Lesson 5 Companion

Simulating a Null Distribution:

While the applets make it very easy to simulate a null distribution, I want to provide you with some code to demonstrate how to do a similar thing in *R*. I find that more exposure to code make students better coders (radical thought... I know) so don't just skip over this section. I will use the halloween treat example in your book for this code.

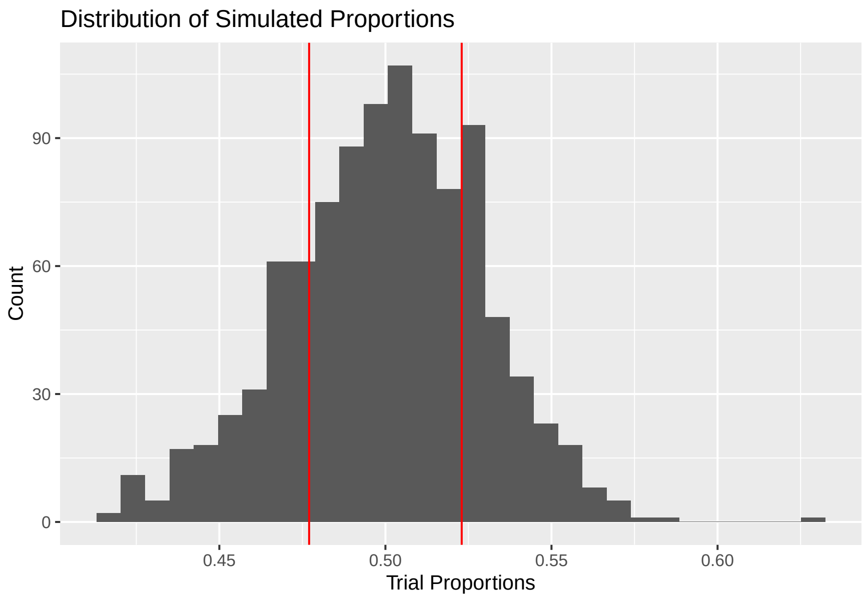
```r
library(tidyverse)

replications_dataframe = NULL

num_reps = 1000

sample_size = 135 + 148

null_prop = 0.5

sample_stat = 148 / (135 + 148)

for (i in 1:num_reps){

  trial = sample(x = c(1,0),
                 size = sample_size,
                 prob = c(null_prop, 1-null_prop),
                 replace = TRUE)

  trial_stat = sum(trial) / sample_size

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

}
```

After turning on the awesome, this code sets up several "parameters" for our simulation. This is "parameter" in the general sense, not to be confused with statistical parameters. We create a dataframe, or storage spot, to put our simulated proportions. We then define the number of repetitions (1000), the size of each simulated repetitions (283), the null probability (null hypothesis of random choice between two options), and the sample statistic (what we observed in our experiment).

What follows is a **for** loop that serves as the "simulation" in our experiment. The code here is a bit more advanced but nothing insurmountable. The variable *trial* is basically a list of size 12 (sample size) of ones and zeros. This list is populated using the **sample** function which chooses from the vector we give it in *x* with the probability of each elements being chosen as defined by the *prob* parameter in the function. Said another way: the function will choose *1* with a probability of *null_prob* and *0* with the probability of *1-null_prob*. Finally, we need to sample *with replacement* otherwise you can't get twelve numbers out of a list of two.

We calculate the proportion of scissors (ones) in our vector by summing the elements of the vector (again, a bunch of ones and zeros) and dividing by the sample size. Finally, we take this *trial_proportion* and "bind" it to the end of our big list of simulated proportions. What we are left with is 1000 proportions that are simulated under the assumption that the probability of picking scissors is 1/3... the null hypothesis. We can visualize the distribution of these numbers using a histogram.

```r
#Calculate how far away from the null proportion
# our sample proportion was.
dist_from_null = abs(sample_stat - null_prop)

#Value above the null proportion
above = null_prop + dist_from_null

#Value below the null proportion
below = null_prop - dist_from_null

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



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

```
##   pvalue
## 1  0.439
```

One-proportion z-test:

Here is some example code for conducting a one-proportion z-test in *R*. I will continue with the "Halloween Treats" example from your textbook. Before we can use a theory-based test, we need to check the validity conditions: with this sample we clearly have at least 10 successes and 10 failures.

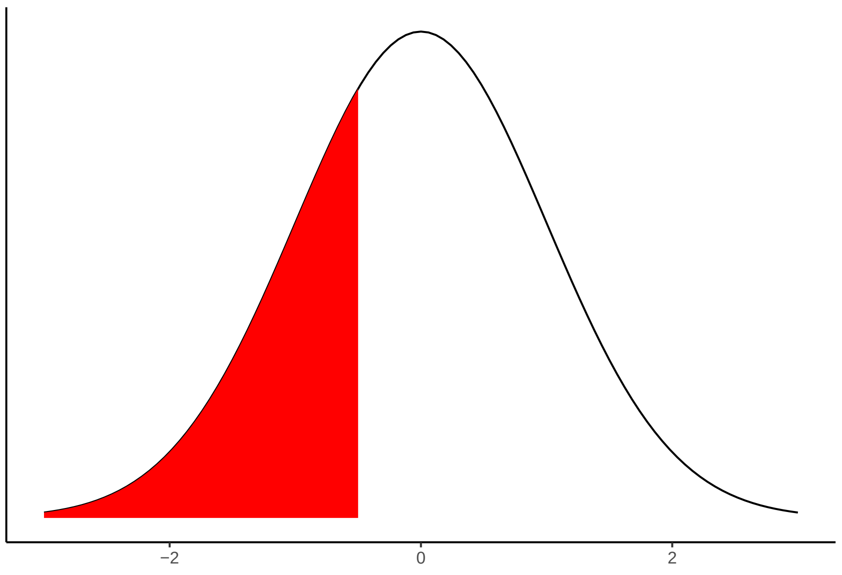
```r
sample_proportion = 148/(135+148)

std_null = sqrt(0.5 * (1-0.5) / 283)
```

```r
z = (sample_proportion - 0.5) / std_null

p = 2 * (1 - pnorm(abs(z)))

p
```

```
## [1] 0.4396586
```

Hopefully, the first three steps here are pretty clear from your reading in your book. The last equation (to get the p-value) may be a little new because of the **pnorm** function. Please spend some time experimenting with this function because it is going to become very useful to you in this class. You can see this function as saying "give me an x and I'll give you an area under the normal curve from negative infinity to that x." Now maybe you think I'm crazy because functions talk to me but that's ok as long as you remember that. Here are a couple examples of using the **pnorm** function. Note: You won't be responsible for knowing all functions below but I'm using a few new ones to illustrate my point.

```r
ggplot(data.frame(x = c(-3, 3)), aes(x = x)) +
  stat_function(fun = dnorm,
                args = list(mean = 0, sd = 1)) +
  stat_function(fun = dnorm,
                args = list(mean = 0, sd = 1),
                xlim = c(-3, -0.5), #This is (-3, 0) for the purposes of plotting.
                geom = "area", fill = "red") +
  labs(x = "", y = "") + theme_classic() +
  scale_y_continuous(breaks = NULL)
```

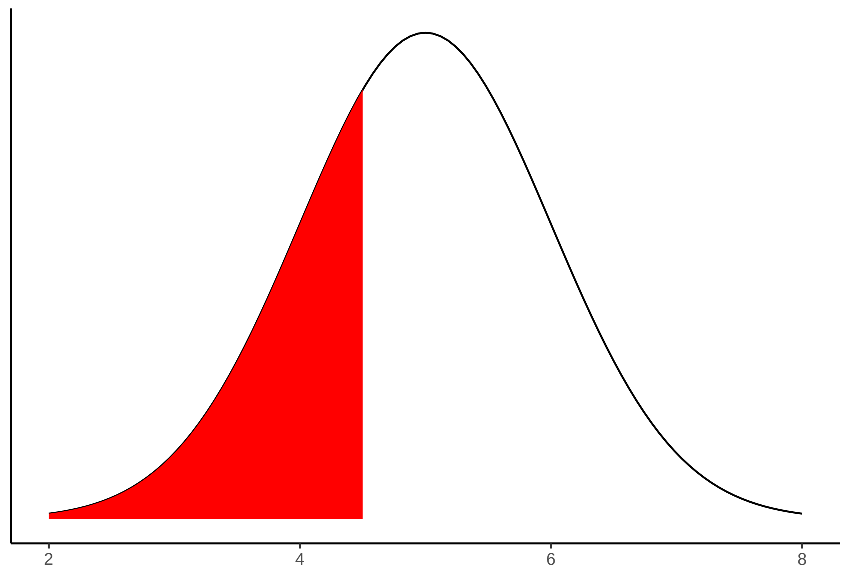


```r
pnorm(-0.5)
```

```
## [1] 0.3085375
```

In this case, **pnorm** gives us the area of the region shaded in red above. When we use **pnorm** without any other parameters, it defaults to the *standard normal distribution* which has a mean of 0 and a standard deviation of 1. One of the great things about *R*, however, is that we don't have to "standardize" our sample statistics to calculate our areas. We can also do something like the following two examples if one or both of our parameters are different than the standard normal.

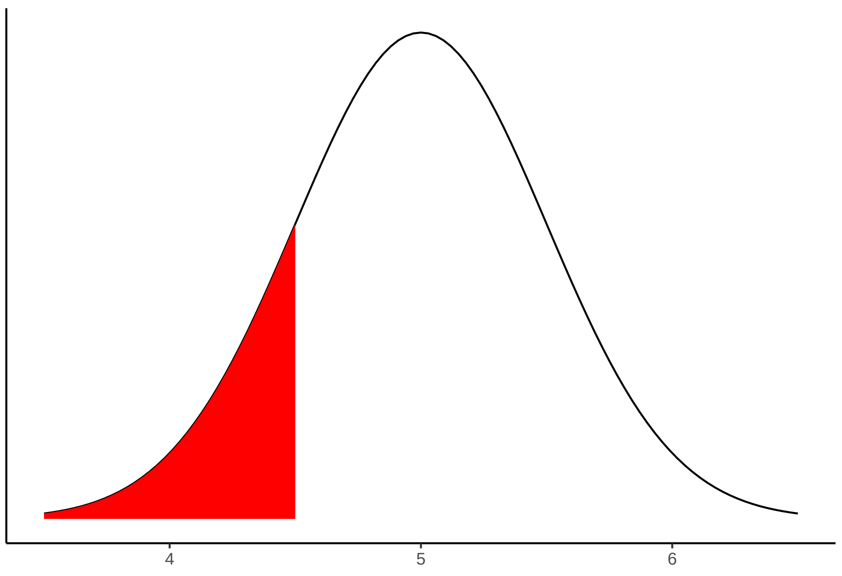
```r
ggplot(data.frame(x = c(2, 8)), aes(x = x)) +
  stat_function(fun = dnorm,
                args = list(mean = 5, sd = 1)) +
  stat_function(fun = dnorm,
                args = list(mean = 5, sd = 1),
                xlim = c(2, 4.5),
                geom = "area", fill = "red") +
  labs(x = "", y = "") + theme_classic() +
  scale_y_continuous(breaks = NULL)
```



```r
pnorm(4.5, mean = 5, sd = 1)
```

```
## [1] 0.3085375
```

```r
ggplot(data.frame(x = c(3.5, 6.5)), aes(x = x)) +
  stat_function(fun = dnorm,
                args = list(mean = 5, sd = 0.5)) +
  stat_function(fun = dnorm,
                args = list(mean = 5, sd = 0.5),
                xlim = c(3.5, 4.5),
                geom = "area", fill = "red") +
  labs(x = "", y = "") + theme_classic() +
  scale_y_continuous(breaks = NULL)
```



```r
pnorm(4.5, mean = 5, sd = 0.5)
```

```
## [1] 0.1586553
```

Why can we use the theory-based test?:

In using the theory-based test we are relying on the central limit theorem (CLT) that tells us: "as long as our sample size is big enough, our sample proportions will be approximately normally distributed." So we have to be cognizant of our sample size to use the CLT… sound familiar? Hopefully you recognize that our validity conditions for the theory-based test help satisfy our sample size requirement.

More specifically, the central limit theorem tells us that our sample proportions will be distributed normally with a mean of π (the population proportion) and a standard deviation of $\sqrt{\frac{\pi \times (1-\pi)}{n}}$. Let's take a look at the statistics for our simulated null distribution above:

```r
#Mean of simulated null distribution
mean(replications_dataframe$trial_stat)
```

```
## [1] 0.4998975
```

```r
#Standard deviation of simulated null distribution
sd(replications_dataframe$trial_stat)
```

```
## [1] 0.02940567
```

```r
#Mean dictated by the CLT
null_prop
```

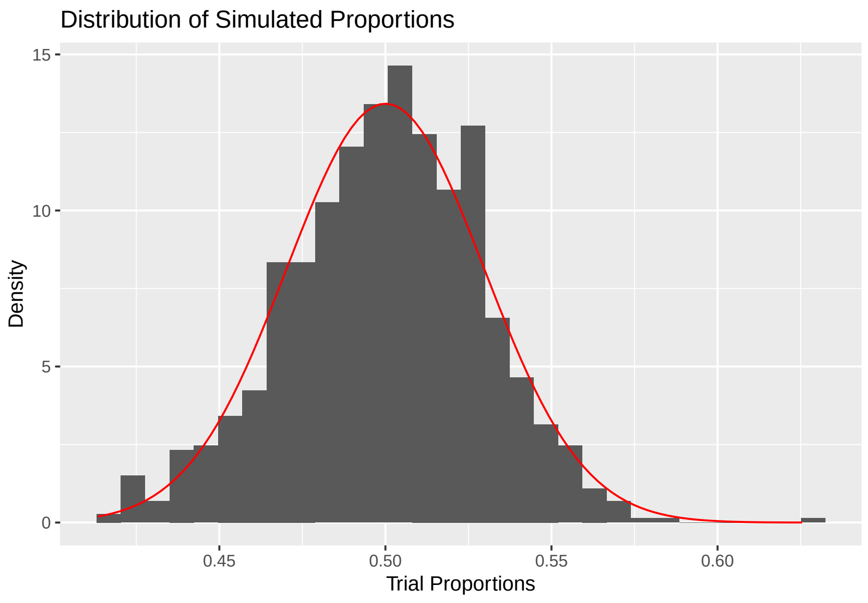
```
## [1] 0.5
```

```r
#Standard debiation dictated by the CLT
sqrt((null_prop * (1 - null_prop)) / sample_size)
```

```
## [1] 0.02972191
```

The values from our simulated null distribution and those dictated by the CLT are pretty close. Hopefully this enough to convince of the magic of the central limit theorem but I'm going to overlay the normal distribution on the simulated null distribution as a parting *R* flex.

```
#Standard debiation dictated by the CLT
CLT_sd = sqrt((null_prop * (1 - null_prop)) / sample_size)

replications_dataframe %>%
  ggplot(aes(x = trial_stat)) +
  geom_histogram(aes(y = ..density..)) +
  labs(x = "Trial Proportions", y = "Density", title = "Distribution of Simulated Proportions") +
  stat_function(fun = dnorm,
                args = c(mean = null_prop, sd = CLT_sd),
                color = "red")
```


Distribution of Simulated Proportions
Density
Trial Proportions