STAT 302: Homework 5

Jaiden Atterbury

Due on 05-17-23

1. Create and store a vector of 100,000 simulations from a Normal distribution with mean 3 and standard deviation 5 (sometimes written as N(3,5)). In this question, set.seed(10).

Below we will use the rnorm function to create and store a vector of 100,000 simulations from a Normal distribution with mean 3 and standard deviation 5.

```
# Set the seed for reproducibility:
set.seed(10)

# Create the simulation vector:
norm_sim <- rnorm(n=100000, mean=3, sd=5)
```

- a) Print out only the first 5 elements of your vector using head().

Below we will use the head function to print out the first 5 elements of the simulated vector.

```
# Print the first 5 elements of the simulated vector:
head(norm_sim, 5)
```

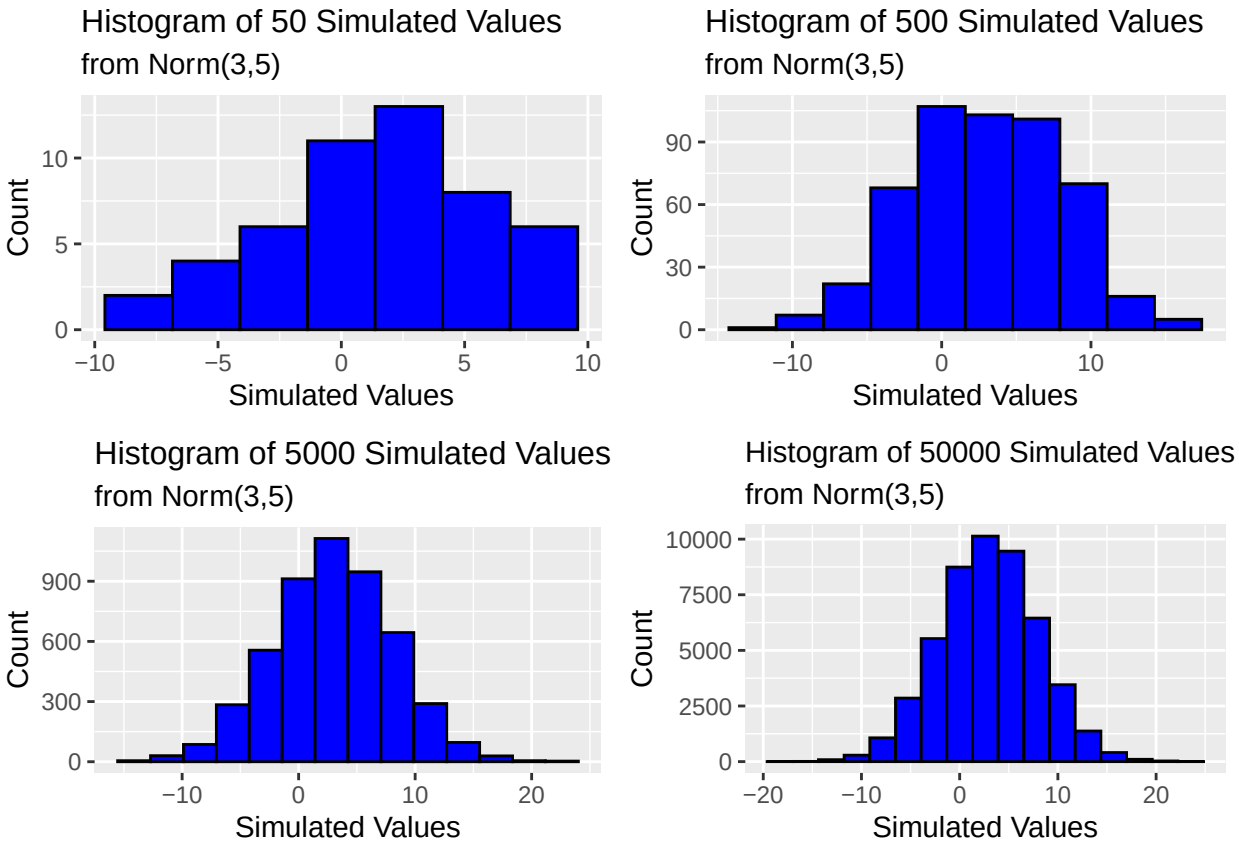
```
## [1]  3.093730855  2.078737290 -3.856652750  0.004161421  4.472725633
```

- b) Generate 4 histograms. The histograms should include the first 50, 500, 5000, and 50000 elements of your simulations, respectively. Be sure to change the title of your histograms to write what they display in plain text. What do you notice about the histograms? Explain why you think this is.

In order to generate the 4 histograms that include the first 50, 500, 5000, and 50000 elements of the normal simulation, we will use Sturge's rule Number of bins $= \log_2(n) + 1$, where n is the sample size

```
# Histogram 1:
n1 <- ggplot() +
  geom_histogram(mapping = aes(x=norm_sim[1:50]), bins=7, color="black",
                 fill="blue") +
  labs(title="Histogram of 50 Simulated Values",
       subtitle ="from Norm(3,5)",
       x="Simulated Values",
       y="Count") +
  theme(plot.title=element_text(size=12))

# Histogram 2:
```

```r
n2 <- ggplot() +
  geom_histogram(mapping = aes(x=norm_sim[1:500]), bins=10, color="black",
                 fill="blue") +
  labs(title="Histogram of 500 Simulated Values ",
       subtitle ="from Norm(3,5)",
       x="Simulated Values",
       y="Count") +
  theme(plot.title=element_text(size=12))

# Histogram 3:
n3 <- ggplot() +
  geom_histogram(mapping = aes(x=norm_sim[1:5000]), bins=14, color="black",
                 fill="blue") +
  labs(title="Histogram of 5000 Simulated Values ",
       subtitle ="from Norm(3,5)",
       x="Simulated Values",
       y="Count") +
  theme(plot.title=element_text(size=12))

# Histogram 4:
n4 <- ggplot() +
  geom_histogram(mapping = aes(x=norm_sim[1:50000]), bins=17, color="black",
                 fill="blue") +
  labs(title="Histogram of 50000 Simulated Values",
       subtitle ="from Norm(3,5)",
       x="Simulated Values",
       y="Count") +
  theme(plot.title=element_text(size=11))

# Create plotting grid:
grid.arrange(n1, n2, n3, n4, nrow=2, ncol=2)
```



As can be seen from the above histograms including the first 50, 500, 5000, and 50000 elements of the normal simulations, is that every histogram seems to resemble the normal distribution centered around 3. However, the key takeaway is that as the number of sample sizes/runs of the simulation increases, the shape of the histogram becomes more normal and centered around the true mean value with smaller and smaller standard deviation. This happens because, in general, as the sample size increase, the histogram converges to the true shape of its simulated distribution.

- c) Compute the mean and standard deviation of your vector from part (a). Report them using in-line R code.

```
# Find the mean of the simulation vector:
xbar <- mean(norm_sim)

# Find the standard deviation of the simulation vector:
sd <- sd(norm_sim)
```

As computed above, the mean of the vector for part (a) is 2.9702081, and the standard deviation of the vector from part (a) is 5.0183993.

- d) In order to standardize vectors, we take each element and subtract the mean and then divide by the standard deviation (computed in part (c)). Create and store a new vector that is the standardization of your simulations from part (a). Create a histogram for all of these standardized simulations (do not forget to change the title again!). What do you notice?

Below, we will create and store a new vector that is the standardization of our simulations from part (a). We will standardize our vector from part (a) by taking each element and subtracting the mean and then dividing by the standard deviation.

```r
# Standardize the simulation vector:
standardized <- (norm_sim - xbar) / sd
```

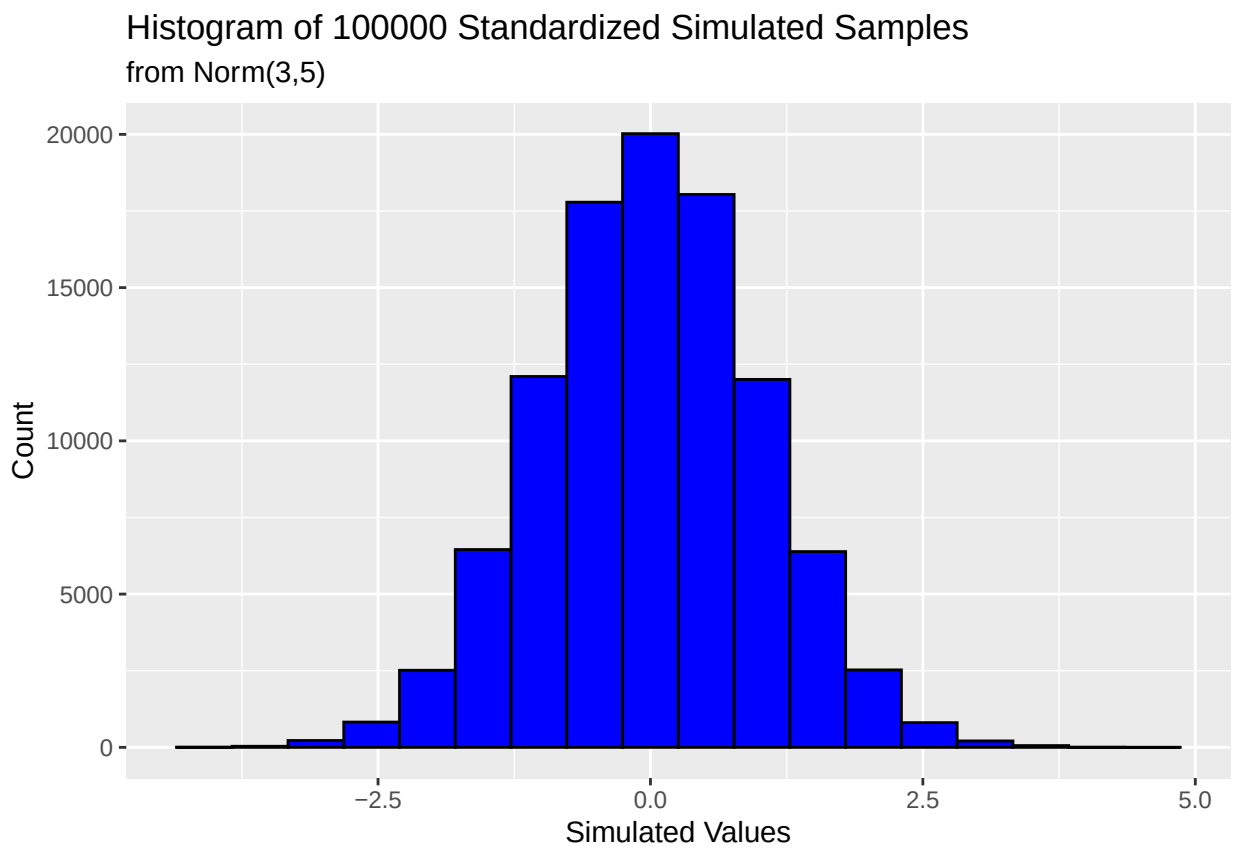
Below we will show the first 5 elements of this standardized vector using the head function.

```r
# Show the first 5 observations of standardized:
head(standardized, 5)
```

```
## [1]  0.02461398 -0.17764047 -1.36036622 -0.59103442  0.29940175
```

Below we will create a histogram for all of these standardized simulations.

```r
# Standardized histogram:
ggplot() +
  geom_histogram(mapping = aes(x=standardized), bins=18, color="black",
                 fill="blue") +
  labs(title="Histogram of 100000 Standardized Simulated Samples",
       subtitle = "from Norm(3,5)",
       x="Simulated Values",
       y="Count")
```



As can be seen from the above histogram, the histogram is symmetric with a center of zero and a small standard deviation. This happens because standardizing observations of a normal random variable turns those observations into $Norm(0, 1)$ observations, hence the histogram of those observations should look like the standard normal curve.

- e) Calculate the percent of simulations from a N(0,1) that you expect to be above 1.644854. How does this compare to the observed proportion of your standardized simulations that are above 1.644854?

For a bit of theoretical background, in order to find the expected number of observations to be above 1.644854 we will need to set up some statistical machinery. Let us denote 100000 random variables X_i, where each X_i denotes if the *ith* sample is above 1.644854 or not. It turns out that these X_i's are Bernoulli random variables such that $X_1, X_2, X_3, \ldots, X_{100000} \sim Binom(1, p)$, where p represents the probability that X_i is above 1.644854. Since these standardized observations are taken from a $Norm(0, 1)$, it turns out that $p = P(Z > 1.644854)$, however we can simply find this using the pnorm function, therefore $p = 0.05$. Furthermore, since the sum of n independent Bernoulli random variables is $Binom(n, p)$, it turns out that $S = X_1 + X_2 + \cdots + X_{100000} \sim Binom(100000, 0.05)$. Lastly, as we know, the expected value of a binomial random variable is np, thus the expected number of simulations from a $N(0, 1)$ that we expect to be above 1.64485 is $100000 * 0.05 = 5000$. For formality, this number is computed in r below, and hence the observed proportion would be 0.05.

```
expected <- 100000 * pnorm(q=1.644854,  lower.tail=FALSE)
actual <- length(standardized[standardized > 1.644854])
```

As computed above, the expected number of simulations from a $N(0, 1)$ that we expect to be above 1.64485 is 5000, with an expected proportion of 0.05. On the other hand, the observed number of the standardized simulations that were above 1.644854 was actually 4926, with an observed proportion of 0.04926 this is due to random sampling variability. The larger the sample we take, the closer and closer the true value would converge to 5000.

- f) How does the percent of simulations from a N(0,1) that you expect to be above 1.644854 (from part (e)) compare to the observed proportion of your first 10 standardized simulations that are above 1.644854? Repeat this for your first 100, 1000, and 10,000 standardized simulations. What do you notice?

```
# Show the proportion above 1.644854 for different number of samples:
for (i in 1:4) {
  print(paste0("Proportion of ", 10^i, " simulations of a N(0, 1)",
               " that are above 1.644854 is: ",
               length(standardized[1:10^i][standardized[1:10^i] > 1.644854]) / 10^i))
}
```

```
## [1] "Proportion of 10 simulations of a N(0, 1) that are above 1.644854 is: 0"
## [1] "Proportion of 100 simulations of a N(0, 1) that are above 1.644854 is: 0.02"
## [1] "Proportion of 1000 simulations of a N(0, 1) that are above 1.644854 is: 0.04"
## [1] "Proportion of 10000 simulations of a N(0, 1) that are above 1.644854 is: 0.0493"
```

As can be seen from the above for loop, as the number of simulations increase, the observed proportion of simulations that are above 1.644854 increase and converges to the expected proportion of simulations that are above 1.644854. In fact, in general, the larger the sample we take, the closer and closer the true value would converge to 5000.

2. A Binomial distribution with n trials and probability of success p, sometimes shorthanded Bin(n,p), represents the number of success out of n independent trials, each with probability of success p.

- a) Initialize two empty matrices. One should have 10 rows and 4 columns, the other should have 10,000 rows and 4 columns. Be sure to give them informative names that follow style guidelines.

Below we will initialize two empty matrices, one with 10 rows and 4 columns, and the other with 10,000 rows and 4 columns.

```
# Initialize matrix 1:
binom_sim_10 <- matrix(data=NA, nrow=10, ncol=4)

# Initialize matrix 2:
binom_sim_10000 <- matrix(data=NA, nrow=10000, ncol=4)
```

- b) Separately fill the first column of each matrix with independent draws from a Binomial distribution with probability 0.2 and n=5. Repeat this process for the second through fourth columns using probabilities of 0.4, 0.6, and 0.8, respectively.

Below we will use a for loop to fill the columns of the matrix with independent draws from a Binomial distribution with probability 0.2 and n=5. We will repeat this process for the second through fourth columns using probabilities of 0.4, 0.6, and 0.8, respectively.

```
# Set the see for reproducibility:
set.seed(10)

# Fill in the two matrices separately:
index = 0
for (prob in seq(from=0.2, to=0.8, by=0.2)) {
  index = index + 1
  binom_sim_10[, index] = rbinom(n=10, size=5, prob=prob)
}

index = 0
for (prob in seq(from=0.2, to=0.8, by=0.2)) {
  index = index + 1
  binom_sim_10000[, index] = rbinom(n=10000, size=5, prob=prob)
}
```

- c) Use four well-labeled histograms to plot the values of each column (if you want to change the number of bars you can use the breaks parameter). Discuss what you see.

Below we will plot the first 4 histogram corresponding to the first matrix; the one with only 10 simulations of the binomial.

```
# Histogram 1:
b1 <- ggplot() +
  geom_histogram(mapping = aes(x=binom_sim_10[,1]), binwidth=1, color="purple",
                 fill="gold") +
  xlim(-1, 6) +
  labs(title="Histogram of 10 Simulated Values",
       subtitle="from Binom(5, 0.2)",
       x="Simulated Values",
       y="Count")

# Histogram 2:
b2 <- ggplot() +
  geom_histogram(mapping = aes(x=binom_sim_10[,2]), binwidth=1, color="purple",
```

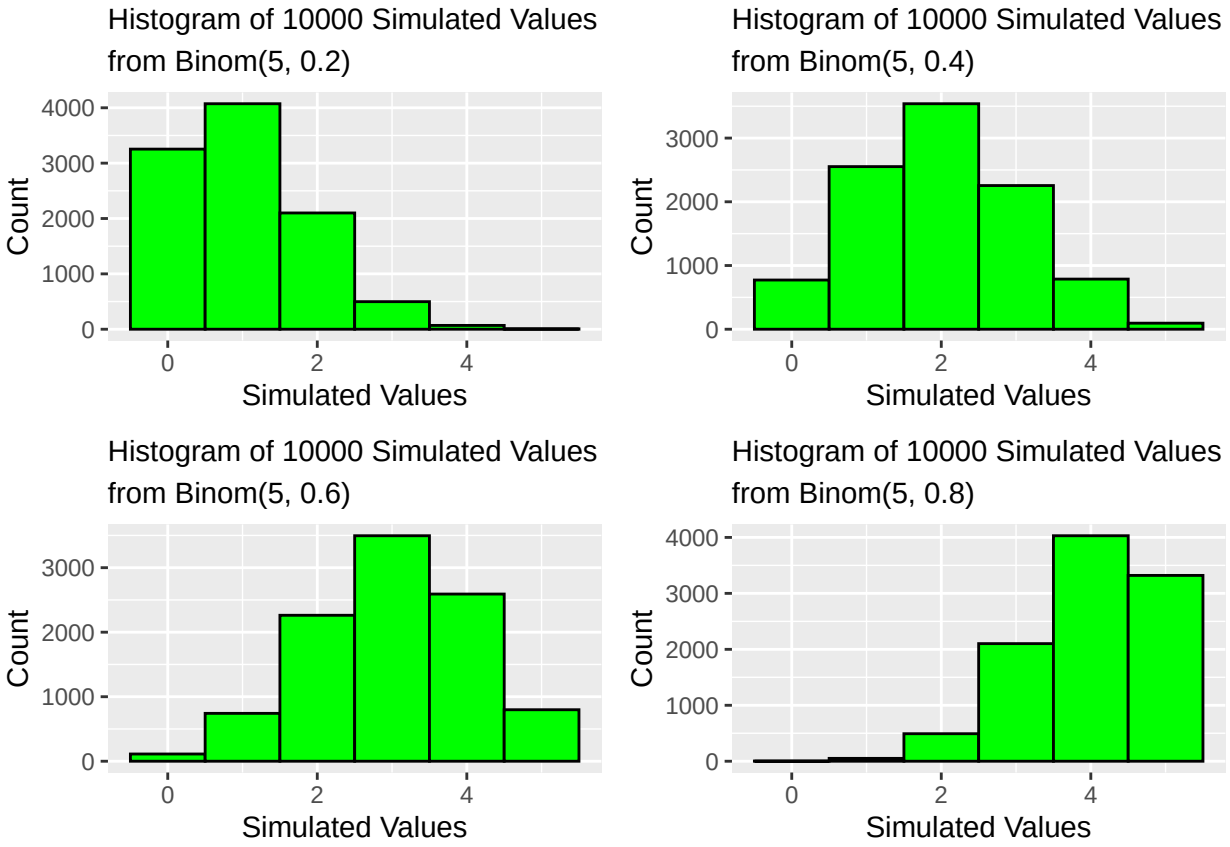
```
                 fill="gold") +
  xlim(-1, 6) +
  labs(title="Histogram of 10 Simulated Values",
       subtitle="from Binom(5, 0.4)",
       x="Simulated Values",
       y="Count")

# Histogram 3:
b3 <- ggplot() +
  geom_histogram(mapping = aes(x=binom_sim_10[,3]), binwidth=1, color="purple",
                 fill="gold") +
  xlim(-1, 6) +
  labs(title="Histogram of 10 Simulated Values",
       subtitle="from Binom(5, 0.6)",
       x="Simulated Values",
       y="Count")

# Histogram 4:
b4 <- ggplot() +
  geom_histogram(mapping = aes(x=binom_sim_10[,4]), binwidth=1, color="purple",
                 fill="gold") +
  xlim(-1, 6) +
  labs(title="Histogram of 10 Simulated Values",
       subtitle="from Binom(5, 0.8)",
       x="Simulated Values",
       y="Count")

grid.arrange(b1, b2, b3, b4, nrow=2, ncol=2)
```

As can be seen from the 4 above histograms, as the probability goes up, the number of successes increase, and thus the most amount of density shifts from left-to-right. Furthermore, due to such a small amount of replications, the shape of the histogram doesn't match what we'd expect from a Binomial PMF of the same parameters, this is because it isn't feasible to have the appropriate number of observations in each bin in such a small sample.

Below we will plot the next 4 histogram corresponding to the second matrix; the one with 10000 simulations of the binomial.

```
# Histogram 1:
p1 <- ggplot() +
  geom_histogram(mapping = aes(x=binom_sim_10000[,1]), binwidth=1, color="black",
                 fill="green") +
  labs(title="Histogram of 10000 Simulated Values",
       subtitle="from Binom(5, 0.2)",
       x="Simulated Values",
       y="Count") +
  theme(plot.title=element_text(size=11))

# Histogram 2:
p2 <- ggplot() +
  geom_histogram(mapping = aes(x=binom_sim_10000[,2]), binwidth=1, color="black",
                 fill="green") +
  labs(title="Histogram of 10000 Simulated Values",
       subtitle="from Binom(5, 0.4)",
       x="Simulated Values",
       y="Count") +
```

```
  theme(plot.title=element_text(size=11))

# Histogram 3:
p3 <- ggplot() +
  geom_histogram(mapping = aes(x=binom_sim_10000[,3]), binwidth=1, color="black",
                 fill="green") +
  labs(title="Histogram of 10000 Simulated Values",
       subtitle="from Binom(5, 0.6)",
       x="Simulated Values",
       y="Count") +
  theme(plot.title=element_text(size=11))

# Histogram 4:
p4 <- ggplot() +
  geom_histogram(mapping = aes(x=binom_sim_10000[,4]), binwidth=1, color="black",
                 fill="green") +
  labs(title="Histogram of 10000 Simulated Values",
       subtitle="from Binom(5, 0.8)",
       x="Simulated Values",
       y="Count") +
  theme(plot.title=element_text(size=11))

grid.arrange(p1, p2, p3, p4, nrow=2, ncol=2)
```



As can be seen from the 4 above histograms, as the probability goes up, the number of successes increase, and thus the most amount of density shifts from left-to-right. Furthermore, unlike the first 4 histograms,

due to a large number of simulations, the shape of the histograms do in fact match what we would expect from the corresponding Binomial PMF with the same parameters. Again, this happens because, in general, as the sample size increase, the histogram converges to the true shape of its simulated distribution.

- d) Calculate the column means of both matrices and present these results in a single table. The rows and columns of your tables should be easy to read and interpret. I suggest using kableExtra package.

Below we will calculate the column means of both matrices and present these results in a single table. The rows correspond to the different simulation sizes, the columns correspond to the different distribution we drew simulations from, and lastly the value in each cell corresponds to the mean of each column of the corresponding matrix.

```
# Find the column means of the first matrix:
binom_col_means_10 <- t(as.matrix(colMeans(binom_sim_10)))

# Find the column means of the second matrix:
binom_col_means_10000 <-t(as.matrix(colMeans(binom_sim_10000)))

# Combine the matrices for presentation:
mean_table <- rbind(binom_col_means_10, binom_col_means_10000)

# Create the row and column names for presentation
rownames(mean_table) <- c("n=10", "n=10000")
colnames(mean_table) <- c("Binom(5, 0.2)", "Binom(5, 0.4)",
                          "Binom(5, 0.6)", "Binom(5, 0.8)")

# Create the table:
kable(mean_table)
```

	Binom(5, 0.2)	Binom(5, 0.4)	Binom(5, 0.6)	Binom(5, 0.8)
n=10	0.5000	1.7000	2.6000	3.7000
n=10000	1.0078	2.0018	3.0105	4.0066

- e) What is the expected column mean for each column? Which matrix has observed column means that are closer to this expectation? Why do you think that is?

Since each column of the matrix has n realizations of a $Binom(n,p)$, in other words $X_1, X_2, \ldots, X_n \sim Binom(n,p)$, it follows that $E[X_1 + X_2 + \cdots + X_n] = \frac{nE[X_i]}{n} = E[X_i] = np$. Hence the expected column mean is simply the expected value of the corresponding Binomial distribution that we drew simulations from.

Below are the expected column means for each column of both matrices in which we simulated from a Binomial distribution of size 5 with different probabilities of success.

```
# Print out the expected column means:
index = 0
for (p in seq(from=0.2, to=0.8, by=0.2)) {
  index = index + 1
  print(paste0("The expected column mean for column ", index, " is ", 5 * p))
}
```

```
## [1] "The expected column mean for column 1 is 1"
## [1] "The expected column mean for column 2 is 2"
## [1] "The expected column mean for column 3 is 3"
## [1] "The expected column mean for column 4 is 4"
```

As can be seen from the expected column mean values and our actual simulated column mean values, as the number of simulations increases, in general, the mean of the column gets closer and closer to the true value of the mean. This happens due to the Central Limit Theorem which states that as the sample size increases, the distribution of the sample mean becomes more and more normal and centered around its mean even if the original sample values weren't taken from a normal distribution. Furthermore, it follows that $\bar{X} \sim Norm(\mu, \frac{\sigma}{\sqrt{n}})$. Hence, since the sample mean is an unbiased estimator of the population mean, on average, we expect to see sample means close to this value, and as the sample size increases, the variance of this value decreases and thus we see values closer and closer to the true population mean.

- f) What is the variance for the values in each column? Which matrix has observed column sample variances that are closer to these values? Why do you think that is?

Below we will calculate the column variances of both matrices and present these results in a single table. The rows correspond to the different simulation sizes, the columns correspond to the different distribution we drew simulations from, and lastly the value in each cell corresponds to the variance of each column of the corresponding matrix.

```
# Find the column means of the first matrix:
binom_col_var_10 <- t(as.matrix(colVars(binom_sim_10)))

# Find the column means of the second matrix:
binom_col_var_10000 <-t(as.matrix(colVars(binom_sim_10000)))

# Combine the matrices for presentation:
var_table <- rbind(binom_col_var_10, binom_col_var_10000)

# Create the row and column names for presentation
rownames(var_table) <- c("n=10", "n=10000")
colnames(var_table) <- c("Binom(5, 0.2)", "Binom(5, 0.4)",
                         "Binom(5, 0.6)", "Binom(5, 0.8)")

# Create the table:
kable(var_table)
```

	Binom(5, 0.2)	Binom(5, 0.4)	Binom(5, 0.6)	Binom(5, 0.8)
n=10	0.2777778	0.6777778	0.4888889	0.9000000
n=10000	0.8078199	1.1899158	1.2021100	0.7914356

Since each column of the matrix has n realizations of a $Binom(n, p)$, in other words $X_1, X_2, \ldots, X_n \sim Binom(n, p)$, it follows that $Var[X_1 + X_2 + \cdots + X_n] = \frac{nVar[X_i]}{n} = Var[X_i] = np(1-p)$. It is important to note that we could only split apart the variances due to the independence of the sample. Hence the expected column variance is simply the variance of the corresponding Binomial distribution that we drew simulations from.

Below are the expected column variances for each column of both matrices in which we simulated from a Binomial distribution of size 5 with different probabilities of success.

```
# Print out the expected column variances:
index = 0
for (p in seq(from=0.2, to=0.8, by=0.2)) {
  index = index + 1
  print(paste0("The expected column variance for column ", index, " is ",
               5 * p * (1 - p)))
}
```

```
## [1] "The expected column variance for column 1 is 0.8"
## [1] "The expected column variance for column 2 is 1.2"
## [1] "The expected column variance for column 3 is 1.2"
## [1] "The expected column variance for column 4 is 0.8"
```

As can be seen from the expected column variance values and our actual simulated column variance values, as the number of simulations increases, in general, the variance of the column gets closer and closer to the true value of the variance. The reason why this happens is that the sample variance is an unbiased estimator of the population variance. Furthermore, the sampling distribution of the sample variance is χ^2_{n-1}. It follows that the variance of the sample variance is $\frac{2\sigma^4}{n-1}$ thus as n increases, the variance of the sample variance decreases. Hence the larger samples will have values closer to the true variance on average, while the smaller samples will suffer from more variable observations (hence why they are far off from the true variance).

3. Using the lake.csv dataset on Canvas. Is there a statistical significant relationship between pH and Mercury? Answer the following questions below:

- a) Perform a simple linear regression. Report the intercept and slope with its corresponding 95% confidence interval and interpret your results.

Since decreased pH levels have been shown in numerous studies to decrease the loss of mercury from lake water and increase mercury binding to particulates in the water itself, we will use pH as the independent/predictor variable, and mercury as the dependent/response variable. In testing the below model assumptions, we will run all tests/assumptions at the 5% level.

Before we run the model we will first see if there is a correlation between the independent and dependent variables, as well as see if the dependent variable is approximately normally distributed.

Correlation between the response and the independent variables:

```
# Test for correlation between the dependent and independent variable:
cor.test(Lake$pH, Lake$Mercury, method="pearson", conf.level = 0.95)
```

```
##
##  Pearson's product-moment correlation
##
## data:  Lake$pH and Lake$Mercury
## t = -5.3626, df = 49, p-value = 2.207e-06
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.7568749 -0.3995094
## sample estimates:
##        cor
## -0.6081381
```

In the above Pearson correlation test, H_0 is that the true correlation between pH and Mercury is zero, while H_A is that the true correlation between pH and Mercury is not zero. The observed t statistic from the sample was -5.3626 on 49 degrees of freedom, where degrees of freedom is the number of data points minus 2, and lastly the p-value was 2.207e-06. Due to the extremely low p-value we reject the null hypothesis that the true correlation between pH and Mercury is zero, and conclude that there is a statistically significant correlation between pH and Mercury. The correlation coefficient between pH and Mercury was -0.6081381, showing a moderately high negative linear association between the pH levels and Mercury levels. Namely, as pH levels increase, Mercury levels decrease, and vice verse. The coefficient of determination is 0.36, which means that roughly 36% of the variability in Mercury can be explained by the pH levels in the water.

Normality of the response:

```r
# Test the normality of the dependent variable:
shapiro.test(Lake$Mercury)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  Lake$Mercury
## W = 0.94551, p-value = 0.02055
```

As can be seen from the above Shapiro-Wilk test, since the p-value is 0.02055 which is less than 0.05, we reject the null hypothesis and we assume that the dependent variable is not approximately normally distributed. Since the dependent variable isn't approximately normally distributed, we will take a square root transformation.

Normality of the transformed response:

```r
# Test the normality of the dependent variable:
shapiro.test(sqrt(Lake$Mercury))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  sqrt(Lake$Mercury)
## W = 0.98023, p-value = 0.5485
```

As can be seen from the above Shapiro-Wilk test, since the p-value is 0.5485 which is greater than 0.05, we fail to reject the null hypothesis and we assume that the square root of dependent variable is approximately normally distributed. We can now move on with our model fitting.

Creating the model/summary of the model:

```r
# Create the simple linear model:
model_1 <- lm(sqrt(Mercury) ~ pH, data=Lake)

# Get a summary of the model:
summary(model_1)
```

```
##
## Call:
## lm(formula = sqrt(Mercury) ~ pH, data = Lake)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.3308 -0.1223  0.0045  0.1049  0.3827
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.43435    0.13432  10.678 2.18e-14 ***
## pH          -0.11607    0.01998  -5.811 4.57e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
```

```
## Residual standard error: 0.1823 on 49 degrees of freedom
## Multiple R-squared:  0.408,  Adjusted R-squared:  0.3959
## F-statistic: 33.77 on 1 and 49 DF,  p-value: 4.573e-07
```

As computed above, the overall model is statistically significant with an F-statistic of 33.77 on 1 and 49 degrees of freedom with a p-value of 4.573e-07. Furthermore, the `pH` coefficient estimate is -0.11607 and is a highly significant predictor of `Mercury` with a p-value of 22.18e-14. Furthermore, the intercept estimate is 1.43435 and is also highly significant with a p-value of 4.57e-07.

For completeness, here are the 95% confidence intervals for the above estimates:

95% confidence intervals for the slope and intercept:

```
# Find the corresponding 95% confidence intervals of the parameters:
confint(model_1)
```

```
##                   2.5 %      97.5 %
## (Intercept)  1.1644108  1.70427995
## pH          -0.1562147 -0.07593055
```

As mentioned above the `pH` coefficient is significant and this is shown by it's interval not containing zero. Also, as mentioned above the intercept estimate is also significant and this is shown by it's interval not containing zero.

In total, the regression equation is $\widehat{\sqrt{\text{Mercury}}} = -0.11607 \cdot \text{pH} + 1.43435$.

Estimate interpretation for the model:
As computed above, the regression slope for pH was -0.11607 with a 95% confidence interval of [-0.1562147, -0.07593055]. This means, on average, for every 1 unit increase in the pH level of the lake water the square root of the mercury level of that lake is estimated to decrease by about 0.11607 units over the sampled range of mercury levels.

As computed above, the regression y-intercept was 1.43435 with a 95% confidence interval of [1.1644108, 1.70427995]. This means that the estimated mean square root mercury level in the lake water is equal to about 1.43435 when the pH level of the lake water is zero.

Overall without running the model assumptions we can't justify the model fit, however due to the significant parameters and low residual standard error we can say that if the model assumptions hold, this linear model fits the data well with pH having decent predicting power for mercury.

Checking for Linearity:

```
# Run a Rainbow test to check the linearity of the variables:
raintest(model_1)
```

```
##
##  Rainbow test
##
## data:  model_1
## Rain = 0.9176, df1 = 26, df2 = 23, p-value = 0.5866
```

As can be seen from the above Rainbow test, since the p-value is 0.5866 which is greater than 0.05, thus we fail to reject the null hypothesis and we assume that the relationship between the independent and the square root of the dependent variables is linear.

Checking for Normality of the Residuals:

```r
# Calculate the residuals:
resid_1 <- residuals(model_1)

# Run a Shapiro test to check the normality of the residuals:
shapiro.test(resid_1)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  resid_1
## W = 0.97705, p-value = 0.4221
```

As can be seen from the above Shapiro-Wilk test, since the p-value is 0.4221 which is greater than 0.05, we fail to reject the null hypothesis and we assume that the residuals are normally distributed.

Checking for equal variance:

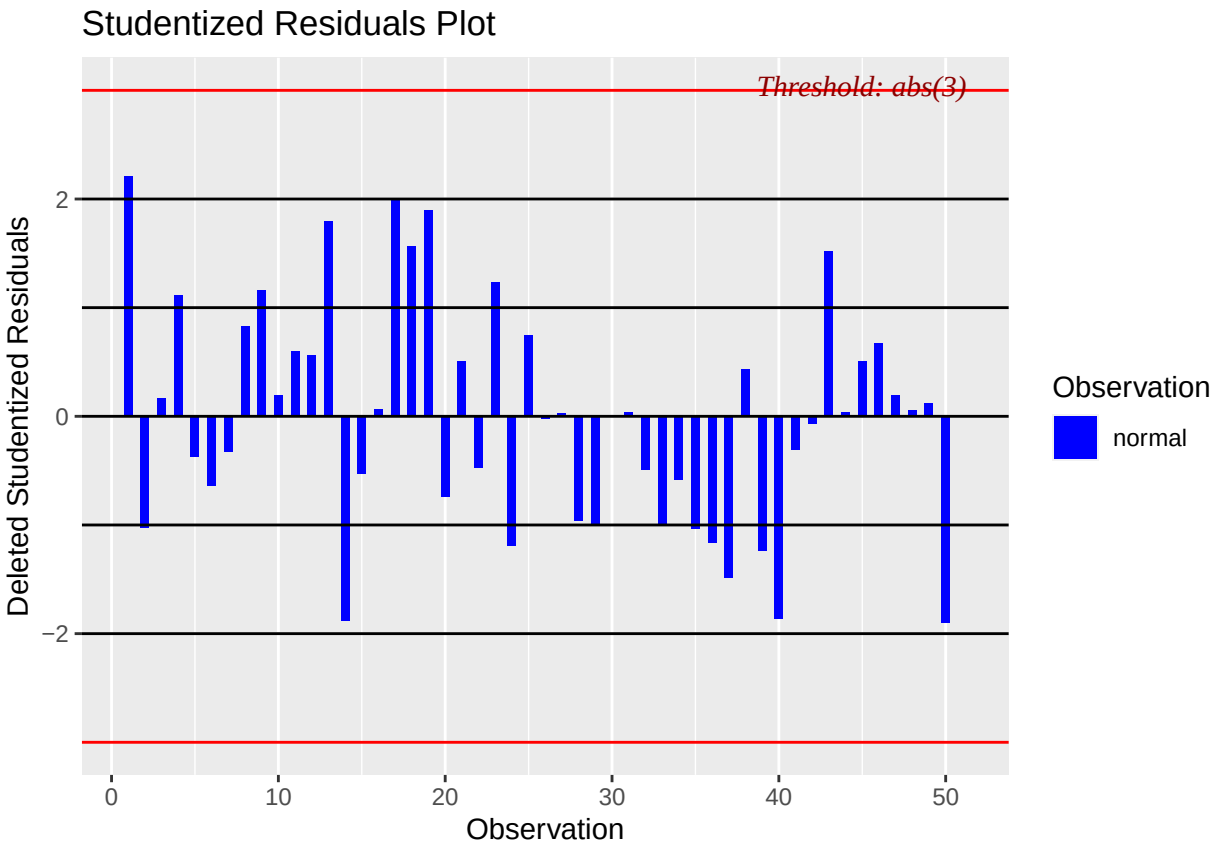
```r
# Run the Breusch Pagan Test for Heteroskedasticity to test for equal variance:
ols_test_breusch_pagan(model_1)
```

```
##
##  Breusch Pagan Test for Heteroskedasticity
##  -----------------------------------------
##  Ho: the variance is constant
##  Ha: the variance is not constant
##
##                  Data
##  -----------------------------------------
##  Response : sqrt(Mercury)
##  Variables: fitted values of sqrt(Mercury)
##
##         Test Summary
##  ----------------------------
##  DF            =    1
##  Chi2          =    0.07075233
##  Prob > Chi2   =    0.7902445
```

As can be seen from the above Breusch Pagan Test for Heteroskedasticity, since the p-value is 0.7902445 which is greater than 0.05, we fail to reject the null hypothesis, thus we have don't have evidence that the equal/constant variance assumption is violated.

Checking for no outliers:

```r
# Create a Studentized Residual Plot to check for outliers:
ols_plot_resid_stud(model_1)
```



As can be seen from the above studentized residual plot, there are no outliers present in the data.

Due to the fact that we violated none of the residual or model assumptions it is safe to say that we can use this model for predictions.

- b) Simulate the independent variable from a normal distribution using the mean 0 and standard deviation 2. Here, set.seed (1) and using n from the real data. In addition, simulate the dependent variable using the intercept and slope of 2 and 0.2 respectively. Assume, the error term is from a normal distribution with mean 0 and standard deviation 2. Calculate the slope and intercept for the least-squares regression line for the simulated data.

Below we will simulate the independent variable from a normal distribution using the mean 0 and standard deviation 2. In addition, we will simulate the dependent variable using the intercept and slope of 2 and 0.2 respectively. Lastly, we will assume that the error term is from a normal distribution with mean 0 and standard deviation 2. With that being said, we will calculate the slope and intercept for the least-squares regression line for the simulated data.

```
# Set the seed for reproducibility:
set.seed(1)

# Find the number of observations for simulation
n <- nrow(Lake)

# Simulate predictor variable:
x <- rnorm(n, 0, 2)
```

```r
# Simulate the error term:
e <- rnorm(n, 0, 2)

# Compute the outcome via the model:
y <- 0.2 * x + 2 + e

# Get the summary of the outcomes:
summary(y)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.6279  0.9735  2.2435  2.2343  3.4845  6.6811
```

As can be seen above, as we'd expect the outcome variable seems to be normally distributed around a mean of about 2.

```r
# Create the linear model of the simulated variables:
model_2 <- lm(y ~ x)

# Get the summary of the simulated model:
summary(model_2)
```

```
##
## Call:
## lm(formula = y ~ x)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.7318 -1.2792 -0.2423  1.0634  4.8016
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   2.1427     0.2695   7.950 2.28e-10 ***
## x             0.4309     0.1638   2.631   0.0113 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.909 on 49 degrees of freedom
## Multiple R-squared:  0.1238, Adjusted R-squared:  0.1059
## F-statistic: 6.922 on 1 and 49 DF,  p-value: 0.01135
```

Below we will calculate the corresponding 95% confidence intervals of the above model.

```r
# Find the corresponding 95% confidence intervals of the parameters:
confint(model_2)
```

```
##                 2.5 %    97.5 %
## (Intercept) 1.6010325 2.6843291
## x           0.1017787 0.7600245
```

In total, the regression equation is $\hat{y} = 2.1427 + 0.4308 \cdot x$.

Estimate interpretation for the model:
As computed above, the regression slope was 0.4308 with a 95% confidence interval of [0.1017787, 0.7600245]. This means, on average, for every 1 unit increase in the value of x the mean value of y is estimated to increase by 0.4308 over the sampled range of y values.

As computed above, the regression y-intercept was 2.1427 with a 95% confidence interval of [1.6010325, 2.6843291]. This means that the estimated mean level of y 2.1427 is when x is zero.

As can be seen by the confidence intervals and corresponding p-values, both of these estimates are significant at the 5% level.

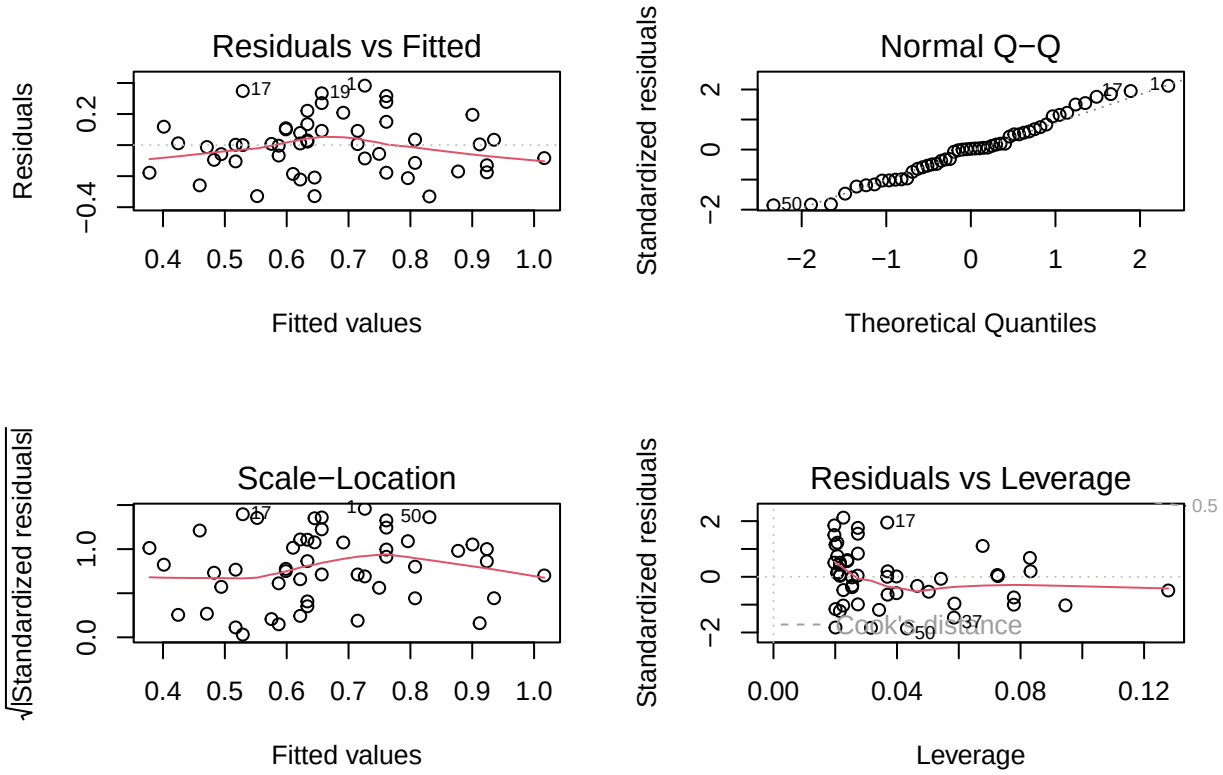
- c) Compare the results of the regression model from the real data and the simulated data. Explain clearly!

To compare the two models we will simply compare the results of the linear model summary for both models. The intercept in our simulated data is 2.1427 while the intercept for the real data with the transformation is 1.43435, so the intercept for the transformed data is smaller than that of the simulated data. Furthermore, the slope for the independent variable in the simulated data is 0.4309 and the slope for the transformed data is -0.11607, so the independent variable slope is greater and is positive in the simulated data, unlike the real data with the transformation in which the slope is smaller and negative. The R-squared value for the simulated model is significantly less than the transformed model, with a value of only 0.1238, meaning that only 12.38% of the variability in the model is explained by the independent variable. On the other hand, the transformed model had an R-squared value of 0.408, meaning that a decent 40.8% of the variability in the transformed model is explained by the independent variable. All variables and intercepts for both models were significant at the 5% level of significance.

- d) Examine the residuals from the real data and the simulated data. Make sure to comment.

For completeness we will re-run the model assumptions for model 1.

```
# Set up the frame:
par(mfrow = c(2, 2))

# Plot the residual assumptions for model 1:
plot(model_1)
```

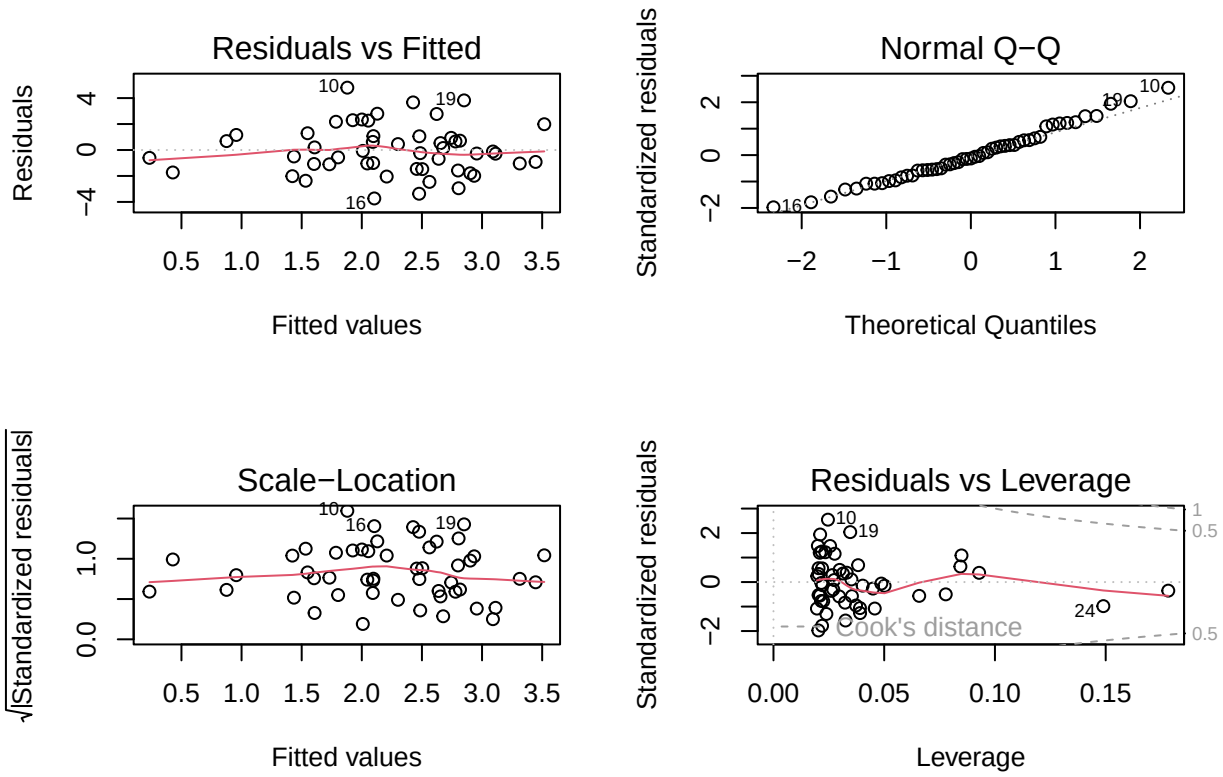


As can be seen from the above plots, due to the fact that the residuals vs fitted plot shows no patterns and seem to be evenly spaced around the red line, we assume that the linearity assumption holds. Next, since we see that the standardized residuals line up pretty well with the 45 degree line, we assume that the normality of the residuals assumption is not violated. Also, due to the fact that the scale-location plot shows no patterns and seems to be evenly spaced around the red line, we assume that the constant variance assumption holds. Lastly, since none of the standardized residuals fall past cook's distance we assume that no extreme outliers assumption holds.

These model diagnostics all agree with the manual residual assumption tests we did above.

Below we will assess the model assumptions for model 2.

```
# Set up the frame:
par(mfrow = c(2, 2))

# Plot the residual assumptions for model 2:
plot(model_2)
```



As can be seen from the above plots, due to the fact that the residuals vs fitted plot shows no patterns and seem to be evenly spaced around the red line, we assume that the linearity assumption holds. Next, since we see that the standardized residuals line up perfectly with the 45 degree line, we assume that the normality of the residuals assumption holds. Also, due to the fact that the scale-location plot shows no patterns and seem to be evenly spaced around the red line, we assume that the constant variance assumption holds. Lastly, since none of the standardized residuals fall past cook's distance we assume that no extreme outliers assumption holds.

All of these residual assumptions are as we'd expect since we purposely made all of the model assumptions hold for our simulated model.

As can be seen above, both models have the same conclusions for their residuals plots, and the plots themselves look very similar.