Lesson 7 Companion

Research question:

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.

Remember the research question is whether or not people can accurately estimate the length of a short song snippet. The parameter of interest is the population average length estimate (μ), the statistic is the sample average length estimate ($\bar{x}$), and the hypothesis are as follows:

$H_0 : \mu = 10$

$H_a : \mu \neq 10$

Simulation-based Approach

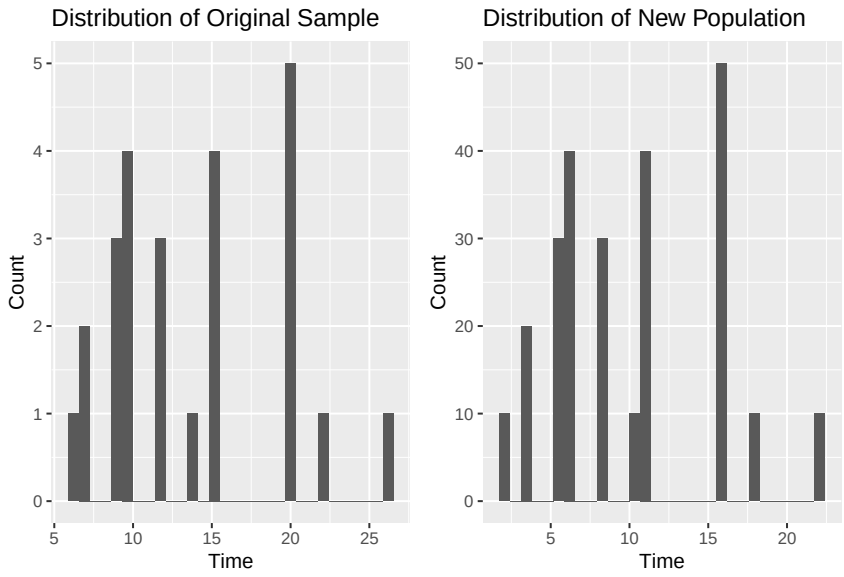
I always tell my students that, at the end of this discussion, you feel like we've done something sketchy then you probably grasp the concepts. Rest assured that, however sketchy it may feel, what we are about to do is grounded in solid statistics.

```
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.

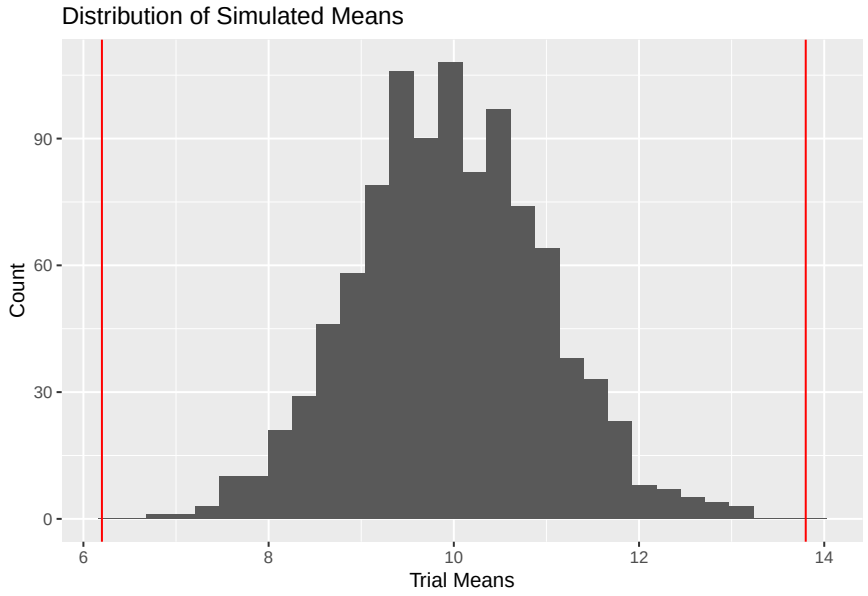
The next step is to shift the mean of our new population. Recall that the purpose of building this new population is to have a population to sample from to build the null distribution. Of course you recall that, when you are building the null distribution, you assume the truth of the null hypothesis. 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
```

```r
sample_size = 25 #Original sample size

for (i in 1:num_reps){

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

  trial_stat = mean(trial)

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

}

#Calculate how far away from the null mean
# our sample mean was.
dist_from_null = abs(sample_stat - null_mean)

#Value above the null mean
above = null_mean + dist_from_null

#Value below the null mean
below = null_mean - dist_from_null

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




```r
#Find the two-sided p-value
replications_dataframe %>%
  summarise(pvalue = (sum(trial_stat <= below) +
```

```
            sum(trial_stat >= above)) / n())
```

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

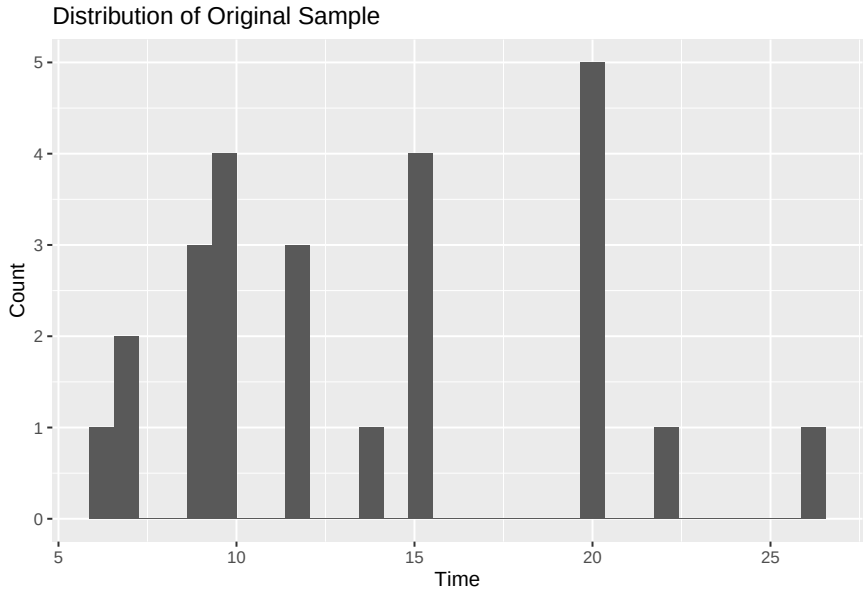
One-sample t-test

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()
```

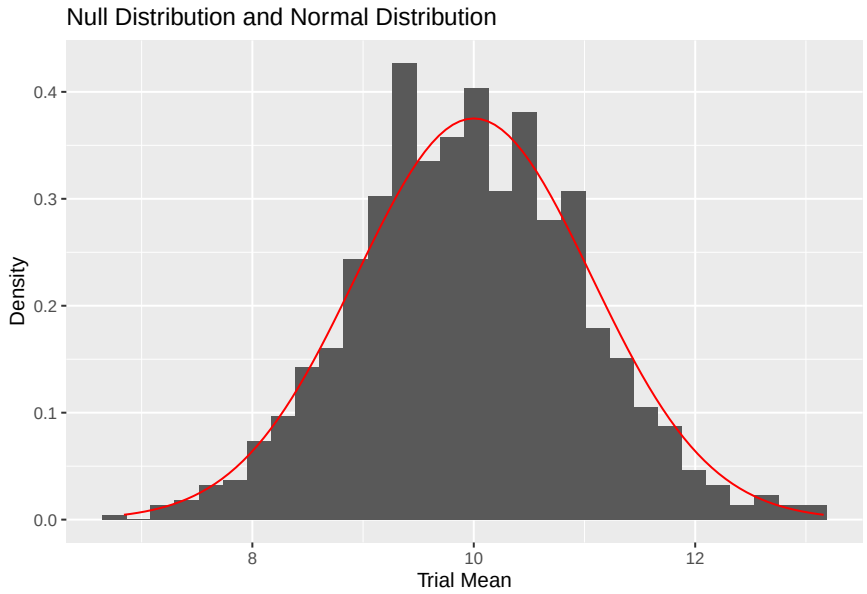


A question you're going to ask again and again is "how skewed is *strongly skewed*" and I would advise you to go check out my "Normality Testing" document on GitHub. In the end, I assess this distribution is not strongly skewed. Now that we've met our validity conditions, the central limit theorem (CLT) tells us that our "trial means" will be distributed normally if the population is distributed normally or approximately normally if the sample size is large enough (regardless of the population distribution). These normal distributions will have a mean of μ and a standard deviation of $\frac{\sigma}{\sqrt{n}}$.

But wait, we don't have our population standard deviation (σ) so what do we do? The best we can do is approximate it using the sample standard deviation(s). Note: Even though it is named *new_population*, our somewhat artificial *population* is not truely our broader population.

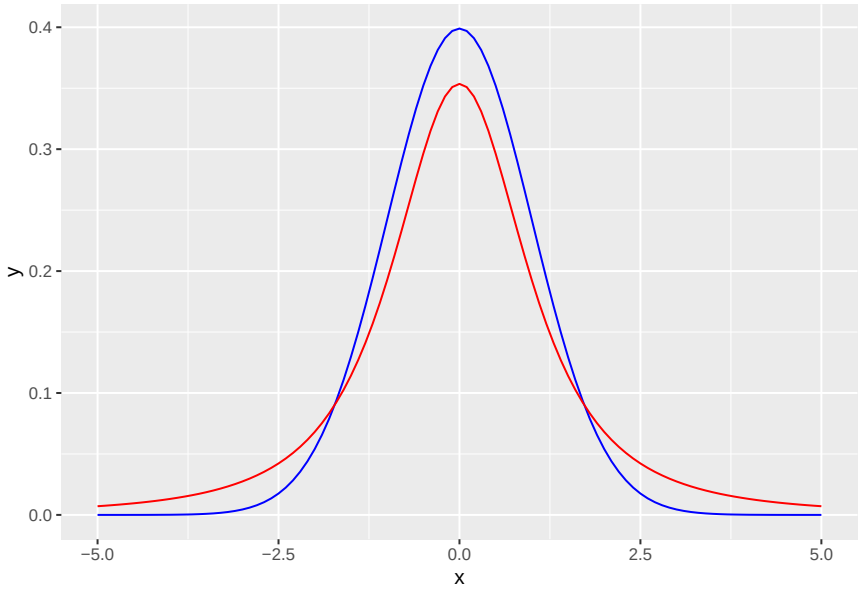
```
CLT_sd = sd(new_population$Time) / sqrt(sample_size)

replications_dataframe %>%
  ggplot(aes(x = trial_stat)) +
```

```
  labs(x = "Trial Mean", y = "Density",
       title = "Null Distribution and Normal Distribution") +
  geom_histogram(aes(y = ..density..)) +
  stat_function(fun = dnorm,
                args = list(mean = null_mean, sd = CLT_sd),
                color = "red")
```



You might ask "but how do we account for the uncertainty introduced by using the sample standard deviation as opposed to the population standard deviation?" That's a great question and it explains why the **t**-distribution is used instead of the normal distribution. The t-distribution has "fatter tails" than the normal distribution for a given sample size and therefore you need have a more extreme sample statistic in order to reject the null hypothesis at a given significance level (Type 1 Error/False Alarm).

```
#Just picking a sample size for demonstration
n = 3

ggplot(data.frame(x = c(-5,5)), aes(x = x)) +
  stat_function(fun = dnorm,
                color = "blue") +
  stat_function(fun = dt,
                args = list(df = n - 1),
                color = "red")
```



Other than the extra area in the tails, the t-distribution functions exactly like the normal distribution for finding our p-value.

```
#Back to using the estimates dataframe
standard_deviation = sd(estimates$Time)

sample_size = nrow(estimates)

sample_t_stat = (sample_stat - null_mean) / (standard_deviation / sqrt(sample_size))

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

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

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

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

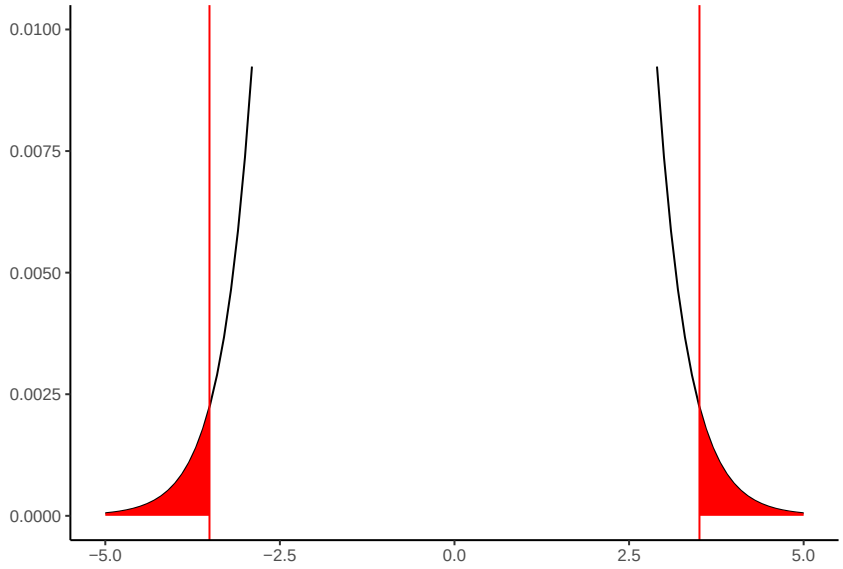
```
#Both tails together
2 * (1 - pt(abs(sample_t_stat), df = sample_size - 1))
```

```
## [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(data.frame(x = c(-5, 5)), aes(x = x)) +
    stat_function(fun = dt,
                  args = list(df = sample_size - 1)) +
  geom_vline(xintercept = sample_t_stat, color = "red") +
  geom_vline(xintercept = -sample_t_stat, color = "red") +
  stat_function(fun = dt,
                args = list(df = sample_size - 1),
                xlim = c(-5, -sample_t_stat),
```

```
                geom = "area", fill = "red") +
    stat_function(fun = dt,
                args = list(df = sample_size - 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.