Programming with R Assignment #2 (75 points) - Desilets_Steve

Test Items starts from here - There are 5 sections - 75 points total

Section 1: (15 points)

(1) R has probability functions available for use (Kabacoff, Section 5.2.3). Using one distribution to approximate another is not uncommon.

(1)(a) (6 points) The Poisson distribution may be used to approximate the binomial distribution if n > 20 and np < 7. Estimate the following binomial probabilities using *dpois()* or *ppois()* with probability p = 0.05, and n = 100. Then, estimate the same probabilities using *dbinom()* or *pbinom()*. Show the numerical results of your calculations.

i. The probability of exactly 0 successes.

```
#Estimate the probability of 0 successes using the Poisson Distribution
dpois(x = 0, lambda = .05*100)
```

```
## [1] 0.006737947
```

```
#Calculate the probability of 0 successes using the Binomial Distribution
dbinom(x = 0, size = 100, prob = .05)
```

```
## [1] 0.005920529
```

ii. The probability of fewer than 7 successes. Please note the following, taken from the Binomial Distribution R Documentation page, regarding the “lower.tail” argument:

lower.tail logical; if TRUE (default), probabilities are P[X ??? x], otherwise, P[X > x].

```
#Estimate the probability of fewer than 7 successes using the Poisson Distribution
ppois(q = 6, lambda = 100*.05, lower.tail = TRUE)
```

```
## [1] 0.7621835
```

```
#Calculate the probability of fewer than 7 successes using the Binomial Distribution
pbinom(q = 6, size = 100, prob = 0.05, lower.tail = TRUE)
```

```
## [1] 0.766014
```

The binomial may also be approximated via the normal distribution. Estimate the following binomial probabilities using *dnorm()* or *pnorm()*, this time with probability p = 0.2 and n = 100. Then, calculate the same probabilities using *dbinom()* and *pbinom()*. Use continuity correction. Show the numerical results of your calculations.

iii. The probability of exactly 25 successes.

```
#Estimate the probability of exactly 25 successes with the Normal Distribution
n_1aiii <- 100
p_1aiii <- 0.2
q_1aiii <- 1 - p_1aiii


pnorm(q = 25.5, mean = n_1aiii * p_1aiii, sd = sqrt(n_1aiii * p_1aiii * q_1aiii)) - pnorm(q = 2
4.5, mean = n_1aiii * p_1aiii, sd = sqrt(n_1aiii * p_1aiii * q_1aiii))
```

```
## [1] 0.04572879
```

```
#Calculate the probability of exactly 25 successes with the Binomial Distribution
dbinom(x = 25, size = n_1aiii, prob = p_1aiii)
```

```
## [1] 0.04387783
```

iv. The probability of fewer than 25 successes. Please note the following, taken from the Normal Distribution R Documentation page, regarding the “lower.tail” argument:

lower.tail logical; if TRUE (default), probabilities are P[X ??? x], otherwise, P[X > x].

```
#Estimate the probability of fewer than 25 successes with the Normal Distribution
pnorm(q = 24.5, mean = n_1aiii * p_1aiii, sd = sqrt(n_1aiii * p_1aiii * q_1aiii))
```

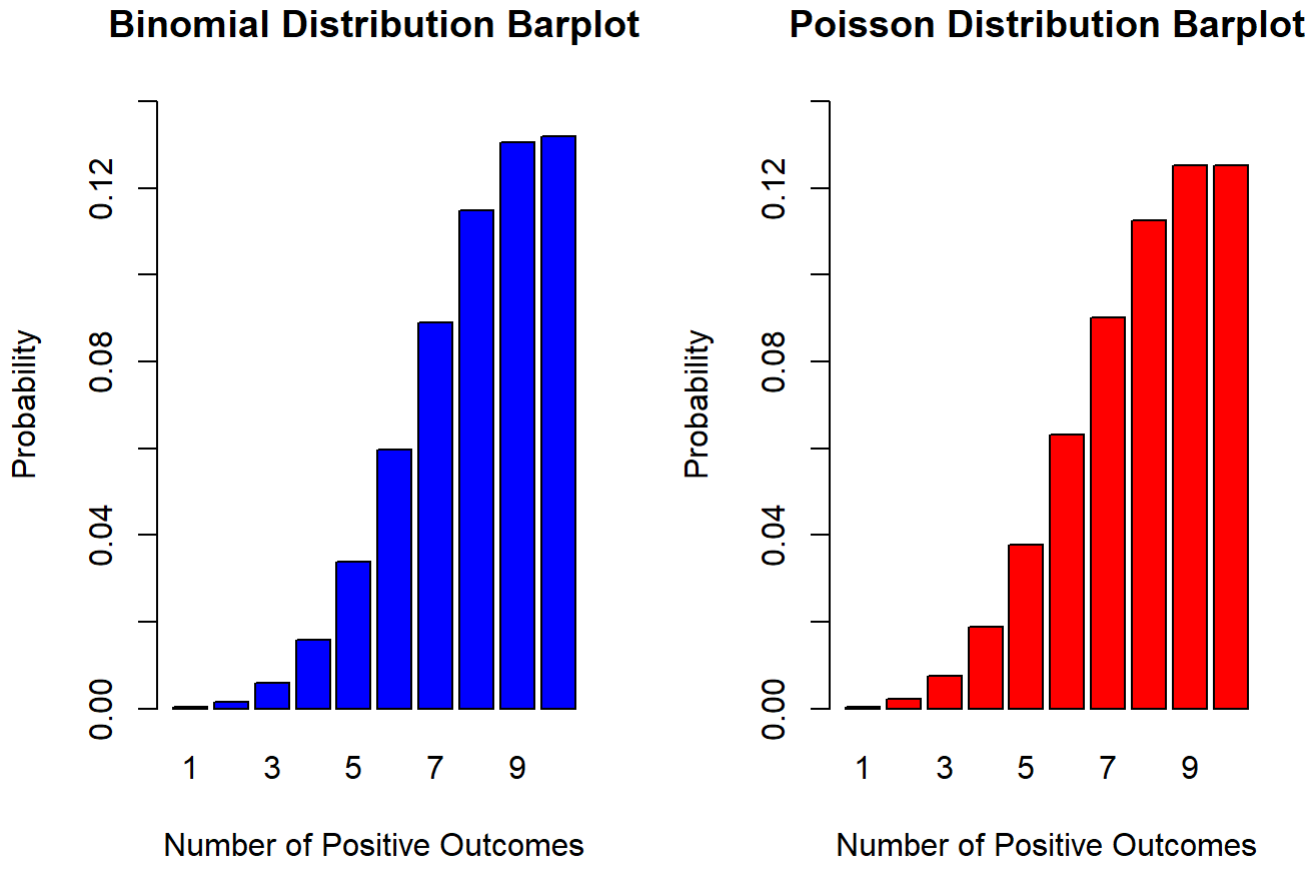
```
## [1] 0.8697055
```

```
#Calculate the probability of fewer than 25 successes with the Binomial Distribution
pbinom(q = 24, size = n_1aiii, prob = p_1aiii, lower.tail = TRUE)
```

```
## [1] 0.8686468
```

(1)(b) (3 points) Generate side-by-side barplots using *par(mfrow = c(1,2))* or *grid.arrange()*. The left barplot will show Poisson probabilties for outcomes ranging from 0 to 10. The right barplot will show binomial probabilities for outcomes ranging from 0 to 10. Use p = 0.1 and n = 100. Title each plot, present in color and assign names to the bar; i.e. x-axis value labels.

```
#Generate side-by-side barplots (using the Poisson and Binomial distributions) showing the proba
bilities associated with x = [1, 10] outcomes for binomial draws when p = 0.1 and n = 100
par(mfrow = c(1,2))


barplot(dbinom(x = 1:10, size = 100, prob = 0.1), names.arg = c(seq(1:10)), col = "blue", axisna
mes = TRUE, xlab = "Number of Positive Outcomes", ylab = "Probability", main = "Binomial Distrib
ution Barplot", ylim = c(0, 0.14))


barplot(dpois(x = 1:10, lambda = 0.1 * 100), names.arg = c(seq(1:10)), col = "red", axisnames =
TRUE, xlab = "Number of Positive Outcomes", ylab = "Probability", main = "Poisson Distribution B
arplot", ylim = c(0, 0.14))
```



(1)(c) (6 points) For this problem, refer to Sections 5.2 of Business Statistics. A discrete random variable has outcomes: 0, 1, 2, 3, 4, 5, 6. The corresponding probabilities in sequence with the outcomes are: 0.215, 0.230, 0.240, 0.182, 0.130, 0.003, 0.001. In other words, the probabilty of obtaining “0” is 0.215.

i. Calculate the expected value and variance for this distribution using the general formula for mean and variance of a discrete distribution. To do this, you will need to use integer values from 0 to 6 as outcomes along with the corresponding probabilities. Round your answer to 1 decimal place.

```
#Calculate the Expected Value of this Distribution

x_1C <- 0:6
Y_1C <- c(0.215, 0.230, 0.240, 0.182, 0.130, 0.003, 0.001)
Mu_1C <- sum(x_1C * Y_1C)
round(Mu_1C, digits = 1)
```

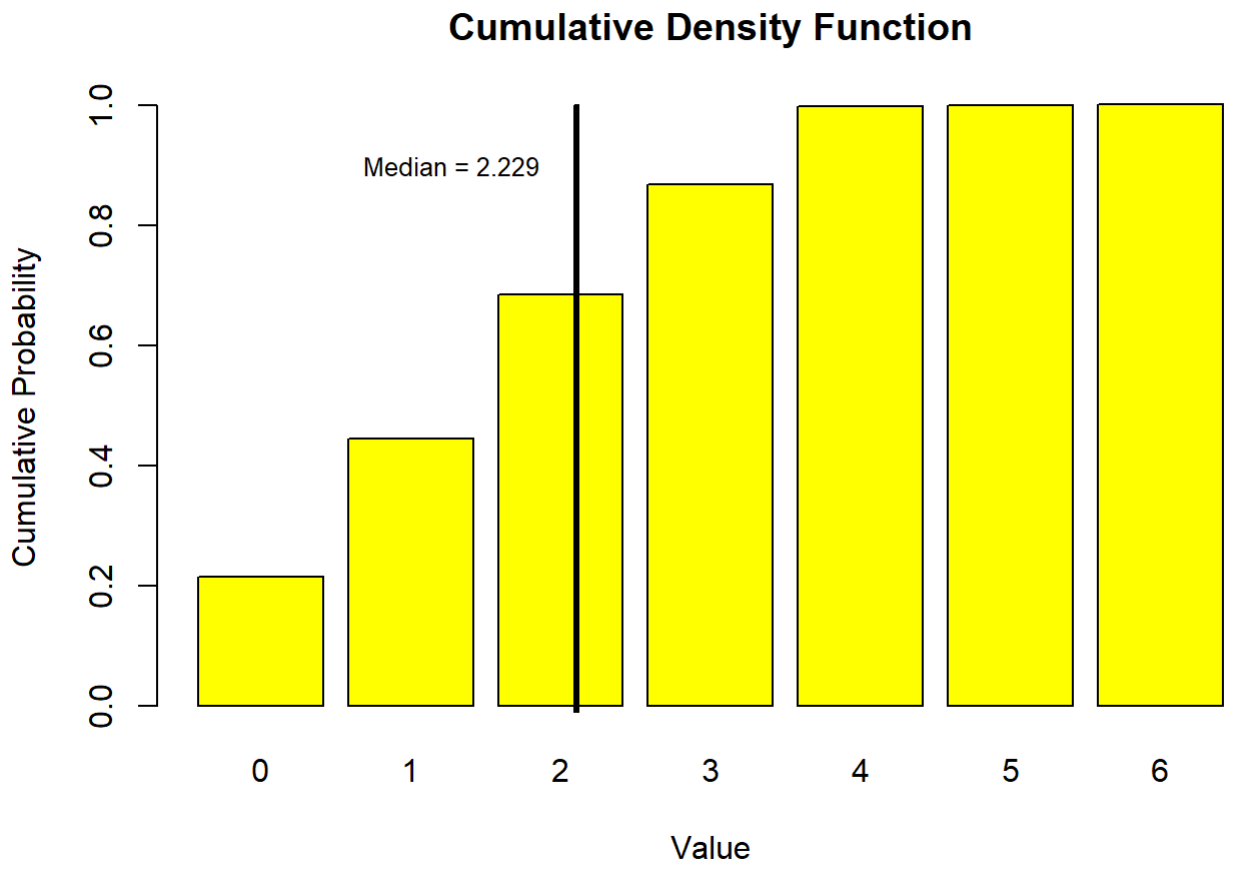
```
## [1] 1.8
```

```
#Calculate the Variance of this Distribution
Deviations_1C <- x_1C - Mu_1C


Var_1C <- sum(Y_1C * (Deviations_1C^2))
round(Var_1C, digits = 1)
```

```
## [1] 1.8
```

ii. Use the *cumsum()* function and plot the cumulative probabilties versus the corresponding outcomes. Detemine the value of the median for this distribution and show on this plot. Note that there are methods for interpolating a median. However, we can identify an appropriate median from our set of our outcomes - 0 through 6 - that satisfies the definition. Creating a stair-step plot of the cumulative probability as a function of the outcomes may be helpful in identifying it.

```
#Plot the cumulative probabilities versus the corresponding outcomes.  Determine the median valu
e for this distribution and show on this plot
barplot(height = cumsum(Y_1C), names.arg = 0:6, col = "yellow", axisnames = TRUE, xlab = "Valu
e", ylab = "Cumulative Probability", main = "Cumulative Density Function")


#Please note that I am using the median definition from Section 3.3 of Business Statistics

median_1Cii <- 2 + ((((1000/2) - (215 + 230)) / 240 ) * 1)

abline(v = median_1Cii+1, col = "black",lwd = 3)
text(x = median_1Cii , y = 0.9,  paste("Median =", round(median_1Cii, digits = 3)), col = "blac
k", cex = 0.8)
```



Section 2: (15 points)

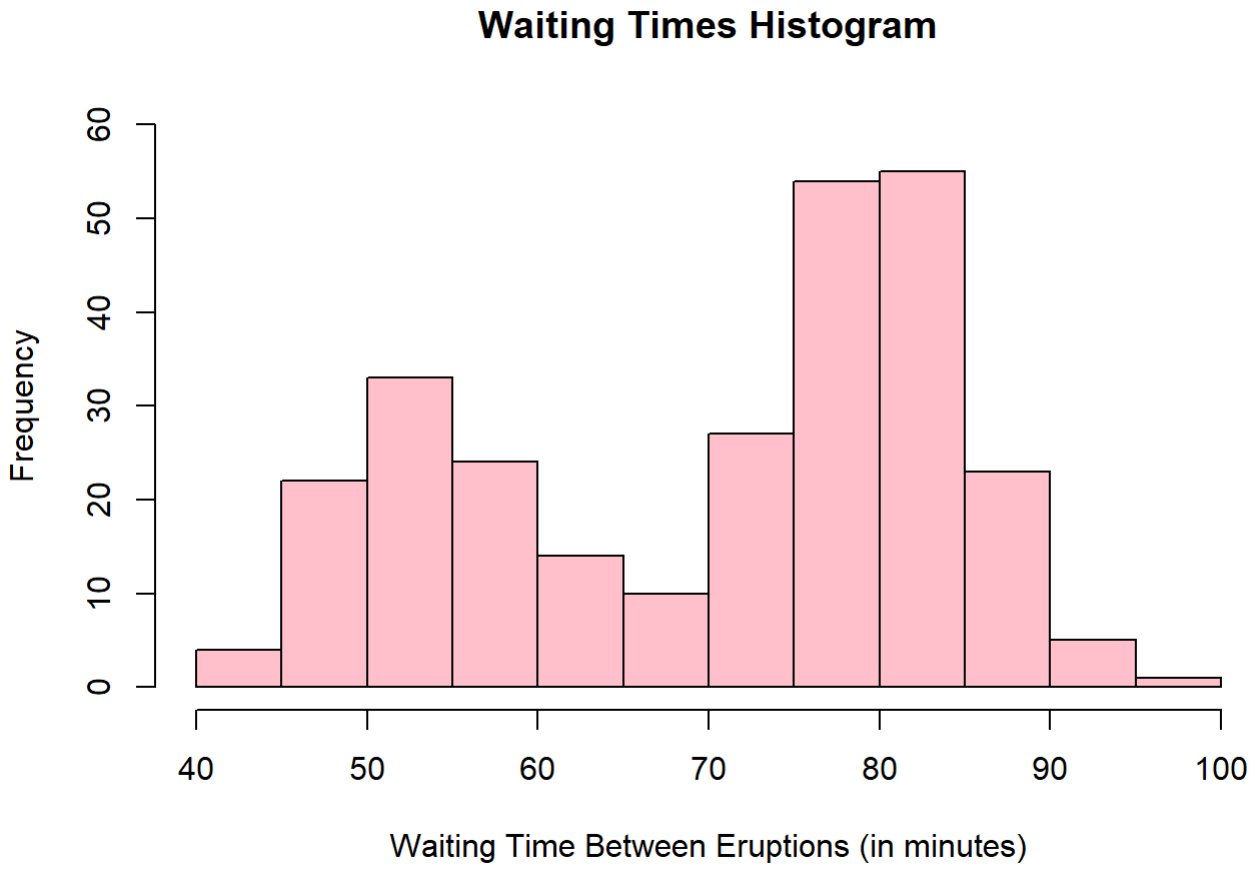
(2) Conditional probabilities appear in many contexts and, in particular, are used by Bayes' Theorem. Correlations are another means for evaluating dependency between variables. The dataset "faithful"" is part of the "datasets" package and may be loaded with the statement *data(faithful)*. It contains 272 observations of 2 variables; waiting time between eruptions (in minutes) and the duration of the eruption (in minutes) for the Old Faithful geyser in Yellowstone National Park.

(2)(a) (6 points) Load the "faithful" dataset and present summary statistics and a histogram of waiting times. Additionally, compute the empirical conditional probability of an eruption less than 3.5 minutes, if the waiting time exceeds 70 minutes.

```
data(faithful, package = "datasets")

#Present summary statistics and a histogram of waiting times
summary(faithful)
```

```
##    eruptions        waiting
##  Min.   :1.600   Min.   :43.0
##  1st Qu.:2.163   1st Qu.:58.0
##  Median :4.000   Median :76.0
##  Mean   :3.488   Mean   :70.9
##  3rd Qu.:4.454   3rd Qu.:82.0
##  Max.   :5.100   Max.   :96.0
```

```
hist(faithful$waiting, xlab = "Waiting Time Between Eruptions (in minutes)", ylab = "Frequency",
main = "Waiting Times Histogram", col = "pink", ylim = c(0, 60))
```

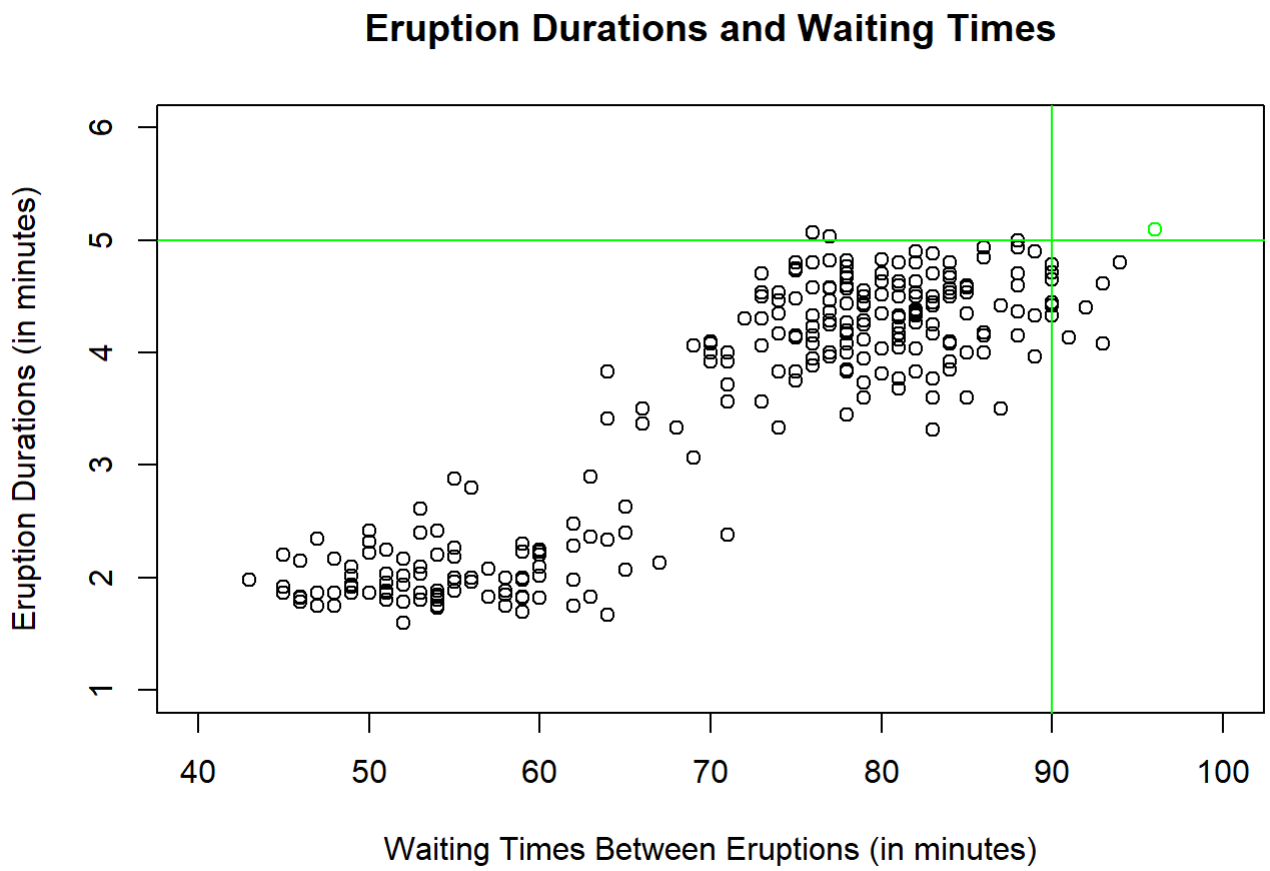



```
#Compute the empirical probability of an eruption less than 3.5 minutes if the waiting time exce
eds 70 minutes

wait_time_over_70 <- faithful[faithful$waiting > 70, ]

wt_over_70_and_eruption_less_than_3point5 <- wait_time_over_70[wait_time_over_70$eruptions < 3.
5,]

nrow(wt_over_70_and_eruption_less_than_3point5) / nrow(wait_time_over_70)
```

```
## [1] 0.02424242
```

i. Identify any observations in “faithful” for which the waiting time exceeds 90 minutes and the eruptions last longer than 5 minutes. List and show any such observations in a distinct color on a scatterplot of all eruption (vertical axis) and waiting times (horizontal axis). Include a horizontal line at eruption = 5.0, and a vertical line at waiting time = 90. Add a title and appropriate text.

```
#Create the plot described in the directions
high_value_eruptions <- faithful[faithful$eruptions > 5 & faithful$waiting > 90, ]

low_value_eruption <- faithful[faithful$eruptions <= 5 | faithful$waiting <= 90, ]

plot(x = low_value_eruption$waiting, y = low_value_eruption$eruptions, xlab = "Waiting Times Bet
ween Eruptions (in minutes)", ylab = "Eruption Durations (in minutes)", main = "Eruption Duratio
ns and Waiting Times", ylim = c(1, 6), xlim = c(40, 100))

abline(h = 5, col = "green")

abline(v = 90, col = "green")

points(x = high_value_eruptions$waiting, y = high_value_eruptions$eruptions, col = "green")
```



ii. What does the plot suggest about the relationship between eruption time and waiting time?

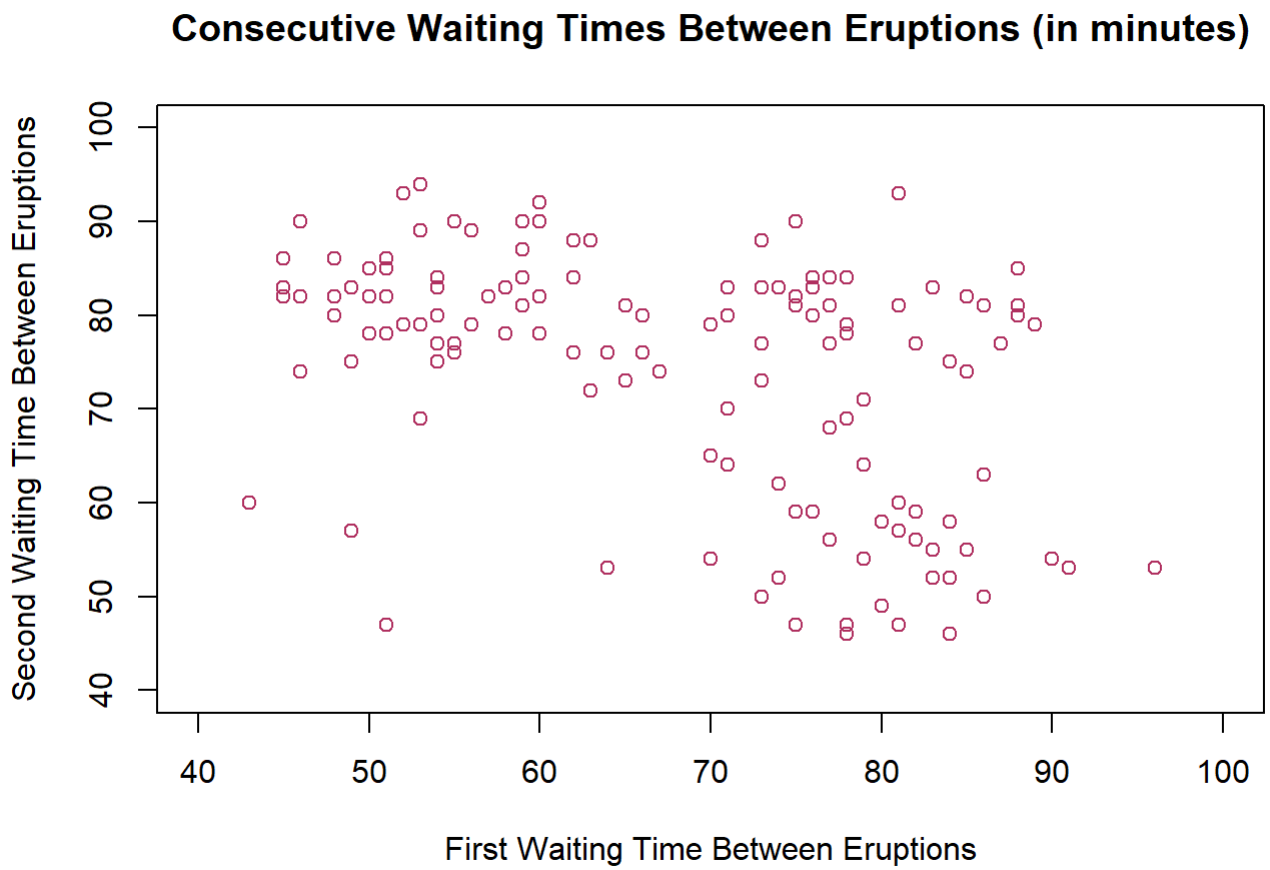
Answer: (Enter your answer here.)

This scatterplot suggests that there exists a positive correlation between the waiting time between eruptions and the duration of the eruption. Furthermore, this scatterplot suggests that the points tend to fit roughly into two clusters. Since the distributions for waiting time and duration seem to cluster in this way, it suggests that measures of center like mean and median may not be the most useful way to represent the average eruption.

(2)(b) (6 points) Past research indicates that the waiting times between consecutive eruptions are not independent. This problem will check to see if there is evidence of this. Form consecutive pairs of waiting times. In other words, pair the first and second waiting times, pair the third and fourth waiting times, and so forth. There are 136 resulting consecutive pairs of waiting times. Form a data frame with the first column containing the first waiting time in a pair and the second column with the second waiting time in a pair. Plot the pairs with the second member of a pair on the vertical axis and the first member on the horizontal axis.

One way to do this is to pass the vector of waiting times - faithful$waiting - to *matrix()*, specifying 2 columns for our matrix, with values organized by row; i.e. byrow = TRUE.

```
#Form consecutive pairs of waiting times
wait_time_pairs <- as.data.frame(matrix(data = faithful$waiting, ncol = 2, byrow = TRUE))



#Plot the pairs of waiting times
plot(x = wait_time_pairs$V1, y = wait_time_pairs$V2, xlab = "First Waiting Time Between Eruption
s", ylab = "Second Waiting Time Between Eruptions", xlim = c(40, 100), ylim = c(40, 100), main =
"Consecutive Waiting Times Between Eruptions (in minutes)", col = "maroon")
```



(2)(c) (3 points) Test the hypothesis of independence with a two-sided test at the 99% confidence level using the Kendall correlation coefficient. The *cor.test()* function can be used to structure this test and specify the appropriate - Kendall's tau - method.

```
#Test the hypothesis of independence
cor.test(x = wait_time_pairs$V1, y = wait_time_pairs$V2, alternative = c("two.sided"), method =
c("kendall"), conf.level = 0.95)
```

```
##
##  Kendall's rank correlation tau
##
## data:  wait_time_pairs$V1 and wait_time_pairs$V2
## z = -4.9482, p-value = 7.489e-07
## alternative hypothesis: true tau is not equal to 0
## sample estimates:
##        tau
## -0.2935579
```

Section 3: (15 points)

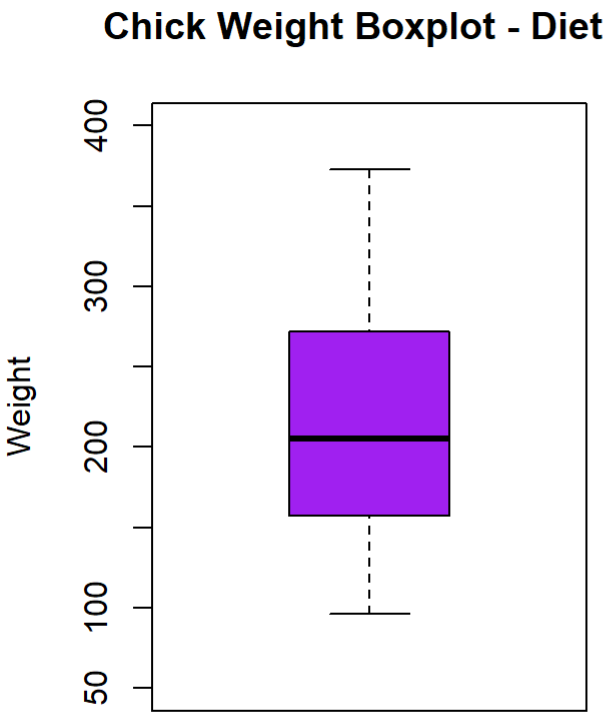
(3) Performing hypothesis tests using random samples is fundamental to statistical inference. The first part of this problem involves comparing two different diets. Using “ChickWeight” data available in the base R, “datasets” package, we will create a subset of the “ChickWeight” data frame. Specifically, we want to create a data frame that includes only those rows where Time == 21 AND Diet == 1 or 3.

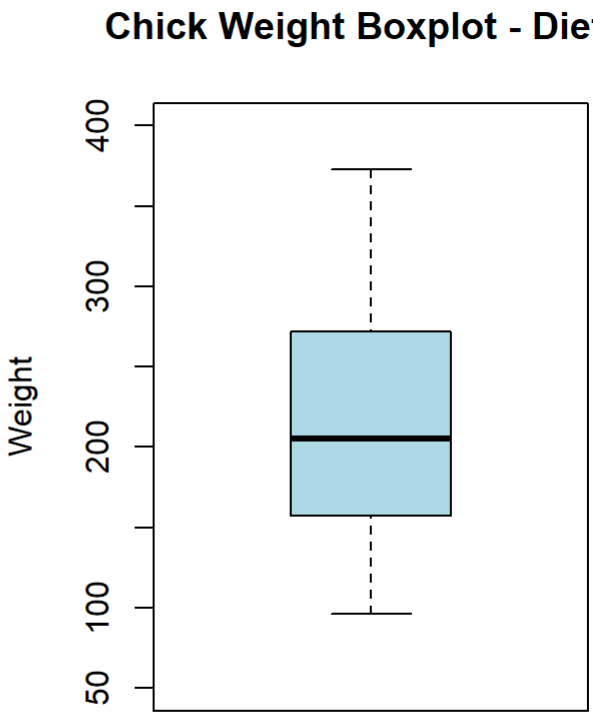
```
# load "ChickWeight" dataset
data(ChickWeight, package = "datasets")

# There are multiple ways to approach the subsetting task. The method you choose is up
# to you.



# The values in your subsetted data frame should match those below:
# > head(df)
#    weight Time Chick Diet
# 12    205   21     1    1
# 24    215   21     2    1
# 36    202   21     3    1
# 48    157   21     4    1
# 60    223   21     5    1
# 72    157   21     6    1



#Create a data frame that includes only those rows where Time = 21 and Diet = 1 or 3
result <- ChickWeight[ChickWeight$Time == 21 & (ChickWeight$Diet == 1 | ChickWeight$Diet == 3),
]
```

The data frame, “result”, has chick weights for two diets, identified as diet “1” and “3”. Use the data frame, “result,” to complete the following item.

(3)(a) (3 points) Display two side-by-side vertical boxplots using par(mfrow = c(1,2)). One boxplot would display Diet “1” and the other Diet “3”.

```
#Display the boxplots described in the directions


par(mfrow = c(1,2))

boxplot(result$weight, subset = result$Diet == "1", ylab = "Weight", main = "Chick Weight Boxplo
t - Diet 1", col = "purple", ylim = c(50, 400))


boxplot(result$weight, subset = result$Diet == "3", ylab = "Weight", main = "Chick Weight Boxplo
t - Diet 3", col = "light blue", ylim = c(50, 400))
```





(3)(b) (3 points) Use the “weight” data for the two diets to test the null hypothesis of equal population mean weights for the two diets. Test at the 95% confidence level with a two-sided t-test. This can be done using *t.test()* in R. Do not assume equal variances. Display the results of *t.test()*.

```
#Conduct the t-test described in the directions

diet_1_chicks <- result[result$Diet == "1", ]

diet_3_chicks <- result[result$Diet == "3", ]

t.test(x = diet_1_chicks$weight, y = diet_3_chicks$weight, alternative = c("two.sided"), paired
= FALSE, var.equal = FALSE, conf.level = 0.95)
```

```
##
##  Welch Two Sample t-test
##
## data:  diet_1_chicks$weight and diet_3_chicks$weight
## t = -3.4293, df = 16.408, p-value = 0.003337
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -149.64644  -35.45356
## sample estimates:
## mean of x mean of y
##    177.75    270.30
```

Working with paired data is another common statistical activity. The “ChickWeight” data will be used to illustrate how the weight gain from day 20 to 21 may be analyzed. This time, we will look only at those individuals on Diet == “3”. You will need to add code below creating two (2) vectors. One (1) vector should include all the Time == 20 weights of those individuals on Diet == “3”; a second should include all the Time == 21 weights of those individuals on Diet == “3”.

```
# There are multiple ways to approach the subsetting task. The method you choose is up
# to you.


# The first six (6) elements of your Time == 20 vector should match those below:
# [1] 235 291 156 327 361 225


#Create a vector including all Chick Weight observations where time = 20 and diet = 3
chickweight_time_20_diet_3 <- ChickWeight[ChickWeight$Time == 20 & ChickWeight$Diet == 3 , ]
vector_cw_time_20_diet_3 <- as.vector(x = chickweight_time_20_diet_3$weight, mode = "any")
vector_cw_time_20_diet_3
```

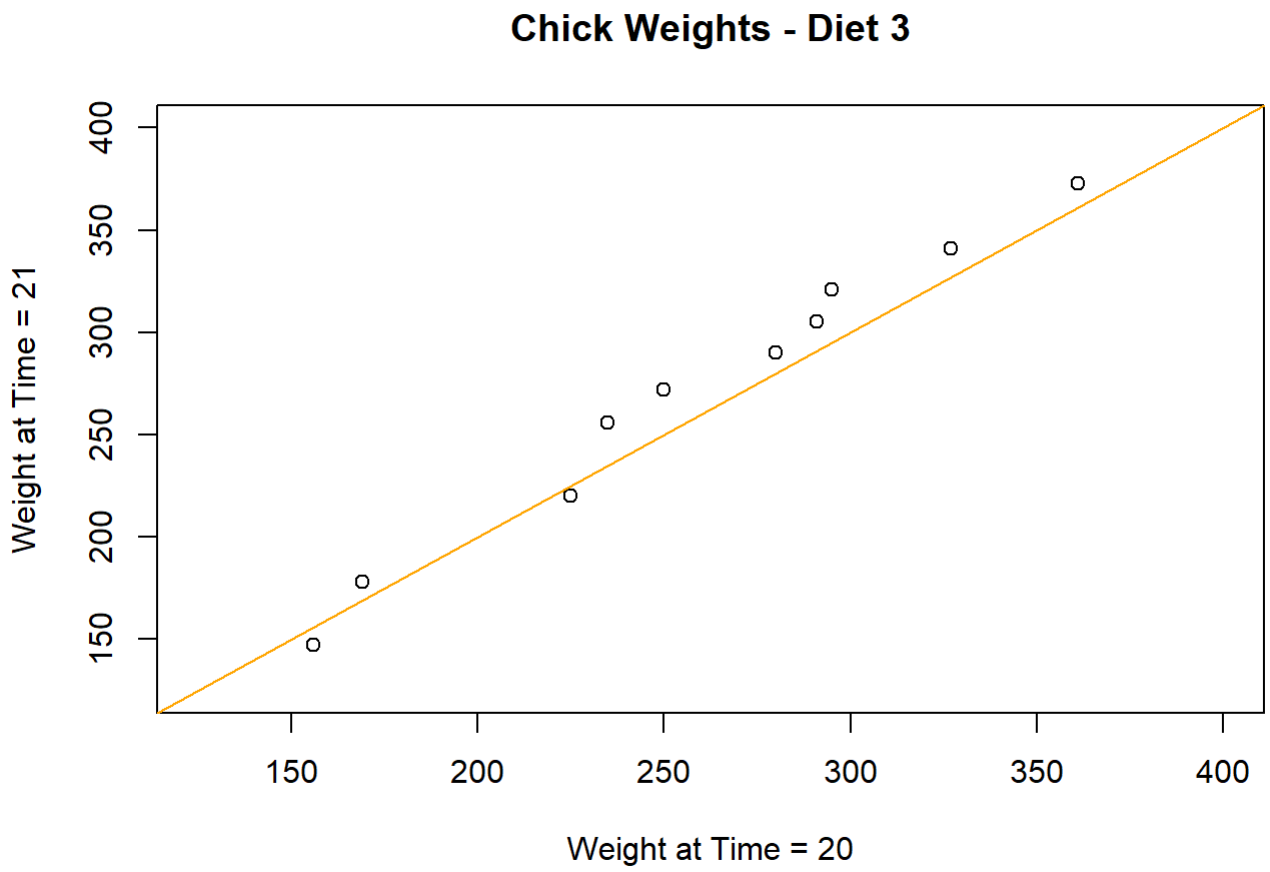
```
##  [1] 235 291 156 327 361 225 169 280 250 295
```

```
#Create a vector including all chick weight observations where time = 21 and diet = 3
chickweight_time_21_diet_3 <- ChickWeight[ChickWeight$Time == 21 & ChickWeight$Diet == 3 , ]
vector_cw_time_21_diet_3 <- as.vector(x = chickweight_time_21_diet_3$weight, mode = "any")
vector_cw_time_21_diet_3
```

```
##  [1] 256 305 147 341 373 220 178 290 272 321
```

(3)(c) (3 points) Present a scatterplot of the Time == 21 weights as a function of the Time == 20 weights. Include a diagonal line with zero intercept and slope equal to one. Title and label the variables in this scatterplot.

```
#Create the scatterplot defined in the directions.  Please note that I'm only including chicks w
ith Diet = 3 per the directions between question 3B and question 3C
plot(x = vector_cw_time_20_diet_3, y = vector_cw_time_21_diet_3, xlab = "Weight at Time = 20", y
lab = "Weight at Time = 21", main = "Chick Weights - Diet 3", xlim = c(125, 400), ylim = c(125,
400))

abline(a = 0, b = 1, lty = 7, col = "orange")
```



(3)(d) (6 points) Calculate and present a one-sided, 95% confidence interval for the average weight gain from day 20 to day 21. Write the code for the paired t-test and for determination of the confidence interval endpoints. **Do not use *t.test()**, although you may check your answers using this function. Present the resulting test statistic value, critical value, p-value and confidence interval.

```
#Conduct the t-test described above and present the test statistic value, critical value, and p-value.  Please note that I'm only using chicks on diet 3 per the directions between question 3B and question 3C.

n_3D <- length(vector_cw_time_20_diet_3)

df_3D <- n_3D - 1

alpha_3D <- .05

t_3D <- qt(p = 1 - alpha_3D, df = df_3D, lower.tail = TRUE)

differences <- vector_cw_time_21_diet_3 - vector_cw_time_20_diet_3

d_bar_3D <- mean(differences)

deviations_3D <- differences - d_bar_3D

squared_deviations_3D <- deviations_3D^2

SSD_3D <- sum(squared_deviations_3D)

SD_3D <- sqrt((SSD_3D / df_3D))

t_statistic_3D <- (d_bar_3D - 0) / (SD_3D / sqrt(n_3D) )

p_value_3D <- pt(q = t_statistic_3D, df = df_3D, lower.tail = FALSE)


print("The test statistic value is")
```

```
## [1] "The test statistic value is"
```

```
t_statistic_3D
```

```
## [1] 3.225267
```

```
print("The critical value is")
```

```
## [1] "The critical value is"
```

```
t_3D
```

```
## [1] 1.833113
```

```
print("The p-value is")
```

```
## [1] "The p-value is"
```

```
p_value_3D
```

```
## [1] 0.00520061
```

```
#Calculate the confidence interval described above

lower_bound_CI_3D <- d_bar_3D - ( t_3D * (SD_3D / sqrt(n_3D) )  )

cat("The one-sided confidence extends from ", lower_bound_CI_3D, " to infinity.")
```

```
## The one-sided confidence extends from  4.920696  to infinity.
```

Section 4: (15 points)

(4) Statistical inference depends on using a sampling distribution for a statistic in order to make confidence statements about unknown population parameters. The Central Limit Theorem is used to justify use of the normal distribution as a sampling distribution for statistical inference. Using Nile River flow data from 1871 to 1970, this problem demonstrates sampling distribution convergence to normality. Use the code below to prepare the data. Refer to this example when completing (4)(c) below.

```
data(Nile, package = "datasets")
```

(4)(a) (3 points) Using Nile River flow data and the “moments” package, calculate skewness and kurtosis. Present a QQ plot and boxplot of the flow data side-by-side using *qqnorm()*, *qqline()* and *boxplot()*; *par(mfrow = c(1, 2))* may be used to locate the plots side-by-side. Add features to these displays as you choose.

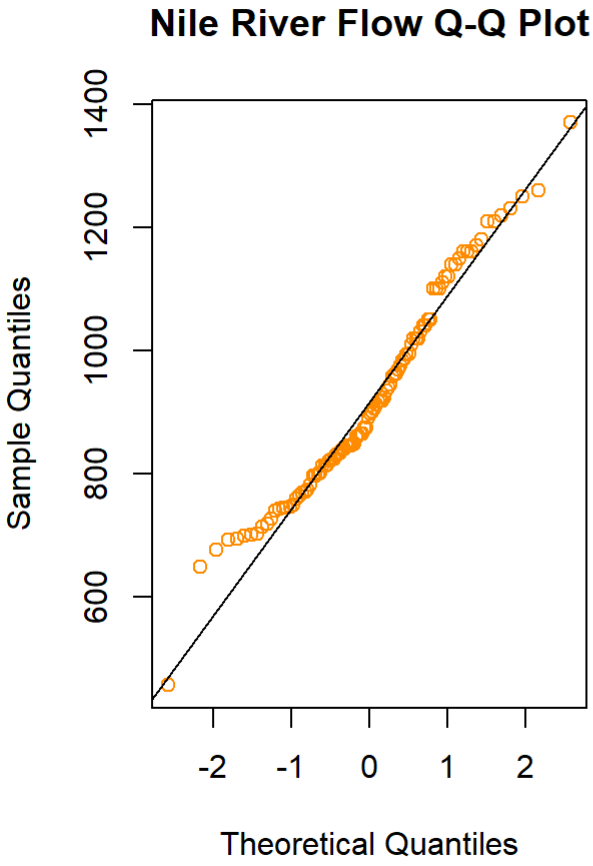
```
#Calculate the skewness and kurtosis of the Nile River data
skewness(Nile)
```

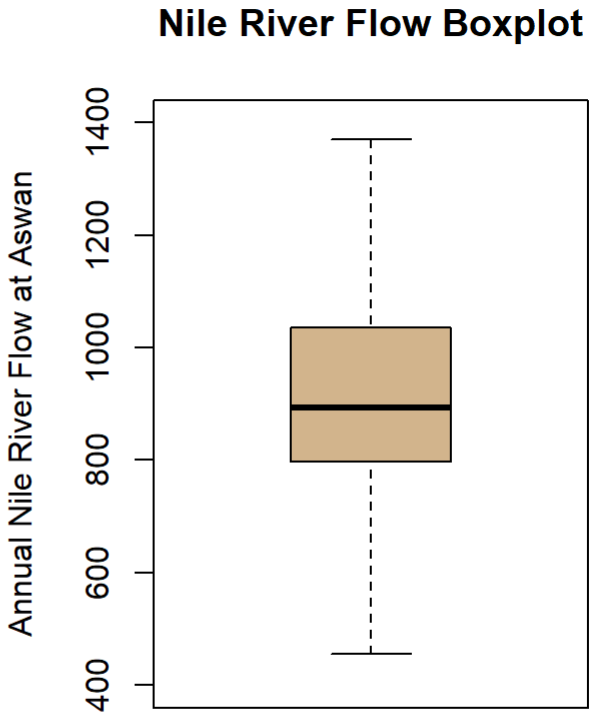
```
## [1] 0.3223697
```

```
kurtosis(Nile)
```

```
## [1] 2.695093
```

```
#Create the QQ Plot and Boxplot requested in the directions
par(mfrow = c(1,2))

qqnorm(y = Nile, xlab = "Theoretical Quantiles", ylab = "Sample Quantiles", col = "dark orange",
main = "Nile River Flow Q-Q Plot")
qqline(y = Nile, distribution = qnorm, probs = c(0.25, 0.75), qtype = 7, col = "black")

boxplot(Nile, main = "Nile River Flow Boxplot", ylab = "Annual Nile River Flow at Aswan", col =
"tan", ylim = c(400, 1400))
```





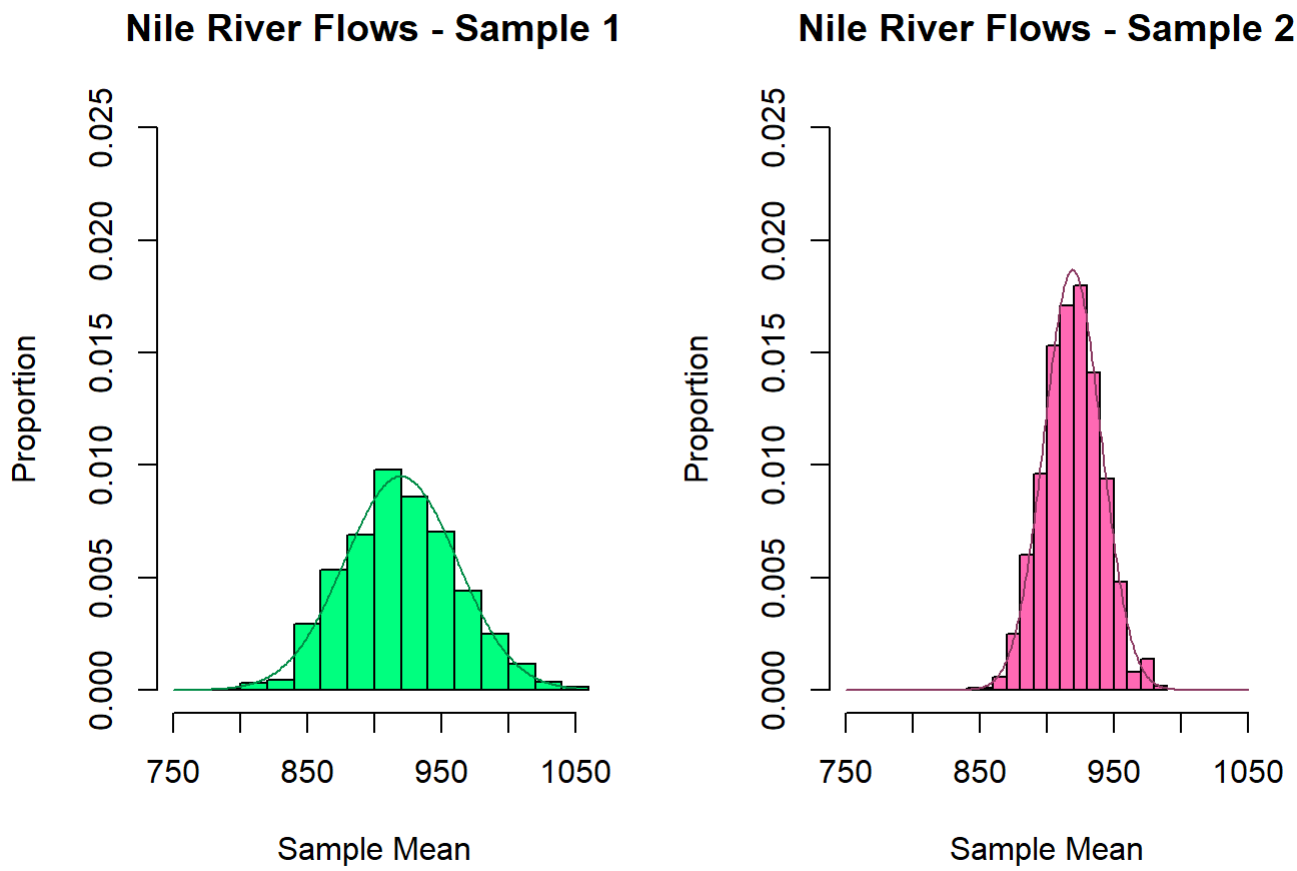
(4)(b) (6 points) Using *set.seed(456)* and the Nile data, generate 1000 random samples of size n = 16, with replacement. For each sample drawn, calculate and store the sample mean. This can be done with a for-loop and use of the *sample()* function. Label the resulting 1000 mean values as “sample1”. **Repeat these steps using *set.seed(789)* - a different “seed” - and samples of size n = 64.** Label these 1000 mean values as “sample2”. Compute and present the means, sample standard deviations and sample variances for “sample1” and “sample2” in a table with the first row for “sample1”, the second row for “sample2” and the columns labled for each statistic.

```
#Generate 1000 random samples of size n = 16 with replacement twice.  For each sample drawn, cal
culate and store the sample mean.  Present the mean, standard deviation, and sample variance for
each sample.

set.seed(456)
sample1 <- data.frame(matrix(ncol = 1, nrow = 0))
colnames(sample1) <- c('Var1')

for(x_4B in 1:1000){
  current_sample <- sample(Nile, size = 16, replace = TRUE)
  current_mean <- mean(current_sample)
  sample1 <- rbind(sample1, data.frame(current_mean))
}

set.seed(789)
sample2 <- data.frame(matrix(ncol = 1, nrow = 0))
colnames(sample2) <- c('Var1')

for(x_4B in 1:1000){
  current_sample <- sample(Nile, size = 64, replace = TRUE)
  current_mean <- mean(current_sample)
  sample2 <- rbind(sample2, data.frame(current_mean))
}

Final_Output_4B <- data.frame(matrix(ncol = 4, nrow = 0))

sample_1_df <- data.frame(matrix(c('Sample 1', mean(sample1$current_mean), sd(sample1$current_me
an), var(sample1$current_mean)), nrow = 1))

sample_2_df <- data.frame(matrix(c('Sample 2', mean(sample2$current_mean), sd(sample2$current_me
an), var(sample2$current_mean)), nrow = 1))

Final_Output_4B <- rbind(Final_Output_4B, sample_1_df, sample_2_df)
colnames(Final_Output_4B) <- c('Sample', 'Sample Mean', 'Sample Standard Deviation', 'Sample Var
iance')

Final_Output_4B
```

```
##     Sample   Sample Mean Sample Standard Deviation  Sample Variance
## 1 Sample 1   919.7673125           41.950209988171 1759.82011805165
## 2 Sample 2 919.066203125          21.3571132334929 456.126285668237
```

(4)(c) (6 points) Present side-by-side histograms of “sample1” and “sample2” with the normal density curve superimposed. To prepare comparable histograms, it will be necessary to use “freq = FALSE” and to maintain the same x-axis with “xlim = c(750, 1050)”, and the same y-axis with “ylim = c(0, 0.025).” **To superimpose separate density functions, you will need to use the mean and standard deviation for each “sample” - each histogram - separately.**

```
# Create histograms of "sample1" and "sample2" with normal density curves superimposed


par(mfrow = c(1,2))

x_values_4C <- seq(750, 1050)
y_values_4C_i <- dnorm(x_values_4C, mean = mean(sample1$current_mean), sd = sd(sample1$current_m
ean))
y_values_4C_ii <- dnorm(x_values_4C, mean = mean(sample2$current_mean), sd = sd(sample2$current_
mean))

hist(sample1$current_mean, xlab = "Sample Mean", ylab = "Proportion", main = "Nile River Flows -
Sample 1", xlim = c(750, 1050), freq = FALSE, ylim = c(0, 0.025), col = "springgreen")
lines(x = x_values_4C, y = y_values_4C_i, col = "springgreen4")

hist(sample2$current_mean, xlab = "Sample Mean", ylab = "Proportion", main = "Nile River Flows -
Sample 2", xlim = c(750, 1050), freq = FALSE, ylim = c(0, 0.025), col = "hotpink")
lines(x = x_values_4C, y = y_values_4C_ii, col = "hotpink4")
```



Section 5: (15 points)

(5) This problem deals with contingency table analysis. This is an example of categorical data analysis (see Kabacoff, pp. 145-151). The “warpbreaks” dataset gives the number of warp breaks per loom, where a loom corresponds to a fixed length of yarn. There are 54 observations on 3

variables: breaks (numeric, the number of breaks), wool (factor, type of wool: A or B), and tension (factor, low L, medium M and high H). These data have been studied and used for example elsewhere. For the purposes of this problem, we will focus on the relationship between breaks and tension using contingency table analysis.

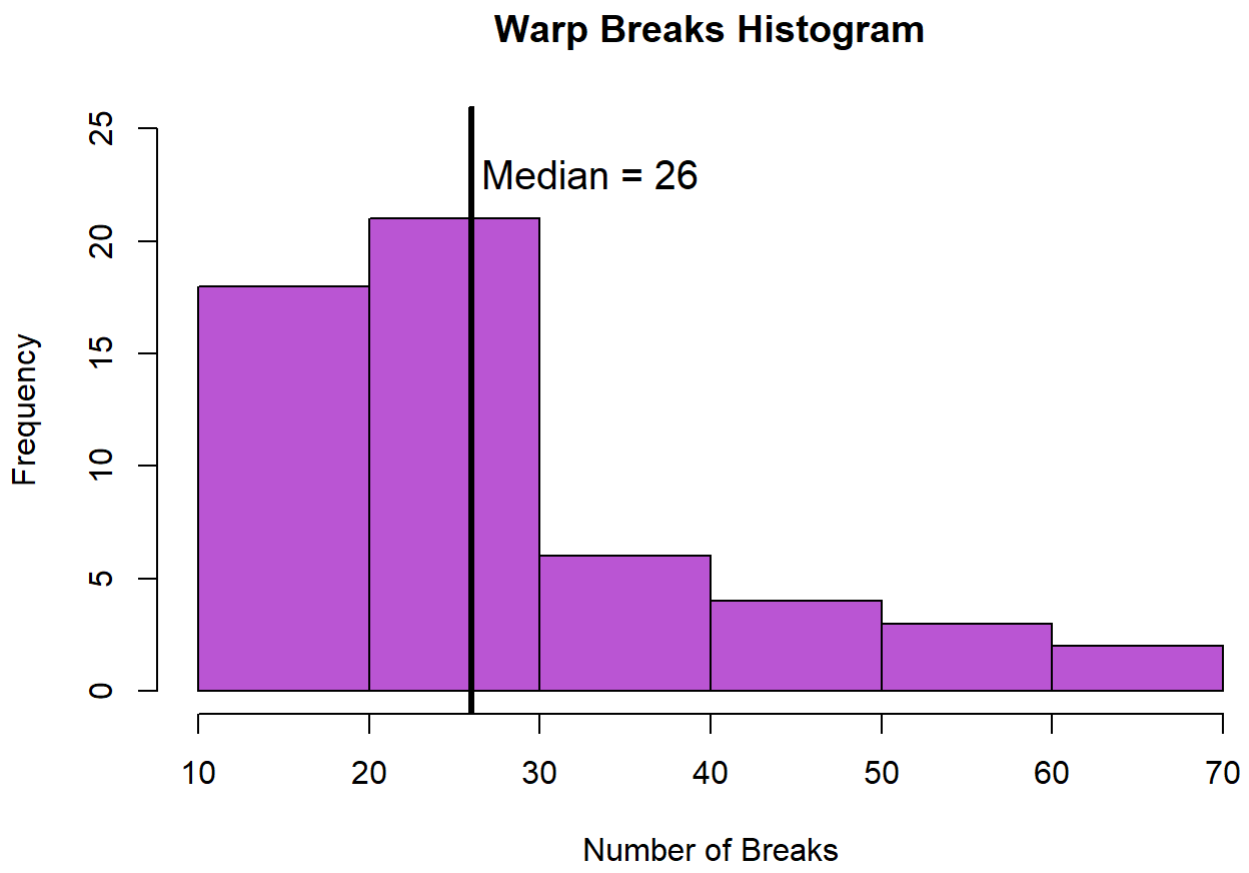
(5)(a)(5 points) warpbreaks is part of the “datasets” package and may be loaded via *data(warpbreaks)*. Load “warpbreaks” and present the structure using *str()*. Calculate the median number of breaks for the entire dataset, disregarding “tension” and “wool”. Define this median value as “median_breaks”. Present a histogram of the number of breaks with the location of the median indicated.

Create a new variable “number” as follows: for each value of “breaks”, classify the number of breaks as either strictly below “median_breaks”, or the alternative. Convert the “above”|“below” classifications to a factor, and combine with the dataset warpbreaks. Present a summary of the augmented dataset using *summary()*. Present a contingency table of the frequency of breaks using the two variables “wool” and “number”. There should be four cells in this table.

```
data(warpbreaks, package = "datasets")

#Present the structure of warpbreaks. Calculate the median number of breaks.  Present a histogra
m of the number of breaks.
str(warpbreaks)
```

```
## 'data.frame':    54 obs. of  3 variables:
##  $ breaks : num  26 30 54 25 70 52 51 26 67 18 ...
##  $ wool   : Factor w/ 2 levels "A","B": 1 1 1 1 1 1 1 1 1 1 ...
##  $ tension: Factor w/ 3 levels "L","M","H": 1 1 1 1 1 1 1 1 1 2 ...
```

```
median_breaks <- median(warpbreaks$breaks)

hist(warpbreaks$breaks, xlab = "Number of Breaks", ylim = c(0,25), ylab = "Frequency", main = "W
arp Breaks Histogram", col = "mediumorchid")
abline(v = median_breaks, col = "black",lwd = 3)
text(x = median_breaks + 7, y = 23,  paste("Median =", median_breaks), col = "black", cex = 1.2
5)
```




```
#Create a new variable called number.  Convert this variable to a factor and combine it with the warpbreaks dataset
warpbreaks$number <- ifelse(warpbreaks$breaks < median_breaks, "below", "above")
warpbreaks <- data.frame(warpbreaks, as.factor(x = warpbreaks$number))
warpbreaks <- warpbreaks[ -c(4) ]
colnames(warpbreaks)[4] ="number"

#Present a summary of the augmented dataset
summary(warpbreaks)
```

```
##      breaks      wool   tension   number
##  Min.   :10.00   A:27   L:18    above:29
##  1st Qu.:18.25   B:27   M:18    below:25
##  Median :26.00          H:18
##  Mean   :28.15
##  3rd Qu.:34.00
##  Max.   :70.00
```

```
#Present a contingency table as described in the directions
df_wool_number <- data.frame(wool = warpbreaks$wool, number = warpbreaks$number)

table(df_wool_number)
```

```
##     number
## wool above below
##    A    16    11
##    B    13    14
```

(5)(b)(3 points) Using the table constructed in (5)(a), test at the 5% level the null hypothesis of independence using the uncorrected *chisq.test()* (Black, Business Statistics, Section 16.2). Show the results of this test and state your conclusions.

```
#Conduct a Chi-Square Test of Independence using the table from 5A

qchisq(p = 0.95, df = 1, lower.tail = TRUE)
```

```
## [1] 3.841459
```

```
chisq.test(x = df_wool_number$wool, y = df_wool_number$number, correct = FALSE)
```

```
##
##  Pearson's Chi-squared test
##
## data:  df_wool_number$wool and df_wool_number$number
## X-squared = 0.67034, df = 1, p-value = 0.4129
```

```
print("Since the observed value of the Chi-Square Statistic, 0.67034, is less than the critical
value for a Chi-Square test with alpha = 0.05, 3.841459, we fail to reject the null hypothesis t
hat the type of wool and category of break counts are independent.")
```

```
## [1] "Since the observed value of the Chi-Square Statistic, 0.67034, is less than the critical
value for a Chi-Square test with alpha = 0.05, 3.841459, we fail to reject the null hypothesis t
hat the type of wool and category of break counts are independent."
```

(5)(c) (3 points) 'Manually' calculate the chi-squared statistic and p-value of the table from (5)(a). The *addmargins()* function can be used to add row and column sums to the table; useful for calculating the expected values for each cell. You should be able to match the chi-squared and p-values from (5)(b). The underlying code for the *chisq.test()* function can be viewed by entering *chisq.test* - without parentheses - in the Console. You are given code below to create the table, add row and column sums and calculate the expected values for the for the first of two (2) rows. You will need to add code to calculate the expected values for the second row and the chi-squared. The *pchisq()* function can be used to return the p-value.

```
#Manually calculate the chi-squared statistic and p-value of the table from 5a
tbl_wool_number <- table(df_wool_number$wool, df_wool_number$number)
tbl_wool_number_margins <- addmargins(tbl_wool_number)
tbl_wool_number_margins
```

```
##
##       above below Sum
##   A      16    11  27
##   B      13    14  27
##   Sum    29    25  54
```

```
e11 <- tbl_wool_number_margins[3, 1] * tbl_wool_number_margins[1, 3] / tbl_wool_number_margins
[3, 3]
e12 <- tbl_wool_number_margins[3, 2] * tbl_wool_number_margins[1, 3] / tbl_wool_number_margins
[3, 3]
e21 <- tbl_wool_number_margins[3, 1] * tbl_wool_number_margins[2, 3] / tbl_wool_number_margins
[3, 3]
e22 <- tbl_wool_number_margins[3, 2] * tbl_wool_number_margins[2, 3] / tbl_wool_number_margins
[3, 3]

chi11 <- ((tbl_wool_number_margins[1, 1] - e11)^2) / e11
chi12 <- ((tbl_wool_number_margins[1, 2] - e12)^2) / e12
chi21 <- ((tbl_wool_number_margins[2, 1] - e21)^2) / e21
chi22 <- ((tbl_wool_number_margins[2, 2] - e22)^2) / e22

chi_square_statistic <- chi11 + chi12 + chi21 + chi22

chi_square_statistic
```

```
## [1] 0.6703448
```

```
pchisq(q = chi_square_statistic, df = 1, lower.tail = FALSE)
```

```
## [1] 0.4129314
```

(5)(d) (4 points) Build a user-defined function, using your code for (5)(c).We want to pass our (5)(a) table to our function and have it return the chi-squared statistic and p-value. You're provided with the 'shell' of a function and will need to add code to calculate the expected values, the chi-squared statistic, the p-value and return (i.e. output) the chi-squared and p-value.

```
chisq_function <- function(x_5D) {
  # Code for calculating the expected values
  function_tbl <- addmargins(x_5D)
  e11_function <- function_tbl[3, 1] * function_tbl[1, 3] / function_tbl[3, 3]
  e12_function <- function_tbl[3, 2] * function_tbl[1, 3] / function_tbl[3, 3]
  e21_function <- function_tbl[3, 1] * function_tbl[2, 3] / function_tbl[3, 3]
  e22_function <- function_tbl[3, 2] * function_tbl[2, 3] / function_tbl[3, 3]

  # Code for calculating the chi-squared
  chi11_function <- ((function_tbl[1, 1] - e11_function)^2) / e11_function
  chi12_function <- ((function_tbl[1, 2] - e12_function)^2) / e12_function
  chi21_function <- ((function_tbl[2, 1] - e21_function)^2) / e21_function
  chi22_function <- ((function_tbl[2, 2] - e22_function)^2) / e22_function

  chi_square_statistic_function <- chi11_function + chi12_function + chi21_function + chi22_func
tion

  # Code for calculating the degrees of freedom and p-value

  df_function <- (nrow(function_tbl) - 2) * (ncol(function_tbl) - 2)
  p_value_function <- pchisq(q = chi_square_statistic_function, df = df_function, lower.tail = F
ALSE)

  # Code to ouput the chi-squared, degrees of freedom and p-value
  function_output <- data.frame("chi-square statistic" = c(chi_square_statistic_function),
                                "degrees of freedom" = c(df_function),
                                "p-value" = c(p_value_function))

  function_output
  }

chisq_function(table(df_wool_number))
```

```
##   chi.square.statistic degrees.of.freedom   p.value
## 1            0.6703448                  1 0.4129314
```

You do not need to do anything with the below. It is provided only for demonstration purposes. In (5)(d), we know the size of the table - 2 x 2 - and write a function to match. Often, though, we'll want to write functions that are flexible in some way.

```
# Below is a function that should return the same values as chisq.test() and your
# function from (5)(d). Here, though, the function loops over the rows and columns
# to calculate the expected values. Ideally, this function would work for any sized
# table.

chisqfun <- function(t) {
   x <- addmargins(t)
   e <- matrix(0, nrow = nrow(t), ncol = ncol(t), byrow = T)
   r <- matrix(0, nrow = nrow(t), ncol = ncol(t), byrow = T)
   for (i in 1:dim(t)[1]) {
       for (j in 1:dim(t)[2]) {
          e[i,j] = x[nrow(x),j] * x[i,ncol(x)]/x[nrow(x), ncol(x)]
         r[i,j] = ((x[i,j] - e[i,j])^2)/e[i,j]
         }
     }
  chi <- sum(r)
  xdf <- (nrow(t) - 1) * (ncol(t) - 1)
  pv <- pchisq(chi, df = xdf, lower.tail = FALSE)
  return(list("chi-squared" = chi, "degrees_of_freedom" = xdf, "p-value" = pv))
  }
```