John Barker, Programming with R Assignment 1

#(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

```
(a=seq(0,4))
(b=13)
(c=(rep(c(2,-5.1,-23),3)))
(d=sum(c(7/42,3,35/42)))
 (abcd=c(a,b,c,d))
```

```
> (a=seq(0,4))
[1] 0 1 2 3 4
> (b=13)
[1] 13
> (c=(rep(c(2,-5.1,-23),3)))
[1]   2.0  -5.1 -23.0   2.0  -5.1 -23.0   2.0  -5.1 -23.0
> (d=sum(c(7/42,3,35/42)))
[1] 4
>
> (abcd=c(a,b,c,d))
 [1]   0.0   1.0   2.0   3.0   4.0  13.0   2.0  -5.1 -23.0   2.0  -5.1 -23.0   2.0  -5.1 -23.0   4.0
>
```

#(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.

```
(abcds = sort(abcd, decreasing = FALSE))

(L=length(abcds))

(Ldesc=seq(L,1))

(sumseqandabcd=Ldesc + abcd)
```

```
> (abcds = sort(abcd, decreasing = FALSE))
 [1] -23.0 -23.0 -23.0  -5.1  -5.1  -5.1   0.0   1.0   2.0   2.0   2.0   2.0   3.0   4.0   4.0
13.0
> (L=length(abcds))
[1] 16
> (Ldesc=seq(L,1))
 [1] 16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1
> (sumseqandabcd=Ldesc + abcds)
 [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  6.0 14.0
```

#(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.

```
(fl=sumseqandabcd[c(1,L)])

(notfl=sumseqandabcd[-c(1,L)])
```

```
> (fl=sumseqandabcd[c(1,L)])
[1] -7 14
> (notfl=sumseqandabcd[-c(1,L)])
 [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.

(origb=c(fl[1],notfl,fl[2]))
origb
(round(sum(origb),digits=2))
```

```
> (origb=c(fl[1],notfl,fl[2]))
 [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  6.0 14.0
> origb
 [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  6.0 14.0
> (round(sum(origb),digits=2))
[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."

sumsincosofhalfx <- 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
# 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(-2,2,length.out = 4001)
y=sumsincosofhalfx(x)

(i = which.max(y))
cat(sprintf("(x,y) = (%.03f,%0.3f)",round(x[i],3),round(y[i],3)))
```

```
> x=seq(-2,2,length.out = 4001)
> y=sumsincosofhalfx(x)
>
> (i = which.max(y))
[1] 3572
> cat(sprintf("(x,y) = (%.03f,%0.3f)",round(x[i],3),round(y[i],3)))
(x,y) = (1.571,1.414)
```

#(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

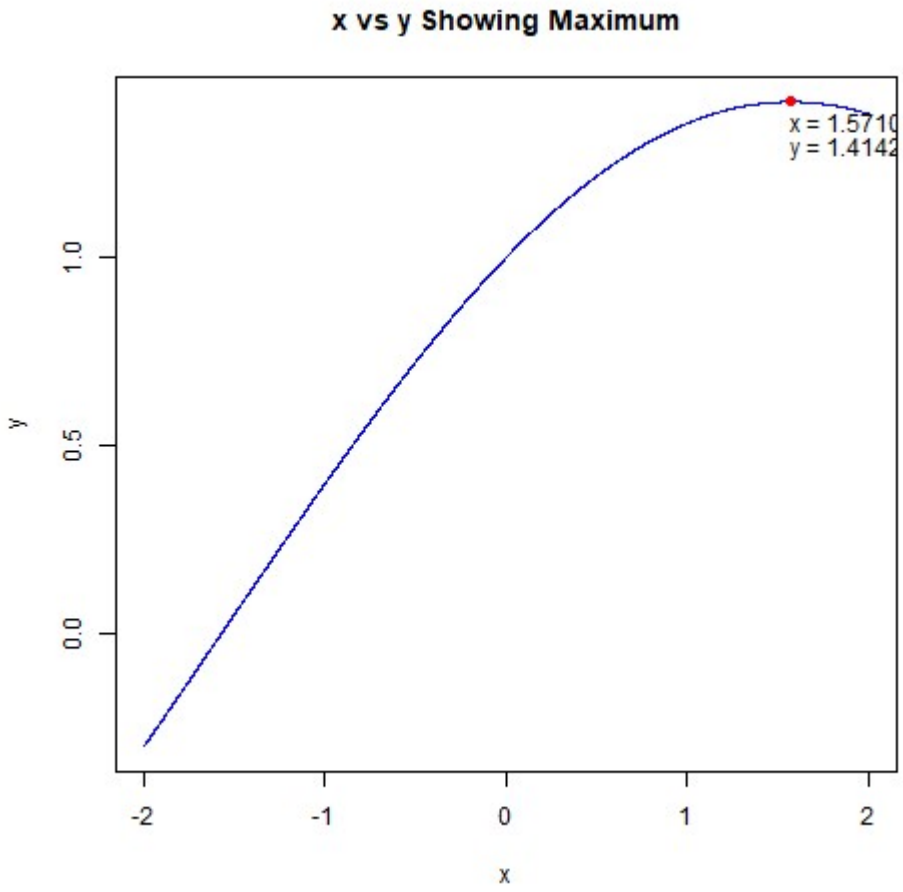
maximum value of y in the display. Add a title and other features such as text annotations.

Text annotations may be added via *text()* for base R plots and

geom_text() or *geom_label()* for ggplots.

```
png("fig2c.png")
plot(x,y,col='blue',type = 'l',main="x vs y Showing Maximum")
points(x[i],y[i],pch=16,col='red')
msgx=sprintf("x = %f", x[i])
msgy=sprintf("y = %f", y[i])
text(x[i],y[i],labels=msgx,adj=c(0,1.5))
text(x[i],y[i],labels=msgy,adj=c(0,2.7))
dev.off()
```

```
> png("fig2c.png")
> plot(x,y,col='blue',type = 'l',main="x vs y Showing Maximum")
> points(x[i],y[i],pch=16,col='red')
> msgx=sprintf("x = %f", x[i])
> msgy=sprintf("y = %f", y[i])
> text(x[i],y[i],labels=msgx,adj=c(0,1.5))
> text(x[i],y[i],labels=msgy,adj=c(0,2.7))
> dev.off()
```

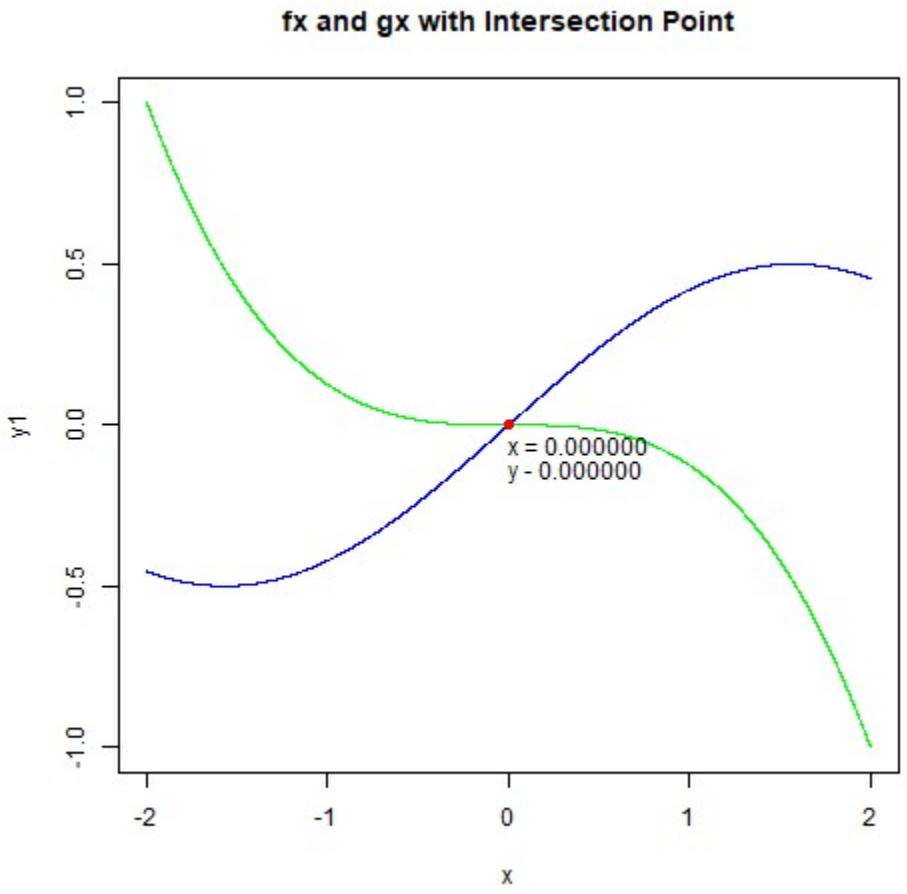

x vs y Showing Maximum
x = 1.5710
y = 1.4142
y
0.0
0.5
1.0
-2
-1
0
1
2
x

```
##### 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.

#define functions
fx <- function(x){
  y = sin(x/2) * cos(x/2)
  return(y)
}

gx <- function(x){
  y = -(x/2)^3
  return(y)
}

#Calculate y's for fx and gx from x
y1=fx(x)
y2=gx(x)

#Get the intersection of fx and gx
i = which(y1==y2)

#find the min and max for both curves
#to scale so both curves fit in plot
ylim = c(min(y1,y2),max(y1,y2))

png("fig3.png")
plot(x,y1,col='blue',type='l',ylim=ylim,main="fx and gx with Intersection Point")
```

```
lines(x,y2,col='green',type='l')
points(x[i],y1[i],pch=16,col='red')
msgx=sprintf("x = %f",x[i])
msgy=sprintf("y - %f",y1[i])
text(x[i],y1[i],labels=msgx,adj=c(0,1.5))
text(x[i],y1[i],labels=msgy,adj=c(0,2.7))
dev.off()
```

```
> fx <- function(x){
+   y = sin(x/2) * cos(x/2)
+   return(y)
+ }
>
> gx <- function(x){
+   y = -(x/2)^3
+   return(y)
+ }
>
> #Calculate y's for fx and gx from x
> y1=fx(x)
> y2=gx(x)
>
> #Get the intersection of fx and gx
> i = which(y1==y2)
>
> #find the min and max for both curves
> #to scale so both curves fit in plot
> ylim = c(min(y1,y2),max(y1,y2))
>
```

```
> png("fig3.png")
> plot(x,y1,col='blue',type='l',ylim=ylim,main="fx and gx with Intersection Point")
> lines(x,y2,col='green',type='l')
> points(x[i],y1[i],pch=16,col='red')
> msgx=sprintf("x = %f",x[i])
> msgy=sprintf("y - %f",y1[i])
> text(x[i],y1[i],labels=msgx,adj=c(0,1.5))
> text(x[i],y1[i],labels=msgy,adj=c(0,2.7))
> dev.off()
```



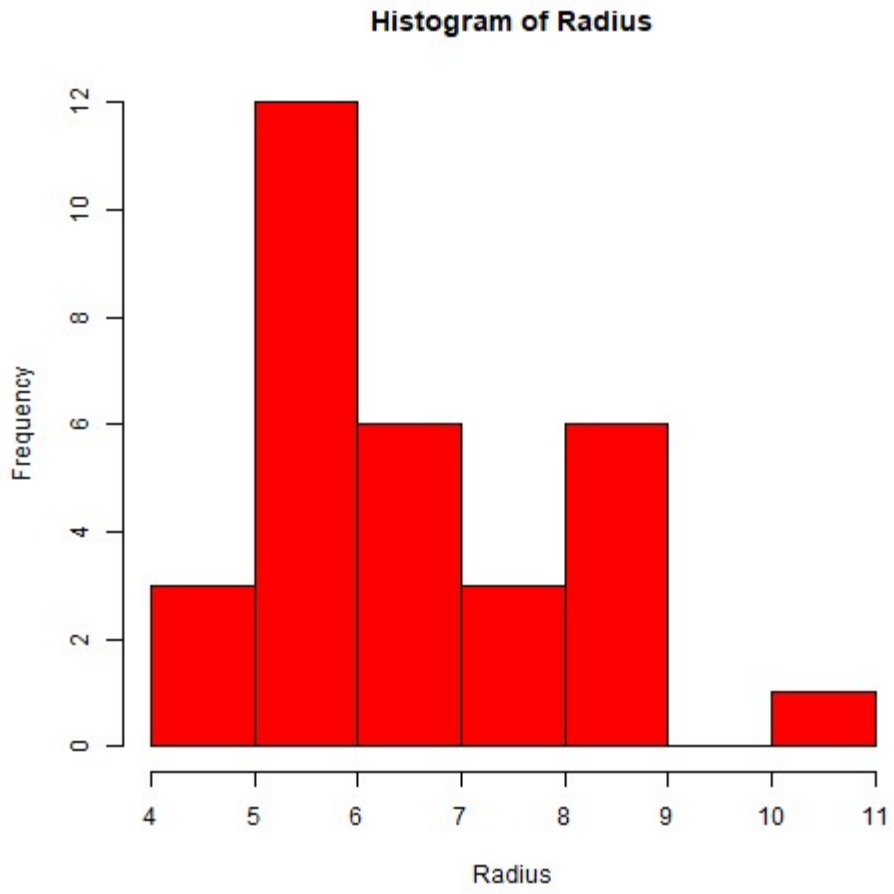
```
##### Section 4: (12 points) Use the "trees" dataset for the following items.
# This dataset has three variables (Girth, Height, Volume) on 31 felled black cherry trees.

#(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.

# --- test4a code ---

data(trees)
str(trees)

#find the median values
(mediantrees = apply(trees,MARGIN=2, median))

#show trees with median girth

#which value in mediantrees is Girth
g = which(names(mediantrees)=="Girth")

#Which trees have median girth
(i=which(mediantrees[g]==trees$Girth))

trees[i,]
```

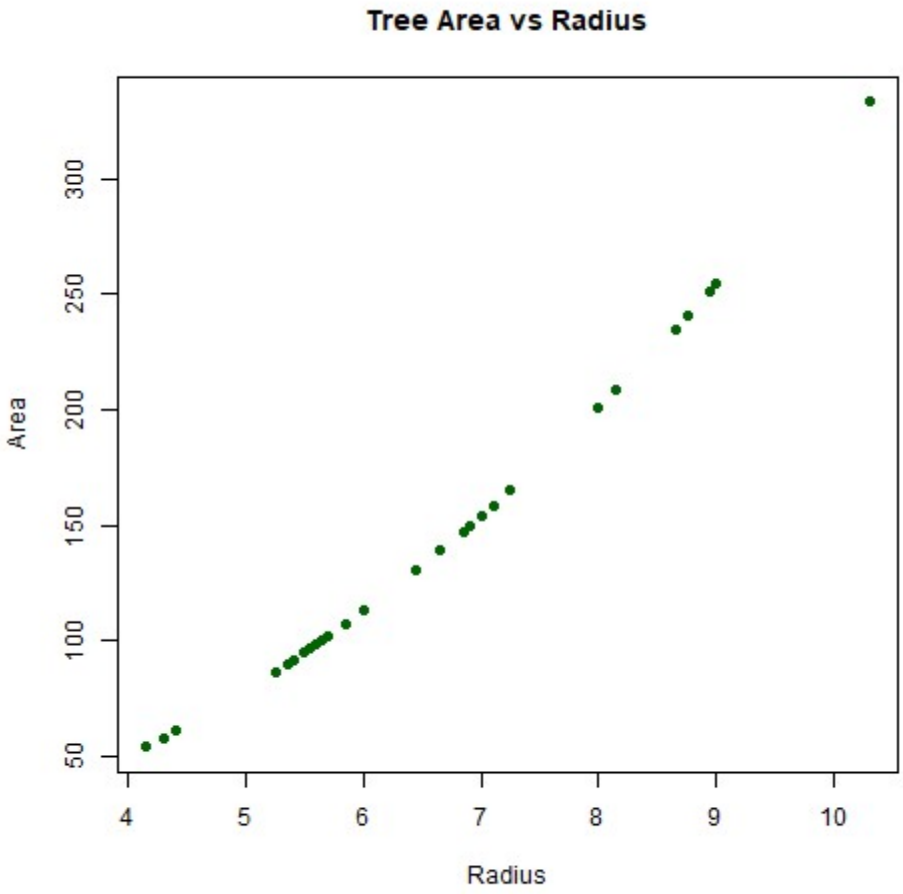
```
> data(trees)
> str(trees)
'data.frame':	31 obs. of  3 variables:
 $ Girth : num  8.3 8.6 8.8 10.5 10.7 10.8 11 11 11.1 11.2 ...
 $ Height: num  70 65 63 72 81 83 66 75 80 75 ...
 $ Volume: num  10.3 10.3 10.2 16.4 18.8 19.7 15.6 18.2 22.6 19.9 ...
>
> #find the median values
> (mediantrees = apply(trees,MARGIN=2, median))
 Girth Height Volume
  12.9   76.0   24.2
>
> #show trees with median girth
>
> #which value in mediantrees is Girth
> g = which(names(mediantrees)=="Girth")
>
> #Which trees have median girth
> (i=which(mediantrees[g]==trees$Girth))
[1] 16 17
>
> trees[i,]
   Girth Height Volume
16  12.9     74   22.2
17  12.9     85   33.8
>
```

```
#(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.

trees$Radius = trees$Girth/2
trees$Area = pi * trees$Radius^2

#stem plot of tree raii
stem(trees$Radius)

#histogram of radii
png("fig4bhist.png")
hist(trees$Radius,col='red', xlab = "Radius",breaks = "FD",main = "Histogram of Radius")
dev.off()

#scatter plot radius x area
png("fig4bscatter.png")
plot(x=trees$Radius, y=trees$Area,pch=16,col='darkgreen',main="Tree Area vs
Radius",xlab="Radius",ylab="Area")
dev.off()
```

```
> trees$Radius = trees$Girth/2
> trees$Area = pi * trees$Radius^2
>
> #stem plot of tree raii
> stem(trees$Radius)
```

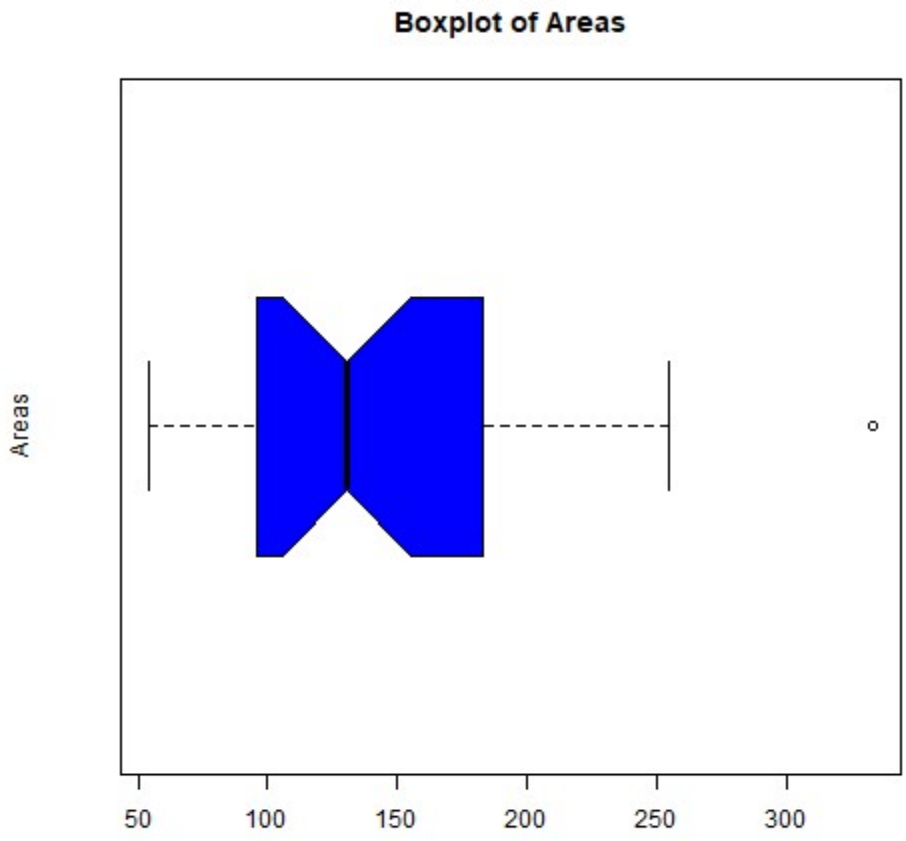
```
  The decimal point is at the |

   4 | 234
   5 | 34455667779
   6 | 055799
   7 | 013
   8 | 0278
   9 | 000
  10 | 3


>
> #histogram of radii
> png("fig4bhist.png")
> hist(trees$Radius,col='red', xlab = "Radius",breaks = "FD",main = "Histogram of Radius")
> dev.off()
null device
          1
>
> #scatter plot radius x area
> png("fig4bscatter.png")
> plot(x=trees$Radius, y=trees$Area,pch=16,col='darkgreen',main="Tree Area vs
Radius",xlab="Radius",ylab="Area")
> dev.off()
```


Histogram of Radius
Frequency
0
2
4
6
8
10
12
4
5
6
7
8
9
10
11
Radius


Tree Area vs Radius
Area
Radius
50
100
150
200
250
300
4
5
6
7
8
9
10

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

```
png("fig4c.png")
boxplot(trees$Area, col = 'blue', range=1.5, main="Boxplot of Areas", ylab="Areas",
    notch = TRUE, horizontal = TRUE)
dev.off()
```



```
#(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.

(areastats = boxplot.stats(trees$Area))

#get the outliers from boxplot stats
(outliers = areastats$out)
(med = areastats$stats[3])

#see if there are any extreme outliers
(iqr = areastats$stats[4]-areastats$stats[2])
(threshold = areastats$stats[4] + 3 * iqr)
outliers > threshold

#show measurements of largest tree without radius and area
trees[which.max(trees$Area),c(1:3)]
```

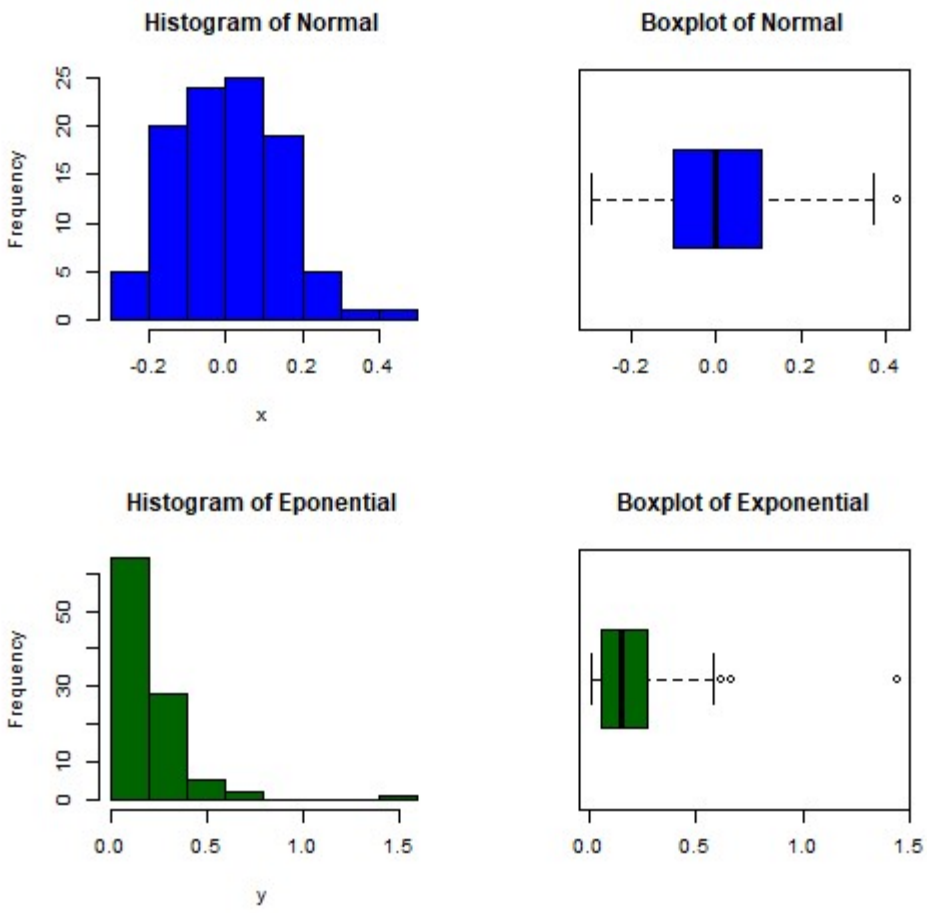
```
> (areastats = boxplot.stats(trees$Area))
$stats
[1]  54.10608  95.90104 130.69811 183.09595 254.46900

$n
[1] 31

$conf
[1] 105.9543 155.4420
```

```
$out
[1] 333.2916


>
> #get the outliers from boxplot stats
> (outliers = areastats$out)
[1] 333.2916
> (med = areastats$stats[3])
[1] 130.6981
>
> #see if there are any extreme outliers
> (iqr = areastats$stats[4]-areastats$stats[2])
[1] 87.1949
> (threshold = areastats$stats[4] + 3 * iqr)
[1] 444.6807
> outliers > threshold
[1] FALSE
>
> #show measurements of largest tree without radius and area
> trees[which.max(trees$Area),c(1:3)]
   Girth Height Volume
31  20.6     87     77
```

```
##### 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.**

#random exponential distribution
set.seed(124)
y = rexp(n=100, rate=5.5)

#random normal distribution
set.seed(127)
x = rnorm(n=100, mean=0, sd=.15)

#create a matrix from the two distributions
z = cbind(x,y)

#compute the inter-quartile range for each column
#note margin = 2 (by column)
round(apply(z,MARGIN=2, IQR),digits=4)
```

```
> #random exponential distribution
> set.seed(124)
> y = rexp(n=100, rate=5.5)
>
> #random normal distribution
> set.seed(127)
> x = rnorm(n=100, mean=0, sd=.15)
>
> #create a matrix from the two distributions
> z = cbind(x,y)
>
> #compute the inter-quartile range for each column
> #note margin = 2 (by column)
> round(apply(z,MARGIN=2, IQR),digits=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.


png("fig5b.png")

par(mfrow = c(2,2))

hist(x,col = 'blue',main = "Histogram of Normal")

boxplot(x,col = 'blue', horizontal = TRUE,main="Boxplot of Normal")

hist(y,col = 'darkgreen',main = "Histogram of Eponential")

boxplot(y,col = 'darkgreen',horizontal=TRUE,main="Boxplot of Exponential")

par(mfrow = c(1,1))

dev.off()
```



#(5)(c) QQ plots are useful for detecting the presence of heavy-tailed distributions.

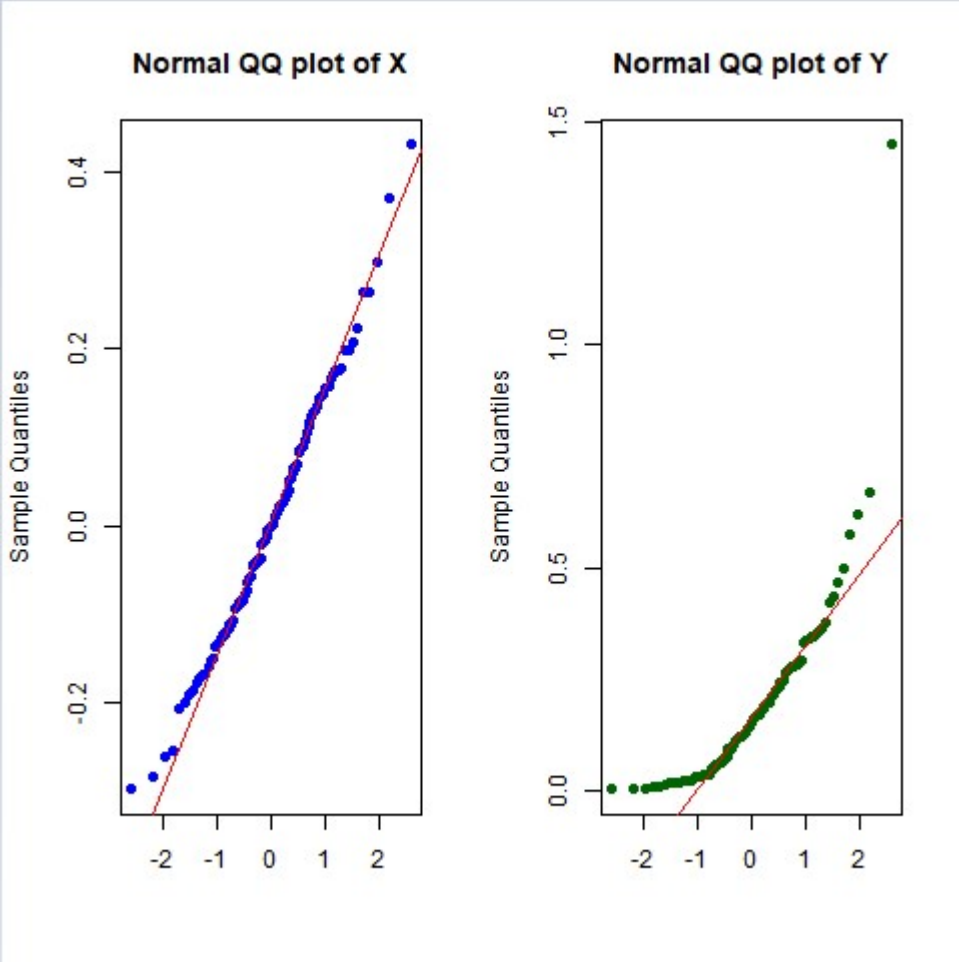
Present side-by-side QQ plots, one for each sample, using *qqnorm()* and *qqline()*.

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.

```
png("fig5c")
par(mfrow = c(1,2))
qqnorm(x,xlab="",col='blue',pch=16,main="Normal QQ plot of X")
qqline(x,col='red')
```

```
qqnorm(y,xlab="",col='darkgreen',pch=16,main="Normal QQ plot of Y")
qqline(y,col='red')
par(mfrow = c(1,1))
dev.off()
```



There is one outlier in the Normal QQ plot of Y.