Data Science for Political Science

2024-09-01

Section Contents

Course Notes 4

1 Introduction 5

1.1 What have I signed up for? 5
1.1.1 Data Science Can Help Social Scientists 6
1.1.2 Course Goals 8
1.2 Setup in R 9
1.3 Open R Script in RStudio 10
1.3.1 Using R as a Calculator 12
1.3.2 Working in an R Script 13
1.3.3 Saving the R Script 14
1.3.4 Annotating your R script 14
1.3.5 Running Commands in your R script 16
1.3.6 Objects 17
1.4 R Markdown 19
1.4.1 Getting started with RMarkdown 20
1.5 Assignment 1 25

2 Description 27

2.1 Process of Describing 27
2.1.1 Example Process 28
2.2 Summarizing univariate data 28
2.3 Functions to summarize univariate data 31
2.3.1 Step-by-step: Using functions in R (overview) 31
2.4 Loading data into R 33
2.4.1 Working with datasets in R 34
2.4.2 Measuring the Turnout in the US Elections 36
2.4.3 Getting to know your data 36
2.5 Comparing VEP and VAP turnout 38
2.5.1 Creating new variables in R 38
2.6 Comparing Presidential vs. Midterm turnout 40
2.6.1 R shortcut for writing vectors 42
2.7 Creating dataframes from within R 43
2.8 Wrapping Up Description 44
2.8.1 Summary of R tools 44

3 Causation with Experiments **47**
3.1 What separates causation from correlation? . . . 47
3.1.1 Potential Outcomes Framework . . . 48
3.1.2 Causal Effects . . . 50
3.1.3 Fundamental Problem of Causal Inference . . . 51
3.2 Randomized Controlled Trials . . . 51
3.2.1 Experiments: Why Randomize? . . . 52
3.2.2 Experiments: How to Analyze . . . 53
3.2.3 Ingredients of an Experiment . . . 53
3.3 Application: Is there racial discrimination in the labor market? . . . 54
3.3.1 Variable classes . . . 55
3.4 Making tables . . . 57
3.4.1 Crosstabulation . . . 57
3.5 Conditional Means . . . 57
3.6 Relational Operators in R . . . 58
3.7 Subsetting data in R . . . 59
3.7.1 Getting Boooooooolean . . . 60
3.8 Creating New Variables using Conditional statements . . . 60
3.8.1 ifelse statements . . . 62
3.9 Types of Experiments . . . 63
3.10 Wrapping Up Causation with Experiments . . . 64
3.10.1 Summary of R tools in this section . . . 64

Course Notes

This document will include important links and course notes for 01:790:391:01: Data Science for Political Science for the fall 2024 semester.

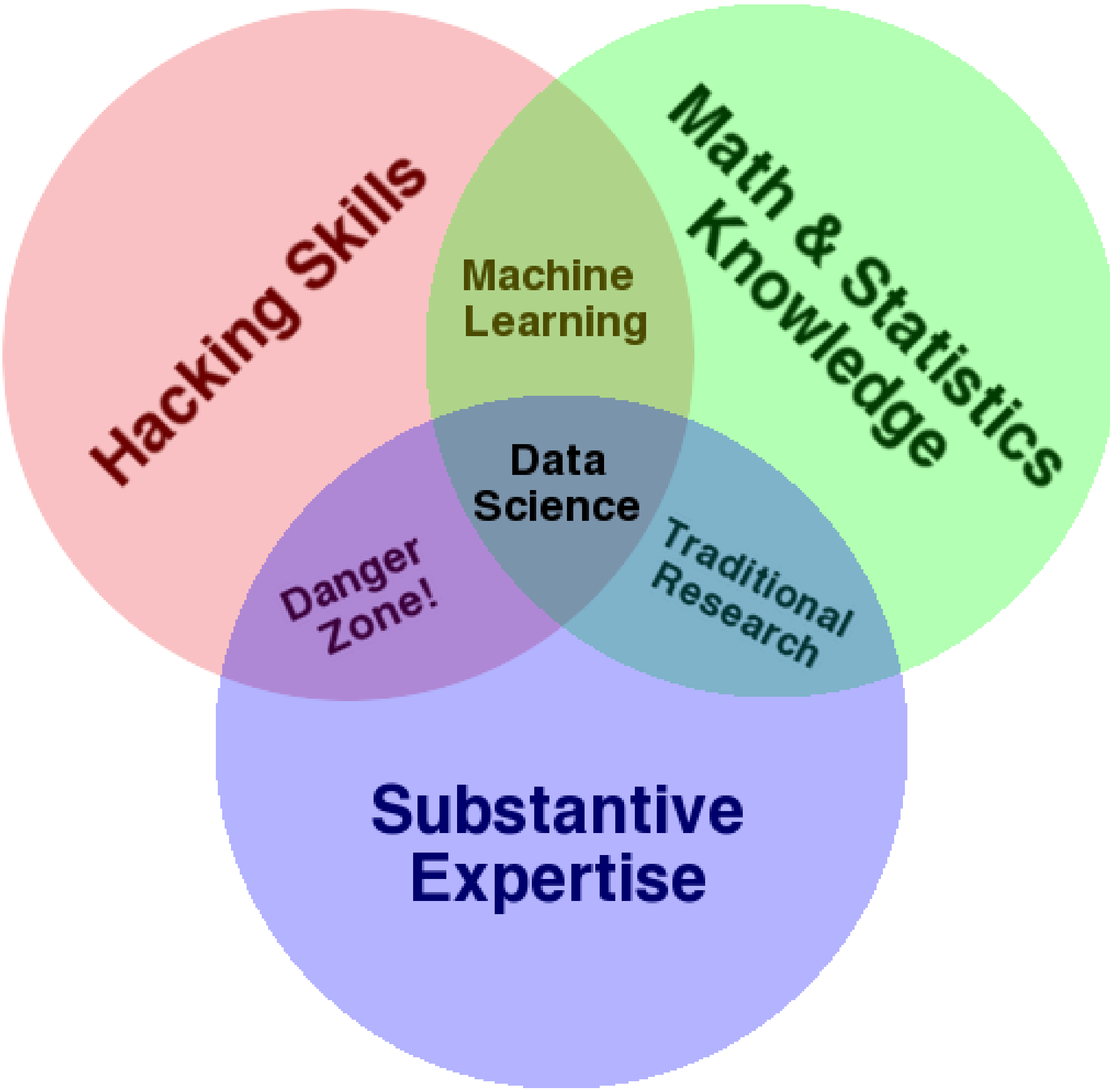
- This site will be updated throughout the semester with new content.
- The Canvas modules will provide links to the relevant sections to review for a given week of the course.
- The primary text for the course is Quantitative Social Science: An Introduction by Kosuke Imai. We will refer to this as QSS in the notes.
- This is a living document. If you spot errors or have questions or suggestions, please email me at k.mccabe@rutgers.edu or post to the course Canvas site.
- Occasionally the notes are updated with embedded video explainers of portions of the code in different sections.

1 Introduction

1.1 What have I signed up for?

First: What is Data Science?

- Data Science involves a combination of math/statistics and programming/coding skills, which, for our purposes, we will combine with social science knowledge.
  - Drew Conway has a nice venn diagram of how these different skill sets intersect.
  - Note: This course will not assume prior familiarity with data science in general or coding, specifically. For those brand new to data science, the idea of learning to code may seem intimidating, but anyone can succeed with a bit of patience and an open mind.



Next: What is political science?

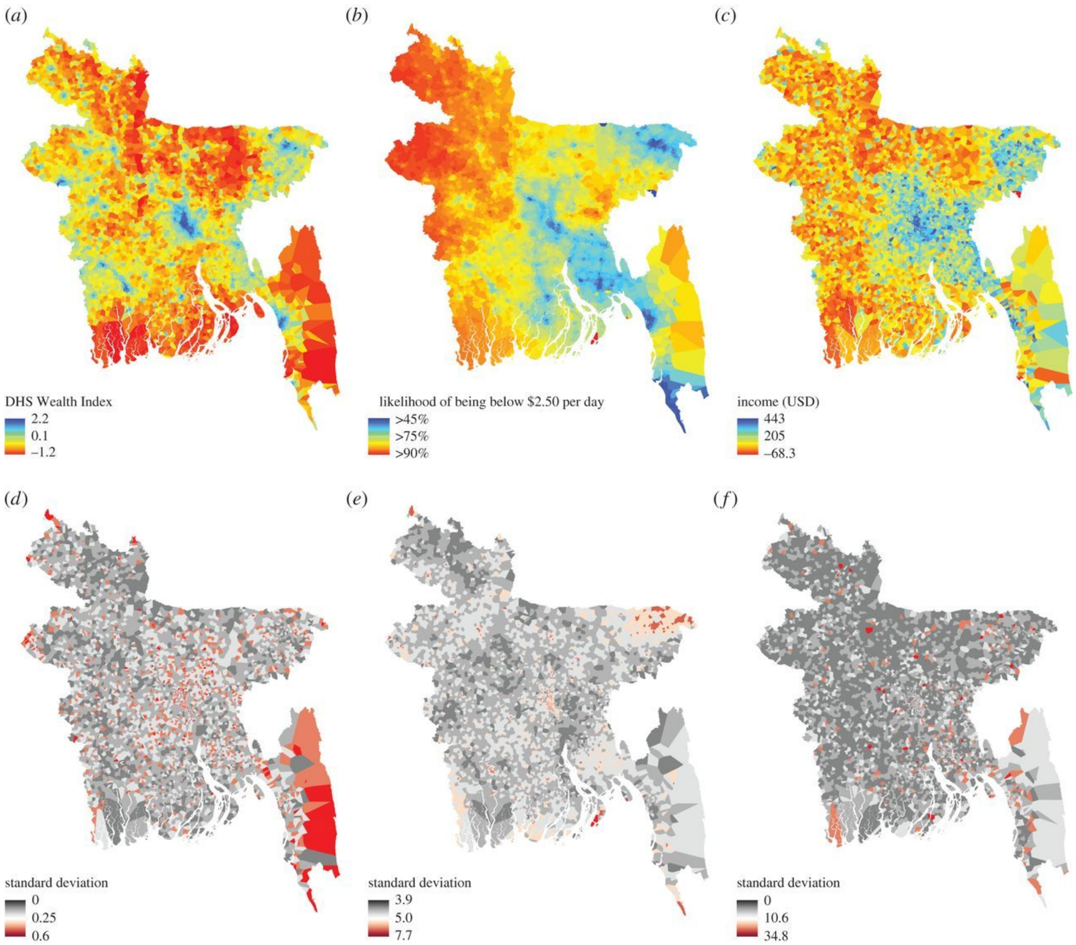
- The science of politics, of course! Politics focuses on studying governance and the distribution of power in society, broadly conceived.
  - How else might you define politics and political science? What do we study in political science?

1.1.1 Data Science Can Help Social Scientists

Example: Mapping poverty using mobile phone and satellite data

Researchers used modern data sources, including mobile phone data, as a way to ***predict*** and ***describe*** poverty in different geographic regions. These tools helped social scientists come up with methods that are much more cost-effective and efficient, but still as accurate as traditional methods for this type of measurement.

- How might measures of global poverty be useful to political scientists?



Steele et al. 2017: "Poverty is one of the most important determinants of adverse health outcomes globally, a major cause of societal instability and one of the largest causes of lost human potential. Traditional approaches to measuring and targeting poverty rely heavily on census data, which in most low- and middle-income countries (LMICs) are unavailable or out-of-date. Alternat emeasures are needed to complement and update estimates between censuses. This study demonstrates how public and private data sources that are commonly available for LMICs can be used to provide novel insight into the spatial distribution of poverty.We evaluate the relative value of modelling three traditional poverty measures using aggregate data from mobile operators and widely available geospatial data."

1.1.2 Course Goals

Social Science Goals

We have several goals in social science. Here are four that data science can help us pursue:

- **Describe** and measure
  - Has the U.S. population increased?
- **Explain**, evaluate, and recommend (study of causation)
  - Does expanding Medicaid improve health outcomes?
- **Predict**
  - Who will win the next election?
- **Discover**
  - How do policies diffuse across states?

What are other examples of these goals?

Note: In this course, we are exploiting the benefits of quantitative data to help achieve goals of social science. However, quantitative data have their shortcomings, too. We will also discuss the limitations of various applications of social science data, and we encourage you to always think critically about how we are using data.

This course will provide you with a taste of each of these social science goals, and how the use of data can help achieve these goals. By the end of the course, you should be able to

- Provide examples of how quantitative data may be used to help answer social science research questions.
- Compare and contrast the goals of description, causation, prediction, and discovery in social science research.
- Use the programming language R to import and explore social science data and conduct basic statistical analyses.
- Interpret and describe visual displays of social science data, such as graphs and maps.
- Develop your own analyses and visualizations to understand social science phenomena.

If you are someone that loves data, we hope you will find this course engaging. If you are someone who loathes or finds the idea of working with data and statistics alarming, we hope you keep an open mind. We will meet you where you are. This course will not assume knowledge of statistical software, and there will be plenty of opportunities to ask questions and seek help from classmates and the instructor throughout the semester.

The first section of course will walk people through how to use the statistical program– R– that we will employ this semester.

Will this course help me in the future?

Even if you do not plan on becoming a social scientist or a data scientist, an introduction to these skills may prove helpful throughout your academic and professional careers.

- To become an informed consumer of news articles and research involving quantitative analyses.
- To practice analytical thinking to make informed arguments and decisions.
- To expand your toolkit for getting a job that may involve consuming or performing some data analysis, even if that is not the traditional role.
  - Example: Journalism- How 5 Data Dynamos Do Their Jobs

1.2 Setup in R

Goal

By the end of the first week of the course, you will want to have R and RStudio installed on your computer (both free), feel comfortable using R as a calculator, and making documents using the R Markdown file type within RStudio.

R is an application that processes the R programming language. RStudio is also an application, which serves as a user interface that makes working in R easier. We will primarily open and use RStudio to work with R.

In other classes, you may come across Stata, SPSS, Excel, or SAS, which are programs that also conduct data analysis. R has the advantage of being free and open-source. Even after you leave the university setting, you will be able to use R/RStudio for free. As an open-source program, it is very flexible, and a community of active R/RStudio users is constantly adding to and improving the program. You might also encounter the Python language at some point. R and Python have similarities, and learning R can also make learning Python easier down the road.

R and RStudio Installation

IMPORTANT: Note the 2 Steps. These are 2 separate applications you must install. Installing one without the other will not work for our purposes.

This content follows and reinforces section QSS 1.3 in our book. Additional resources are also linked below.

- This video from Professor Christopher Bail explains why many social scientists use R and describes the R and RStudio installation process. This involves
  1. Going to cran, select the link that matches your operating system, and then follow the installation instructions, and

2. Visiting RStudio and follow the download and installation instructions. R is the statistical software and programming language used for analysis. RStudio provides a convenient user interface for running R code.

https://www.youtube.com/watch?v=ulIv0NiVTs4

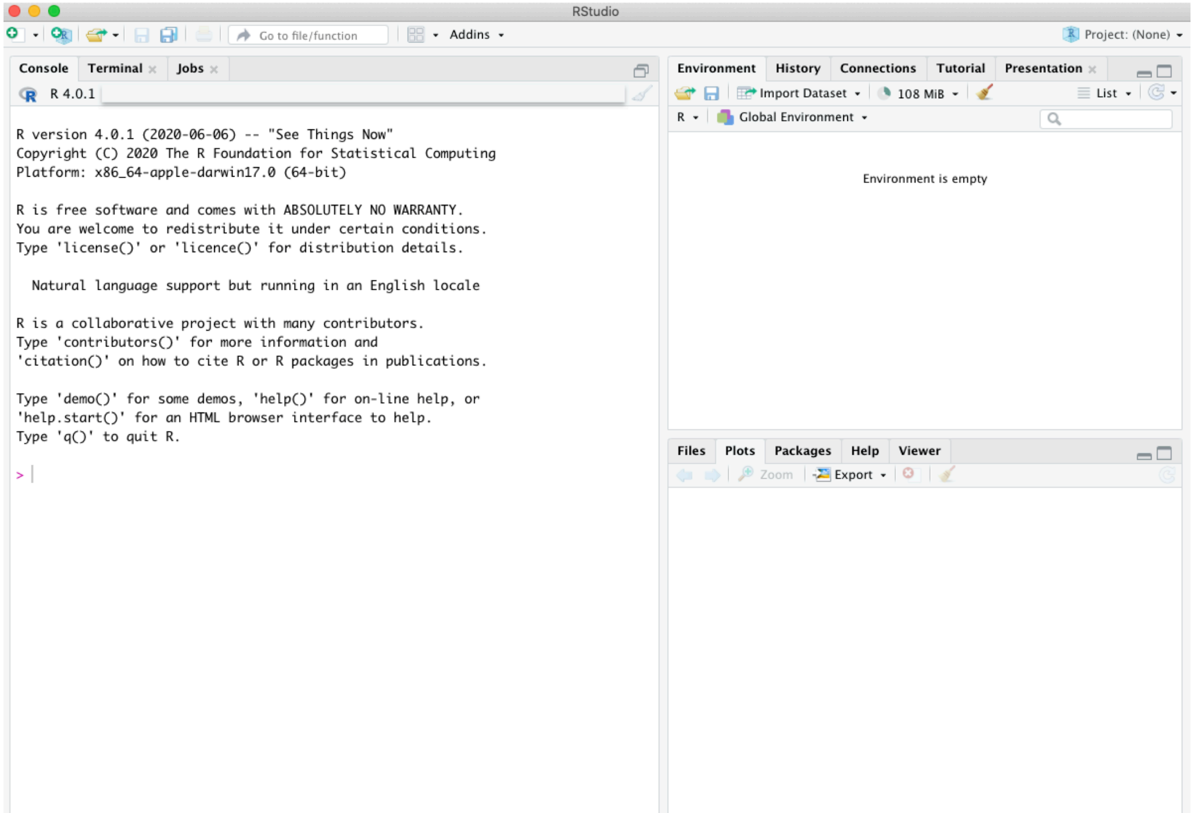
1.3 Open R Script in RStudio

This next section provides a few notes on using R and RStudio now that you have installed it. In this section, we cover the following materials:

- Using R as a calculator and assigning objects using `<-`
- Setting your working directory and the `setwd()` function.
- Creating and saving an R script (.R file)
- Creating, saving, and compiling an R Markdown document (.Rmd) into an html document (.html)

This section highlights important concepts from QSS chapter 1.

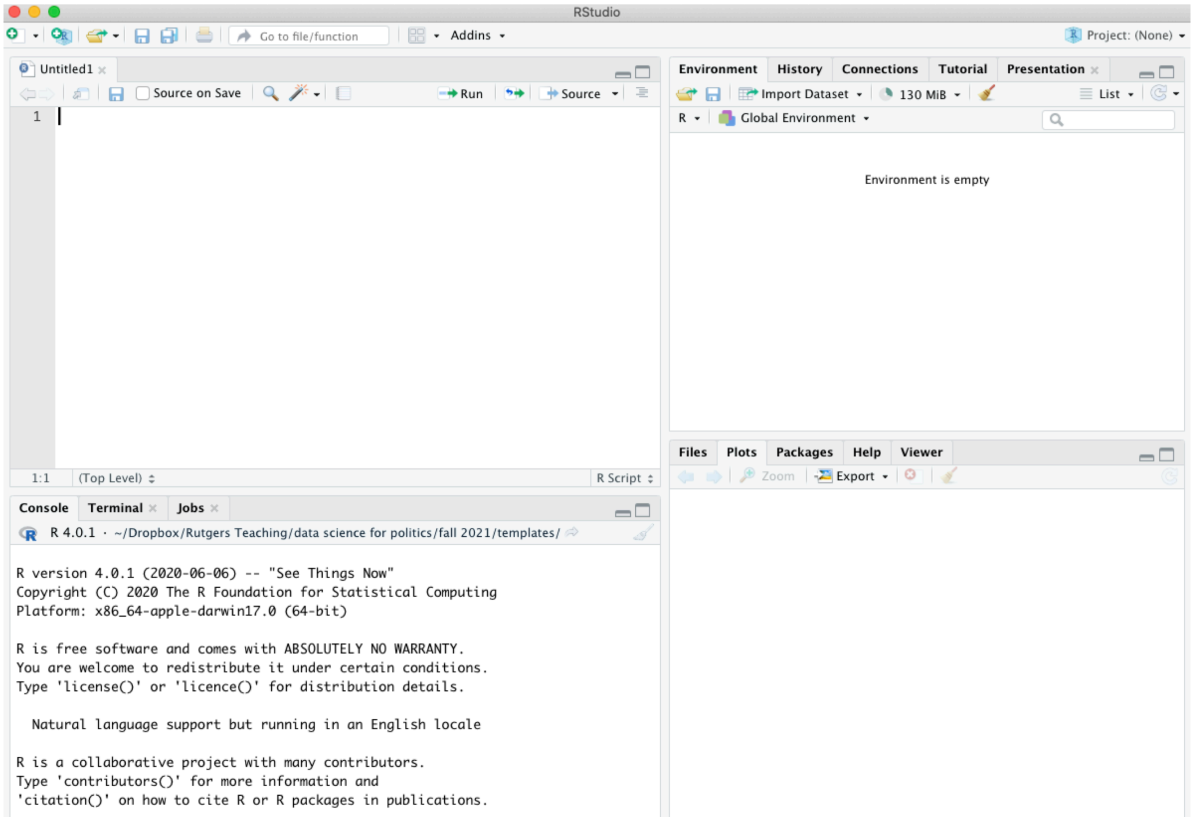
RStudio is an open-source and free program that greatly facilitates the use of R, especially for users new to programming. Once you have downloaded and installed R and RStudio, to work in R, all you need to do now is ***open RStudio*** (it will open R). It should look like this, though your version numbers will be different:



Note: The first time you open RStudio, you likely only have the three windows above. We will want to create a fourth window by **opening an R script** to create the fourth window.

- To do this, in RStudio, click on File -> New -> R script in your computer's toolbar. This will open a blank document for text editing in the upper left of the RStudio window. We will return to this window in a moment.
  - You can alternatively click on the green + sign indicator in the top-left corner of the RStudio window, which should give you the option to create a new R script document.

Now you should have something that looks like this, similar to Figure 1.1. in QSS:



- The upper-left window has our .R script document that will contain code.
- The lower-left window is the console. This will show the output of the code we run. We will also be able to type directly in the console.
- The upper-right window shows the environment (and other tabs, such as the history of commands). When we load and store data in RStudio, we will see a summary of that in the environment.
- The lower-right window will enable us to view plots and search help files, among other things.

1.3.1 Using R as a Calculator

The *bottom left* window in your RStudio is the Console. You can type in this window to use R as a calculator or to try out commands. It will show the raw output of any commands you type. For example, we can try to use R as a calculator. Type the following in the Console (the bottom left window) and hit "enter" or "return" on your keyboard:

```
5 + 3
```

```
[1] 8
```

```r
5 - 3
```

```
[1] 2
```

```r
5^2
```

```
[1] 25
```

```r
5 * 3
```

```
[1] 15
```

```r
5/3
```

```
[1] 1.666667
```

```r
(5 + 3) * 2
```

```
[1] 16
```

Again, in the other RStudio windows, the upper right will show a history of commands that you have sent from the text editor to the R console, along with other items. The lower right will show graphs, help documents and other features. These will be useful later in the course.

1.3.2 Working in an R Script

Earlier, I asked you to open an R script in the upper left window by doing File, then New File, then R Script. Let's go back to working in that window.

Set your working directory `setwd()`

Many times you work in RStudio, the first thing you will do is set your working directory. This is a designated folder in your computer where you will save your R scripts and datasets.

There are many ways to do this.

- An easy way is to go to Session -> Set Working Directory -> Choose Directory. I suggest choosing a folder in your computer that you can easily find and that you will routinely use for this class. Go ahead and create/select it.
- Note: when you selected your directory, code came out in the bottom left Console window. This is the `setwd()` command which can also be used directly to set your working directory in the future.
- If you aren't sure where your directory has been set, you can also type `getwd()` in your Console. Try it now

```
## Example of where my directory was
getwd()
```

```
[1] "/Users/ktmccabe/Dropbox/GitHub2/dsps24"
```

If I want to change the working directory, I can go to the top toolbar of my computer and use Session -> Set Working Directory -> Choose Directory or just type my file pathway using the `setwd()` below:

```
## Example of setting the working directory using setwd().
## Your computer will have your own file path.
setwd("/Users/ktmccabe/Dropbox/Rutgers Teaching/")
```

1.3.3 Saving the R Script

Let's now save our R script to our working directory and give it an informative name. To do so, go to File, then Save As, make sure you are in the same folder on your computer as the folder you chose for your working directory.

Give the file an informative name, such as: "McCabeWeek1.R". Note: all of your R scripts will have the .R extension.

1.3.4 Annotating your R script

Now that we have saved our R script, let's work inside of it. Remember, we are in the top-left RStudio window now.

- Just like the beginning of a paper, you will want to title your R script. In R, any line that you start with a # will not be treated as a programming command. You can use this to your advantage to write titles/comments– annotations that explain what your code is doing. Below is a screenshot example of a template R script.

- You can specify your working directory at the top, too. Add your own filepath inside `setwd()`

```
#############################################
## Problem Set XX ###########################
## Name: Your name   #############
## People you worked with:        ###########
#############################################

# enter the path of your working directory
setwd()


###########################################
# Problem 1
###########################################

## add comments like this to help explain your steps

# I added two numbers
sum53 <- 5 + 3
sum53

###########################################
# Problem 2
###########################################


###########################################
# Problem 3
###########################################


```

- Then you can start answering problems in the rest of the script.
- Think of the R script as where you write the final draft of your paper. In the Console (the bottom-left window), you can mess around and try different things, like you might when you are taking notes or outlining an essay. Then, write the final programming

steps that lead you to your answer in the R script. For example, if I wanted to add 5 + 3, I might try different ways of typing it in the Console, and then when I found out `5 + 3` is the right approach, I would type that into my script.

1.3.5 Running Commands in your R script

The last thing we will note in this section is how to execute commands in your R script.

To run / execute a command in your R script (the upper left window), you can

1. Highlight the code you want to run, and then hold down "command + return" on a Mac or "control + enter" on Windows
2. Place your cursor at the end of the line of code (far right), and hit "command + return" on a Mac or "control + return" on Windows, or
3. Do 1 or 2, but instead of using the keyboard to execute the commands, click "Run" in the top right corner of the upper-left window.

Try it: Type `5 + 3` in the R script. Then, try to execute `5 + 3`. It should look something like this:

```
#############################################
## Problem Set 1 ###########################
## Name: Katherine McCabe   #############
## People you worked with: Just me      ###########
#############################################

## enter the path of your working directory
setwd("/Users/ktmccabe/Dropbox/Rutgers Teaching/data science for politics")

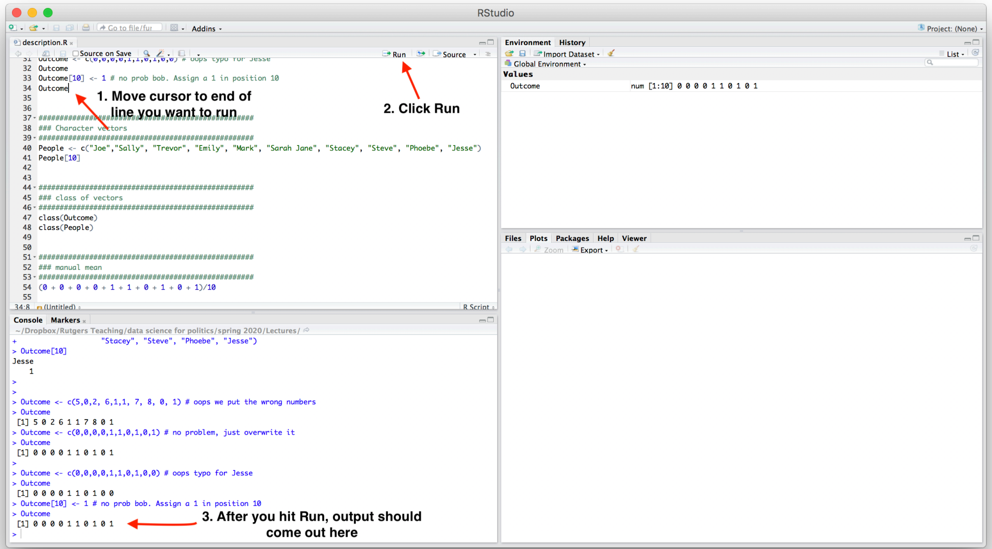

###########################################
## Problem 1
###########################################

## Add 5 + 3
5 + 3
```

After you executed the code, you should see it pop out in your Console:

```
5 + 3
```

```
[1] 8
```



Note: The symbol # also allows for annotation behind commands or on a separate line. Everything that follows # will be ignored by R. You can annotate your own code so that you and others can understand what each part of the code is designed to do.

```
## Example
sum53 <- 5 + 3 # example of assigning an addition calculation
```

1.3.6 Objects

Sometimes we will want to store our calculations as "objects" in R. We use `<-` to assign objects by placing it to the left of what we want to store. For example, let's store the calculation `5 + 3` as an object named `sum53`:

```
sum53 <- 5 + 3
```

After we execute this code, `sum53` now stores the calculation. This means, that if we execute a line of code that just has `sum53`, it will output 8. Try it:

```
sum53
```

```
[1] 8
```

Now we no longer have to type `5 + 3`, we can just type `sum53`. For example, let's say we wanted to subtract 2 from this calculation. We could do:

```
sum53 - 2
```

```
[1] 6
```

Let's say we wanted to divide two stored calculations:

```
ten <- 5 + 5
two <- 1 + 1
ten / two
```

```
[1] 5
```

The information stored does not have to be numeric. For example, it can be a word, or what we would call a character string, in which case you need to use quotation marks.

```
mccabe <- "professor for this course"
mccabe
```

```
[1] "professor for this course"
```

Note: Object names cannot begin with numbers and no spacing is allowed. Avoid using special characters such as % and $, which have specific meanings in R. Finally, use concise and intuitive object names.

- GOOD CODE: `practice.calc <- 5 + 3`
- BAD CODE: `meaningless.and.unnecessarily.long.name <- 5 + 3`

While these are simple examples, we will use objects all the time for more complicated things to store (e.g., like full datasets!) throughout the course.

We can also store an array or "vector" of information using `c()`

```
somenumbers <- c(3, 6, 8, 9)
somenumbers
```

```
[1] 3 6 8 9
```

Importance of Clean Code

Ideally, when you are done with your R script, you should be able to highlight the entire script and execute it without generating any error messages. This means your code is clean. Code with typos in it may generate a red error message in the Console upon execution. This can happen when there are typos or commands are misused.

For example, R is case sensitive. Let's say we assigned our object like before:

```
sum53 <- 5 + 3
```

However, when we went to execute `sum53`, we accidentally typed `Sum53`:

```
Sum53
```

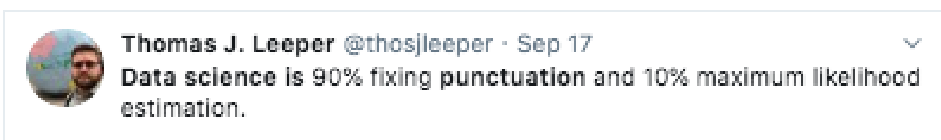
```
Error in eval(expr, envir, enclos): object 'Sum53' not found
```

Only certain types of objects can be used in mathematical calculations. Let's say we tried to divide `mccabe` by 2:

```
mccabe / 2
```

```
Error in mccabe/2: non-numeric argument to binary operator
```

A big part of learning to use R will be learning how to troubleshoot and detect typos in your code that generate error messages.

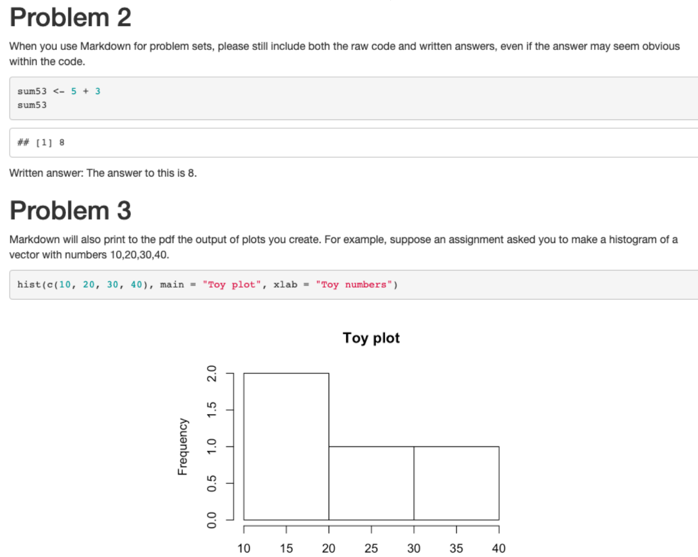


1.4 R Markdown

An R Markdown document, which you can also create in RStudio, allows you to weave together regular text, R code, and the output of R code in the same document. This can be very convenient when conducting data analysis because it allows you more space to explain what you are doing in each step. We will use it as an effective platform for writing up problem sets.

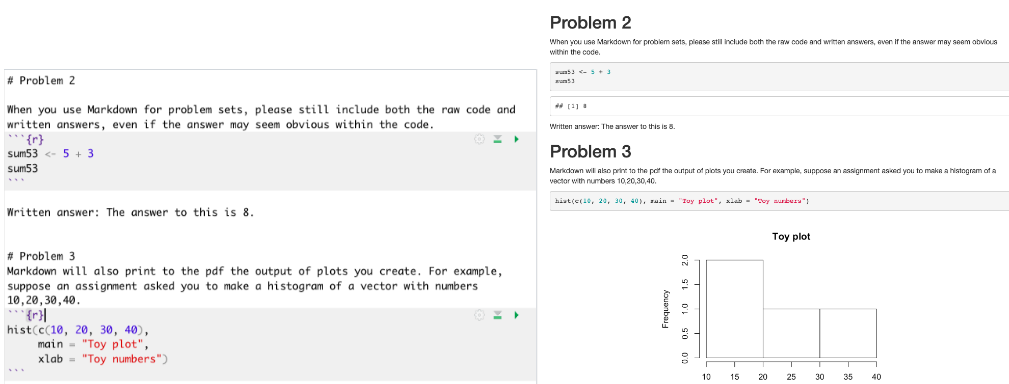
R Markdown documents can be "compiled" into html, pdf, or docx documents by clicking the `Knit` button on top of the upper-left window. Below is an example of what a compiled html file looks like.

- Note that the image has both written text and a gray chunk, within which there is some R code, as well as the output of the R code (e.g., the number 8 and the image of the



  histogram plot.

We say this is a "compiled" RMarkdown document because it differs from the raw version of the file, which is a .Rmd file format. Below is an example of what the raw .Rmd version looks like, compared to the compiled html version.



1.4.1 Getting started with RMarkdown

Just like with a regular R script, to work in R Markdown, you will open up RStudio.

- For additional support beyond the notes below, you can also follow the materials provided by RStudio for getting started with R Markdown https://rmarkdown.rstudio.com/lesson-1.html.

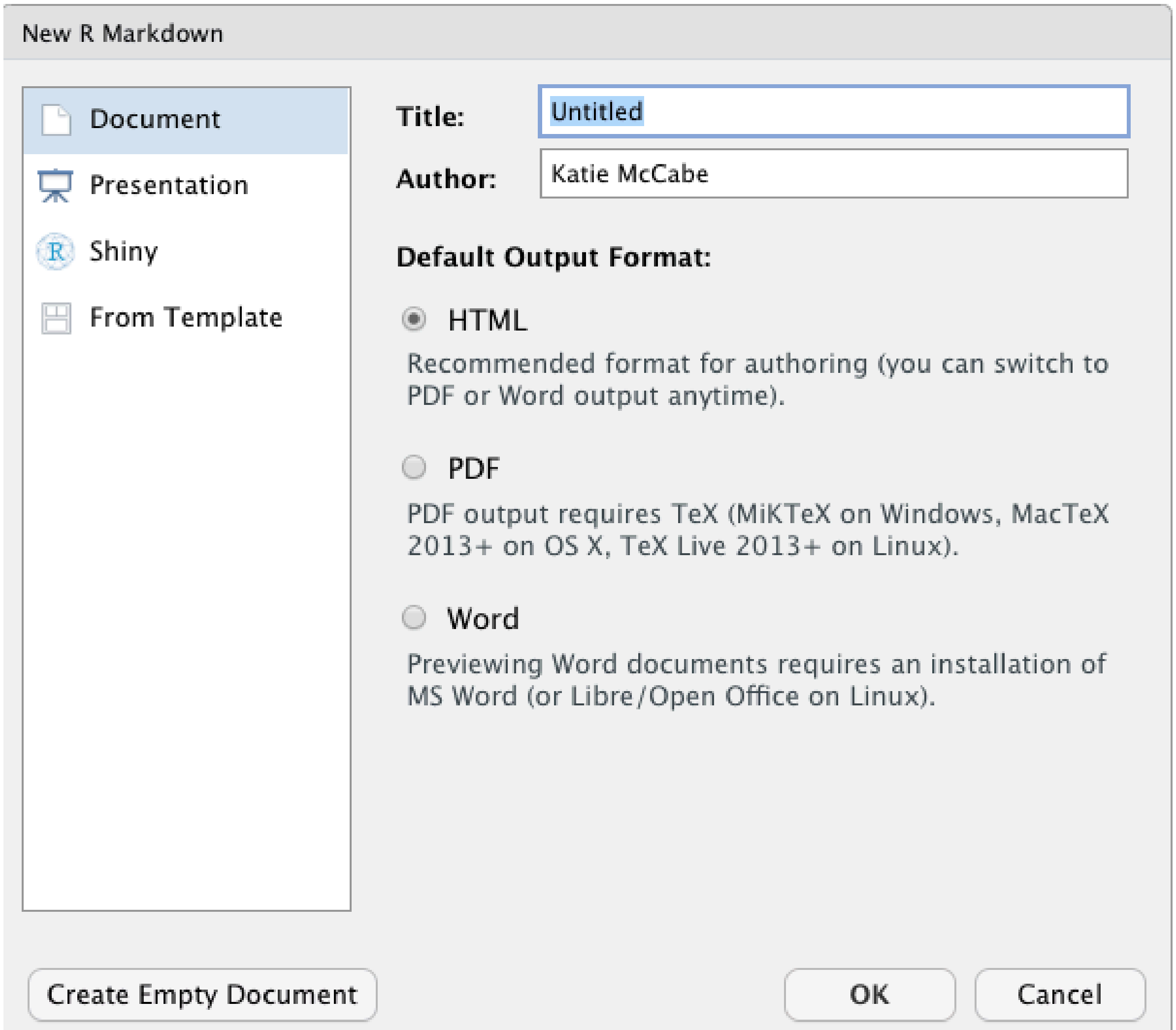
The **first time** you will be working in R Markdown, you will want to install two packages: `rmarkdown` and `knitr`. You can do this in the Console window in RStudio (remember the lower-left window!).

Type the following into the Console window and hit enter/return.

```
install.packages("rmarkdown")
install.packages("knitr")
```

Once you have those installed, now, each time you want to create an R Markdown document, you will open up a .Rmd R Markdown file and get to work.

1. Go to File -> New File -> R Markdown in RStudio
    - Alternatively, you can click the green + symbol at the top left of your RStudio window
2. This should open up a window with several options, similar to the image below
    - Create an informative title and change the author name to match your own
    - For now, we will keep the file type as html. In the future, you can create pdf or .doc documents. However, these require additional programs installed on your computer, which we will not cover in the course.



3. After you hit "OK" a new .Rmd script file will open in your top-left window with some template language and code chunks, similar to the image below. Alternatively, you can start from scratch by clicking "Create Empty Document" or open a template .Rmd file of your own saved on your computer.

```
---
title: "Problem Set 1"
author: "Katie McCabe"
date: "9/7/2021"
output: html_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
```

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see <http://rmarkdown.rstudio.com>.

When you click the **Knit** button a document will be generated that includes both content as well as the output of any embedded R code chunks within the document. You can embed an R code chunk like this:

```{r cars}
```

4. ***Save as .Rmd file.*** Save the file by going to "File -> Save as" in RStudio

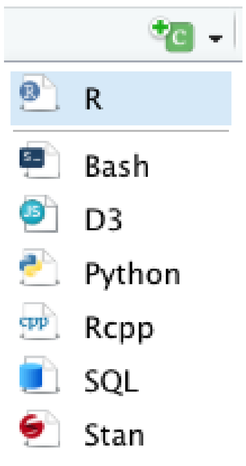
- Give the file an informative name like your LastnamePractice1.Rmd

5. ***Key Components.*** Now you are ready to work within the Rmd script file. We will point to four basic components of this file, and you can build your knowledge of RMarkdown from there.

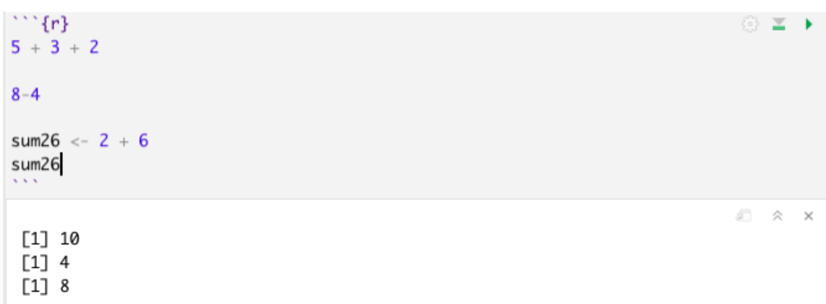
    1. The top part bracketed by `---` on top and bottom is the YAML component. This tells RStudio the pertinent information about how to "compile" the Rmd file.
        - Most of the time you can leave this alone, but you can always edit the title, author, or date as you wish.
    2. The next component are the global options for the document. It is conveniently labeled "setup." By default what this is saying is that the compiled version will "echo" (i.e., display all code chunks and output) unless you specifically specify otherwise. For example, note that it says `include = FALSE` for the setup chunk. That setting means that this code chunk will "run" but it will not appear in the nicely compiled .html file.
        - Most of the time you will not need to edit those settings.
    3. The third component I want to bring attention to is the body text. The # symbol in RMarkdown is used to indicate that you have a new section of the document. For example, in the compiled images at the beginning, this resulted in the text being larger and bolded when it said "Problem 2." In addition to just using a single #,
```

using ## or ### can indicate subsections or subsubsections. Other than that symbol, you can generally write text just as you would in any word processing program, with some exceptions, such as how to make text bold or italicized.

4. The final component I want to call attention to are the other main body code chunks. These are specific parts of the document where you want to create a mini R script. To create these, you can simply click the + C symbol toward the top of the top left window of RStudio and indicate you want an R chunk.



6. ***Writing R Code.*** Within a code chunk, you can type R code just like you would in any R script, as explained in the previous section. However, in RMarkdown, you also have the option of running an entire code chunk at once by hitting the green triangle at the top-right of a given code chunk.



7. ***Knitting the document.*** Once you have added a code chunk and/or some text, you are ready to compile or "Knit" the document. This is what generates the .html document.

   - To do so, click on the Knit button toward the top of the top-left window of Rstudio. After a few moments, this should open up a preview window displaying the compiled html file.
   - It will also save an actual .html file in your working directory (the same location on your computer where you have saved the .Rmd file)

- Try to locate this compiled .html file on your computer and open it. For most computers, .html files will open in your default web browser, such as Google Chrome or Safari.
- This step is a common place where errors are detected and generated. Sometimes the compiling process fails due to errors in the R code in your code chunks or an error in the Markdown syntax. If your document fails to knit, the next step is to try to troubleshoot the error messages the compiling process generates. The best way to reduce and more easily detect errors is to "knit as you go." Try to knit your document after each chunk of code you create.

1.5 Assignment 1

Below is an exercise that will demonstrate you are able to use R as a calculator, create R scripts, and create and compile R Markdown files. You should be able to complete this assignment after reviewing the course notes from this section and QSS chapter 1.

We will start walking through this assignment together during class, but you are welcome to try to do this ahead of time on your own.

You will submit three documents on Canvas:

- An **R script** (.R) file with your code. Follow the best practices by titling your script and using # comments to explain your steps. This code should be clean. I should be able to run your code to verify that the code produces the answers you write down.
- An **.Rmd document and**
- A compiled RMarkdown **.html document** that you get after "knitting" the .Rmd file. This should also have a title including your name and use text or # comments to explain your steps.

You can create these documents from scratch using the guidance in the previous sections, or you can download and open the .R and .Rmd templates, provided on Canvas, in RStudio to get started.

This video provides a brief overview of opening an R script and R Markdown file in RStudio with similar problems to those asked of you in the assignment. The notes in previous sections provide additional details.

https://www.youtube.com/watch?v=g37_-icdPMc

Assignment Exercises

1. Create a .R script saved as "LastnameSetup1.R" (use your last name). If you use the template on Canvas, after opening in RStudio, use File -> Save As to change to create this file name. Within the body of this file, make sure to title it and provide your name.

    1. Set your working directory using the Session tab in RStudio.
    2. Do the calculation 8 + 3 - 5 in your R script. Store it as an object with an informative name. Report the answer as a comment # below the code.
    3. Do the calculation 7 x 3 in your R script. Store it as an object with an informative name. Report the answer as a comment # below the code.
    4. Add these two calculations together. Note: do this by adding together the objects you created, not the underlying raw calculations. Report the answer as a # below the code.

2. In this problem, we will just re-format what we did in the first problem in an R Markdown format. Create a .Rmd R Markdown file saved as "LastnameSetup1.Rmd." If you use the template on Canvas, after opening in RStudio, use File -> Save As to change to create this file name. Within this file, make sure to title it and provide your name.

    1. Create a Markdown heading # Problem 2.1. Underneath this, create an R code chunk in which you do the calculation 8 + 3 - 5. Store it as an object with an informative name. Report the answer in plain language below the code chunk.
    2. Create a Markdown heading # Problem 2.2. Underneath this, create an R code chunk in which you do the calculation 7 x 3 in your R script. Store it as an object with an informative name. Report the answer in plain language below the code chunk.
    3. Create a Markdown heading # Problem 2.3. Underneath this, create an R code chunk in which you add the previous two calculations together. Note: do this by adding together the objects you created, not the underlying raw calculations. Report the answer in plain language below the code chunk.
    4. Create a Markdown heading # Problem 2.4. Write down how you will complete your R assignments this semester. For example, if you have a personal laptop with R and RStudio on it, you will simply write "I will use my personal laptop." If you don't have a personal computer or laptop, please indicate where on campus or off-campus you will have regular access to a computer with R/RStudio to do your work. It is ***essential*** that you have regular access to a computer so that you will not fall behind in this course.

3. Create a compiled .html file by "knitting" the .Rmd file into a .html document. Save the file as "LastnameSetup1.html." This should happen automatically. The file will be located in the folder where the .Rmd file is also saved.

All done! Submit the three documents (.R, .Rmd, and .html) on Canvas.

2 Description

What are things we want to describe in political science?

- Unemployment rate, GDP
- Voter turnout, vote share for a party in an election
- Percentage of women in the labor force
- Poverty rates over time

What else? What does description help us achieve?

- Identify tendencies
- Identify patterns or trends
- Identify relationships between two or more factors
- Help us generalize from anecdotes, what is common vs. what is uncommon?
- Diagnose demand, needs, potential problems, likely outcomes

Generate ideas for other goals, such as explanation and prediction

2.1 Process of Describing

How do we go about a descriptive quantitative analysis?

1. Substantive Expertise: Start with a topic, puzzle, or question (e.g., How is the economy doing?)
2. Find outcome data relevant to that question (e.g., GDP)
    - Start from a concept: what we want to describe (i.e., health of the economy)
    - Move toward an "operationalization" (i.e., a way to measure it)
    - Easy! except… social science is messy. Our concepts are rich, while our measures may be very narrow or concrete.
        - For example, GDP is one way to measure economic health, but is it the only measure?
        - Choose measures based on validity, reliability, cost
        - Validity: how well does the empirical measure reflect the concept it is trying to measure (too broad vs. narrow, susceptible to external biases?)

- Reliability: how reproducible and stable is the measure across different researchers and slight variation in sample or measures

3. Find multiple relevant units or "data points" depending on the descriptions and comparisons you want to make
   - E.g., Multiple years of data (e.g., U.S., from 1900 to 2020)
   - E.g., Multiple countries from one year (e.g., U.S. to Germany to other countries)
4. Summarize the data to help answer the question

2.1.1 Example Process

1. How is the economy doing?
2. Find outcome data relevant to that question
   - Let's ask people
3. Find multiple relevant units or data points
   - We will ask several people. Each person will be a data point.
4. Summarize the data
   - Let's take the mean

Let's say we ask 10 people, "Is the economy doing well?" We will give a person a 1 if they say yes, a 0 if they say no. We will index each person by i, and we have a total of $N = 10$ people.

i	People	Outcome
1	Joe	0
2	Sally	0
3	Trevor	0
4	Emily	0
5	Mark	1
6	Sarah Jane	1
7	Stacey	0
8	Steve	1
9	Phoebe	0
10	Jesse	1

How would you summarize information in explaining it to another person? You would probably want to describe how most people feel about the economy. In other words, you would describe the "central tendency" of people's responses (the central tendency of the data).

2.2 Summarizing univariate data

For a video explainer of the code in this section, see below. The video only discusses the code. Use the notes and lecture discussion for additional context. (Via youtube, you can speed up the playback to 1.5 or 2x speed.)

https://www.youtube.com/watch?v=80tbdiWuljc

Univariate data refers to data coming from one "variable," where a variable captures the values of a changing characteristic.

Our set of values is Outcome = {0,0,0,0,1,1,0,1,0,1}.

- We will call this a vector of values, where a vector is just a collection of things.
- Because our vector contains only numbers, we will call it a *numeric* vector.
- Each value can be indexed by `i`, denoting the position of the value in the
- For example, Jesse is in position `i=10` of the vector, and his value is 1

We can create vectors in R by using `c()` and assigning `<-` it to an object we will call `Outcome`. Note: you will use the `c()` and `<-` assignment tool all of the time in this course!

```
Outcome <- c(0,0,0,0,1,1,0,1,0,1) # Use commas to separate values
```

We can extract a particular value within our vector using brackets and the value's numeric position in the vector.

```
Outcome[10] # what value is in the 10th position?
```

```
[1] 1
```

We can label our outcomes using `names()`

```
names(Outcome) <-c("Joe","Sally", "Trevor", "Emily", "Mark",
                   "Sarah Jane", "Stacey", "Steve", "Phoebe", "Jesse")
Outcome[10]
```

```
Jesse
    1
```

We can overwrite whole vectors or values within a vector

```
Outcome <- c(5,0,2, 6,1,1, 7, 8, 0, 1) # oops we put the wrong numbers
Outcome
```

```
 [1] 5 0 2 6 1 1 7 8 0 1
```

```r
Outcome <- c(0,0,0,0,1,1,0,1,0,1) # no problem, just overwrite it
Outcome
```

```
 [1] 0 0 0 0 1 1 0 1 0 1
```

Oops we accidentally type a 0 for Jesse.

```r
Outcome <- c(0,0,0,0,1,1,0,1,0,0) # oops typo for Jesse
Outcome
```

```
 [1] 0 0 0 0 1 1 0 1 0 0
```

```r
Outcome[10] <- 1 # no prob bob. Assign a 1 in position 10
Outcome
```

```
 [1] 0 0 0 0 1 1 0 1 0 1
```

Vectors do not have to be numeric. Character vectors contain a collection of words and phrases. In R, we use quotations around character values

Example: let's create a vector of names that we will call `People`.

```r
People <- c("Joe","Sally", "Trevor", "Emily", "Mark", "Sarah Jane", "Stacey", "Steve", "Ph
People[10]
```

```
[1] "Jesse"
```

We can use the R function `class()` to tell us the type of object we have.

```r
class(Outcome)
```

```
[1] "numeric"
```

```r
class(People)
```

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

2.3 Functions to summarize univariate data

We will use many "functions" in R, which are actions that we request R to perform with data. Functions take one or more inputs that you provide, "under the hood" perform a series of actions, and then produces one or more outputs. For example, if we supply the `mean()` function in R with a set of numbers, R will take the average of those numbers and then report the result.

For univariate data (data based on a single variable or changing characteristic), often we are interested in describing the range of the values and their central tendency. A central tendency reflects the mean or median.

- range: the minimum (`min()`) and maximum (`max()`) values
- mean: the average value (`mean()`)

The average is the sum of the values divided by the number of values:

$\bar{X} = \frac{\text{sum of values}}{\text{number of values}} = \frac{x_1 + x_2 + ... + x_N}{N} = \frac{1}{N}\sum_{i=1}^{i=N} x_i$

Let's do this in R for our set of 10 values

```
(0 + 0 + 0 + 0 + 1 + 1 + 0 + 1 + 0 + 1)/10
```

```
[1] 0.4
```

The average outcome is .4. Note: when a variable contains only 0's and 1's its mean is the proportion of 1's. 40% of people think the economy is doing well.

2.3.1 Step-by-step: Using functions in R (overview)

A function is an action(s) that you request R to perform on an object or set of objects. For example, we will use the `mean()` function to ask R to take the mean or "average" of a vector.

- Inside the function you place inputs or "arguments."

```
mean(Outcome)
```

```
[1] 0.4
```

R also has functions that take the sum `sum()` of a vector of values.

```
sumofvalues <- sum(Outcome)
```

And that count the total number of values or "length" `length()` of the vector.

```
numberofvalues <- length(Outcome)
```

Note that the below is also equivalent to the mean

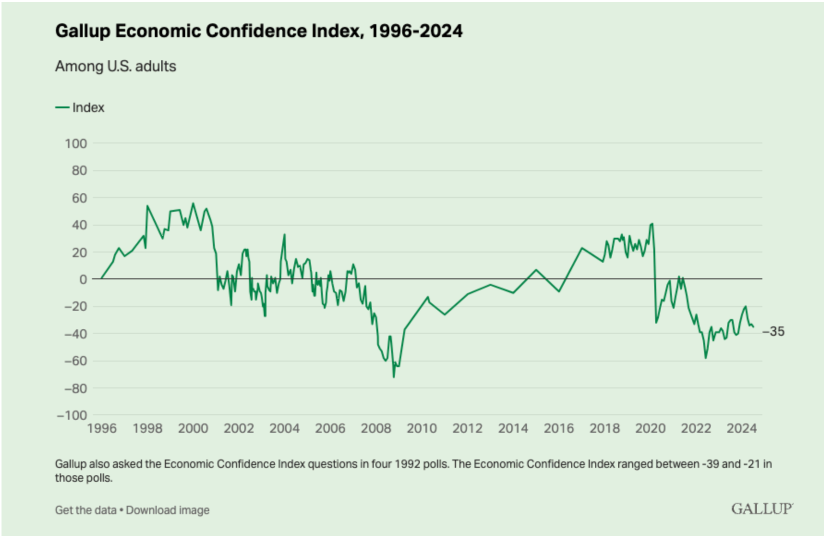
```
sumofvalues / numberofvalues
```

```
[1] 0.4
```

Returning to our example, we found that 40% of people surveyed thought the economy was doing well. Surveying people about their opinions on how the country doing is a common way that social scientists use description. We could extend this exercise in many ways going forward, even with the same question.

- Start with a question: How is the economy doing?
- Let's find a measure: Ask people if the economy is doing well.
- Find data points: Multiple people (we could stop there with the average!), or add more variables:
  - Across time: Survey people across multiple years
  - Across type of people: Survey different partisan groups

These types of trends are often used by news organizations and public opinion organizations like, Gallup.



This was just a first example of description in political science. There are many other ways to describe how the economy is doing and many other topics we might want to describe in politics.

2.4 Loading data into R

For this section, our motivating example will be methods to measure voter turnout in the United States.

Describing voter turnout

- What is a typical level of voter turnout?
- How has turnout changed over time?
- Is turnout higher in presidential years or in midterm years?

How can we measure turnout? Think about the validity, reliability, and cost of different approaches.

Example: Dataset on Voter Turnout in the U.S. across multiple years

	year	VEP	VAP	total	ANES	felons	noncit	overseas
1	1980	159635	164445	86515	71	802	5756	1803
2	1982	160467	166028	67616	60	960	6641	1982
3	1984	167702	173995	92653	74	1165	7482	2361
4	1986	170396	177922	64991	53	1367	8362	2216
5	1988	173579	181955	91595	70	1594	9280	2257
6	1990	176629	186159	67859	47	1901	10239	2659
7	1992	179656	190778	104405	75	2183	11447	2418
8	1994	182623	195258	75106	56	2441	12497	2229
9	1996	186347	200016	96263	73	2586	13601	2499
10	1998	190420	205313	72537	52	2920	14988	2937
11	2000	194331	210623	105375	73	3083	16218	2937
12	2002	198382	215462	78382	62	3168	17237	3308
13	2004	203483	220336	122295	77	3158	18068	3862
14	2008	213314	230872	131304	78	3145	19392	4972

In this dataset, each row is an election year. Each column contains information about the population, potential voters, or voter turnout. These will help us compute the turnout rate in a given year. To work with this dataset, we need to load it into R.

2.4.1 Working with datasets in R

For a video explainer of the code in this section, see below. The video only discusses the code. Use the notes and lecture discussion for additional context. (Via youtube, you can speed up the playback to 1.5 or 2x speed.)

https://www.youtube.com/watch?v=rm_g0rrglEQ

Often the variables we care about are stored inside of rectangular datasets

- These have a number of rows `nrow()` and columns `ncol()`
- Each row is an "observation," representing the information collected from an individual or entity
- Each column is a variable, representing a changing characteristic across multiple observations

When we import a dataset into R, we have a few options.

Option 1: Download dataset to your computer

- Move the dataset to your working directory
- Identify the file type (e.g., csv, dta, RData, txt)

- Pick the appropriate R function to match the type (e.g., `read.csv()`, `read.dta()`, `load()`, `read.table()`)
- Assign the dataset to an object. This object will now be `class()` of `data.frame`

```
turnout <- read.csv("turnout.csv")
```

Click here for an alternative function for csv files.

Some scholars prefer to use the function `read_csv` to load csv data. It is better at handling more complicated types of data. We will not need to use this function in this course, but you may encounter it elsewhere.

To use this function, the first time we will go about using it, we have to first install a "package" called `readr`. Packages in R give us additional tools beyond what the base version of R provides. It is like installing an extra app on your phone.

```
install.packages("readr")
```

Once we have that installed, now anytime we want to use the function, we will call (open) the "readr" package using `library()`, and then the syntax is just like using the `read.csv` function.

```
library(readr)
turnout <- read_csv("turnout.csv")
```

Option 2: Read file from a url provided

- Need an active internet connection for this to work
- URL generally must be public
- Include the url inside the function used to read the data

```
turnout <- read.csv("https://raw.githubusercontent.com/ktmccabe/teachingdata/main/turnout.
```

```
class(turnout)
```

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

You can also open up a window to view the data:

```
View(turnout)
```

2.4.2 Measuring the Turnout in the US Elections

Relevant questions with voter turnout

- What is a typical level of voter turnout?
- Is turnout higher in presidential years or in midterm years?
- Is turnout higher or lower based on voting-eligible (VEP) or voting-age (VAP) populations? We have a lot of people who are citizens 18 and older who are ineligible to vote. This makes the VEP denominator smaller than the VAP.

Voter Turnout in the U.S.

- Numerator: `total`: Total votes cast (in thousands)
- Denominator:
  - VAP: (voting-age population) from Census
  - VEP (voting-eligible population) VEP = VAP + overseas voters - ineligible voters
- Additional Variables and Descriptions
  - `year`: election year
  - `ANES`: ANES self-reported estimated turnout rate
  - `VEP`: Voting Eligible Population (in thousands)
  - `VAP`: Voting Age Population (in thousands)
  - `total`: total ballots cast for highest office (in thousands)
  - `felons`: total ineligible felons (in thousands)
  - `noncitizens`: total non-citizens (in thousands)
  - `overseas`: total eligible overseas voters (in thousands)
  - `osvoters`: total ballots counted by overseas voters (in thousands)

2.4.3 Getting to know your data

```r
## How many observations (the rows)?
nrow(turnout)
```

```
[1] 14
```

```r
## How many variables (the columns)?
ncol(turnout)
```

```
[1] 9
```

```r
## What are the variable names?
names(turnout)
```

```
[1] "year"     "VEP"      "VAP"      "total"    "ANES"     "felons"   "noncit"
[8] "overseas" "osvoters"
```

```r
## Show the first six rows
head(turnout)
```

```
  year    VEP    VAP total ANES felons noncit overseas osvoters
1 1980 159635 164445 86515   71    802   5756     1803       NA
2 1982 160467 166028 67616   60    960   6641     1982       NA
3 1984 167702 173995 92653   74   1165   7482     2361       NA
4 1986 170396 177922 64991   53   1367   8362     2216       NA
5 1988 173579 181955 91595   70   1594   9280     2257       NA
6 1990 176629 186159 67859   47   1901  10239     2659       NA
```

Extract a particular column (vector) from the data using the $.

```r
turnout$year
```

```
 [1] 1980 1982 1984 1986 1988 1990 1992 1994 1996 1998 2000 2002 2004 2008
```

Extract the 10th year. Just like before! We use 10 to indicate the value of the `year` column in position (row 10) of the data.

```r
turnout$year[10]
```

```
[1] 1998
```

We can take the `mean()` of a particular column, too. Let's take it of the total number of voters.

```r
mean(turnout$total)
```

```
[1] 89778.29
```

And get the `class()` (Note: integer is just a type of numeric variable)

```r
class(turnout$total)
```

```
[1] "integer"
```

We can also use brackets in the full data frame, but because our data frame has BOTH rows and columns, we cannot just supply one position `i`. Instead, we have to tell R which row AND which column by using a comma between the positions.

```r
turnout[1,2] # value in row 1, column 2
```

```
[1] 159635
```

We can use the column name instead

```r
turnout[1, "VEP"]
```

```
[1] 159635
```

If we leave the second entry blank, it will return all columns for the specified row

```r
turnout[1,] # All variable values for row 1
```

```
  year    VEP    VAP total ANES felons noncit overseas osvoters
1 1980 159635 164445 86515   71    802   5756     1803       NA
```

The opposite is true if we leave the first entry blank.

```r
turnout[,2] # VEP for all rows
```

```
 [1] 159635 160467 167702 170396 173579 176629 179656 182623 186347 190420
[11] 194331 198382 203483 213314
```

2.5 Comparing VEP and VAP turnout

2.5.1 Creating new variables in R

Let's create a new variable that is VAP that adds overseas voters.

```
# Use $ to add a new variable (i.e., column) to a dataframe
turnout$VAPplusoverseas <- turnout$VAP + turnout$overseas
```

Under the hood, what this is doing is taking each value of `turnout$VAP` and adding it to its corresponding values of `turnout$overseas`.

And, yes, this new variable shows up as a new column in `turnout`. Go ahead, `View()` it

```
View(turnout)
```

This does not change the underlying `turnout.csv` file, only the `turnout data.frame` we are working with in the current R session.

- This is an advantage of using an R script.
- You don't have to worry about overwriting/messing up the raw data.
- You start from the original raw data when you load `turnout.csv`, and then everything else is done within R.

This is our new denominator. Now we can calculate turnout based on this denominator.

```
turnout$newVAPturnout <- turnout$total / turnout$VAPplusoverseas
```

Just like with adding two vectors, when we divide, each value in the first vector is divided by its corresponding value in the second vector.

```
turnout$newVAPturnout
```

```
 [1] 0.5203972 0.4024522 0.5253748 0.3607845 0.4972260 0.3593884 0.5404097
 [8] 0.3803086 0.4753376 0.3483169 0.4934211 0.3582850 0.5454777 0.5567409
```

Let's calculate the VEP turnout rate and turn it into a percentage. This time, we do it in one step.

- (total votes / VEP) $\times$ 100:

```
turnout$newVEPturnout <- (turnout$total / turnout$VEP) * 100
turnout$newVEPturnout
```

```
 [1] 54.19551 42.13701 55.24860 38.14115 52.76848 38.41895 58.11384 41.12625
 [9] 51.65793 38.09316 54.22449 39.51064 60.10084 61.55433
```

Let's change it from a proportion to a percentage. How? Multiply each value of `turnout$newVAP` by 100

```
turnout$newVAPturnout <- turnout$newVAPturnout * 100
```

This multiplies each number within the vector by 100.

```
turnout$newVAPturnout
```

```
 [1] 52.03972 40.24522 52.53748 36.07845 49.72260 35.93884 54.04097 38.03086
 [9] 47.53376 34.83169 49.34211 35.82850 54.54777 55.67409
```

What is typical turnout?

```
mean(turnout$newVAPturnout)
```

```
[1] 45.45658
```

```
mean(turnout$newVEPturnout)
```

```
[1] 48.94937
```

We find that turnout based on the voting age population is lower than turnout based on the voting eligible population. This is a pattern that political scientists have examined, going back several decades. For example, in a 2001 article McDonald and Popkin show that is it the ineligible population that grew from the 1970s onward and not the population of people who simply prefer not to vote. (See more here.)

2.6 Comparing Presidential vs. Midterm turnout

How does turnout compare in presidential vs. midterm years? Sometimes using a single summary of turnout may obscure important underlying differences in the data. To detect these differences, we may want to summarize different parts of the data.

Oh dear. We need to extract specific years from the turnout data frame. Which rows contain the years we want?

```
turnout$year
```

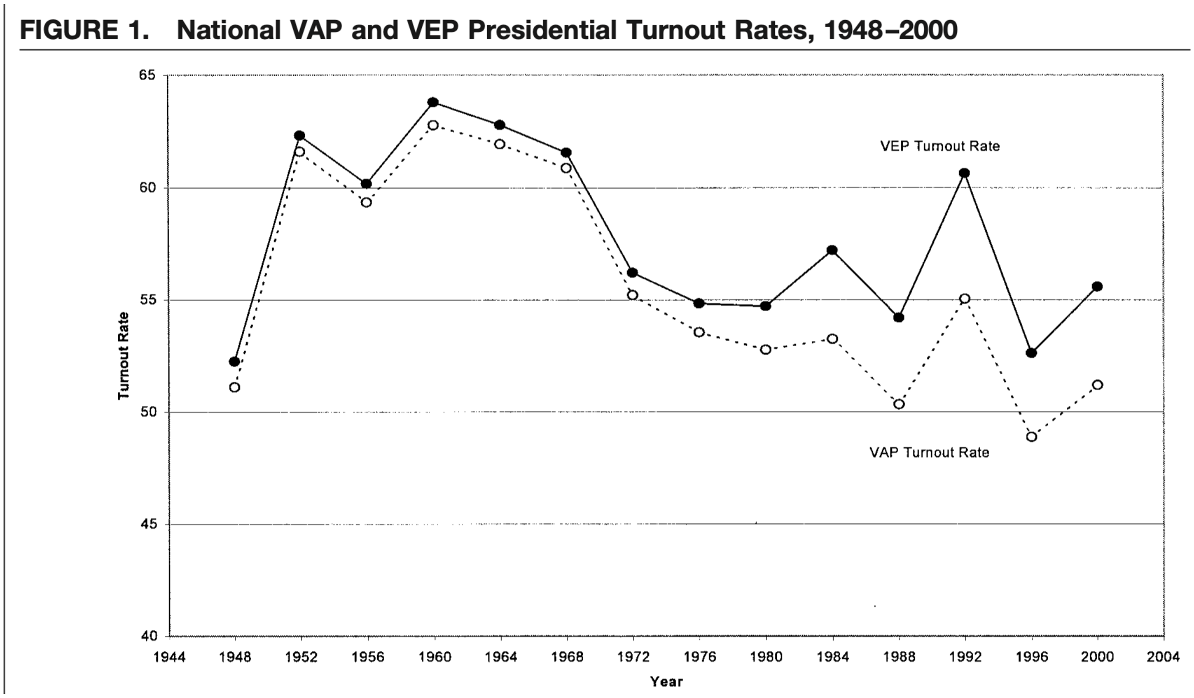


Figure 2.1: McDonald and Popkin 2001

```
[1] 1980 1982 1984 1986 1988 1990 1992 1994 1996 1998 2000 2002 2004 2008
```

Ok: rows 1,3,5,7,9,11,13,14 are the presidential. And rows 2,4,6,8,10,12 are midterms.

```
## we can extract all of these at once by using c()
turnout$year[c(1,3,5,7,9,11,13,14)] # presidential
```

```
[1] 1980 1984 1988 1992 1996 2000 2004 2008
```

Let's take the mean VEP turnout for presidential years.

```
mean(turnout$newVEPturnout[c(1,3,5,7,9,11,13,14)])
```

```
[1] 55.983
```

Let's take the mean VEP turnout for midterm years.

```
mean(turnout$newVEPturnout[c(2,4,6,8,10,12)])
```

```
[1] 39.5712
```

Let's take the difference by storing each mean and then subtracting

```
mean.VEP.pres <- mean(turnout$newVEPturnout[c(1,3,5,7,9,11,13,14)])
mean.VEP.mid <- mean(turnout$newVEPturnout[c(2,4,6,8,10,12)])
mean.VEP.pres -  mean.VEP.mid
```

```
[1] 16.41181
```

Presidential turnout, on average, is higher than midterm turnout.

2.6.1 R shortcut for writing vectors

Sometimes we write numbers that are in a predictable sequence (e.g., 1,2,3,4,5). In R, we have functions that prevent us from having to type each number when this is the case.

```
c(1,2,3,4,5) # is equivalent to:
```

```
[1] 1 2 3 4 5
```

```
1:5 # is equivalent to:
```

```
[1] 1 2 3 4 5
```

```
seq(from = 1, to = 5, by = 1)
```

```
[1] 1 2 3 4 5
```

We can use the last one to our advantage to extract the midterm years, which go by 2

```
mean(turnout$newVEPturnout[c(2,4,6,8,10,12)]) # is the same as
```

```
[1] 39.5712
```

```r
mean(turnout$newVEPturnout[seq(2, 12, 2)])
```

```
[1] 39.5712
```

Not a big deal now, but imagine if you had to write 100 numbers or 1 MILLION NUMBERS!

2.7 Creating dataframes from within R

While importing data from outside of R is the most common way to work with dataframes in R, you can also create dataframes from inside R. Ultimately, a dataframe just binds together multiple vectors / columns to create a rectangular object.

For example, let's say we want to create a dataframe with columns indicating just the midterm years and their VEP turnout. These correspond to the two vectors:

- `turnout$newVEPturnout[seq(2, 12, 2)]`
- `turnout$year[seq(2, 12, 2)]`

In R, you can create a rectangular `data.frame` object with the `data.frame` function.

- Within this function, you can make several entries that follow the syntax `colname = values`. We supply what we would like the name of the column to be, such as `midyear`, and then provide R with a set of values. We can then provide a comma and add more columns.
  - You just want to make sure each column has the same number of values.

```r
midtermdata <- data.frame(midyear = turnout$year[seq(2, 12, 2)],
                          VEPturnout = turnout$newVEPturnout[seq(2, 12, 2)])
```

You can supply the values for each column using objects or just vectors of raw numeric values like the below:

```r
midtermdata <- data.frame(midyear = c(1982, 1986, 1990, 1994, 1998, 2002),
                          VEPturnout = c(42.13701, 38.14115, 38.41895, 41.12625, 38.09316,
```

The result is a nice rectangular dataframe similar to what we loaded using the `turnout.csv` dataset from outside of R.

```r
midtermdata
```

```
  midyear VEPturnout
1    1982   42.13701
2    1986   38.14115
3    1990   38.41895
4    1994   41.12625
5    1998   38.09316
6    2002   39.51064
```

Now, because our dataframe has a different name. If we want to access columns from this dataframe, we start with `midterm$` followed by the variable name.

```
midtermdata$midyear
```

```
[1] 1982 1986 1990 1994 1998 2002
```

2.8 Wrapping Up Description

In this section, we have described voter turnout using multiple measures and types of elections. There are several other questions that political scientists may be interested in when it comes to voter turnout.

For example, during and following the 2020 elections, many states passed laws that changed election procedures: Ability to vote by mail, Ballot dropboxes, Length of early voting. What else?

- What effect (if any) do these laws have on voter turnout?

In the next section, we start to examine how to evaluate causal claims.

2.8.1 Summary of R tools

We have touched on a number of R tools thus far. Here is a summary of some of the key items to remember going forward:

- `setwd()`: sets the working directory in R, which tells R which folder on your computer contains the datasets or other R files where you will be working. You should get into the habit of setting your working directory each time you work in RStudio.
  - Can set this in the toolbar Session -> Set Working Directory -> Choose Directory, followed by clicking the "Open" button on the folder where you want to work.
  - Example: `setwd("~/Downloads/Data Science")`

- `##`: Hashtags are used to help annotate your code. Anything behind a hashtag is treated as plain text
- `+ - * /`: These are some of the mathematical operators you can use in R
  - You can also control which operations are performed first using `()` just like you would do with math outside of R. For example, try to compare the answer to `6 + 4 * 3` with `(6 + 4) * 3`
- `<-`: This is an assignment tool that allows us to store calculations, vectors, datasets, and more as *objects* in R.
  - Example: `sum53 <- 5 + 3` creates an object called `sum53` that stores the calculation on the right.
- `[]`: Brackets are used to extract specific components of objects we create. The number(s) inside the brackets tell us which entries to extract.
  - Example: `Outcome[2]` will tell us to extract the second entry in the object `Outcome`
  - Note: when we use datasets, the brackets will have two entries, one corresponding to the row entry and one corresponding to the column. Example `turnout[1,2]` means the entry in the first row and second column.

Functions We have already started using a number of *functions* in R, which are operations we ask R to do for us, such as creating vectors, importing data, or summarizing data by finding the mean, range, etc. Functions come in the same format, which starts with the function name followed by parentheses. Example: `mean()`. Each function then takes a particular input(s). When you "run" a line of code with a function, R applies the function to the input.

- `c()`: This is a function that combines a set of values into a vector in R. The values can be numbers or text items and should be separated by commas. If text, each text item should be in quotation marks.
  - Example: `Outcome <- c(3, 4, 6, 2, 1)`
  - Example: `People <- c("Sam", "Julie", "Mark")`
- `mean()`, `median()`, `min()`, `max()`, `range()`: These functions summarize vectors that are numeric/integers in nature.
  - Example `mean(Outcome)` takes the average of the values in the `Outcome` vector
- `read.csv()`: This function loads a rectangular .csv file into R as a data.frame
  - Example: `turnout <- read.csv("turnout.csv")`
  - Not all datasets will be .csv files. In the future, we will use other functions, such as `load()` or `read.dta()` to import datasets of different file types.

Dataframes

We have started working with dataframes in R. These objects are rectangular datasets that include a collection of vectors. Every column in a dataframe generally represents a different concept or "variable," while each row represents a different unit or "observation."

- `$`: When we are working with vectors that are inside of a dataframe (the columns inside of a dataframe), we use the `$` to access them.
  - Example: `turnout$year` will show us the values in the `year` column vector inside our `turnout` rectangular dataframe
- `nrow()`, `ncol()`, `dim()`, `head()`, `names()`: These functions help us explore the dataframes by telling us the number of rows and columns (the dimensions), giving us a sneak peek of the first 6 rows of the dataframe, or showing us the names of the variables (columns) in the data.
  - Example: `nrow(turnout)`

3 Causation with Experiments

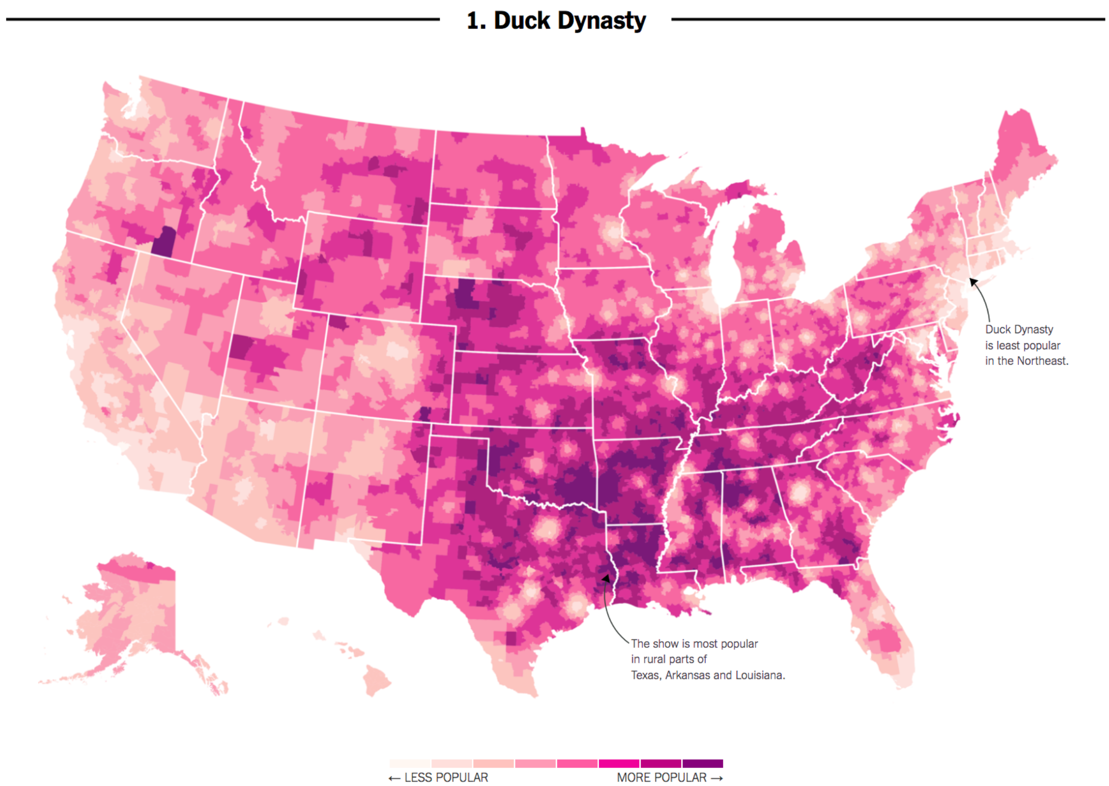
Recall that we said, four primary goals of social science include:

- **Describe** and measure
  - Has the U.S. population increased?
- **Explain**, evaluate, and recommend (study of causation)
  - Does expanding Medicaid improve health outcomes?
- **Predict**
  - Who will win the next election?
- **Discover**
  - How do policies diffuse across states?

In this section, we start to explore the goal of explanation–making causal claims.

3.1 What separates causation from correlation?

Here's an example. In 2016, researchers at the NY Times noticed that areas in the country where the television show *Duck Dynasty* was popular also tended to support Donald Trump at higher rates.



If we put our social scientist hat on, we might want to distinguish whether this is a causal or, more likely, just a correlational relationship:

- Correlation: Areas that watch Duck Dynasty are more likely to support Trump (degree to which two variables “move together”)
- Causality: Watching Duck Dynasty (vs. not watching) increases your support of Trump.

Causal Question: Does the manipulation of one factor (the treatment), (holding everything else constant), cause a change in an outcome?

3.1.1 Potential Outcomes Framework

When studying causal relationships, we distinguish two concepts:

- treatment: variable whose change may produce a change in the outcome (e.g., watching vs. not watching *Duck Dynasty*)
- outcome (Y): what may change as a result (e.g., their support for Trump)

We imagine two states of the world or “potential outcomes.”

- $Y(1)$: the outcome if the treatment is administered (e.g., watching the show)
- $Y(0)$: the outcome if the treatment is NOT administered or maybe something else is (e.g., not watching the show)

Political Science Example: How does voter turnout (Y) change as a result of varying whether someone receives a mail-in ballot (the treatment)?

- Y(sent a mail-in ballot): do you vote or not
- Y(not sent a mail-in ballot): do you vote or not

We compare your likelihood of turning out to vote in a world where you did receive a mail-in ballot vs. a counterfactual state of the world in which you did not receive a mail-in ballot, generally assuming that this is the only thing that is different between these two potential states of the world.

In many cases in social science, we might start by observing some connection in the real world. To make a causal claim, we then have to imagine what that counterfactual state of the world would be. Examples:

Causal Question: Does the minimum wage increase the unemployment rate?

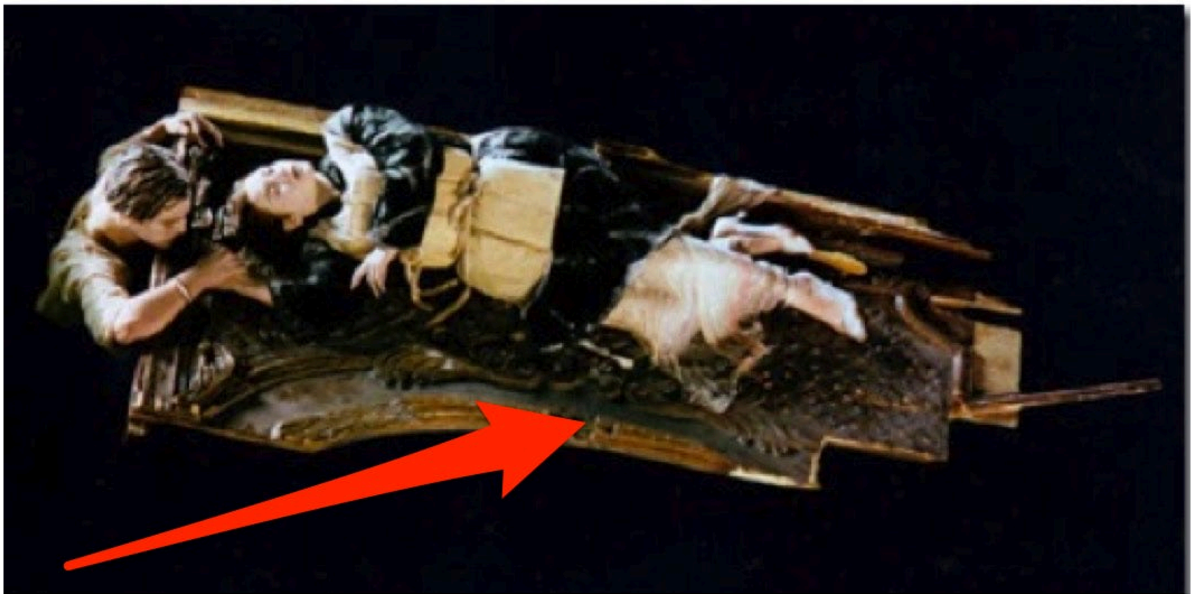
- (Hypothetical) Factual: An unemployment rate went up after the minimum wage increased
- Implied Counterfactual: Would the unemployment rate have gone up, had the minimum wage increase not occurred?

Causal Question: Does the gender of a political messenger influence the persuasiveness of the message?

- (Hypothetical) Factual: Suppose a political messenger perceived as a man had a somewhat persuasive effect delivering a message on abortion.
- Implied Counterfactual: Would a political messenger perceived as a woman have a similar or different persuasive effect?

We use causal logic all of the time outside of social science.

For example, many viewers get angry after watching the movie *Titanic* because they believe Jack did not have to die. We can place their claims in our causal framework:



- Outcome: Jack Surviving the Titanic
- Potential Outcomes in two states of the world
    - Rose did not share the floating door, and Jack died.
    - Counterfactual question: If Rose had shared the floating door, would Jack have lived?

In *Bit by Bit*, Matt Salganik notes that sometimes cause-and-effect questions are implicit. For example, in more general questions about maximization of some performance metric, we might want to compare several alternatives:

The question "What color should the donate button be on an NGO's website?" is really lots of questions about the effect of different button colors on donations.

- Factual: A voter donates some amount with a black button
- Counterfactual: What would a voter donate if the button were blue?
- Counterfactual: What would a voter donate if the button were red?

What other causal questions might social scientists or data scientists ask?

3.1.2 Causal Effects

When we are conducting a causal analysis, we will want to estimate a causal effect.

- Causal effects are all about ideal comparisons between treated vs. untreated

A causal effect is the change in the outcome Y that is caused by a change in the treatment variable.

- $Y(1) - Y(0)$ = causal effect or "treatment effect"
- e.g., Donation if contacted - Donation if not contacted

We often want to know the **average treatment effect** in some population, not just the causal effect for a single individual. Here, we might ask, on average, how much would our outcome change if our units were treated instead of untreated. To do so, we simply sum up all of the causal effects and divide them by the number of units in our population.

- $\frac{1}{N}\sum_{i=1}^{N}(Y_i(1) - Y_i(0))$ = "average treatment effect" (ATE)
  - e.g., Average donations if contacted - Average donations if not contacted

Note: If the math above is helpful, you can use it. If it is difficult to read, focus on the plain language definitions that go before it. The notation here is less important than the conceptual understanding.

3.1.3 Fundamental Problem of Causal Inference

The problem: Fundamental Problem of Causal Inference

What makes the evaluation of causal claims difficult, is that in the real world, we suffer from the fundamental problem of causal inference:

- For any individual, we only get to see (observe) the result from one state of the world
  - This makes that subtraction of potential outcomes impossible.

(Unless we are in Groundhog Day)

3.2 Randomized Controlled Trials

One approach for addressing the fundamental problem of causal inference is to simulate two potential states of the world through random assignment: Randomized Controlled Trials / Experiments

Experiments approximate an ideal factual vs. counterfactual comparison

- We randomly assign one group to receive a "treatment" and another not to receive a treatment (the control)
  - There can be more than two groups. The key is that each group varies (is manipulated) in some way.

- When treatment assignment is **randomized**, the only thing that distinguishes the treatment group from the control group, besides the treatment itself, is chance.

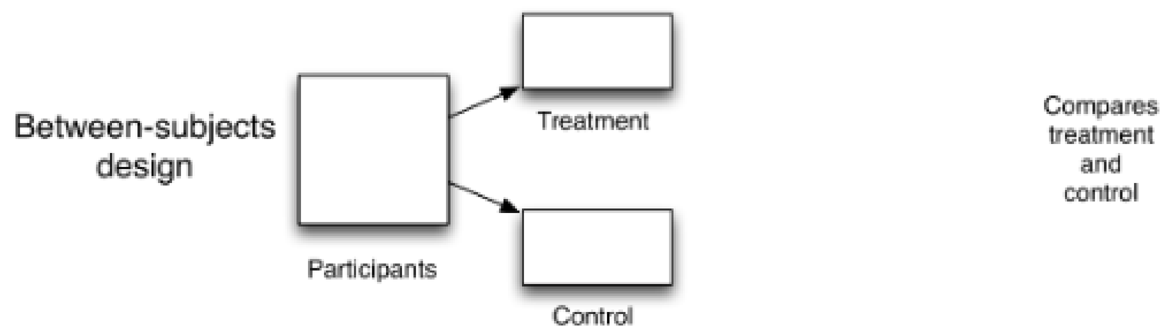


Figure 3.1: Salganik Bit by Bit Chapter 4.4

This allows us to compare the average outcomes between groups in order to estimate our causal effects (more on this below).

3.2.1 Experiments: Why Randomize?

Randomization is essential for being able to *identify* and *isolate* the causal effect of the treatment on the outcome.

Without randomization, there may be several reasons why two groups differ beyond the treatment of interest.

- For example, if we randomly assigned half of Rutgers seniors to go to a Sabrina Carpenter concert and half to go to a Bruce Springsteen concert we would expect the groups to have about equal proportions of female students, average age, racial composition, majors, etc.
  - (If we didn't randomly assign, and just let people "select" into watching a particular concert, the groups could look very different.)

But because we randomized assignment, on average, we'd expect the two groups to be identical except for the treatment– in this case, which concert they attended.

- Great news! This means any differences in the outcomes between the two groups can be attributed to the treatment. So if we wanted to see if going to a Bruce Springsteen show leads people to hold more favorable opinions of New Jersey, we could compare the average attitudes toward NJ among seniors who attended the show vs. attended the Carpenter show.

3.2.2 Experiments: How to Analyze

Difference in Means: We compare each group's average outcome by subtracting one from the other to estimate the average treatment effect (ATE) aka the average causal effect of the treatment.

- $\widehat{ATE} = \bar{Y}(treatment) - \bar{Y}(control)$

This is an estimate of, on average, how much our outcome would change if units went from being untreated to treated.

- E.g., on average how much a person donates to a campaign if contacted by phone compared to if not contacted by phone.

3.2.3 Ingredients of an Experiment

From *Bit by Bit*

4.2 What are experiments?

Randomized controlled experiments have four main ingredients: recruitment of participants, randomization of treatment, delivery of treatment, and measurement of outcomes.

For every experiment, you should be able to

- State the causal question or relationship of interest
- Describe how the experiment will be implemented (e.g., recruitment of subjects)
- Identify and describe the randomization into treatment group(s) and control group and what happens in each group
- Identify the outcome of interest, how it is measured
- Evaluate the relevant comparison (between two different experimental conditions)

We will turn to an example in the next section.

3.3 Application: Is there racial discrimination in the labor market?

Marianne Bertrand and Sendhil Mullainathan. 2004. "Are Emily and Greg more employable than Lakisha and Jamal? A field experiment on labor market discrimination."

"We perform a field experiment to measure racial discrimination in the labor market. We respond with fictitious resumes to help-wanted ads in Boston and Chicago newspapers."

- Recruitment: Construct resumes to send to ads
- Randomization: To manipulate perception of race, each resume is (randomly) assigned
- Treatment: either a very African American sounding name
- Control: or a very White sounding name
- Outcome: Does the resume receive a callback?
- Comparison: Callback rates for African American (sounding) names vs. White (sounding) names (the difference in means between groups)

For a video explainer of the code in this section, see below. The video only discusses the code. Use the notes and lecture discussion for additional context. (Via youtube, you can speed up the playback to 1.5 or 2x speed.)

https://www.youtube.com/watch?v=LeJkRydMruM

Let's load the data. Note: When we have variables that are text-based categories, we may want to tell R to treat these "strings" of text information as factor variables, a particular type of variable that represents data as a set of nominal (unordered) or ordinal (ordered) categories. We do this with the `stringsAsFactors` argument.

```
resume <- read.csv("resume.csv", stringsAsFactors = T)
```

```
resume <- read.csv("https://raw.githubusercontent.com/ktmccabe/teachingdata/main/resume.cs
                   stringsAsFactors = T)
```

Variables and Description

- `firstname`: first name of the fictitious job applicant
- `sex`: sex of applicant (female or male)
- `race`: race of applicant (black or white)
- `call`: whether a callback was made (1 = yes, 0 = no)

The data contain 4870 resumes and 4 variables.

```
nrow(resume) # number of rows
```

```
[1] 4870
```

```r
ncol(resume) # number of columns
```

```
[1] 4
```

```r
dim(resume) # number of rows and columns
```

```
[1] 4870    4
```

Note: These data look a little different from what we used last week. For example, the `sex` and `race` variables contain words, not numbers.

```r
head(resume)
```

```
  firstname    sex  race call
1   Allison female white    0
2   Kristen female white    0
3   Lakisha female black    0
4   Latonya female black    0
5    Carrie female white    0
6       Jay   male white    0
```

3.3.1 Variable classes

We can check the class of each variable: Look, we have a new type, a "factor" variable.

```r
class(resume$firstname)
```

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

```r
class(resume$sex)
```

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

```r
class(resume$race)
```

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

```
class(resume$call)
```

```
[1] "integer"
```

We have now encountered `numeric, character`, and `factor` vectors and/or variables in R. Note: This is simply how R understands them. Sometimes R can get it wrong. For example, if we write:

```
somenumbers <- c("1", "3", "4")
class(somenumbers)
```

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

Because we put our numbers in quotation marks, R thinks the values in `somenumbers` are text. The number "3" might as well be the word "blue" for all R knows. Fortunately, we can easily switch between classes.

```
somenumbers <- as.numeric(somenumbers)
class(somenumbers)
```

```
[1] "numeric"
```

Here, we used `as.numeric()` to overwrite and change the character vector into a numeric vector.

Rules of Thumb

- Usually, we want `character` variables to store text (e.g., open-ended survey responses)
- We want `numeric` variables to store numbers.
- Usually, we want `factor` variables to store categories.
  - Within R, factor variables assign a number to each category, which is given a label or `level` in the form of text.
  - Categories might be ordinal or "ordered" (e.g., Very likely, Somewhat likely, Not likely) or
  - Unordered (e.g., "male", "female")
  - R won't know if a factor variable is ordered or unordered. Alas, we have to be smarter than R.
  - R might think you have a character variable when you want it to be a factor or the reverse.
    * That's when `as.factor()` and `as.character()` are useful.
- Always check `class()` to find out the variable type

3.4 Making tables

A nice thing about numeric and factor variables is we can use the `table` command to see how many observations in our data fall into each category or numerical value.

```
## Example: how many black vs. white sounding resumes
table(resume$race)
```

```
black white
 2435  2435
```

As mentioned, `factor` variables have levels:

```
levels(resume$race)
```

```
[1] "black" "white"
```

3.4.1 Crosstabulation

We can also use the `table` command to show a crosstabulation: a table that displays the frequency of observations across two variables.

```
## Example: how many black vs. white sounding resumes by call backs
## We can label the two dimensions of the table with the =
table(calledback = resume$call, race = resume$race)
```

```
          race
calledback black white
         0  2278  2200
         1   157   235
```

3.5 Conditional Means

Recall how to take a mean of a variable in our data. For example, let's take the mean of the variable `call`.

```
mean(resume$call)
```

```
[1] 0.08049281
```

This gives us the average callbacks (or callback rate) for everyone in our data. In experiments, we want to take the mean for a specific group within our data– the treatment group, and then the mean for the control group.

Somehow, we have to identify, within our data, which rows were part of the treatment group and which were a part of the control group. In this study, we want to identify resumes with an assigned name perceived to be black vs. perceived to be white. This is in our `race` variable.

We will cover a couple of tools to do this, with the first being `tapply`.

To find how the average of one variable (e.g., our outcome- the callback rate) varies across different categories of our factor variable, we use `tapply()`.

```
## take the mean of input1 by categories of input2
## mean of the call variable conducted separately by race
tapply(resume$call, INDEX=resume$race, mean)
```

```
     black      white 
0.06447639 0.09650924 
```

This tells us the callback rate for each group of people in our data. That's not the only way to do this, however. We can also use the tools below.

3.6 Relational Operators in R

Goal: Compare callback rates for white sounding names to black sounding names, so we need to be able to filter by race.

Good news: We have several relational operators in R that evaluate logical statements:

- `==, <, >, <=, >=, !=`
- We have a statement and R evaluates it as `TRUE` or `FALSE`

```
## for each observation, does the value of race equal "black"?
resume$race == "black"
```

By putting this logical statement within `[ ]`, we are asking R to take the `mean()` of the variable `resume$call` for the subset of observations for which this logical statement is `TRUE`.

```
mean(resume$call[resume$race == "black"])
```

```
[1] 0.06447639
```

Ultimately, each of these paths has led us to a place where we can estimate the average treatment effect by calculation the difference in means: the difference in callback rates for black and white applicants.

We said the ATE = $\bar{Y}(treatment) - \bar{Y}(control)$

```
ate <- mean(resume$call[resume$race == "black"]) -
  mean(resume$call[resume$race == "white"])
ate
```

```
[1] -0.03203285
```

How can we interpret this? Do white applicants have an advantage?

3.7 Subsetting data in R

Subsetting Dataframes in R

Maybe we are interested in differences in callbacks for females. One approach for looking at the treatment effect for female applicants, only, is to subset our data to include only female names.

- To do this, we will assign a new `data.frame` object that keeps only those rows where `sex == "female"` and retains all columns
- Below are two approaches for this subsetting, one that uses brackets and one that uses the `subset` function

```
## option one
females <- resume[resume$sex == "female", ]
## option two using subset()- preferred
females <- subset(resume, sex == "female")
```

Now that we have subset the data, this simplifies estimating the ATE for female applicants only.

We said the ATE = $\bar{Y}(treatment) - \bar{Y}(control)$

```
ate.females <- mean(females$call[females$race == "black"]) -
  mean(females$call[females$race == "white"])
ate.females
```

```
[1] -0.03264689
```

3.7.1 Getting Boooooooolean

We can make this slightly more complex by adding more criteria. Let's say we wanted to know the callback rates for just female black (sounding) names.

- R allows use to use & (and) and | (or)

```
femaleblack <- subset(resume, sex == "female" & race == "black")
```

We could now find the callback rate for Black females using the tools from above:

```
mean(femaleblack$call)
```

```
[1] 0.06627784
```

3.8 Creating New Variables using Conditional statements

Note: We will cover each of these tools in this section, time permitting. Otherwise, we will return to it in a future section.

We can instead create a new variable in our main dataframe. Let's make a variable that takes the value 1 if a name is female and black sounding and 0, otherwise

```
# Initialize a new variable called femaleblackname
resume$femaleblackname <- NA
# Assign a 1 to our new variable where sex is female and race is black
resume$femaleblackname[resume$sex == "female" & resume$race == "black"] <- 1
# Assign a 0 if sex is not female OR if race is not black
resume$femaleblackname[resume$sex != "female" | resume$race != "black"] <- 0
```

We can check our work

```
table(name = resume$firstname, femaleblack = resume$femaleblackname)
```

```
          femaleblack
name         0   1
  Aisha      0 180
  Allison  232   0
```

```
Anne      242   0
Brad       63   0
Brendan    65   0
Brett      59   0
Carrie    168   0
Darnell    42   0
Ebony       0 208
Emily     227   0
Geoffrey   59   0
Greg       51   0
Hakim      55   0
Jamal      61   0
Jay        67   0
Jermaine   52   0
Jill      203   0
Kareem     64   0
Keisha      0 183
Kenya       0 196
Kristen   213   0
Lakisha     0 200
Latonya     0 230
Latoya      0 226
Laurie    195   0
Leroy      64   0
Matthew    67   0
Meredith 187   0
Neil       76   0
Rasheed    67   0
Sarah     193   0
Tamika      0 256
Tanisha     0 207
Todd       68   0
Tremayne   69   0
Tyrone     75   0
```

Let's say we wanted to know the callback rates for just female black (sounding) names.

```
mean(femaleblack$call)
```

```
[1] 0.06627784
```

```r
mean(resume$call[resume$femaleblackname == 1])
```

```
[1] 0.06627784
```

BINGO: two ways to do the same thing.

3.8.1 ifelse statements

Remember how we created the variable `femaleblack`, well there is another way to do that in R using what are called conditional statements with `ifelse()`.

- Can be read: If this relational statement is `TRUE`, I assign you A, otherwise I assign you B

```r
resume$femaleblackname <- ifelse(resume$sex == "female" &
                                   resume$race == "black", 1, 0)
```

Can be read: If sex is female and race is black, give the observation in the new variable a 1, otherwise give it a 0.

Like most things, we can also get more complicated here. Let's say we wanted to create a variable that indicated both race and sex.

- Can be read: If this relational statement is `TRUE`, I assign you A,
- Otherwise if this second relational statement is `TRUE`, I assign you B,
- Otherwise if this third relational statement is `TRUE`, I assign you C,
- Otherwise I assign you D

```r
resume$racesex <- ifelse(resume$sex == "female" &
                                   resume$race == "black", "FemaleBlack",
                         ifelse(resume$sex == "female" &
                                   resume$race == "white", "FemaleWhite",
                                ifelse(resume$sex == "male" &
                                   resume$race == "white", "MaleWhite", "MaleBlack")))
```

Note: what you assign can be numeric or text.

3.9 Types of Experiments

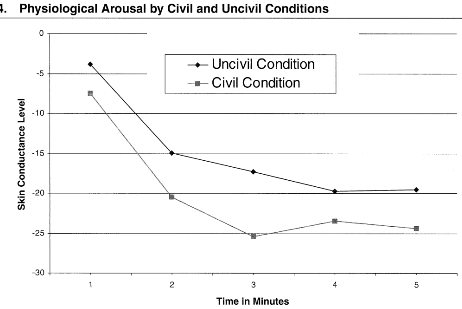
Experiments can vary:

- Setting: Lab, Survey, Field
- Mode: Analog vs. Digital
- And in Validity
    - Internal: were the processes conducted in a correct, reliable way?
    - External: can we generalize from the experiment to the real world, or would the results change?
    - Context: Would people act the same way outside of the experiment?
    - Recruitment: Are the people in our experiment representative of the people we care about?
    - Construct
        * Treatment: Is the experimental treatment similar to what people see in the real world?
        * Outcome: Is the outcome something we care about in the real world? Are we measuring it in a realistic, accurate way?

Review *Bit by Bit* chapter 4 for more examples of social science experiments.

Example: Televised Incivility, Trust and Emotions (Mutz and Reeves)

FIGURE 4. Physiological Arousal by Civil and Uncivil Conditions



Source: Experiment 3.
Note: Points represent averages based on civil and uncivil means. Incivility produced significantly higher levels of physiological arousal ($t = 14.38$, $p < .001$).

Participants sat alone in a room with electrodes attached to their hands to measure skin conductance. Subjects viewed 20 minutes of a political debate created for the experiment, which varied in civility and politeness. Results showed respondents had more of an emotional response to the uncivil condition and expressed less trust in politicians.

Example: Online Survey Experiment

Audience Costs (Tomz)

A country sent its military to take over a neighboring country. The attacking country was led by a [dictator, who invaded ***OR*** democratically elected government, which invaded] [to get more power and resources ***OR*** because of a longstanding historical feud.

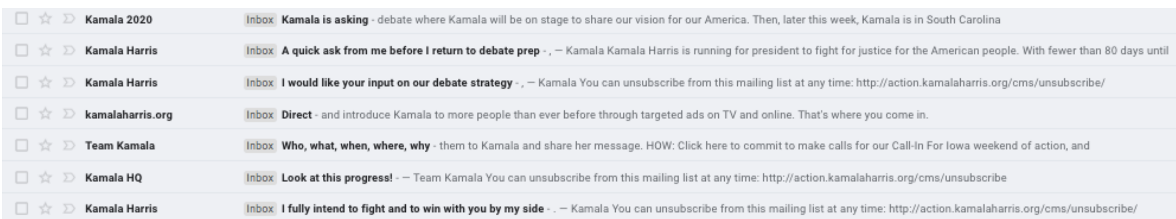
The attacking country had a [strong military, so it would ***OR*** weak military, so it would not] have taken a major effort for the United States to help push them out.

A victory by the attacking country would [hurt ***OR*** not affect] the safety and economy of the United States.

- Participants provided a different version of the vignette above, and a reaction by the president
- Presidential approval varies depending on the president's response and the nature of the situation

Example: Digital Field Experiments in Campaigns

Example: A/B Testing in Campaigns



Emails are virtually costless. Very easy to ask: Are people more likely to open them with X subject or Y subject or Z subject?

3.10 Wrapping Up Causation with Experiments

In this section, we have discussed what it means to make a causal claim, why it is essentially impossible to make causal comparisons in real life due to the fundamental problem of causal inferences, and how experiments can help us make comparisons that approximate our causal ideals.

In the next section, we start to examine how to visualize data.

3.10.1 Summary of R tools in this section

Here are some of the R tools we used in this section:

- `table()`: this function summarizes the frequency of observations that take a particular value. The input is one or more variables in your data.

- E.g., `table(resume$sex)` or `table(resume$sex, resume$call)`

- `tapply()`: this function applies a given operation like `mean` to whichever variable is in the first position, separately or "conditionally" by different values of the variable in the second "index" position.
  - E.g., `tapply(resume$call, INDEX=resume$race, mean)` finds the average callbacks for applicants separately for different races of applicants in the data.
- `== > < >= <= !=`: Relational operators help us set up "logical statements" in R that are evaluated as `TRUE` or `FALSE`
  - E.g., `resume$race == "black"` evaluates whether for each observation in the race column is "black" in which case the statement is `TRUE` or not black, in which case the statement is `FALSE`
  - E.g., `resume$call < 1` evaluates whether for each observation in the call column has a value less than one in which case the statement is `TRUE` or not less than 1, in which case the statement is `FALSE`
  - We can then isolate certain parts of columns using relational operators and the brackets `[]`. For example we can take the mean callbacks for applicants who are black using `mean(resume$call[resume$race == "black"])`
- `&` and `|`: These are boolean operators that allow us to combine multiple relational operators using an AND statement (`&`) or an OR statement `|`. Note the bar is a bar that is usually above your backslash key and not a capitalized i.
  - E.g., `mean(resume$call[resume$race == "black" & resume$sex == "female"])`
- `subset()`: We can subset whole rows of our data using this function. It takes two inputs– the first is the name of the original dataframe, and the second is a relational statement. Usually we store this output in R by assigning the results to a new object, a dataframe that contains only those rows for which the logical statement using the relational operators is true. E.g., `females <- subset(resume, sex == "female")` subsets our data to keep only those rows where applicants were female.