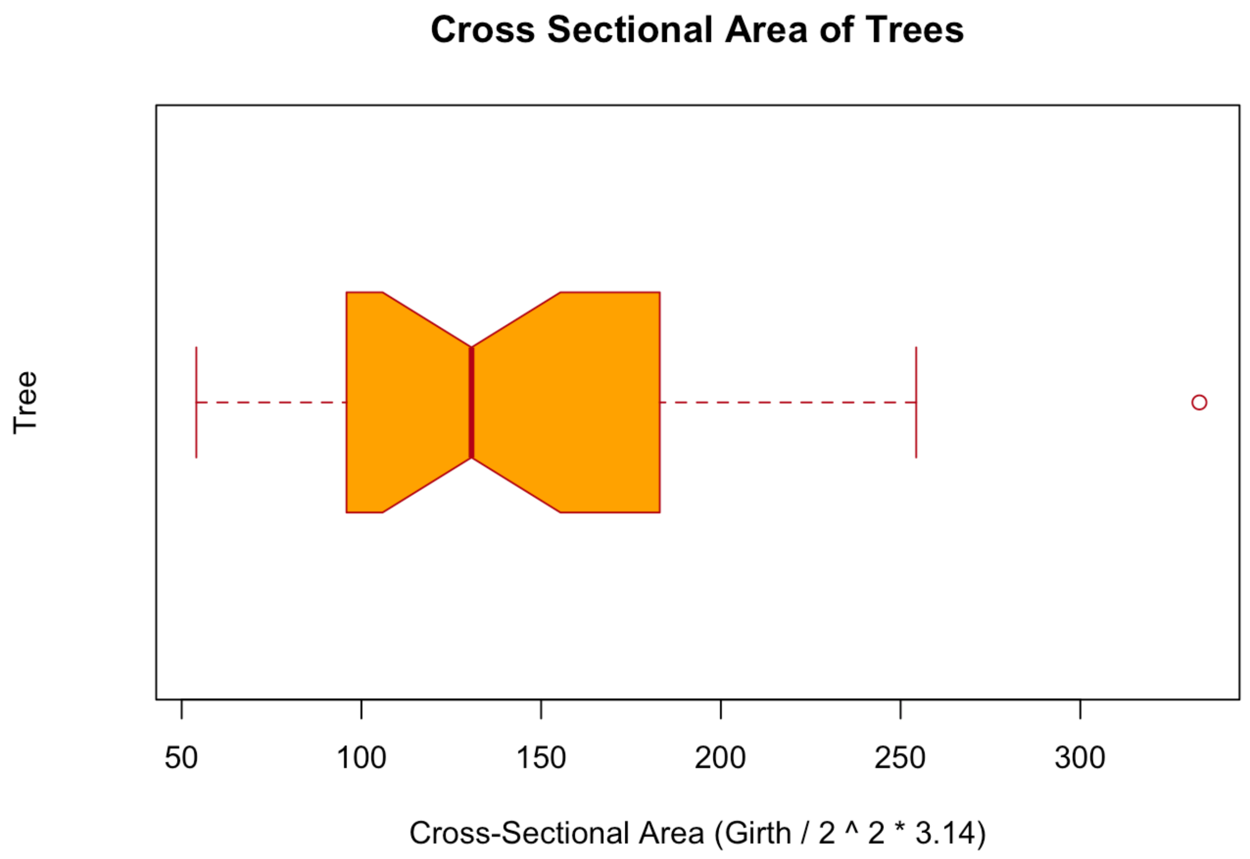
```
##
##   The decimal point is 2 digit(s) to the right of the |
##
##   0 | 56699999
##   1 | 0000011334
##   1 | 55567
##   2 | 0134
##   2 | 555
##   3 | 3
```

```
ggplot(trees, aes(x = radius, y = ..density..)) +
  geom_histogram(binwidth = 1,fill="blue") +
  labs(title = "Tree Radius Histogram", subtitle = "cross-sectional area") +
  theme(plot.title = element_text(hjust = 0.5)) +
  theme(plot.subtitle = element_text(hjust = 0.5)) +
  theme(plot.title = (element_text(size = 16, face = "bold"))) +
  theme(axis.title = (element_text(size = 10, face = "bold"))) +
  labs(caption = "Data Science Exam") +
  xlab('Radius') +
  ylab('Frequency of Cross-Section')
```



(4)(c) Present a horizontal, notched, colored boxplot of the areas calculated in (b). Title and label the axis.

```
boxplot(trees$cross_sectional,
main = "Cross Sectional Area of Trees",
xlab = "Cross-Sectional Area (Girth / 2 ^ 2 * 3.14)",
ylab = "Tree",
col = "orange",
border = "brown",
horizontal = TRUE,
notch = TRUE
)
```



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

```
boxplot.stats(trees$cross_sectional, coef = 1.5, do.conf = TRUE, do.out = TRUE)
```

```
## $stats
## [1]  54.07865  95.85242 130.63185 183.00313 254.34000
##
## $n
## [1] 31
##
## $conf
## [1] 105.9005 155.3632
##
## $out
## [1] 333.1226
```

```
tree_w_largest_area <- trees %>%
  arrange(desc(cross_sectional)) %>%
  slice(1)

tree_w_largest_area
```

```
##   Girth Height Volume row_number radius cross_sectional
## 1  20.6     87     77         31   10.3        333.1226
```

Section 5: (12 points) The exponential distribution is an example of a right-skewed distribution with outliers. This problem involves comparing it with a normal distribution which typically has very few outliers.

5(a) Use *set.seed(124)* and *rexp()* with n = 100, rate = 5.5 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 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.).

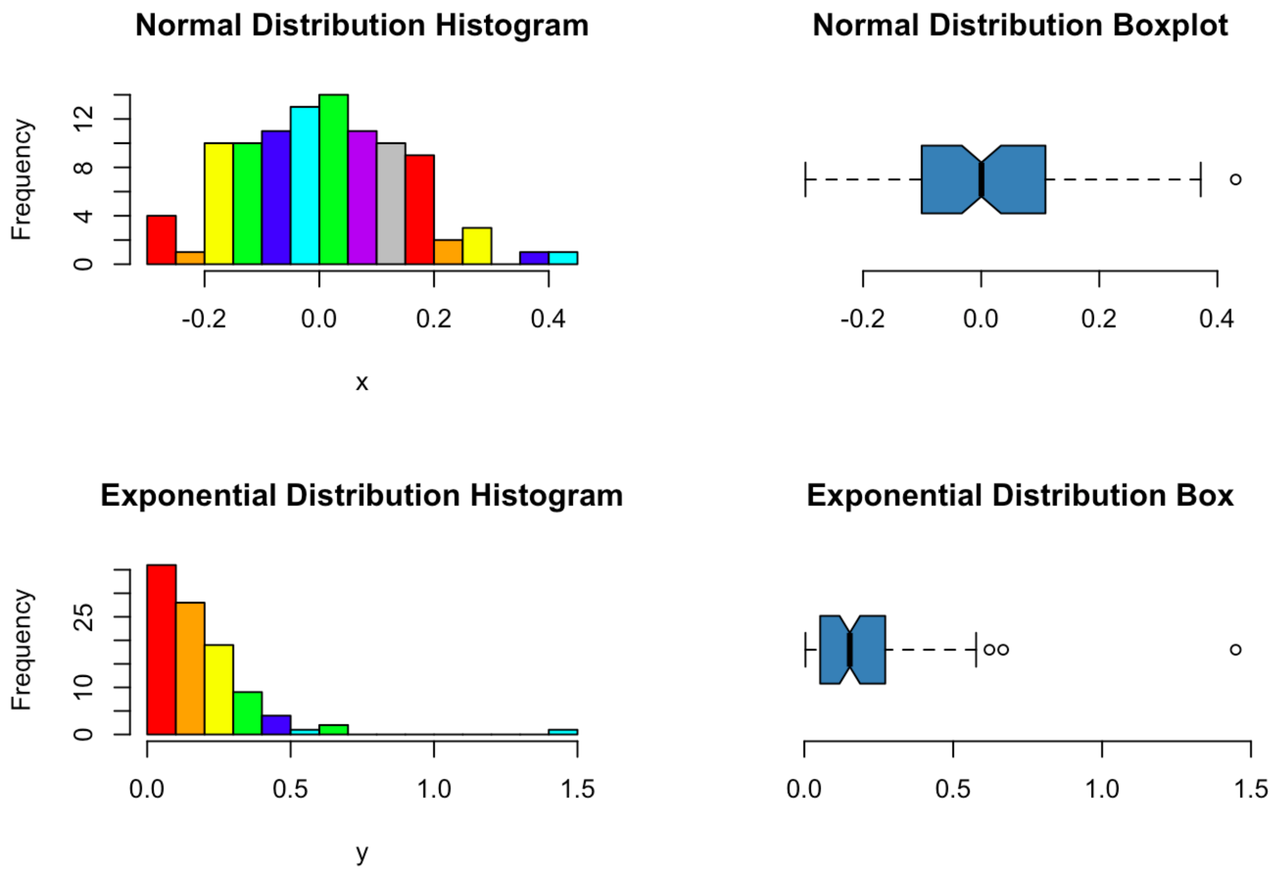
For information about *rexp()*, use *help(rexp)* or *?rexp()*. **Do not output x or y.**

```
?rexp()

set.seed(124)
y =  rexp(n = 100, rate = 5.5)

set.seed(127)
x <- rnorm(n=100, mean=0, sd=.15)

new_cbind_object <- cbind(x, y)

round(apply(new_cbind_object, 2, IQR), 4)
```

```
##      x      y
## 0.2041 0.2164
```

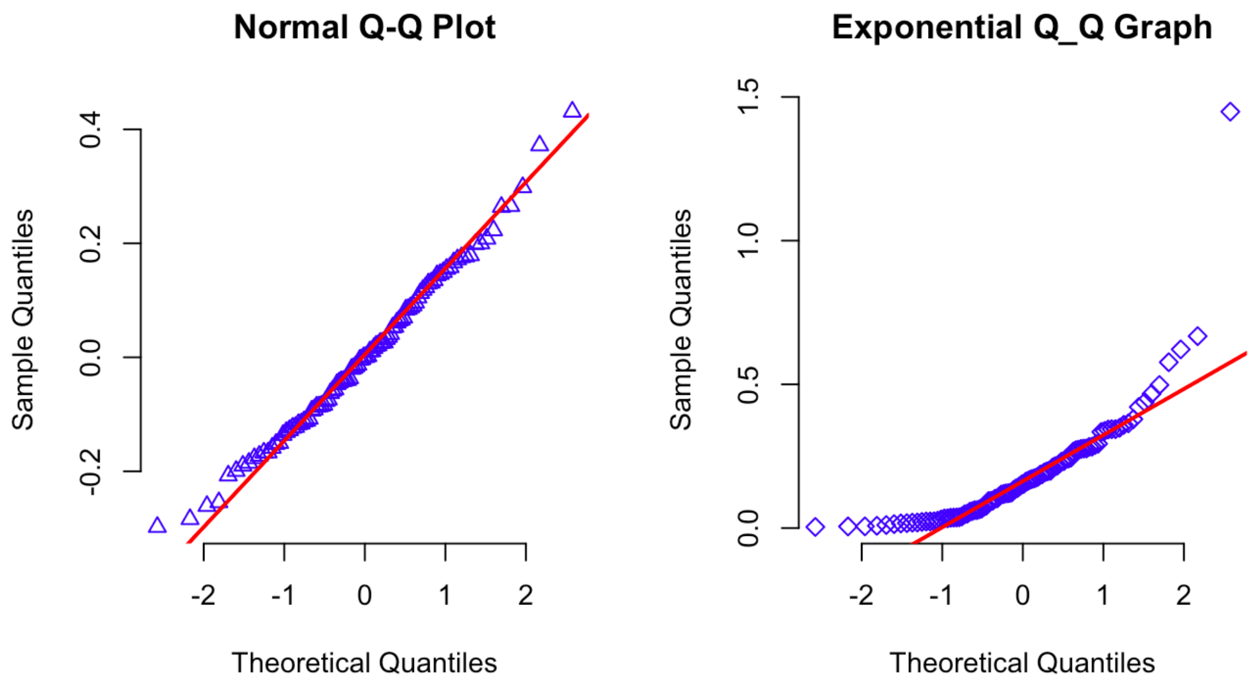
(5)(b) This item will illustrate the difference between a right-skewed distribution and a symmetric one. For base R plots, use *par(mfrow = c(2, 2))* to generate a display with four diagrams; *grid.arrange()* for ggplots. On the first row, for the normal results, present a histogram and a horizontal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

```
par(mfrow = c(2, 2))
colors = c("red", "orange", "yellow", "green", "blue", "cyan", "green", "purple",
"grey")
hist(x, breaks=15, main = "Normal Distribution Histogram", col=colors)
boxplot(x, col = "steelblue", range = 1.5, main = "Normal Distribution Boxplot",
        notch = TRUE, horizontal=TRUE, frame=F)

hist(y, breaks=15, main = "Exponential Distribution Histogram", col=colors)
boxplot(y, col = "steelblue", range = 1.5, main = "Exponential Distribution Box",
        notch = TRUE, horizontal=TRUE, frame=F)
```



(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. Lastly, determine if there are any extreme outliers in either sample.

```
par(mfrow = c(1, 2), pty="s", cex=.9)

qqnorm(x, pch=2, col="blue", frame=FALSE)
qqline(x, col = "red", lwd = 2)

qqnorm(y, pch=5, col="blue", frame=FALSE, main = "Exponential Q_Q Graph")
qqline(y, col = "red", lwd = 2)
```



```
boxplot.stats(x, coef=3)
```

```
## $stats
## [1] -0.2976325808 -0.1007240230  0.0003706968  0.1088532648  0.4310592908
##
## $n
## [1] 100
##
## $conf
## [1] -0.03274251  0.03348391
##
## $out
## numeric(0)
```

```
boxplot.stats(y, coef=3)
```

```
## $stats
## [1] 0.003880211 0.053278194 0.152793270 0.271774062 0.667719381
##
## $n
## [1] 100
##
## $conf
## [1] 0.1182709 0.1873156
##
## $out
## [1] 1.448679
```