Lesson 7 Example Code

Example Code:

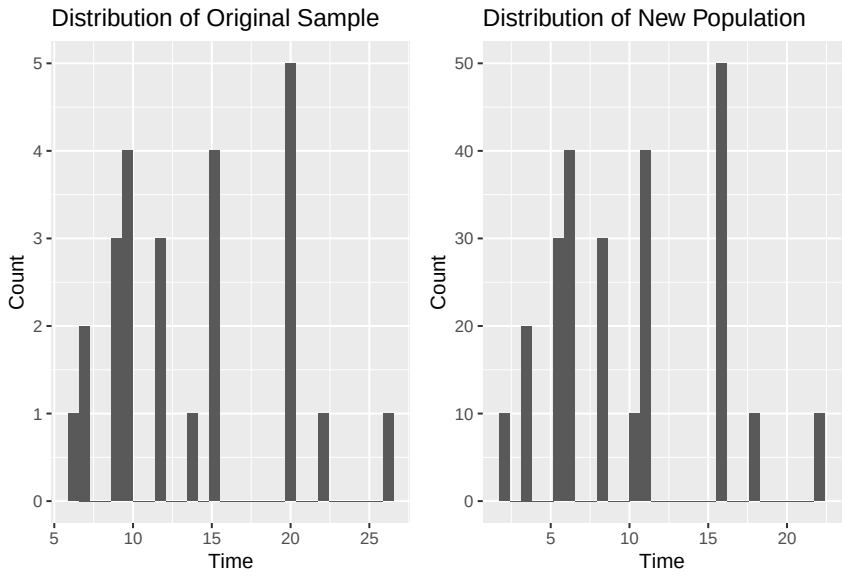
Simulation-based Approach:

I always tell my students that, at the end of this discussion, you feel like we've done something sketchy then you probably grasp the concepts. Rest assured that, however sketchy it may feel, what we are about to do is grounded in solid statistics. For this example I'm going to use the data available on the book's website for the "Time Estimate" problem. It doesn't match the exact data in the textbook (for some reason) but it'll work for this example.

```
library(tidyverse)
library(patchwork) #Install this if you want to use it

estimates = read_table2("http://www.isi-stats.com/isi/data/chap3/TimeEstimate.txt")

head(estimates)
```

```
## # A tibble: 6 x 1
##    Time
##   <dbl>
## 1     6
## 2     7
## 3     7
## 4     9
## 5     9
## 6     9
```

```
sample_stat = mean(estimates$Time)

new_population = do.call("rbind", replicate(10, estimates, simplify = FALSE))

null_mean = 10

new_population = new_population - abs(mean(new_population$Time) - 10)

p1 <- estimates %>%
  ggplot(aes(x = Time)) +
  labs(x = "Time", y = "Count", title = "Distribution of Original Sample") +
  geom_histogram()

p2 <- new_population %>%
  ggplot(aes(x = Time)) +
  labs(x = "Time", y = "Count", title = "Distribution of New Population") +
  geom_histogram()

p1 + p2
```



Let's take this part-by-part to help explain what is going in. After loading the libraries we will need, we load the dataset from the book's website. I can check to ensure that it was loaded correctly (and get the column title) using the head command. My next step is to start building my new population by copying the estimates dataframe ten times. The **do.call** function performs whatever function is listed (in this case **r**ow **bind**) given the arguments that are listed next. Here we are binding these ten replicates for our empirical datasets into a single dataframe.

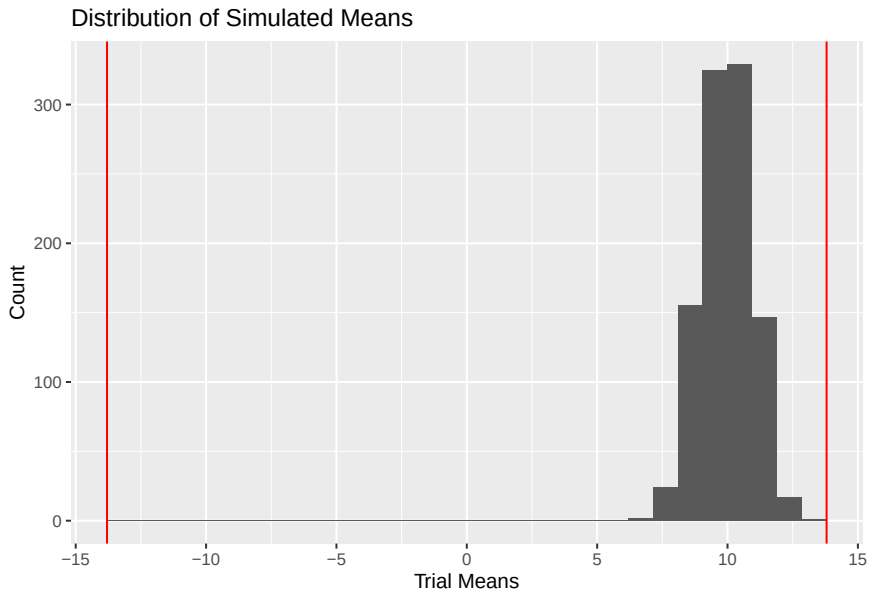
The next step is to shift the mean of our new population. Recall that the purpose of building this new population is to have a population to sample from to build the null distribution. Of course you recall that, when you are building the null distribution, you assume the truth of the null hypothesis. In this problem our null hypothesis is that the average estimate will be 10 seconds. Therefore we need to build a population where the mean is 10 seconds. We can do this by finding the absolute value of the difference between the real mean of the new population and our null mean (**abs(mean(new_population$Time) - 10)**) and subtracting that from every element in the **new_population** dataframe.

So now that we have this new population with a mean of 10 seconds... let's reflect on how this could possibly be an acceptable method. Consider that what we are really looking for when building our null distribution is an idea of how spread out we can expect the results to be. When we built the null distribution using coin flips for the Doris and Buzz problem we were trying to see how spread out the simulated results were. Ultimately, it answered the question: how likely would it be to see 15 heads by chance alone? This is a question of the variation of our results.

Take a look at the two plots here and consider, even when we have shifted our mean, are we capturing the spread of our empirical sample? Of course we are! Since we merely replicated the sample ten times (ignoring the y-axis) our histograms look exactly the same. Now we can get to the actual work of building our simulated null hypothesis.

```
replications_dataframe = NULL

num_reps = 1000

sample_size = 25 #Original sample size

for (i in 1:num_reps){

  trial = sample(new_population$Time,
```

```
                  size = sample_size,
                  replace = FALSE)

  trial_mean = mean(trial)

  replications_dataframe = rbind(replications_dataframe, data.frame(trial_mean))

}

replications_dataframe %>%
  ggplot(aes(x = trial_mean)) +
  geom_histogram() +
  labs(x = "Trial Means", y = "Count", title = "Distribution of Simulated Means") +
  geom_vline(xintercept = sample_stat, color = "red") +
  geom_vline(xintercept = -sample_stat, color = "red")
```




```
#Find the two-sided p-value
replications_dataframe %>%
  summarise(pvalue = sum(abs(trial_mean) >= abs(sample_stat)) / n())
```

```
##   pvalue
## 1      0
```

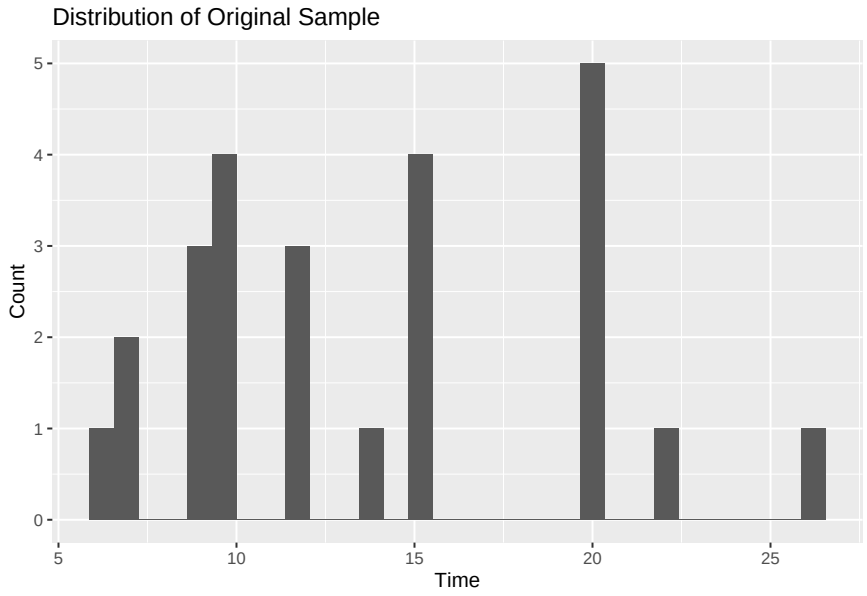
One-sample T-test I'm going to first show you a method that will, hopefully, help you understand calculating the p-value... and then I'm going to show you the easy way. We'll continue with the same example and check our validity conditions.

```
#At least 20 observations...
nrow(estimates)
```

```
## [1] 25
```

```
#...and their distribution should not be strongly skewed
estimates %>%
  ggplot(aes(x = Time)) +
  labs(x = "Time", y = "Count", title = "Distribution of Original Sample") +
```

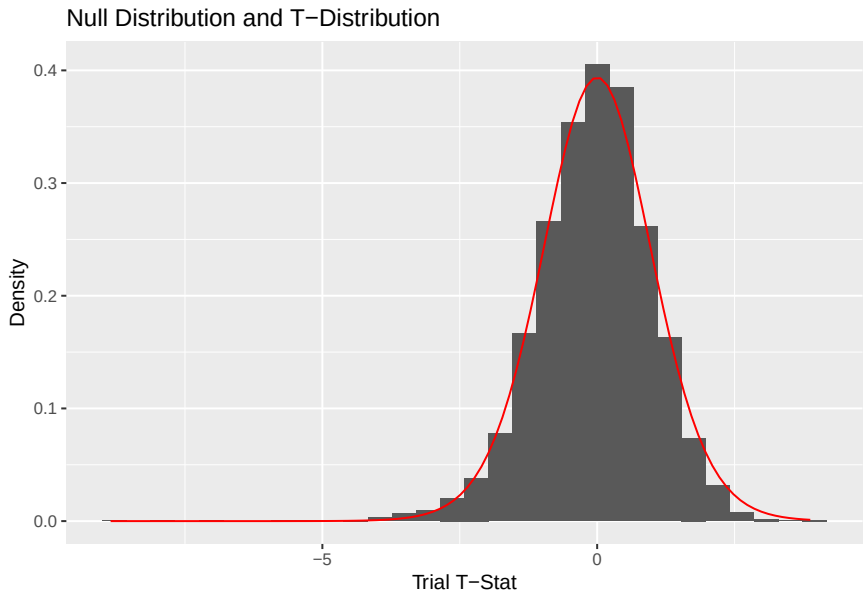
```
  geom_histogram()
```



A question you're going to ask again and again is "how skewed is *strongly skewed*" and I don't have a definite answer for you other than pointing out what I think is *strongly skewed* when I see it. I assess this distribution is not strongly skewed given all of our observations are between 5 and 26 seconds. Now that we've met our validity conditions, we can utilize the t distribution to represent our null distribution.

```
replications_dataframe = NULL

num_reps = 5000 #Increased for smoother null distribution

sample_size = 20

for (i in 1:num_reps){

  trial = sample(new_population$Time,
                 size = sample_size,
                 replace = FALSE)

  standard_error = sd(trial) / sqrt(sample_size)

  trial_t_stat = (mean(trial) - null_mean) / standard_error

  replications_dataframe = rbind(replications_dataframe, data.frame(trial_t_stat))

}



replications_dataframe %>%
  ggplot(aes(x = trial_t_stat)) +
  labs(x = "Trial T-Stat", y = "Density",
       title = "Null Distribution and T-Distribution") +
  geom_histogram(aes(y = ..density..)) +
  stat_function(fun = dt,
```

```
                args = list(df = sample_size - 1),
                color = "red")
```



The code above is here to help demonstrate why a t-distribution is appropriate for modeling our null distribution given the validity conditions are met. It should look relatively familiar as it is building the null distribution in a very similar manner to my previous examples. However, instead of calculating the trial mean or proportion, I calculate the trial t-statistic. You can see from the plot that the t-distribution isn't a perfect representation but fits our null distribution relatively well. Now that (hopefully) I've got your convinced, let's look at using the **pt** function to calculate our p-value.

```
sample_t_stat = (sample_stat - null_mean) / (sd(estimates$Time) / sqrt(nrow(estimates)))

#One of the tails (left for this example)
pt(-sample_t_stat, df = nrow(estimates) - 1)
```

```
## [1] 0.0009028877
```

```
#The other tail (right for this example)
1 - pt(sample_t_stat, df = nrow(estimates) - 1)
```

```
## [1] 0.0009028877
```

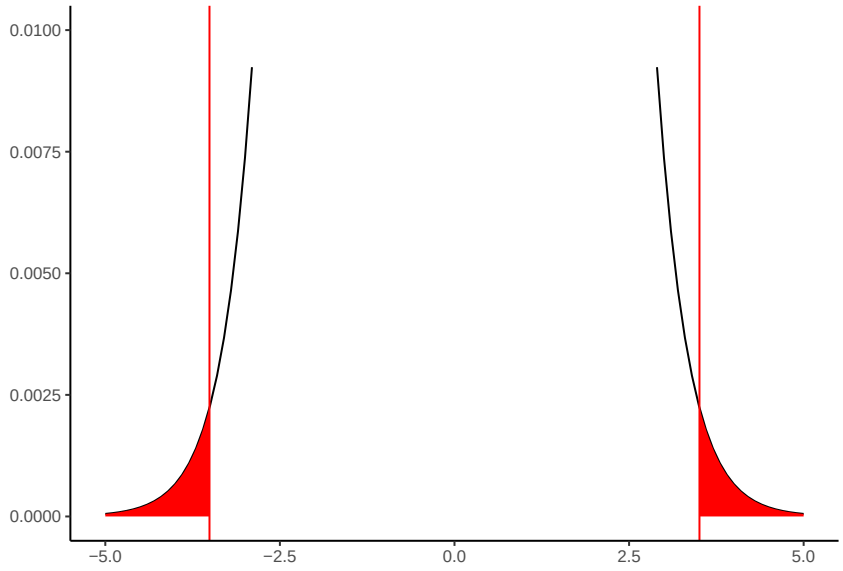
```
#Both tails together
2 * (1 - pt(abs(sample_t_stat), df = nrow(estimates) - 1))
```

```
## [1] 0.001805775
```

The first step here was calculating the t-statistic for our original sample. It might be hard to remember that original sample at this point so maybe scroll up to remind yourself. We can then use the **pt()** function to find the area under the t-distribution curve. It's like **pnorm()** but for the t-distribution instead. In essence we are looking for the red area in the figure below. Clearly, I've zoomed in here so you can see the actual area.

```
ggplot(NULL, aes(x = c(-5, 5))) +
    stat_function(fun = dt,
                  args = list(df = nrow(estimates) - 1)) +
  geom_vline(xintercept = sample_t_stat, color = "red") +
  geom_vline(xintercept = -sample_t_stat, color = "red") +
```

```
  stat_function(fun = dt,
                args = list(df = nrow(estimates) - 1),
                xlim = c(-5, -sample_t_stat),
                geom = "area", fill = "red") +
    stat_function(fun = dt,
                args = list(df = nrow(estimates) - 1),
                xlim = c(sample_t_stat, 5),
                geom = "area", fill = "red") +
  labs(x = "", y = "") + theme_classic() + ylim(c(0, 0.01))
```



While I think it's important for you to know how to use the **pt()** function because it promotes a deeper understanding, *R* has another function that will allow you to quickly calculate the p-value.

```
t.test(estimates, mu = null_mean, alternative = "two.sided")
```

```
##
##  One Sample t-test
##
## data:  estimates
## t = 3.5081, df = 24, p-value = 0.001806
## alternative hypothesis: true mean is not equal to 10
## 95 percent confidence interval:
##  11.56437 16.03563
## sample estimates:
## mean of x
##      13.8
```

Yes, that was **a lot** less code. Spend some time playing around with the parameters of **t.test()** so you become familiar with what each does and how to apply this function to different types of problems.