Introduction to R Short Course

Foundations of Statistical Inference (PLSC 500)

Special thank you to Hikaru Yamagishi and Kyle Peyton! These notes build on their previous hard work!

The goal of this section is to get you up an running in `R`. You will need to bring a laptop to all sections with `R` and RStudio installed. This is so that we can work through examples together. If this presents a problem for anyone please let us know asap!

R is an open-source statistical software language. While other statistical software packages such as Stata, SPSS, or even Excel can of course be used for statistical analysis, R has many advantages. First, (with apologies to Python - which is also highly recommended, but not for this class) it is the programming language of choice of many data scientists and statisticians, and by far the most popular programming language in political science. Second, it makes writing functions very easy. Third, there is a large community of developers who have contributed a huge number of add-ons for R that you will find invaluable. Finally, it's free, and always will be, which is not true of other software. In addition to R, please also download and install RStudio, the top-of-the-line script editor.

Note that there are various approaches to coding in `R`. This guide will use both base `R` and what is called the "tidyverse" or "dplyr," a collection of `R` packages designed for data science. All tidyverse packages share an underlying design philosophy, grammar, and data structures. Note that these are not the only ways to code in `R`, and I am not advocating any package. For example, some prefer the data manipulation library data.table, which has speed and memory efficiency advantages when working with unusually large datasets.

Step 1: R

Please go here https://cran.r-project.org/ to download the latest version of `R` and install it on your machine.

Step 2: RStudio

Please go here https://www.rstudio.com/products/rstudio/download/ and download the **free** desktop version on RStudio on your machine.

Step 3: Latex

If you want to compile this .Rmd document as a PDF then you need to get Latex installed on your machine. It will still compile as an html or word doc just fine without it. However, it is recommended that you get Latex up and running on your machine and generate PDFs for sharing your work, not word documents. You can still work in Markdown and RStudio, Latex is just needed for some behind the scenes stuff.

First, you need to install a Latex distribution from https://www.latex-project.org/. Next, follow the instructions below for your operating system. If you have problems with the instructions below, please reach out to us for help.

- **Windows**: download the "MiKTex" distribution available at https://www.latex-project.org/get/.
- **Mac**: download the "MacTeX" distribution available at https://www.latex-project.org/get/.

Step 4: Workflow

We highly recommend that you start using Markdown for homeworks. We promise it will make your life easier in the long run. It can generate PDFs and Word documents. You can also write in Latex syntax. But the Markdown syntax if very straightforward. The integration with RStudio is seamless. There are many many introductions to Markdown online. Here are two useful reference documents:

- https://www.rstudio.com/wp-content/uploads/2016/03/rmarkdown-cheatsheet-2.0.pdf
- https://www.rstudio.com/wp-content/uploads/2015/03/rmarkdown-reference.pdf

This might seem intimidating, but getting started is super easy. Don't look at the references until you need them. Just dig in:

- Open RStudio
- File > New File > R Markdown ...
- Now you have created your first Markdown (.Rmd) file! Here is a concise and excellent introductory tutorial: http://r4ds.had.co.nz/r-markdown.html

This document was created in Markdown using some Latex syntax to create math. So all the code chunks you see in the next section are things that you can also do very easily. Have a look at the .Rmd file as well so that you can see the code that generates this PDF. It could also serve as a template for your homeworks if you want. Some of us use Latex syntax in Markdown as a matter of habit, but .Rmd files are very flexible, so you can use a combination of Markdown and Latex syntax.

Note that when writing code, we generally do not recommend starting in R Markdown. Instead, create a separate R Script and run your code there, then incorporate into R Markdown as necessary.

Write good code

Finally, be kind to your future self. Write good code. This will make your life easier, and the lives of people who have to read your code. It also signals to other people who might read your code that you are careful and well organized. There are many style guides out there. Have a look at this one: http://adv-r.had.co.nz/Style.html. The Google Style guide is also useful: https://google.github.io/styleguide/Rguide.xml.

One feature of good code is the use of comments. There is virtually no such thing as including too many comments to tell your readers/replicators (and your future self!!) what your code is doing in plain English. My (Trevor's) personal coding style involves the use of headings and subheadings. You will see examples of this below.

Some tips for first time coders:

- Delete or clean up obsolete code. First time coders are often tempted to keep adding to their code and trying new things, but this can quickly lead to a jumbled mess.

- When naming variables, vectors, lists, data frames, etc., use names that make sense! You will often be assigning names to dozens of objects, and you will want to know what each represents. For example, if you are naming regression models with different outcome variables, model1, model2, and model3 are NOT helpful names.

- Do not let your code run on-and-on on one line. R Studio has a useful feature where it will place a vertical line at 80 characters to encourage you not to run past this mark. You can turn this on by going to: File –> Preferences –> Code –> Display –> Show Margin.

- Clear your workspace before loading a new R Script. This can be done by restarting your R session.

How to get help

If you haven't seen `R` before, this might look scary. But learning by example is the best way to get started. This tweet https://twitter.com/kierisi/status/898534740051062785?s=03 is a good description of the basic

workflow in `R`:

1. Install R
2. Install RStudio
3. Google "How do I [THING I WANT TO DO] in R?"
4. Repeat 3 as necessary.

A general rule of thumb is to spend 5-10 minutes Googling before asking for help from a human being. Stack Exchange is your friend.

Useful resources

There are many good introductions to `R` on the internet. Some of my favorite books (in order of difficulty) are,

1. R for Data Science (https://r4ds.had.co.nz/). A problem oriented introduction to data analysis + R via Hadley Wickham' universe of awesome packages.
2. Introduction to Probability and Statistics Using R (https://cran.r-project.org/web/packages/IPSUR/vignettes/IPSUR.pdf). This is a very comprehensive (free) introduction. See Chapter 2 for the basics.
3. The Art of R Programming (https://www.nostarch.com/artofr.htm). I think this is available through Yale Library as an e-book. It's a very useful reference to have.
4. The R Infero (http://www.burns-stat.com/pages/Tutor/R/_inferno.pdf). Author's description: "If you are using R and you think you're in hell, this is a map for you."
5. Advanced R (http://adv-r.had.co.nz/). This is a free resource from Hadley Wickham. It's a very good reference for advanced topics.

There are also several good websites to check out for help. For example:

- https://stackoverflow.com/questions/tagged/r
- https://www.r-bloggers.com/
- https://r-graph-gallery.com (for data visualization)

Practice Activities

Activity 1

Write a sentence about what you hope to gain from this course that uses *italics* and **bold**.

Activity 2

Make an unordered list of things using bullet points.

- This
- Is
- a
- list
- of things.

Activity 3

Make an ordered list of things using numbers.

1. This
2. is
3. number
4. four?

Activity 4

Write the following equation in math mode:

$$y = \alpha + \beta X + \gamma Z$$

Activity 5

Load the csv from http://hdl.handle.net/10079/989c3e0c-4d31-47e5-b461-947ef6b54e1a.

```
# Read in csv directly from URL
toy_df = read.csv("http://hdl.handle.net/10079/989c3e0c-4d31-47e5-b461-947ef6b54e1a")
```

You can also read in files from a directory on your computer. When working with local files, it is recommended that you create an Rstudio Project file first. An Rstudio project sets up its own working directory, making it easier to manage filepaths and collaborate across users (e.g., if working out of a Git repo or a dropbox folder).

Activity 6-7

Install the haven package. Load the Stata .dta from http://hdl.handle.net/10079/1g1jx43 using your favorite method.

```
# First need to install this package if you don't have it already:
# install.packages("haven")

# Load the package and read in a Stata dataset using Method 3 from above.
# Note you can use ?read_dta() to learn more about this function.
library(haven)
#toy_df = read_dta("http://hdl.handle.net/10079/1g1jx43")
```

Activity 8

There are many datasets in base R. Let's load the chickens dataset and do some basic stuff.

```
# Load dataset
data("chickwts")

# What is this thing?
?chickwts
```

In R a dataset is a "data frame". R also understands matrices; but data frames have the useful property that they can be a combination of character strings and numerics.

```
# ------------------------------------------------------------------------
# CREATE A DATAFRAME OF CATS AND RANK THEIR SIZE ----
# ------------------------------------------------------------------------

# Assign names of three types of domestic cats to a list
cats = c("Ragdoll", "Siberian", "Main Coon")

# Add a vector of the ranking of their adult sizes
rank = c(2, 3, 1)

# Combine the vectors/lists into a dataframe
cat_df = data.frame(cats = cats, rank = rank)

# Print the dataframe for viewing
cat_df
```

```
##        cats rank
## 1   Ragdoll    2
## 2  Siberian    3
## 3 Main Coon    1
```

The `class()` function tells you what class (e.g. data.frame, matrix) an object belongs to:

```
class(chickwts)
```

```
## [1] "data.frame"
```

```
class(cat_df)
```

```
## [1] "data.frame"
```

The function `head()` is useful for having a peek at the top of a data frame. If you want to look at the entire data frame in spreadsheet format, you can used the `View()` function. This is not recommended for "large" data frame. Even something with ~40k rows and 100 columns will give you trouble on most machines.

```
# Take a look at the first 6 rows. You can pass the argument n = 10 to print
# the first 10 rows. Check out ?head() to see what arguments the function takes.
head(chickwts)
```

```
##   weight      feed
## 1    179 horsebean
## 2    160 horsebean
## 3    136 horsebean
## 4    227 horsebean
## 5    217 horsebean
## 6    168 horsebean
```

`nrow()` is another basic function that comes in handy. As the name suggests, it tells you how many rows are in a data frame.

```
# How many observations are in here?
nrow(chickwts)
```

```
## [1] 71
```

Activity 9

Compute some summary statistics. The `summary()` function is useful for datasets that do not have too many columns.

```
# Compute summary statistics for each column
summary(chickwts)
```

```
##      weight             feed
##  Min.   :108.0   casein   :12
##  1st Qu.:204.5   horsebean:10
##  Median :258.0   linseed  :12
##  Mean   :261.3   meatmeal :11
##  3rd Qu.:323.5   soybean  :14
##  Max.   :423.0   sunflower:12
```

You can also access variables from within the dataset using the money sign:

```
# Calculate the variance of chicken weights
var(chickwts$weight)
```

```
## [1] 6095.503
```

```
# Calculate the standard deviation
sd(chickwts$weight)
```

```
## [1] 78.0737
```

```
sqrt(var(chickwts$weight))
```

```
## [1] 78.0737
```

The feed type is stored as a factor. This is an important R object. It's like a character string, but it has an ordering to it. By default the ordering is alphabetical.

```
# Feed type is a factor:
class(chickwts$feed)
```

```
## [1] "factor"
```

Let's have a closer look at this variable:

```
# Tabulate the feed types
table(chickwts$feed)
```

```
##
##    casein horsebean   linseed  meatmeal   soybean sunflower
##        12        10        12        11        14        12
```

```
# It has six levels
levels(chickwts$feed)
```

```
## [1] "casein"    "horsebean" "linseed"   "meatmeal"  "soybean"   "sunflower"
```

The six levels tell us how the factor is ordered. We can convert factors to numerics or characters. We cannot convert a character to a numeric, however. This is because it does not have a natural ordering, unless you tell R to give it one.

```
# Convert to character, and store this new variable inside the dataset as
# a new column
chickwts$feed_char = as.character(chickwts$feed)
class(chickwts$feed_char)
```

```
## [1] "character"
```

```
# This will produce an error because factors don't have an implicit ordering:
as.numeric(chickwts$feed_char)
```

```
## Warning: NAs introduced by coercion
```

```
##  [1] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
## [26] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
## [51] NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
```

Note the ordering when we convert the factor to a numeric, "c" comes before "h" in the alphabet, so "casein" gets an 1 and "horsebean" gets a 2, etc.

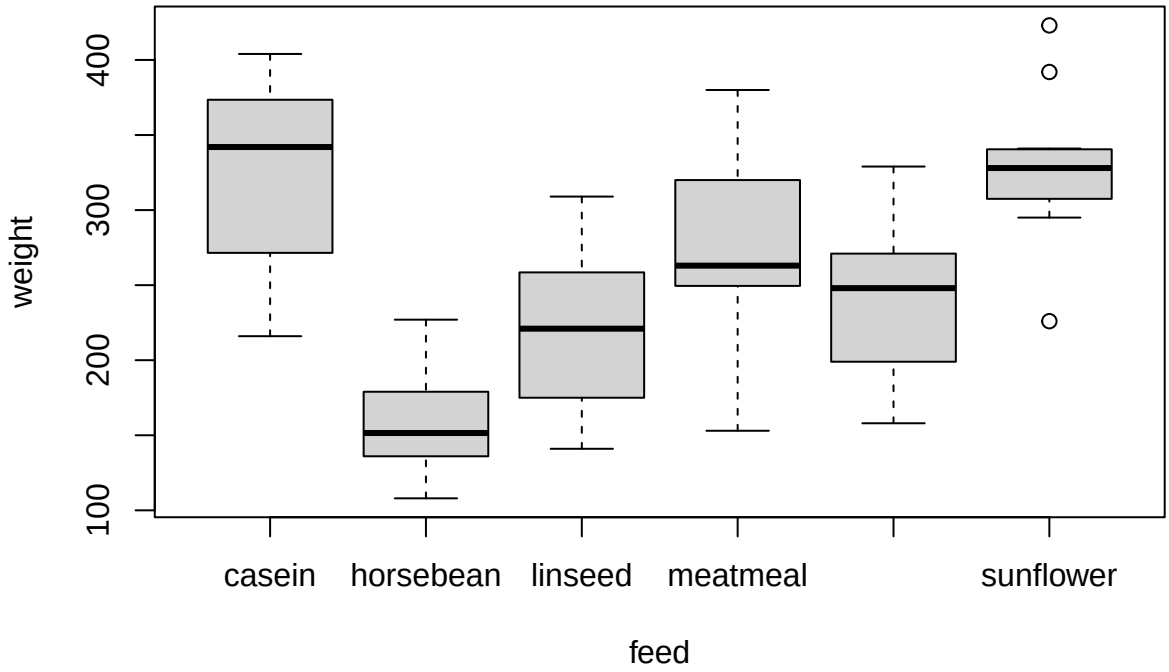
```
# Convert factor to numeric, and cross-tabulate the two vectors
table(as.numeric(chickwts$feed), chickwts$feed_char)
```

```
##
##      casein horsebean linseed meatmeal soybean sunflower
##   1     12         0       0        0       0         0
##   2      0        10       0        0       0         0
##   3      0         0      12        0       0         0
##   4      0         0       0       11       0         0
##   5      0         0       0        0      14         0
##   6      0         0       0        0       0        12
```

Activity 10

Make a plot in base R. Let's make a boxplot using the "formula" notation. This takes a numeric outcome on the left hand side (displayed on vertical axis in the plot) and a factor on the right hand side (displayed on horizontal axis in the plot)

```
# Make a boxplot; see ?boxplot for more info
boxplot(weight ~ feed, data = chickwts)
```



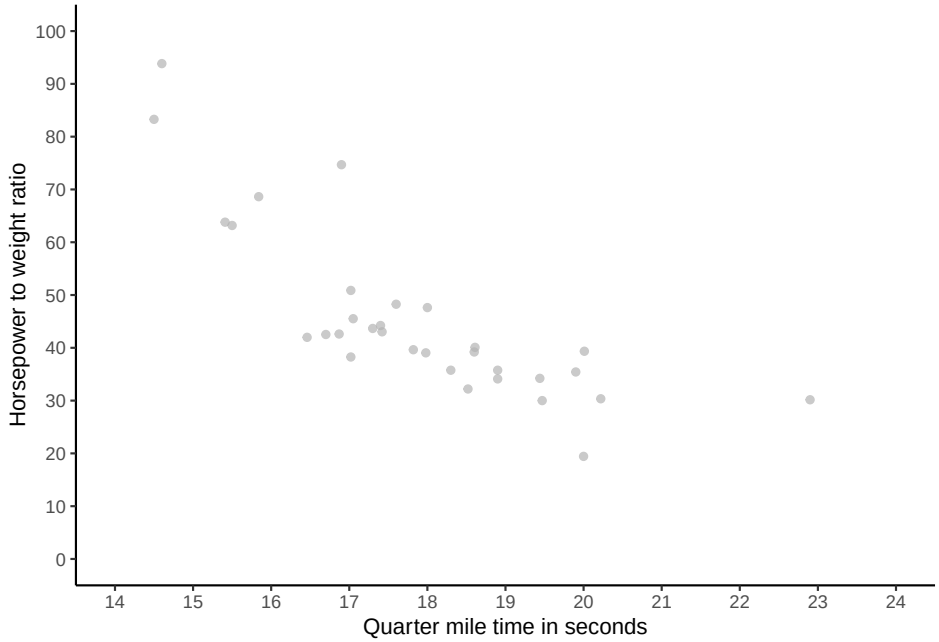
Activity 11

Create a plot using ggplot2, arguably the best and most flexible data visualization platform in existence. Let's create a simple scatterplot using the "mtcars" dataset. This is a built-in `R` data set commonly used for practice. It is from the 1974 Motor Trend magazine, and contains variables for aspects of automobile design and performance for 32 automobiles. Three of these variables are: horsepower, weight, and quarter mile time. Let's say we are interested in viewing the relationship between horsepower/weight and quarter mile time.

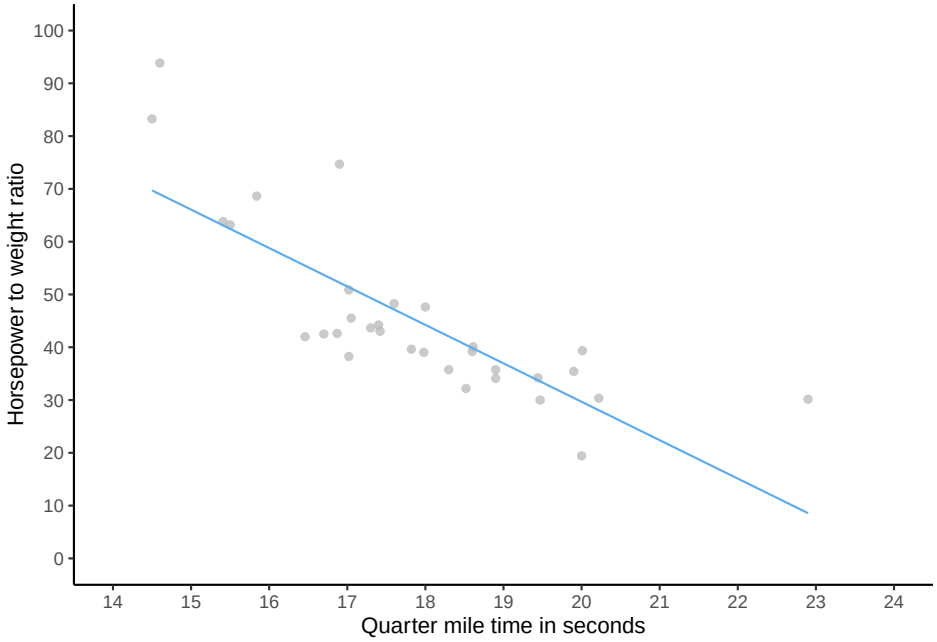
Note that I am showing a contained `R` script below to give you an idea of commenting and workflow from start to finish.

```
# ------------------------------------------------------------------------
# LIBRARIES AND IMPORT ----
# ------------------------------------------------------------------------
```

```r
# Load libraries
library(tidyverse)

# Load dataset
data("mtcars")

# ------------------------------------------------------------------------------
# PLOT THE RELATIONSHIP BETWEEN QUARTER MILE TIME AND POWER/WEIGHT RATIO ----
# ------------------------------------------------------------------------------

# Create new variable of power to weight ratio (using base R)
mtcars$hp_wt = mtcars$hp/mtcars$wt

# Or using tidyverse syntax
mtcars <- mtcars %>% mutate(hp_wt = hp/wt)

# Create scatterplot of hp/weight ratio to quarter mile time
hp_wt_plot =
  ggplot(mtcars, aes(qsec, hp_wt)) +
  geom_point(color = "grey70", alpha = 0.7, size = 1.5) +
  ylab("Horsepower to weight ratio") +
  xlab("Quarter mile time in seconds") +
  scale_x_continuous(limits = c(14, 24), breaks=seq(14, 24, 1)) +
  scale_y_continuous(limits = c(0, 100), breaks=seq(0, 100, 10)) +
  theme_classic()

# Display plot
hp_wt_plot
```




```r
# You can add additional features to or modify a saved plot
# Example: Add regression line to plot
hp_wt_plot +
  geom_smooth(method = lm, size = 0.5, color = "steelblue2", se = FALSE)
```



Activity 12

Using the tools described in Chapter 12 of R for Data Science, manipulate the titled table2, table3, and table4a + table4b so that they are the same as table1. You can access them when you load the tidyverse, as in:

```
library(tidyverse)
table1
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Now let's save the tables we will need and assign names to the tables.

```
# Load tables and save them using more useful names
tb_df = table1
tb_long_2 = table2
tb_rate_3 = table3
tb_count_4a = table4a
tb_pop_4b = table4b
```

Next, let's make table2 match table1. Table2 is currently in "long" format (see below), and we need to reshape it to "wide" in order for it to match table1.

Table 1 (wide)

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Table 2 (long)

```
## # A tibble: 12 x 4
##    country      year type            count
##    <chr>       <int> <chr>           <int>
##  1 Afghanistan  1999 cases             745
##  2 Afghanistan  1999 population   19987071
##  3 Afghanistan  2000 cases            2666
##  4 Afghanistan  2000 population   20595360
##  5 Brazil       1999 cases           37737
##  6 Brazil       1999 population  172006362
##  7 Brazil       2000 cases           80488
##  8 Brazil       2000 population  174504898
##  9 China        1999 cases          212258
## 10 China        1999 population 1272915272
## 11 China        2000 cases          213766
## 12 China        2000 population 1280428583
```

This method of reshaping data uses the pivot_longer and pivot_wider syntax from dplyr. There are other packages that perform similar operations, such as the reshape2 package. Feel free to use whichever method feels more intuitive to you.

```
# Table 2 reshape to wide ----------------------------
# Using syntax from dplyr / tidyverse
tidy2 <- pivot_wider(tb_long_2, names_from = type, values_from = count)
```

"tidy2" now matches table 1 above.

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Now let's look at table3. Table3 currently has the *cases* and *population* variables combined into a single variable called *rate*, with *cases* and *population* separated by a "/". We can use the dplyr "separate" command to split apart variables.

Table 3

```
## # A tibble: 6 x 3
##   country      year rate
##   <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

```
# Split apart rate variable into cases and population  -------------------------
# Here we specify separator the separator as "/"
# We can also convert the variable type to numeric in this step
tidy3 <- tb_rate_3 %>%
  separate(rate, into = c("cases", "population"), sep = "/", convert = TRUE)
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

Now it's time for table4a and table4b, which means we're going to have to merge some data.

These data frames are split apart by the cases and population variables, and in wide format. Let's start by converting the dataframes from wide to long.

Table 4a before

```
## # A tibble: 3 x 3
##   country     `1999` `2000`
##   <chr>        <int>  <int>
## 1 Afghanistan    745   2666
## 2 Brazil       37737  80488
## 3 China       212258 213766
```

Table 4b before

```
## # A tibble: 3 x 3
##   country         `1999`     `2000`
##   <chr>            <int>      <int>
## 1 Afghanistan   19987071   20595360
## 2 Brazil       172006362  174504898
## 3 China       1272915272 1280428583
```

```
# Reshape Tables 4a and 4b from wide to long -----------------------------------
# Reshape Table 4a to long
tidy4a <- pivot_longer(table4a, cols = c(`1999`, `2000`),
                       names_to = "year", values_to = "cases")

# Reshape Table 4b to long
tidy4b <- pivot_longer(table4b, cols = c(`1999`, `2000`),
                       names_to = "year", values_to = "population")
```

Table 4a after

```
## # A tibble: 6 x 3
##   country     year   cases
##   <chr>       <chr>  <int>
## 1 Afghanistan 1999     745
## 2 Afghanistan 2000    2666
## 3 Brazil      1999   37737
## 4 Brazil      2000   80488
## 5 China       1999  212258
## 6 China       2000  213766
```

Table 4b after

```
## # A tibble: 6 x 3
##   country     year   cases
##   <chr>       <chr>  <int>
## 1 Afghanistan 1999     745
## 2 Afghanistan 2000    2666
## 3 Brazil      1999   37737
## 4 Brazil      2000   80488
## 5 China       1999  212258
## 6 China       2000  213766
```

Now let's merge the dataframes together:

```r
# Merge tables 4a and 4b using a left join on country and year
tidy4 <- left_join(tidy4a, tidy4b, by = c("country", "year"))
```

Merged Table 4a and Table 4b

```
## # A tibble: 6 x 4
##   country     year   cases population
##   <chr>       <chr>  <int>      <int>
## 1 Afghanistan 1999     745   19987071
## 2 Afghanistan 2000    2666   20595360
## 3 Brazil      1999   37737  172006362
## 4 Brazil      2000   80488  174504898
## 5 China       1999  212258 1272915272
## 6 China       2000  213766 1280428583
```

Now let's clean up the dataset so that it matches table1.

```r
# Sort Table4 so that it is ordered by country year
tidy4 <- arrange(tidy4, country, year)

# Convert the year variable to a numeric value
tidy4$year <- as.integer(as.character(tidy4$year))
```

Finally, let's confirm that we did everything correctly and all of our tables actually match table1.

```r
# Confirm that reshaped tables 2, 3, and 4 are identical to table 1
if((identical(table1, tidy2)) & (identical(table1, tidy3)) & (identical(table1, tidy4))){
  print("All dataframes are identical")
}
```

```
## [1] "All dataframes are identical"
```

Introduction to R Short Course Part 2

Foundations of Statistical Inference (PLSC500)

Contents

This document contains some additional exercises/examples. We are now moving beyond workflow, data manipulation, and data cleaning and into data analysis. Here we will show you some examples of random sampling, simulation, and hypothesis testing.

As usual, clear your environment. We will be using some functions that are included in the tidyverse, so let's load it now.

```
# Load libraries
library(tidyverse)
```

Activity 13: Simulations and loops - the birthday problem

Let's practice performing simulations using loops. Loops are a powerful tool that tell a computer to repeat an instruction until a specified condition is reached.

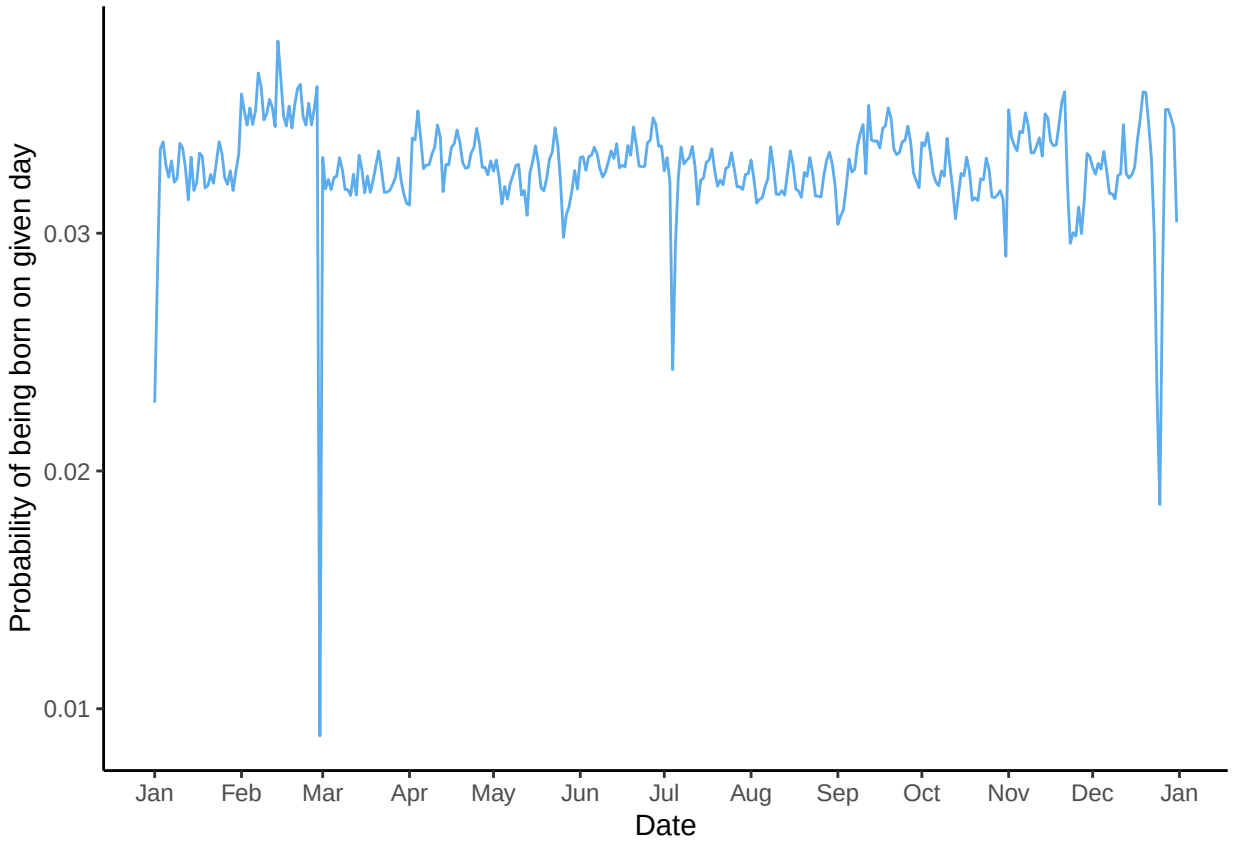
In this activity, we are going to use simulation to find the probability that two people in a room share a birthday. This is often referred to as the "birthday paradox."

We will use the real probability that a person is born on a given day to perform this simulation, so let's import that data first.

```r
# ------------------------------------------------------------------------------
# PREAMBLE, LIBRARIES, AND IMPORT ----
# ------------------------------------------------------------------------------

# Additional libraries
library(scales)

# Import real birthday data (full URL commented below)
# https://raw.githubusercontent.com/fivethirtyeight/
# data/master/births/US_births_2000-2014_SSA.csv

# Two ways to import the data: 1
bdays <- read.csv("https://raw.githubusercontent.com/fivethirtyeight/data/master/births/US_births_2000-

# Two ways to import the data: 2
urlRemote  = "https://raw.githubusercontent.com/"
pathGithub = "fivethirtyeight/data/master/births/"
fileName   = "US_births_2000-2014_SSA.csv"

bdays = paste0(urlRemote, pathGithub, fileName) %>% read.csv()
```

Next let's clean the data up a bit and visualize it in order to make sure everything makes sense.

```r
# ------------------------------------------------------------------------------
# CLEAN DATAFRAME, CALCULATE PROBABILITIES, AND VISUALIZE PROBABILITIES ----
# ------------------------------------------------------------------------------

# Calculate probability someone was born on each day of the year
bdays_months <- bdays %>%
  group_by(month, date_of_month) %>% # Group the data by month and day (no year)
  summarise(births = sum(births)) %>% # Sum births over years by month and day
  mutate(prob = births/sum(births)) # Create new variable

# Convert month and year variables to one date variable
bdays_months <- bdays_months %>%
  mutate(date = as.Date(paste(month, date_of_month, 2016, sep="-"), "%m-%d-%Y"))

# Create a plot of birthday probabilities throughout the year
ggplot(bdays_months, aes(x = date, y = prob)) +
  geom_line(color = "steelblue2") +
  xlab("Date") +
  ylab("Probability of being born on given day") +
  scale_x_date(date_breaks = "1 month", labels = date_format("%b")) +
  theme(axis.text.x=element_text(angle=90, hjust=1)) +
  theme_classic()
```



Now let's use simulation to estimate the probability of two people in class size x sharing a birthday!

Note that this involves the use of a "for loop." "For loops" repeat an instruction until a specified condition is reached - in this case until we have run a simulation our desired number of times (here 10,000 times). This can be done using the *seq_along* command in R, which tells R to "sequence along" a vector (or list), repeating an instruction for each element in the vector, and stopping when it reaches the end of the vector.

```
# ------------------------------------------------------------------------
# SIMILATE THE PROBABILITY OF TWO PEOPLE IN CLASS SIZE X SHARING A BIRTHDAY ----
# ------------------------------------------------------------------------

# Assign class size
class_size = 27

# Create blank vector in which we will sequence along and store results
output <- vector(mode = "integer", length = 10000)

# Pick class_size random days of the year with replacement
# (assuming equal probability) and save the number of shared birthdays
for (i in seq_along(output)) {

  # Randomly sample from birthdays with the probabilities we calculated
  bdays = sample(x = bdays_months$date, size = class_size,
                 replace = TRUE, prob = bdays_months$prob)
```

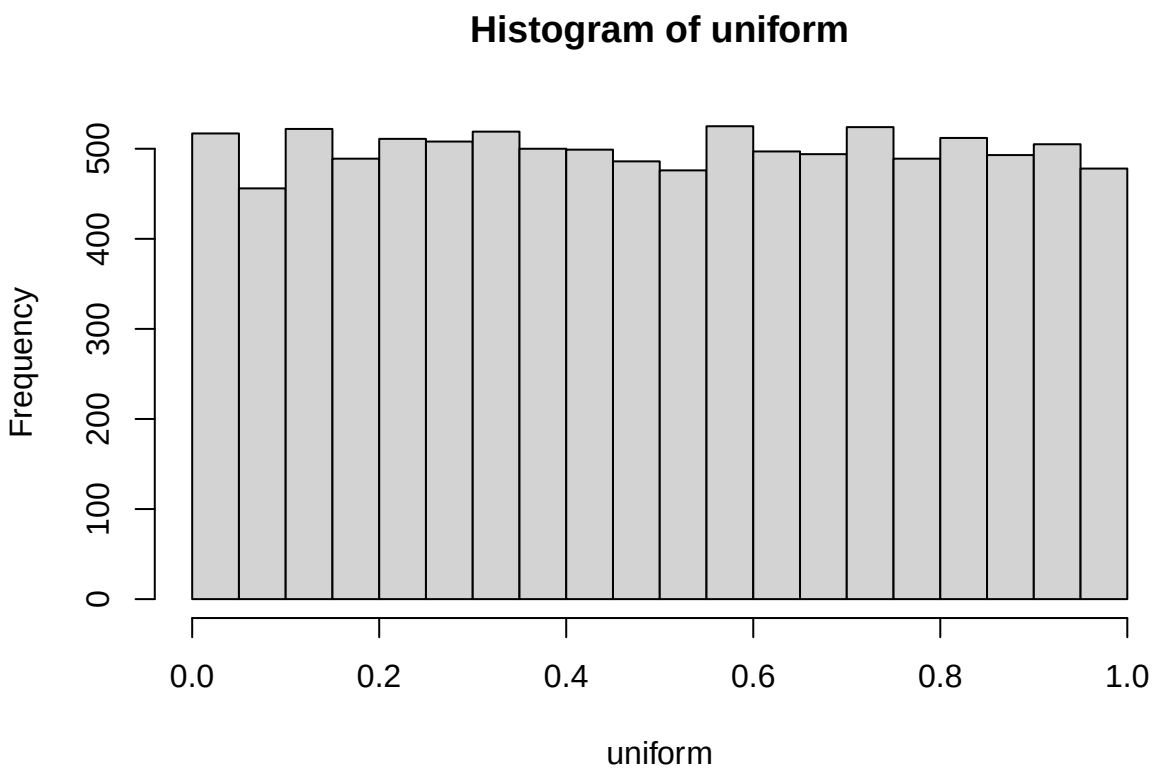
```r
  # Calculate the number of birthdays that appear more than once
  output[[i]] <- class_size - length(unique(bdays))
}

# Calculate the probability that at least two people share a birthday
prob_shared <- sum(output > 0)/length(output)
sprintf("The simulated probability that two people share a birthday is %1.2f%%",
        100*prob_shared)
```

```
## [1] "The simulated probability that two people share a birthday is 62.96%"
```

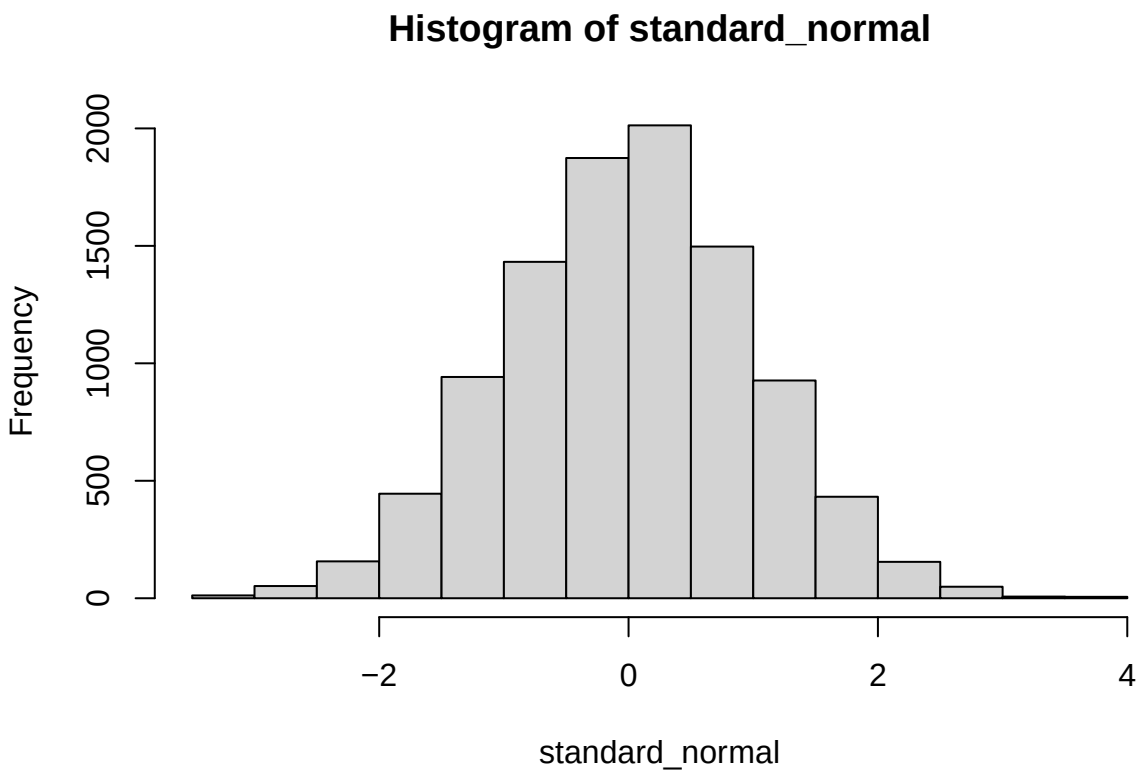
Activity 14: Creating distributions

R also has some useful functions for sampling from or creating various kinds of common distributions. Examples of how to use some of these functions are provided below.

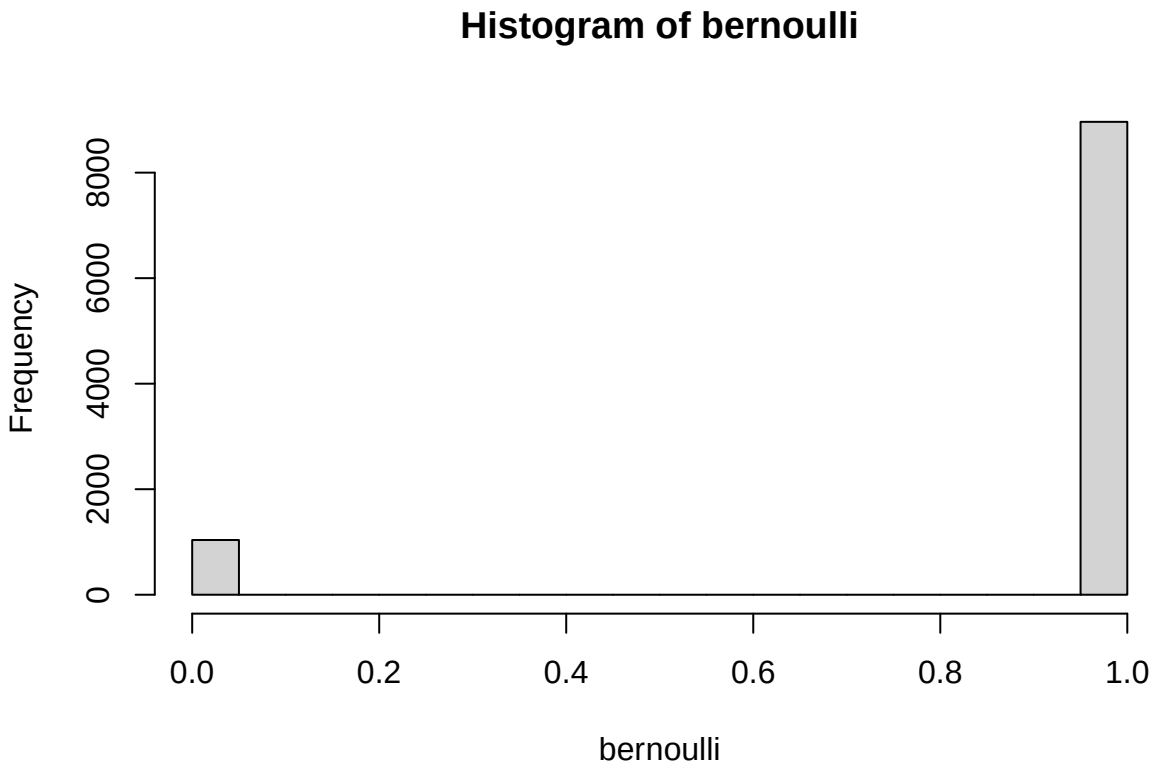
```
# ------------------------------------------------------------------------------
# USING R TO CREATE VECTORS OF DISTRIBUTIONS ----
# ------------------------------------------------------------------------------

# Set sample size (i.e. number of draws from each distribution)
n = 10000

# Create vectors of various distributions using R's built in functions
uniform = runif(n = n) # Uniform distribution
standard_normal = rnorm(n = n, mean = 0, sd = 1) # Standard normal distribution
bernoulli = as.numeric(rbernoulli(n = n, p = 0.9)) # Bernoulli distribution

# Let's plot these to make sure they look right
hist(uniform)
```




```
hist(standard_normal)
```




```
hist(bernoulli)
```



Activity 15: Randomly sampling from populations

However, what if we can't actually see the full underlying population distributions, but we can sample from it and we want to estimate the mean? Now let's randomly sample from one of the distributions above and compute the sample mean. In other words, let's "plug in" the sample mean for the population mean in order to estimate it.

```
# ------------------------------------------------------------------------
# RANDOMLY SAMPLE FROM A LARGER POPULATION/DISTRIBUTION ----
# ------------------------------------------------------------------------

# We're going to need another loop - let's set up the loop parameters
draws = 100 # How many draws will we take from the population each time?
sims = 10000 # How many random samples will we conduct?
means = vector(mode = "integer", length = sims) # Need a place to store means

# Randomly sample n draws from a standard normal distribution and take
# the mean sims times
for (i in 1:sims) {
  sample = sample(standard_normal, draws, replace = TRUE)
  means[i] = mean(sample)
}

# What is our estimated mean?
sn_mean = mean(means)
```

Are we confident that our estimate is accurate? Let's use various metrics for observing the amount of variability in our estimates. We will calculate the variance and the a normal-approximation based confidence interval. Since in this case we know the true population mean (this is not usually true!), we can also observe how close our sample mean is to the population.

```
# First let's calculate the variance of our sample mean
sn_mean_var = var(means)

# Now let's calculate the confidence interval
ci_lower = sn_mean - 1.96 * sqrt(sn_mean_var/n)
ci_upper = sn_mean + 1.96 * sqrt(sn_mean_var/n)

# In this case we know the actual population mean, so let's verify
sn_pop_mean = mean(standard_normal) # Is our estimate a good approximation?
```

Activity 16: Hypothesis testing

Let's do something a bit less abstract: catch the cheating gambler.

A gambler is playing a game that requires dice (I don't know enough about gambling to know which). The management of the casino you work at suspects she is using a loaded die, and has asked you to verify. Use your knowledge of hypothesis testing to be a good employee. You start by watching the suspected cheat cast 100 die rolls. The dataset "rolls" shows the outcomes you recorded. We'll start by importing this data.

```
# The dataset "rolls" shows the outcome of 100 die rolls from the gambler
rolls = read_csv("data/rolls.csv") # Import this dataset
```

Now let's calculate some summary statistics from this dataset. We'll see that the mean of the gambler's 100 die rolls is 3.89.

```
# Let's calculate some summary statistics from this dataset
die_mean = mean(rolls$Y)
die_var = var(rolls$Y)
```

What if we took the mean of a fair die rolled 100 times? We will use the sample function (with replacement) to simulate this process.

```
# What if we took the mean of a fair die rolled 100 times?
fair_mean = sample(x = c(1, 2, 3, 4, 5, 6),
                   size = 100,
                   replace = TRUE,
                   prob = c((1/6), (1/6), (1/6), (1/6), (1/6), (1/6)))

mean(fair_mean)
```

```
## [1] 3.57
```

Definitely different. But this still isn't enough evidence. What we actually want to know is the probability we would observe a mean as extreme or more than the gambler's mean (3.89) if we were to roll a fair die 100 times (what is this called?). Our "null hypothesis" is therefore that the gambler's die is a fair die, and that the mean of its 100 rolls is equal to that of a fair die. We will therefore test this "null hypthesis" by comparing the mean of the gambler's die (3.89) to a distribution of means from 10,000 means of 100 rolls of a fair die. Fortunately, we can use R to simulate this process so we don't actually have to roll a die one million times.

```
# Create an empty vector of length 10,000 to store our fair means
fair_means <- vector(mode = "integer", length = 10000) # Create empty vector

# Simulate 10,000 fair die rolls to get a distribution of means of 100 fair die rolls
for (i in seq_along(fair_means)) {
  fair_means[[i]] <- mean(sample(x = c(1, 2, 3, 4, 5, 6),
                           size = 100, replace = TRUE,
                           prob = c((1/6), (1/6), (1/6), (1/6), (1/6), (1/6))))
```

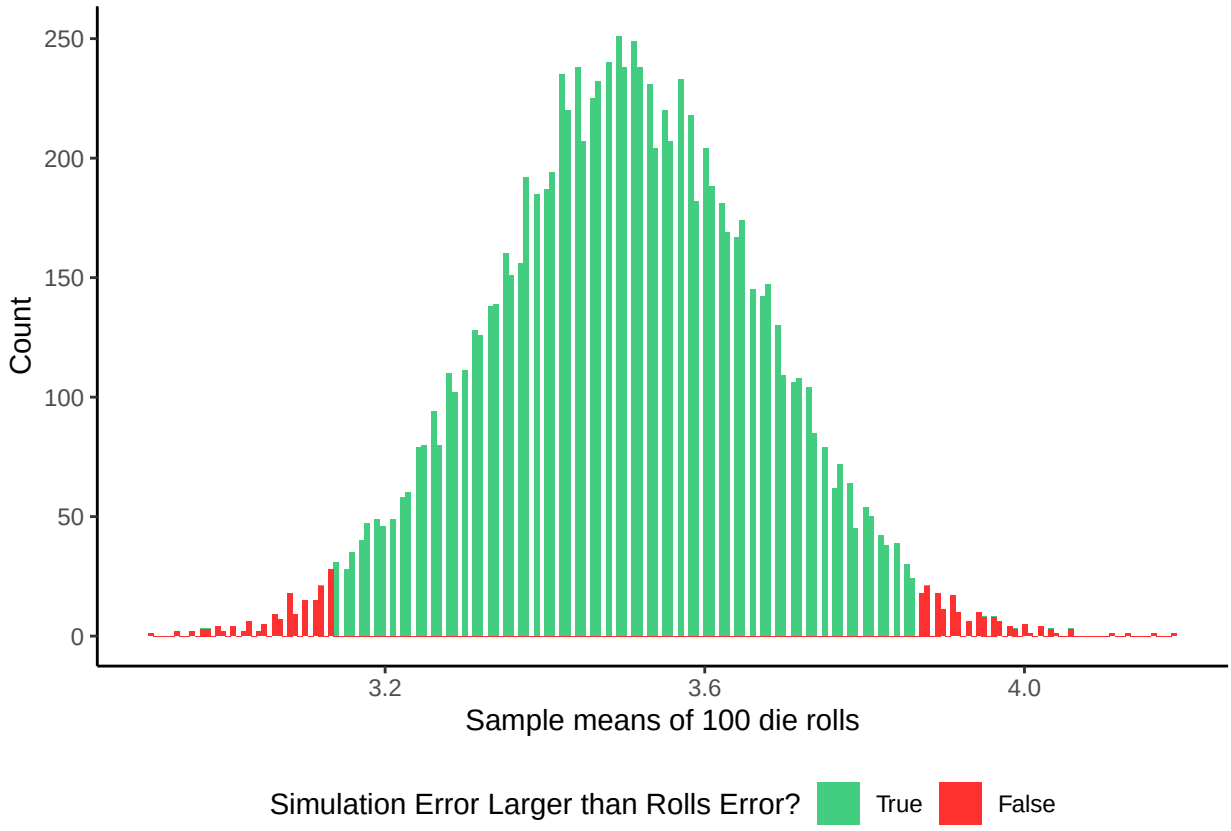
```
}

# Save our null distribution as a data frame
null_dist = as.data.frame(fair_means)

# Add variable for mean of null distribution
null_dist$null_mean = mean(null_dist$fair_means)

# Add variable for gambler's mean
null_dist$gambler_mean = mean(rolls$Y)

# Calculate the percentage of times we observe a result
# as extreme as the one that we saw in the null distribution
p = sum(with(null_dist, fair_means > mean(gambler_mean)))/nrow(null_dist)

p
```

```
## [1] 0.0143
```

The probability of observing a mean as extreme or more as the mean we would observe if the null hypothesis were true is 0.0143. This is known as a p-value.

We can also visualize where the gambler's mean falls on our null distribution (i.e. the distribution of means from 10,000 means of 100 rolls of a fair die). We will also color our plot to show the means of the simulated fair die rolls that have an error as a large or larger than the gambler's mean in absolute value.

```
# Find the proportion of simulated estimates from that have an error
# that is at least as large in absolute value
null_dist <- null_dist %>%
  mutate(
    gambler_error = gambler_mean - null_mean,
    fair_larger = (fair_means >= (null_mean + gambler_error) |
                   fair_means <= (null_mean - gambler_error))
    )

# Create histogram with vertical line showing estimate and coloring errors
# larger than rolls error in absolute value
ggplot(null_dist, aes(fair_means)) +
  geom_histogram(bins = 200, aes(fill = fair_larger)) +
  scale_fill_manual(values = c("seagreen3", "firebrick1"),
                    labels = c("True", "False"),
                    name = "Simulation Error Larger than Rolls Error?") +
  xlab("Sample means of 100 die rolls") +
  ylab("Count") +
  theme_classic() +
  theme(legend.position = "bottom")
```


250
200
150
100
50
0
Count
3.2
3.6
4.0
Sample means of 100 die rolls
Simulation Error Larger than Rolls Error?
True
False