STAT 302 Homework 7

Jaiden Atterbury

Due on: 05/29/2023

1. What does the function below do? What mathematical function is this? Try running the code chunk below and then running $h(x)$ for different values of x.

```
h <- function(x) {
  if (x > 1) {
    return(x * h(x - 1))
  } else {
    return(1)
  }
}
```

The above function h is a function that uses a more advanced technique called recursion. Recursion occurs when a function calls itself creating a "looping" pattern where every time the function is called, it calls itself again and again until some condition is met. In essence it is a while loop in function form.

In particular, this function takes in what is assumed to be a non-negative integer value x and then checks if the integer is greater than 1, if it isn't, then the function returns 1. If its not, then the function returns the value x time the function itself evaluated at $x-1$. This process will repeat until the value of x is less than or equal to 1. If we limit our input values x to only non-negative integers than we can see that this is the mathematical function called the factorial function which takes the product of the non-negative integers up to and including x.

For example if we let the input value x be 5, then the returned value is $5 \cdot h(5-1) = 5 \cdot h(4)$. which will call the h function again. Doing so will now call h with an x value of 4 which yields the result $5 \cdot 4 \cdot h(4-1) = 5 \cdot 4 \cdot h(3)$. This will continue until $x = 1$, which will yield the result $5 \cdot 4 \cdot 3 \cdot 2 \cdot 1 = 120 = 5!$.

However, this function also allows for negative and floating point inputs. Thus, this function serves as a sort of generalization of the factorial function for values that aren't non-negative integers, and the exact factorial functions for non-negative integers inputs.

Below we will create a table of the function h and the built-in factorial function to show the similarity.

Test the above h function for integers:

```
# Create the output tibble:
fact_table <- tibble(x = c(0, 1, 2, 5, 10),
                     `h(x)` = rep(0, 5),
                     `factorial(x)` = rep(0, 5))


# Calculate the value for the functions h and factorial for each value of x:
index = 0
for (value in fact_table$x) {
  index = index + 1
```

```
  fact_table[index, 2] = h(value)
  fact_table[index, 3] = factorial(value)
}

# Show the tibble:
fact_table
```

```
## # A tibble: 5 x 3
##       x  `h(x)` `factorial(x)`
##   <dbl>   <dbl>          <dbl>
## 1     0       1              1
## 2     1       1              1
## 3     2       2              2
## 4     5     120            120
## 5    10 3628800        3628800
```

As can be seen by the above tibble, the values of $h(x)$ are identical to those of factorial(x) for non-negative integers.

However, as discussed below, this function generalizes for non-negative and floating point inputs.

Test the above h function for negative values:

```
# Analyze the function for negative input:
h(-1.2)
```

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

Any negative value will always result in a function value of 1, however, decimal values greater than 1 will be different from 1, which is show below.

Test the above h function for decimal values:

```
# Analyze the function for decimal input:
h(2.4)
```

```
## [1] 3.36
```

Hence this function is more than just the factorial function if we don't narrow our domain of input numbers.

2. Write a function to compute the mean, median, mode, variance, and skewness of a binomial (for (i) n = 10, p = 0.2; and (ii) n = 100, p = 0.5) distribution. A binomial distribution with the correct formulas can be found here: https://en.Wikipedia.org/wiki/Binomial_distribution

Below we will write a function to compute the mean, median, mode, variance, and skewness of a binomial distribution. Then we will test the function on binomial distributions with different parameter values.

In order to understand what is going on in the below functions, we will first explain how to calculate each value. The formula for the mean is np, the formula for the variance is $np(1-p)$, the formula for the skewness is $\frac{(1-p)-p}{\sqrt{np(1-p)}}$. The formula for the mode however is quite difficult and includes three parts/conditions. If the value of $(n+1)p$ is equal to 0 or is a non-integer, the value of the mode is the floor of $(n+1)p$. Secondly, if the values of $(n+1)p$ is a positive integer, then the distribution has two modes: $(n+1)p$ and $(n+1)p-1$.

Lastly if $(n+1)p$ is equal to $n+1$, then the mode is simply n. Lastly, making a function to calculate the median isn't entirely possible since there isn't one formula for finding the median and it may not be unique. However, as mentioned in class we could either use the floor or ceiling of np or we could create a more complicated formula as described below. First off, if np is an integer then the median is simply np. Also, if $p = \frac{1}{2}$ and n is an odd integer, then the any number m in the interval $\frac{1}{2}(n-1) \leq m \leq \frac{1}{2}(n+1)$ is a median of the binomial distribution. The next condition sates that the median is $round(np)$ if $|m-np| \leq min\{p, 1-p\}$ except for the case where $p = \frac{1}{2}$ and n is an odd integer. Lastly, in all other cases, the median m is simply within the interval of the floor of np and the ceiling of np.

Create the binomial statistics function:

```r
# Implement the binomial stats function:
binom_stats <- function(n, p) {
  # Calculate the mean:
  mean <- n * p

  # Calculate the variance:
  variance <- n * p * (1 - p)

  # Calculate the skewness:
  skew <- ((1 - p) - p) / sqrt(variance)

  # Calculate the median:

  # Case 1: np is an integer:
  if (n*p %% 1 == 0) {
    median <- n*p
  }

  # Case 2: p=0.5 and n is odd:
  else if (p == 1 / 2 & n %% 2 == 1) {
    median <- c((n - 1) / 2, (n + 1) / 2)
  }

  # Case 3: median = round(np) if the absolute value of the median minus np
  # is less than or equal to the minimum of p or 1-p:
  else if (abs(round(n * p) - n * p) <= min(p, 1 - p)) {
    median <- round(n * p)
  }

  # Case 4: Every other case:
  else {
    median <- c(floor(n*p), ceiling(n*p))
  }

  # Calculate the mode:
  value <- (n + 1) * p

  # Case 1: success probability is 1:
  if (value == n + 1) {
    mode <- n
  }

  # Case 2: value is a positive integer:
  else if (value != 0 & value %% 1 == 0) {
```

```
    mode <- c(value, value - 1)
  }

  # Case 3: value is 0 or a non-integer:
  else {
    mode <- floor(value)
  }

  # Create the list to return:
  return_list <- list(mean, variance, median, skew, mode)

  # Give the list elements names:
  names(return_list) <- c("Mean", "Variance", "Median", "Skewness", "Mode")

  return(return_list)
}
```

Below we will test the function implementation for a $Binom(10, 0.2)$ distribution.

Test the binom_stats function for specification example 1:

```
# Test case 1:
binom_stats(10, 0.2)
```

```
## $Mean
## [1] 2
##
## $Variance
## [1] 1.6
##
## $Median
## [1] 2
##
## $Skewness
## [1] 0.4743416
##
## $Mode
## [1] 2
```

As can be seen from the above function call, the mean, median, and mode are all 2. This is what we'd expect since np is an integer value. We can also see that the variance is 1.6, and the skewness is 0.4743416. All of these values are what we'd expect from theory.

Below we will test the function implementation for a $Binom(100, 0.5)$ distribution.

Test the binom_stats function for specification example 2:

```
# Test case 2:
binom_stats(100, 0.5)
```

```
## $Mean
## [1] 50
##
## $Variance
```

```
## [1] 25
##
## $Median
## [1] 50
##
## $Skewness
## [1] 0
##
## $Mode
## [1] 50
```

As can be seen from the above function call, the mean, median, and mode are all 250. This is what we'd expect since np is an integer value. We can also see that the variance is 25, and the skewness is 0. The skewness being zero makes sense since p is equal to q in this given binomial distribution. All of these values are what we'd expect from theory.

Most of the function implementations don't matter since we only deal with cases where np is an integer and thus don't hit the other cases but below we will indicate that the other branches do in fact work.

Testing Case 1 of the mode:

```
# Testing Case 1 of the mode:
binom_stats(10, 1)
```

```
## $Mean
## [1] 10
##
## $Variance
## [1] 0
##
## $Median
## [1] 10
##
## $Skewness
## [1] -Inf
##
## $Mode
## [1] 10
```

As can be seen from above, since $(n+1)p = n+1$, our value of the mode simply equals n.

Testing Case 2 of the mode and Case 2 of the median:

```
# Testing Case 2 of the mode and median:
binom_stats(7, 0.5)
```

```
## $Mean
## [1] 3.5
##
## $Variance
## [1] 1.75
##
## $Median
## [1] 3 4
```

```
##
## $Skewness
## [1] 0
##
## $Mode
## [1] 4 3
```

As can be seen from above, since $(n+1)p \neq 0$ and $(n+1)p$ is a positive integer, our mode takes on two values $(n+1)p$ and $(n+1)p-1$. Also since $p = \frac{1}{2}$ and n is an odd integer, it follows that any number m in the interval $\frac{1}{2}(n-1) \leq m \leq \frac{1}{2}(n+1)$ is a median of the binomial distribution. In our case this interval is $[3, 4]$.

Testing Case 3 of the median:

```
# Testing Case 3 of the median:
binom_stats(10, 0.33)
```

```
## $Mean
## [1] 3.3
##
## $Variance
## [1] 2.211
##
## $Median
## [1] 3
##
## $Skewness
## [1] 0.228657
##
## $Mode
## [1] 3
```

As can be seen above, since $|m - np| \leq min\{p, 1-p\}$ and we aren't in the case where $p = \frac{1}{2}$ and n is an odd integer. The median is $round(np)$ if $|m - np| \leq min\{p, 1-p\}$, hence in our case the median is 3.

Testing Case 4 of the median:

```
# Testing Case 4 of the median:
binom_stats(123, 0.67)
```

```
## $Mean
## [1] 82.41
##
## $Variance
## [1] 27.1953
##
## $Median
## [1] 82 83
##
## $Skewness
## [1] -0.06519766
##
## $Mode
## [1] 83
```

As can be seen, since none of the above cases were satisfied we are in case 4, thus all we can say is that the median m is simply within the interval of the floor of np and the ceiling of np. Hence in our case, the median is somewhere in the interval $[82, 83]$.

3. Using Helium data set posted on Canvas. This data set contains data from an experiment in which two identical footballs, one air filled with air and one filled with helium, were each kicked 39 times. The distances traveled were measured for each trial.

In this problem we are given a data set which contains data from an experiment in which two identical footballs, one filled with air and one filled with helium, were each kicked 39 times. The distances traveled were measured for each trial. In reality, this data was a matched pairs example, however for the sake of this class we will consider these as two separate samples.

- a) Use the t.test function to test the null hypothesis that the mean distance traveled by the helium-filled ball is equal to the mean distance traveled by the air-filled ball. Use a one-sided hypothesis (you think that since balloons filled with helium float better than balloons filled with air that your helium filled football might go further). What do you conclude?

In this part of the problem we will use the t.test function to test the null hypothesis that the mean distance traveled by the helium-filled ball is equal to the mean distance traveled by the air-filled ball. The competing alternative hypothesis will be a one-sided hypothesis. In particular the alternative states that the mean distance traveled by the helium-filled ball is greater than that of the air-filled ball. Before we run this test we will check our model assumptions.

First off since the dependent variable is distance traveled it is measured on an interval/ratio scale. Secondly, the independent variable has two discrete levels; helium-filled and air-filled. Next, since we aren't thinking of this data as matched pairs even though in reality it is, we can safely assume the two groups being compared are independent of each other. We can assume independence because one kick from one group does not tell us anything about a kick from another group. Below we will check the normality of the dependent variable assumption, and the equal variance assumption. However, first we will set up the data for comparison.

Reconstruct the data for the tests:

```
# Create the labels for the distance variable:
label <- rep(c("air", "helium"), times=c(39, 39))

# Get the distance values associated with each label:
distance <- c(helium$air, helium$helium)

# Create a tibble of the label and distance values:
combined_data <- tibble(label = label, distance = distance)
```

All of the tests for this problem will be conducted at the 5% level of significance.

Normality of the air-filled subgroup:

```
# Run a Shapiro test for normality of response assumption:
shapiro.test(combined_data$distance[combined_data$label == "air"])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  combined_data$distance[combined_data$label == "air"]
## W = 0.97738, p-value = 0.6089
```

As can be seen from the above Shapiro-Wilk normality test, the p-value is 0.6089 which is greater than 0.05, thus we fail to reject the null hypothesis and assume that the air-filled subgroup of the response variable is approximately normally distributed.

Normality of the air-filled subgroup:

```
# Run a Shapiro test for normality of response assumption:
shapiro.test(combined_data$distance[combined_data$label == "helium"])
```

```
##
##  Shapiro-Wilk normality test
##
## data:  combined_data$distance[combined_data$label == "helium"]
## W = 0.92101, p-value = 0.009347
```

As can be seen from the above Shapiro-Wilk normality test, the p-value is 0.009347 which is less than 0.05, thus we reject the null hypothesis and assume that the helium-filled subgroup of the response variable is not approximately normally distributed.

As can be seen the response variable is not approximately normally distributed across both groups, thus the use of a t-distribution can't be justified. However, for the sake of the problem we will continue with

Equal variance assumption:

```
# Run a F test for equal variance assumption:
var.test(helium$helium, helium$air)
```

```
##
##  F test to compare two variances
##
## data:  helium$helium and helium$air
## F = 1.7576, num df = 38, denom df = 38, p-value = 0.08621
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  0.9216806 3.3518418
## sample estimates:
## ratio of variances
##           1.757648
```

As can be seen from the above F test to compare two variances, the p-value is 0.08621 which is greater than 0.05, thus we fail to reject the null and assume that the two groups have similar variances. We will now run the t-test using the t.test function in R.

Run the two sample t-test:

```
# Factor the label column to get the right order of subtraction:
combined_data$label <- factor(combined_data$label, levels=c("helium", "air"))

# Run the t-test:
t.test(distance~label, data=combined_data, alternative="greater", var.equal=TRUE)
```

```
##
##  Two Sample t-test
##
```

```
## data:  distance by label
## t = 0.37032, df = 76, p-value = 0.3561
## alternative hypothesis: true difference in means between group helium and group air is greater than 0
## 95 percent confidence interval:
##  -1.613768       Inf
## sample estimates:
## mean in group helium    mean in group air
##             26.38462             25.92308
```

As can be seen from the above two sample t-test, the p-value was 0.3561 with a t-statistic value of 0.37032 on 76 degrees of freedom. This means that at the 5% level we fail to reject the null hypothesis. Hence we don't have evidence that true average distance traveled in the helium-group is greater than that in the air-filled group.

- b) Write code to run a permutation test to test the same hypothesis as above.

Below we will write code to run a permutation test to test the same hypothesis as above. Namely we're testing $H_0 : \mu_H - \mu_A$ versus $H_A : \mu_H - \mu_A > 0$, where μ_H is the mean distance traveled in the helium-filled group, and μ_A is the mean distance traveled in the air-filled group. In class we were told that we didn't need to set a unique seed. However, for reproducibility and in order to make sure my analysis matches my results, I set the seed to 1.

Run the permutation test:

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

# Initialize empty difference vector:
permuted_diff <- numeric()

# Run a permutation test:
for (i in 1:10000) {
  # For each iteration, scramble the label variable:
  permuted_data <- combined_data %>% mutate(label = sample(label))

  # Compute the difference in means:
  diff <- mean(permuted_data$distance[permuted_data$label == "helium"]) -
          mean(permuted_data$distance[permuted_data$label == "air"])

  # Store the difference in means:
  permuted_diff[i] <- diff
}
```

Now that we have run the permutation test we will calculate the permuted p-value below.

Compute the permuted p-value:

```
# Compute the observed difference in means from the data:
obs_diff <- mean(combined_data$distance[combined_data$label == "helium"]) -
            mean(combined_data$distance[combined_data$label == "air"])

# Find the permuted p-value:
mean(permuted_diff > obs_diff)
```

```
## [1] 0.3403
```

As computed above, the p-value was 0.3403. We will interpret this value in the next part.

- c) What is your conclusion from the permutation test? Does it match your conclusion from your t-test?

As computed above, the p-value was 0.3403. This means that at the 5% level we fail to reject the null hypothesis. Hence we don't have evidence that true average distance traveled in the helium-group is greater than that in the air-filled group.

As can be seen from the above t-test and permutation tests, the p-values were very similar and the conclusion was identical; that we fail to reject the null and that we don't have evidence that true average distance traveled in the helium-group is greater than that in the air-filled group.