CORE121 Lab 1

Ian McDonald

6/28/2021

```
knitr::opts_chunk$set(echo = T,
                      results = "hide")
source("lab_setup.R")
```

Overview

This starter exercise shows you how to manipulate and manage a basic dataset using the R programming environment. We will use the interactive tool RStudio to produce results with the core R language.
This exercise will mimic the results you saw in Excel Lab #1. All you will do is copy some R statements from this page into an R Script on the server. From there, you will execute the script and observe the results.

Don't be concerned if you're unfamiliar with programming in general or R specifically, The purpose of using R in learn a few basic ideas about managing and calculating data in a two-dimensional data set.

You will copy and paste the content from the gray boxes into a file you'll name lab1.R. You'll execute the file, note some results into lab1.R, download it, then submit it the Google Classroom.

##R and RStudio

Note that R is a programming language that you will run with the RStudio application. RStudio is just one of many ways you can develop work with the R programming language, but it's the most popular, and like R it's openly sourced and completely free to anyone who wants to use it. For now, you only need to concern yourself with RStudio, and there is a good chance it will be the only way you ever interact with R.

Other Numbers classes are using a similar kind of language called Python. Like R, there are many tools that support work in Python. There are several popular Python development tools.

Setup

We will set up our work environment with these six steps.

Step 1: Log into the RStudio server from https://datasci.watzek.cloud/. Follow the links until you see the RStudio display that looks like this:

We will create an RStudio project, which will help keep our work organized and isolated.

Step 2: Follow this menu sequence: *File - New Project*

And you will see a New Project wizard that looks like this:

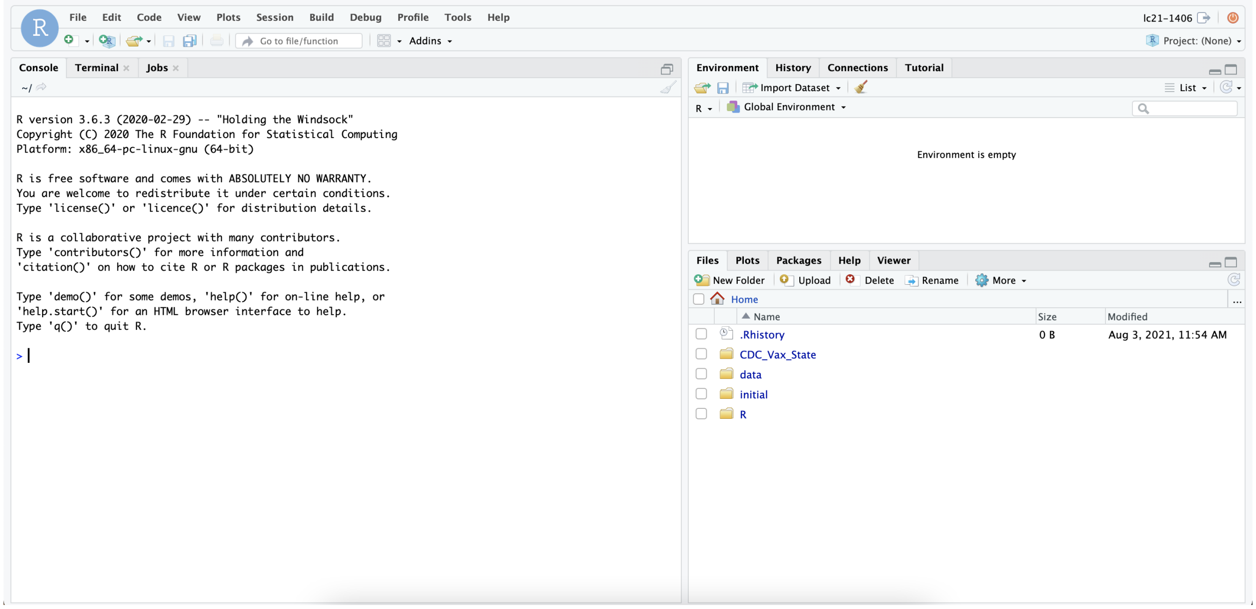
Click on "New Project" and then enter "Lab 1" as your directory name. Leave the two boxes unchecked.

Step 3: We will store our commands into a file called Lab_1.R. Enter this file sequence:

File - New File - New R Script.

Your screen should look similar to this:

Step 4: Save your untitled script using: *File - Save (or Save As)*

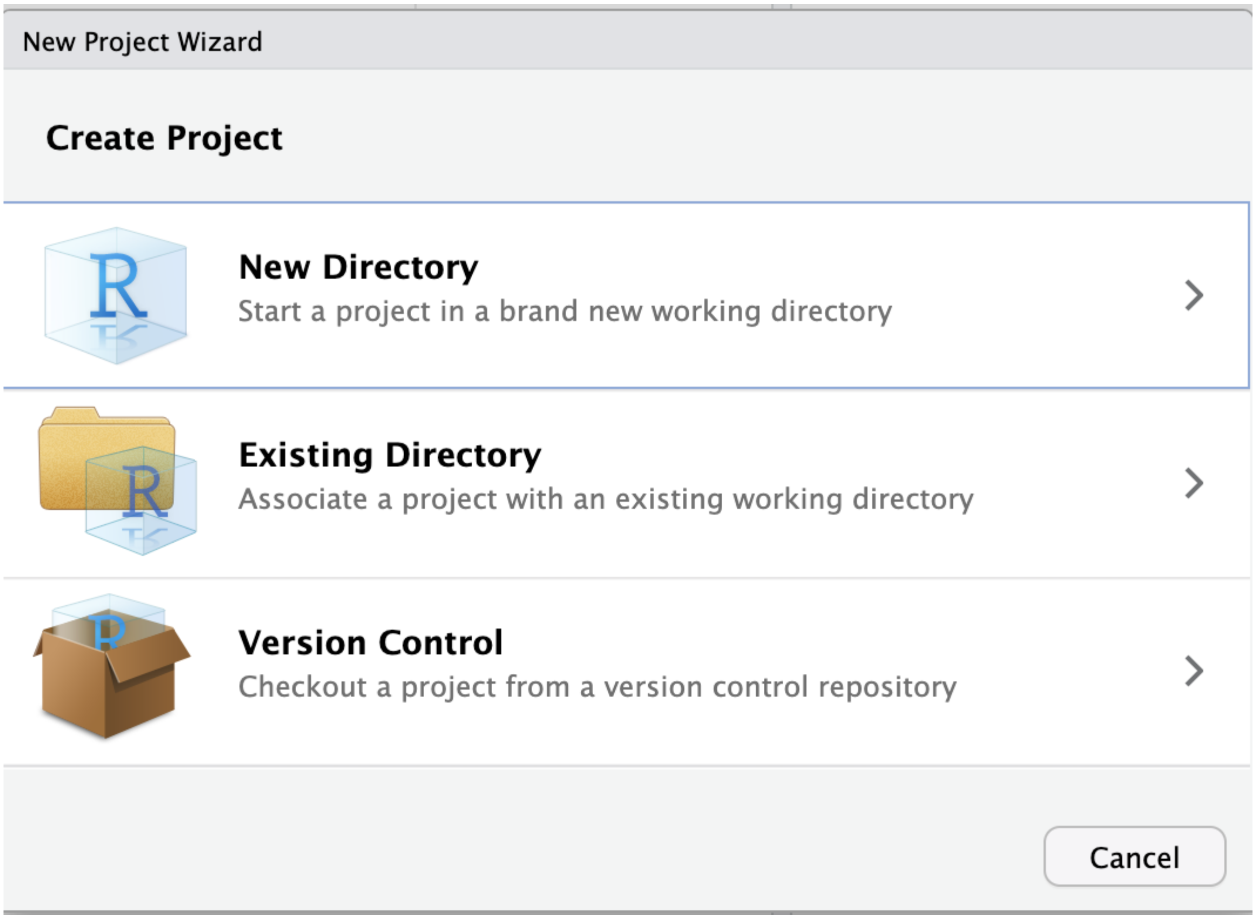


```
R version 3.6.3 (2020-02-29) -- "Holding the Windsock"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-pc-linux-gnu (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Environment is empty

Name	Size	Modified
.Rhistory	0 B	Aug 3, 2021, 11:54 AM
CDC_Vax_State		
data		
initial		
R		

Figure 1: RStudio Screen Capture



New Project Wizard

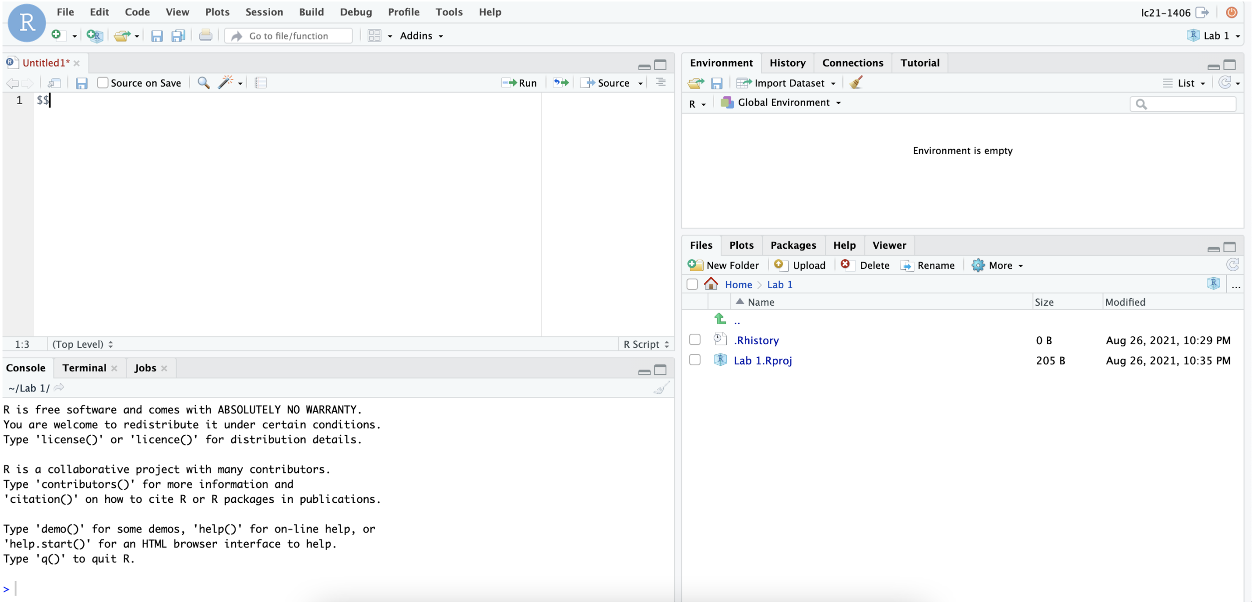
Create Project

New Directory
Start a project in a brand new working directory

Existing Directory
Associate a project with an existing working directory

Version Control
Checkout a project from a version control repository

Figure 2: RStudio Screen Capture



```
$$
```

Environment is empty

Name	Size	Modified
..		
.Rhistory	0 B	Aug 26, 2021, 10:29 PM
Lab 1.Rproj	205 B	Aug 26, 2021, 10:35 PM

```
R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

Figure 3: RStudio Screen Capture

and give it the name **Lab_1**. Note that the underscore isn't strictly necessary, but it could simplify things to avoid a blank space if you need to copy it or move it around later.

Step 5: Copy the one zip file in the directory listed as RStudio Lab 1 in the classroom Google Drive folder. That file lives in this link: https://bit.ly/3DgytCp. Download it to a temporary location where you can retrieve it easily.

Step 6: Take the zip file and upload it into your RStudio directory. Click on the Upload button on the lower right pane of your screen, then choose the zip file. All of its contents will automatically appear in your file list.

Now we ready to load R commands into our script and run them.

Building the R Script

I'll give you a bit of explanation of the commands as you paste them and run them.

The first thing were going to do is load some special R content stored something called a library. The library contains programs and data that your script can retrieve later. R, like all programming languages, is extended beyond its very basic capability by the work of programmers from anywhere (including yourself, perhaps, someday).

The sequence is:

- Copy the lines from the grey box into your R script in the upper left corner of your RStudio session.
- Select the pasted lines, and then click on the "Run" button with the green arrow in front of it. Sometimes you'll see things change on your screen.

Copy and paste these five statements into your script (don't worry if they don't appear in a grey box.)

library(readxl)
library(tigris)
library(sf)

library(tidyverse)
library(plotly)

```
library(readxl)
library(tigris)
```

```
## To enable
## caching of data, set `options(tigris_use_cache = TRUE)` in your R script or .Rprofile.
```

```
library(sf)
```

```
## Linking to GEOS 3.8.1, GDAL 3.2.1, PROJ 7.2.1
```

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.4     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x tidyr::extract()   masks magrittr::extract()
## x dplyr::filter()    masks stats::filter()
## x dplyr::lag()       masks stats::lag()
## x purrr::set_names() masks magrittr::set_names()
```

```
library(plotly)
```

```
## 
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
## 
##     last_plot
```

```
## The following object is masked from 'package:stats':
## 
##     filter
```

```
## The following object is masked from 'package:graphics':
## 
##     layout
```

```
lab_1_data <- read_excel("Excel Lab 1.xlsx")
```

Notice that you store your commands in one window of your display, in the upper left corner. That window is called Source.

When you execute something, the immediate results appear in the bottom left window called the console. If you want, you can enter commands direct in the console at the ">" prompt. Once in a great while you may want to try that but the normal sequence is to enter content in the source, and when you want to run something, see the results in the console.

You should make sure you save your work every time you change something, or at the very least save it frequently.

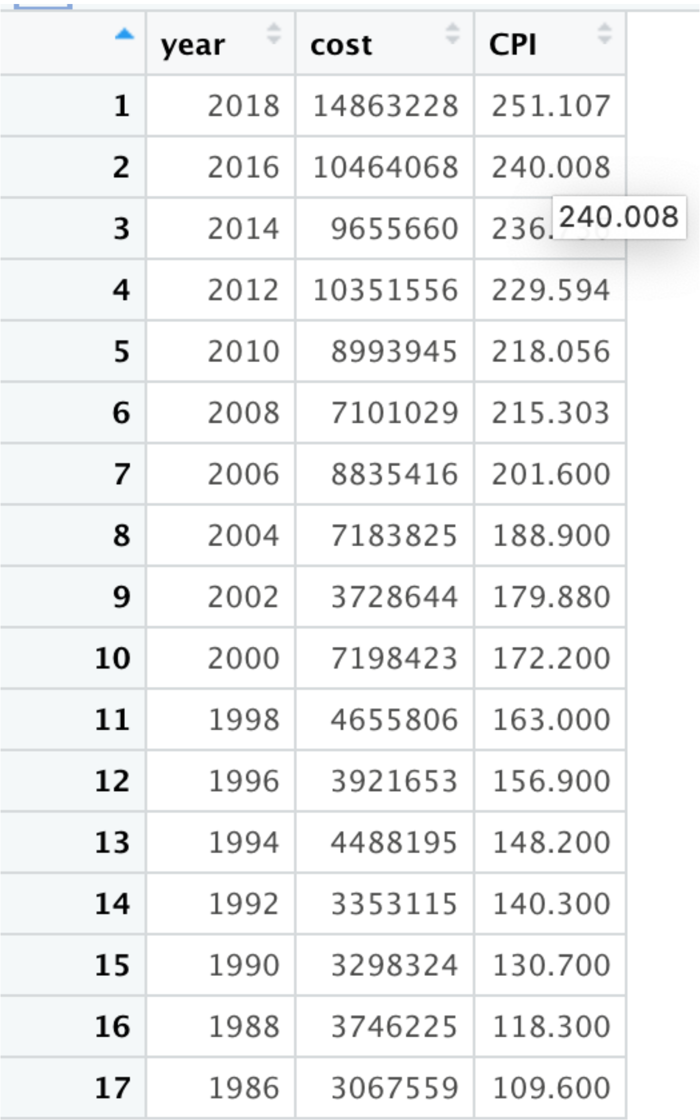
Let's upload some more code. By the way, you can leave as many blank lines as you want in a script, and blank lines will help makes things more readable when placed artfully, just like anything else you write.

The line in this box will read the data from the first worksheet of your Excel file and create an R data table

```
lab_1_data <- read_excel("Excel Lab 1.xlsx")
```

Notice in the upper right corner a new object appeared called *lab_1_data*. Click on the blue arrow beside it, and you will see all the column names from the spreadsheet appear in a list.

In fact, click on the *lab_1_data* name itself and you will see this appear in your source window:



	year	cost	CPI
1	2018	14863228	251.107
2	2016	10464068	240.008
3	2014	9655660	236.736
4	2012	10351556	229.594
5	2010	8993945	218.056
6	2008	7101029	215.303
7	2006	8835416	201.600
8	2004	7183825	188.900
9	2002	3728644	179.880
10	2000	7198423	172.200
11	1998	4655806	163.000
12	1996	3921653	156.900
13	1994	4488195	148.200
14	1992	3353115	140.300
15	1990	3298324	130.700
16	1988	3746225	118.300
17	1986	3067559	109.600

Figure 4: RStudio Screen Capture

This display doesn't give you all the functionality you enjoy with Excel, but you can certainly see the contents. As we go further, you're going to refer to this data object.

Notice something else: everything you have just entered into your script is something you can run from scratch anytime you want. Excel doesn't give you that ability!

You now have two tabs in the upper left console window. Click on the Lab_1.R tab and you'll return to your script. Let's add things in the order of the steps in the Excel lab.

For example, I might discover that I need to change some of the names or labels or content in my new data table. You'll end up referring to these column names, so you might as well make them simple and recognizable.

In this chunk below, the magic word is **mutate**. Here, I use mutate to change some of the entries in the column labled "state". Notice the first part of this statement:

```
lab_1_data <- lab_1_data %>%
        mutate(state = ifelse(state != "District of Columbia", str_to_title(state), state))
```

Notice this part of the chunk:
lab_1_data <- lab_1_data %>%

Here, the code is saying "read from my lab_1_data table and when you've made the changes, store it back into lab_1_data." I can also choose, instead, to store the results into a different object, or just print the results on my screen.

In this line, we just fixed a problem from the Excel worksheet where the "District Of Columbia" should have been "District of Columbia".

One of R's features is its ability, **to a fault**, to support different ways of handling tasks. I could have done this instead (you don't need to copy in this chunk but it won't hurt anything if you do):

```
lab_1_data <- read_excel("Excel Lab 1.xlsx") %>%
        mutate(state = ifelse(state != "District of Columbia", str_to_title(state), state))
```

and I would have accomplished both tasks in one statement. But it's possible I might want to take things one step at a time, usually because I want to see the results from each step before I consolidate them. Or perhaps because I want to document my work in a particular way.

Now let's get down to business. We know from the first lab that I will be interested in creating sums of various columns.

Notice that the column names are funky and inconsistent. We can make the column names shorter and more usable.

```
lab_1_data <- lab_1_data %>%
        rename(pop_2014 = `2014 Citizen Population`,
               voted_2014 = `2014 Total voted`,
               pop_2018 = `2018 citizen Population`,
               voted_2018 = `2018 total voted`)
```

In R, I can store those calculations in a different data table and use them later.

```
sums_of_lab1_data <- lab_1_data %>%
        summarize(pop_2014 = sum(pop_2014),
                  voted_2014 = sum(voted_2014),
                  pop_2014 = sum(voted_2014),
                  voted_2018 = sum(voted_2018))
```

But R is full of alternatives that can condense the statements, such as this one, where we tell it to "add everything that is like a numeric column".

```
sums_of_lab1_data <- lab_1_data %>%
        summarize(across(where(is.numeric), sum))
```

If you click on sums_of_lab1_data in the upper right, you'll see some values that will look familiar from the Excel Lab.

Let's generate some new columns that will resemble what I generated in the original lab. But notice that I don't have to copy formulas into cells with all the risks that entails. I just make issue these lines:

```
lab_1_data <- lab_1_data %>%
        mutate(turnout_pct_increase = (voted_2018 - voted_2014) / voted_2014 * 100) %>%
        arrange(desc(turnout_pct_increase))
```

Now click on your lab_1_data tab. That object reveals content we generated in Excel by copying a bunch of cells.

Now create two new columns that present the percentage of the population that voted in each state for each year (Total Voted/Citizen Population).

```
lab_1_data <- lab_1_data %>%
     mutate(turnout_pct_2014 = voted_2014 / pop_2014 * 100) %>%
     mutate(turnout_pct_2018 = voted_2018 / pop_2018 * 100)
```

Now create an additional variable, called "pct pop change" that subtracts the 2014 percentage from the 2018 percentage. Sort this list from largest to smallest. Utah continues to top the list, but the percentage change is much smaller in magnitude. Discuss in your group the difference between these two percent change calculations. Which do you think should be reported in newspaper coverage?

```
lab_1_data <- lab_1_data %>%
        mutate(pct_pop_change = turnout_pct_2018 - turnout_pct_2014) %>%
        arrange(desc(pct_pop_change))

lab_1_data %>%
        select(state, pct_pop_change)
```

Do not forget to save your script often. Do it right now.

This time, the results appear in the console. You didn't update your lab_1_data object, and notice that you just see this:

lab_1_data %>% select some, with column names in parentheses.

And you get a usable display in the console. But you didn't change anything in your source data table.

The next question we want to ask is "What is the total turnout in the United States for each year?" To do that we need to add up all the states for each year.

```
lab_1_data <- lab_1_data %>%
        mutate(pct_total_2018 = voted_2018 / sum(voted_2018) * 100)

lab_1_data <- lab_1_data %>%
        mutate(pct_dif_us_mean_2018 = sum(voted_2018)/sum(pop_2018)*100 - turnout_pct_2018)
```

```
lab_1_data %>% summarise(median(voted_2014), median(pop_2014), median(voted_2018), median(pop_2018))
```

```
lab_1_data %>% summarize((sum(voted_2018) - sum(voted_2014)) / sum(voted_2014)) * 100
```

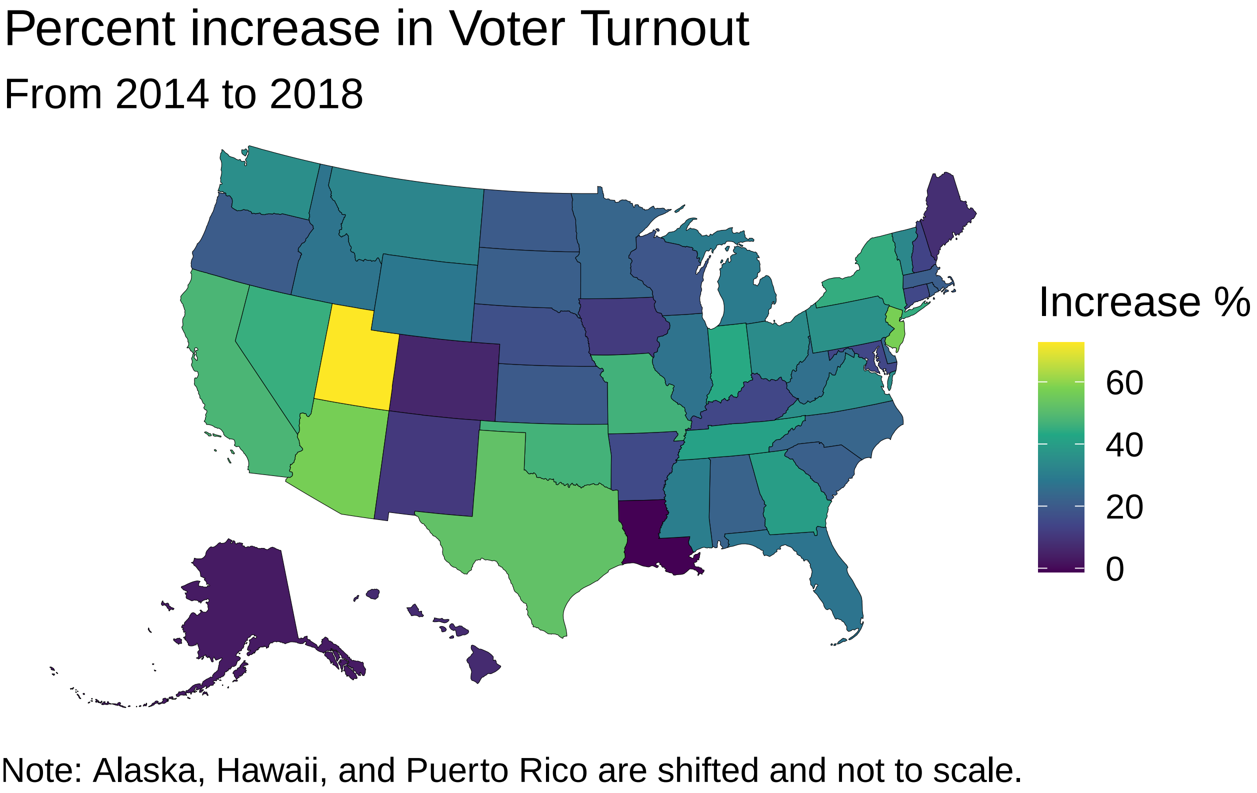
```
lab_1_data %>% summarize(sum(voted_2014) / sum(pop_2014) * 100)
lab_1_data %>% summarize(sum(voted_2018) / sum(pop_2018) * 100)
```

```
lab_1_data %>% summarize(sum(voted_2018) / sum(pop_2018) * 100) - lab_1_data %>% summarize(sum(voted_20
```

```
lab_1_data <- lab_1_data %>%
        arrange(desc(turnout_pct_increase))
```

Now let's make a simple map of this.

```
state_codes <- st_codes_f(full_name = lab_1_data$state)
lab_1_data <- inner_join(lab_1_data, state_codes, by=c("state" = "st_name"))
us_states <- states(cb = TRUE, resolution = "20m") %>%
     shift_geometry()
lab_1_sf <- inner_join(us_states, lab_1_data, by=c("STUSPS" = "stcd"))

m <- lab_1_sf %>%
ggplot(aes(text = str_c(STUSPS," ",format(turnout_pct_increase, digits = 3)))) +
     geom_sf(aes(fill = turnout_pct_increase), color = "black", size = 0.1) +
     scale_fill_viridis_c() +
     theme_void(base_size = 16) +
     labs(title = "Percent increase in Voter Turnout",
          subtitle = "From 2014 to 2018",
          fill = "Increase %",
          caption = "Note: Alaska, Hawaii, and Puerto Rico are shifted and not to scale.") +
     theme(plot.title = element_text(hjust = 0))

m
```


Percent increase in Voter Turnout
From 2014 to 2018
Increase %
60
40
20
0
Note: Alaska, Hawaii, and Puerto Rico are shifted and not to scale.

And we can make the mouse "hover" over individual states display its data:

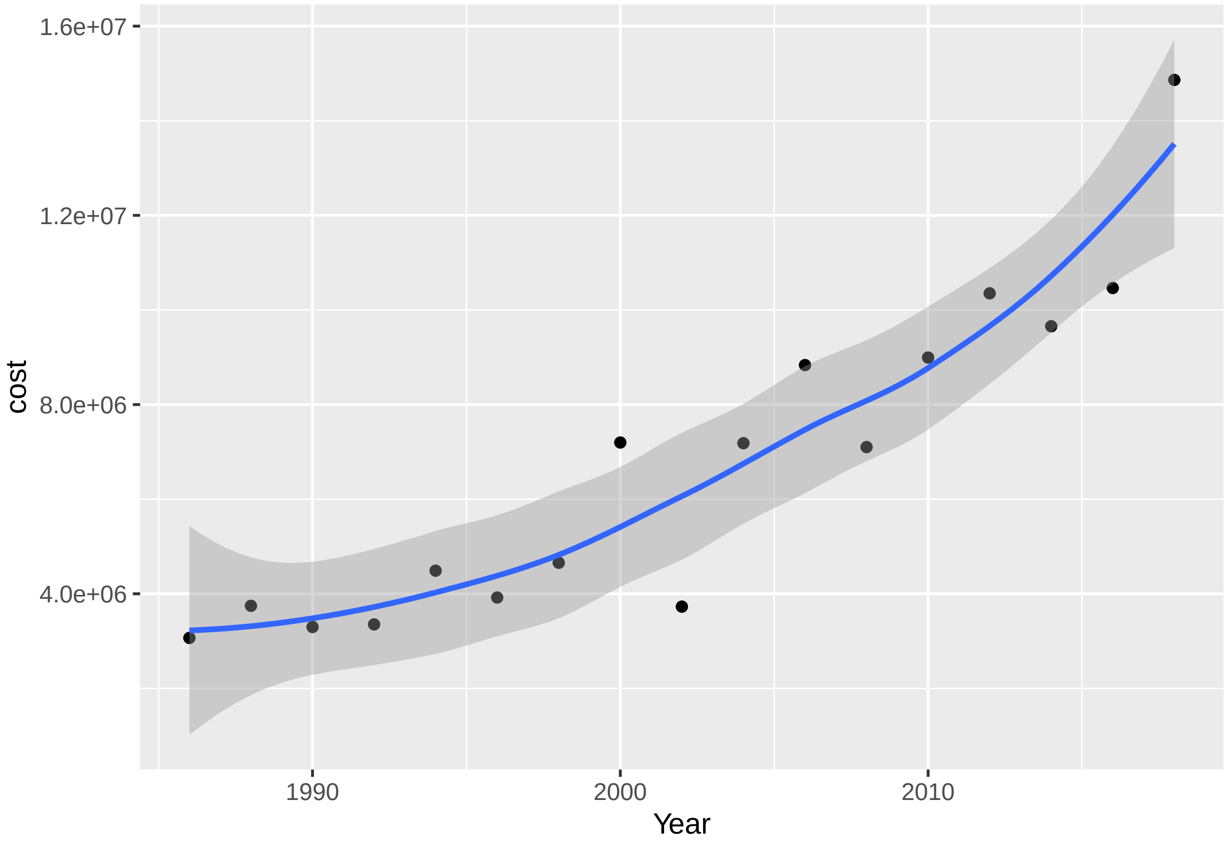
```
gg_2 <- ggplotly(m, tooltip = "text")
gg_2 %>%
    style(
      hoveron = "fills",
      # override the color mapping
      line.color = toRGB("gray40"),
      # don't apply these style rules to the first trace, which is the background graticule/grid
      traces = seq.int(2, length(gg_2$x$data))
    ) %>%
    hide_legend()
```

```
## A line object has been specified, but lines is not in the mode
## Adding lines to the mode...
```

Part 2: Campaign Finance

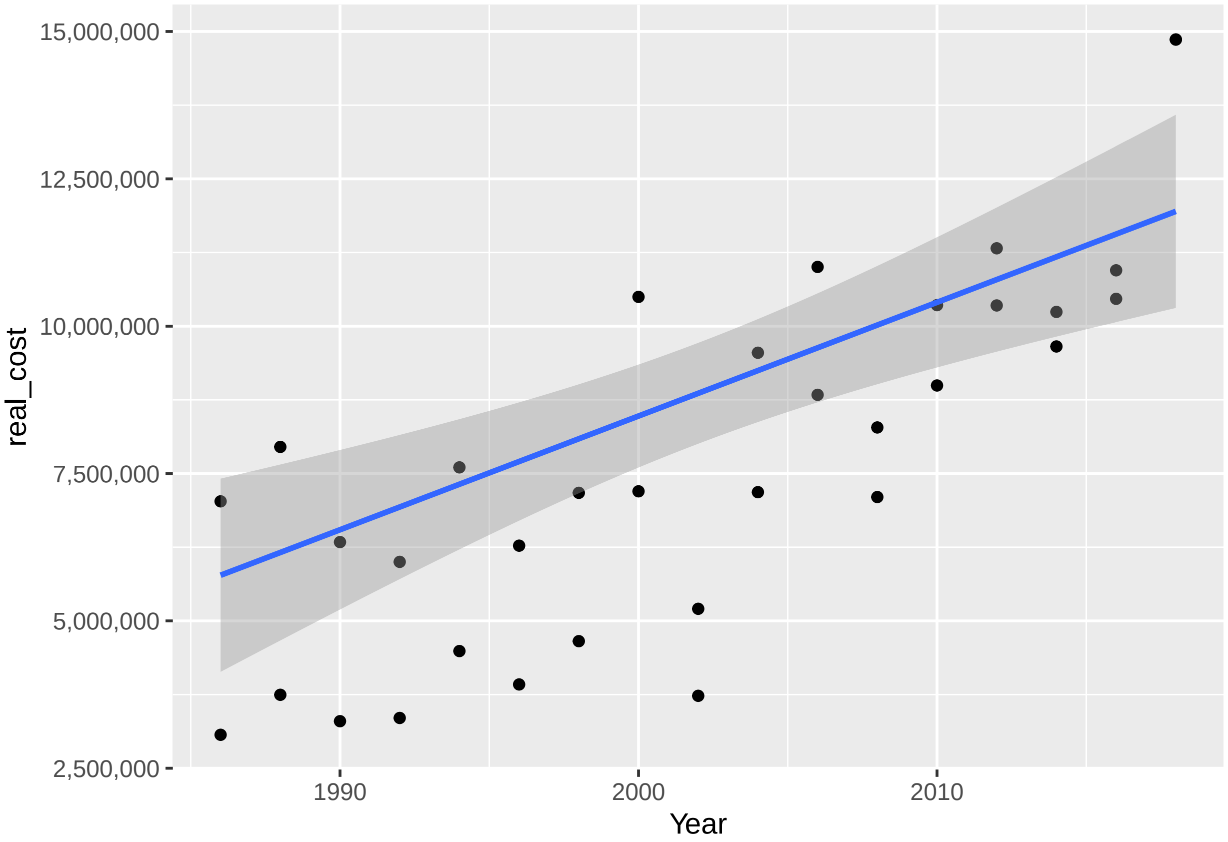
```
lab_2_data <- read_excel("Excel Lab 1.xlsx", sheet = "Campaign Finance") %>%
        rename(cost = `Cost of Senate Elections (Winner)`) %>%
        arrange(Year)

cpi_2018 <- lab_2_data[lab_2_data$Year == 2018,][["CPI"]]

lab_2_data <- lab_2_data %>% mutate(inflation_rate = (CPI - lag(CPI)) / lag(CPI) * 100) %>%
        mutate(real_cost = cpi_2018 / CPI * cost)


lab_2_data %>% ggplot(aes(x = Year, y = cost)) +
        geom_point() +
        stat_smooth(method = loess)
```

```
## `geom_smooth()` using formula 'y ~ x'
```



```
lab_2_data %>% ggplot(aes(x = Year, y = real_cost)) +
        geom_point() +
        geom_point(aes(x = Year, y = cost)) +
        stat_smooth(method = lm, fullrange = TRUE) +

        scale_y_continuous(labels = scales::label_comma())
```

```
## `geom_smooth()` using formula 'y ~ x'
```


15,000,000
12,500,000
10,000,000
7,500,000
5,000,000
2,500,000
real_cost
1990
2000
2010
Year