

Lab 1: Getting Started with R

Rafael Campos-Gottardo*

January 30 & 31, 2025

Welcome to POLI311

POLI311 builds on the introduction to R provided in POLI210. The focus of this course is on running and interpreting linear regression models in R. R is a free open source programming language designed for statistical computing. Despite having a slightly higher learning curve than other statistical software programs used in the social sciences (e.g. STATA, SPSS, etc.) R has a number of advantages. First, R is free which makes it more accessible. Second, the R is an open source programming language which means that the R community has created a number of packages designed to perform specific statistical analyses. These packages range from packages designed for graphing ([ggplot2](#)) to more specialized packages designed for specific tasks (e.g. [calculating adjusted marginal means for conjoint experiments](#)). Many quantitative political science papers published in the *American Political Science Review* now include R packages (See [Mehlhoff, 2023](#) for an example). Finally, R allows political scientists to share their statistical analyses which increases the reproducibility of research in political science. However, as a programming language R comes with a greater learning curve than other statistics software.

Therefore, in these labs we will cover how to use R for the regression analyses commonly performed in political science. Since we only have 50 minutes together each week it is important to come to lab prepared and attend office hours if you need additional clarification.

Learning Objectives

- Open R studio and create an R project.
- Perform basic mathematical operations in R.
- Create your first R objects.
- Download a data set from *mycourses* and load the data into R as a `data.frame`.
- Understand the structure of a `data.frame`.
- Use the `table()` function to inspect the variables in a `data.frame`.

Before Lab

Before your lab this week please ensure that you have completed the following:

- Read this lab guide.
- Download [R](#) and [RStudio](#).
- Familiarize yourself with [R Projects](#). (If you need a refresher on how to create an R project please refer to the following [video](#).)
- Familiarize yourself with your computer's file structure. When working with R it is important to know where files you download go and where you want to store your work for this class.

*This lab guide builds on those created by Aaron Erlich and Colin Scott. I would also like to acknowledge Brendan Szendro, Nicholas Vaxelaire, Guila Cohen, and Natalie Delmonte for their help developing this lab guide.

Performing Basic Mathematical Operations in R

Like most programming languages R can be used as a calculator. In R Studio the console can be used to perform basic mathematical operations. The console is located in the bottom left pane in R studio and has a `>` and cursor. To perform basic mathematical simply type what mathematical operation you want into the console and press **Enter**.

Basic mathematical operations include addition:

```
3 + 5
```

```
## [1] 8
```

R begins each output with a number in square brackets.

Practice Problem 1: What does the number `[1]` in this output indicate?

You can also perform more complex mathematical operations like multiplication and division:

```
4 * 3
```

```
## [1] 12
```

```
5 / 6
```

```
## [1] 0.8333333
```

R also ignores spaces when executing a command. However, it is important to write code that is readable to others (your professors, TAs, and journal reviewers need to be able to read your code). Therefore, it is best practice to separate numbers and operators with a space.

When performing mathematical operations R respects the order of operations. As a result the following operations will yield different results:

```
(3 + 4) * 5^3 / 7
```

```
## [1] 125
```

```
3 + 4 * 5^3 / 7
```

```
## [1] 74.42857
```

Functions in R

In addition to the following basic mathematical operators (+, -, *, /, ^, %%¹, etc.) R comes with a number of built-in functions to perform mathematical or statistical operations and manipulate data. Functions are denoted by a word (or words) followed by brackets (e.g., `function()`). For example, the `sqrt()` function provides the square root.

```
sqrt(16)
```

```
## [1] 4
```

```
sqrt(80)
```

```
## [1] 8.944272
```

Practice Problem 2: How would you calculate the following formula in R: $\sqrt{\frac{14 \times 54^2}{(55-14)+98 \times 2}}$?

¹This operator gives you the remainder from division.

Objects in R

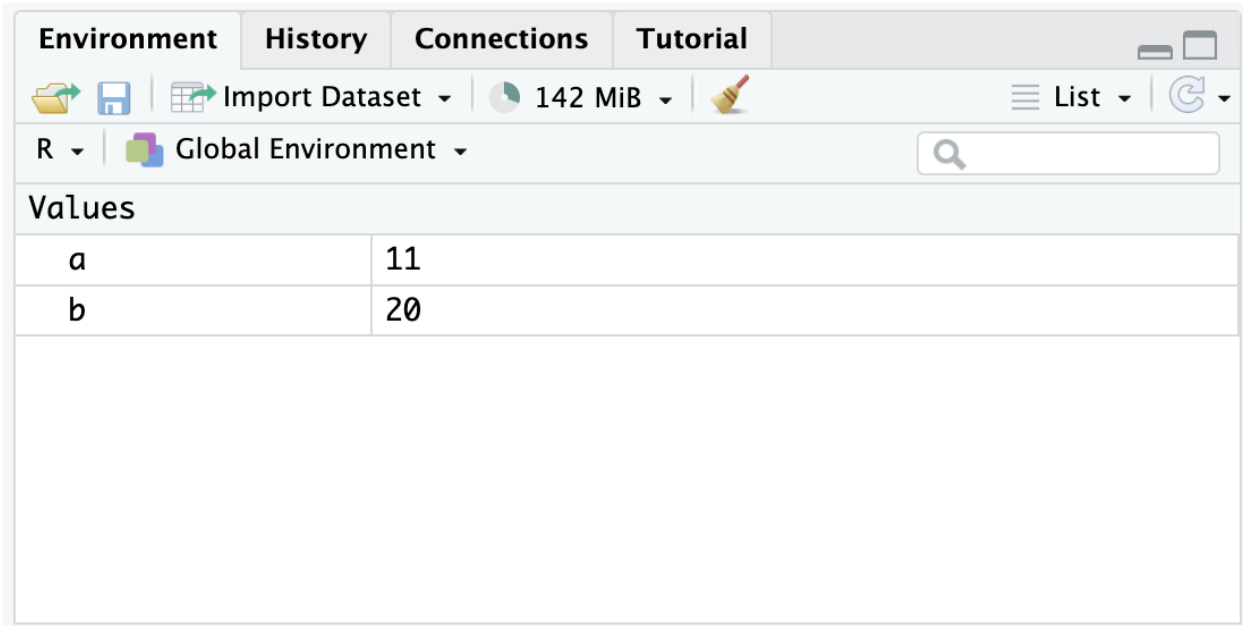
R is an object oriented programming language (OOP) which means that information is stored as objects. In order to store information as an object R uses the `<-` assignment operator (for those familiar with `Python` = also works as an assignment operator in R). One feature of R is that you left (`<-`) and right assign (`->`).

The results from mathematical operations can be assigned to objects:

```
a <- 5 + 6
```

```
4 * 5 -> b
```

Notice that when you create an object it appears in the environment in the top right panel in R Studio.



The environment viewer allows you to see all objects and dataframes that are currently in your R environment. If you try to call an object that is not in the R environment you will receive an error.

For example if we tried to call the object `d` we would receive the following error: `Error: object 'd' not found`²

If you want more practice with objects refer to the *introduction to objects* lab handout.

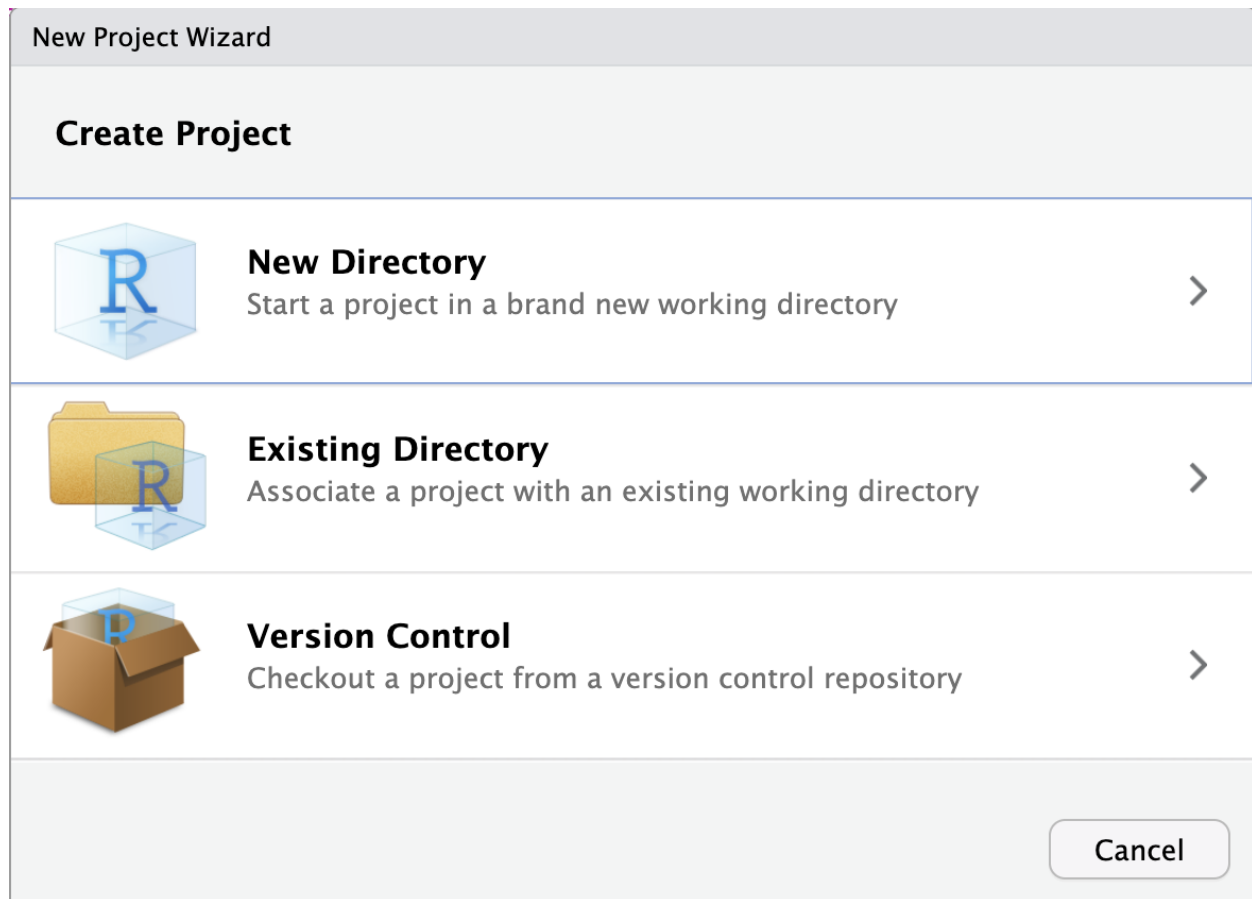
R Projects

Now that we have learned how to create our first objects in R we want to create an R project. An R project helps us organize our files and data. An RStudio project links each R environment to a working directory. Using R projects make it easier to organize and move your work in R and makes your code more reproducible.³

TO create an R Project open R Studio and click **File > New Project...** Once you click **New Project...** the following window will appear:

²One of the most important skills when working in R is understanding and debugging error messages. We will discuss error messages during labs and you can refer to the *Troubleshooting R* section in this lab handout for more tips.

³Some researchers prefer using the `setwd()` function. However, this method makes it harder to work in R and makes your code less reproducible. If you are interested in using `setwd()` [this video](#) explains how to use `setwd()` and why you should use R Projects instead of `setwd()`.










For now we can select **New Directory** which will create a new folder for our lab files (If you select **Existing Directory** R Studio will create a new **R.Proj** file within an existing folder). Once you select **New Directory** the following window will appear:

New Project Wizard

Back

Project Type

 New Project	>
 R Package	>
 Shiny Application	>
 Quarto Project	>
 Quarto Website	>
 Quarto Blog	>
 Quarto Book	>


Cancel

In this window select **New Project**. Finally, you will be given the option to name your new project and select where you want your new directory to be saved on your computer (this is why its important to have a file management system on your computer).

New Project Wizard

Back

Create New Project



Directory name:

POLI311_Labs

Create project as subdirectory of:

~/Documents/Year M2/POLI311

Browse...

☐ Create a git repository

☐ Use renv with this project

☐ Open in new session

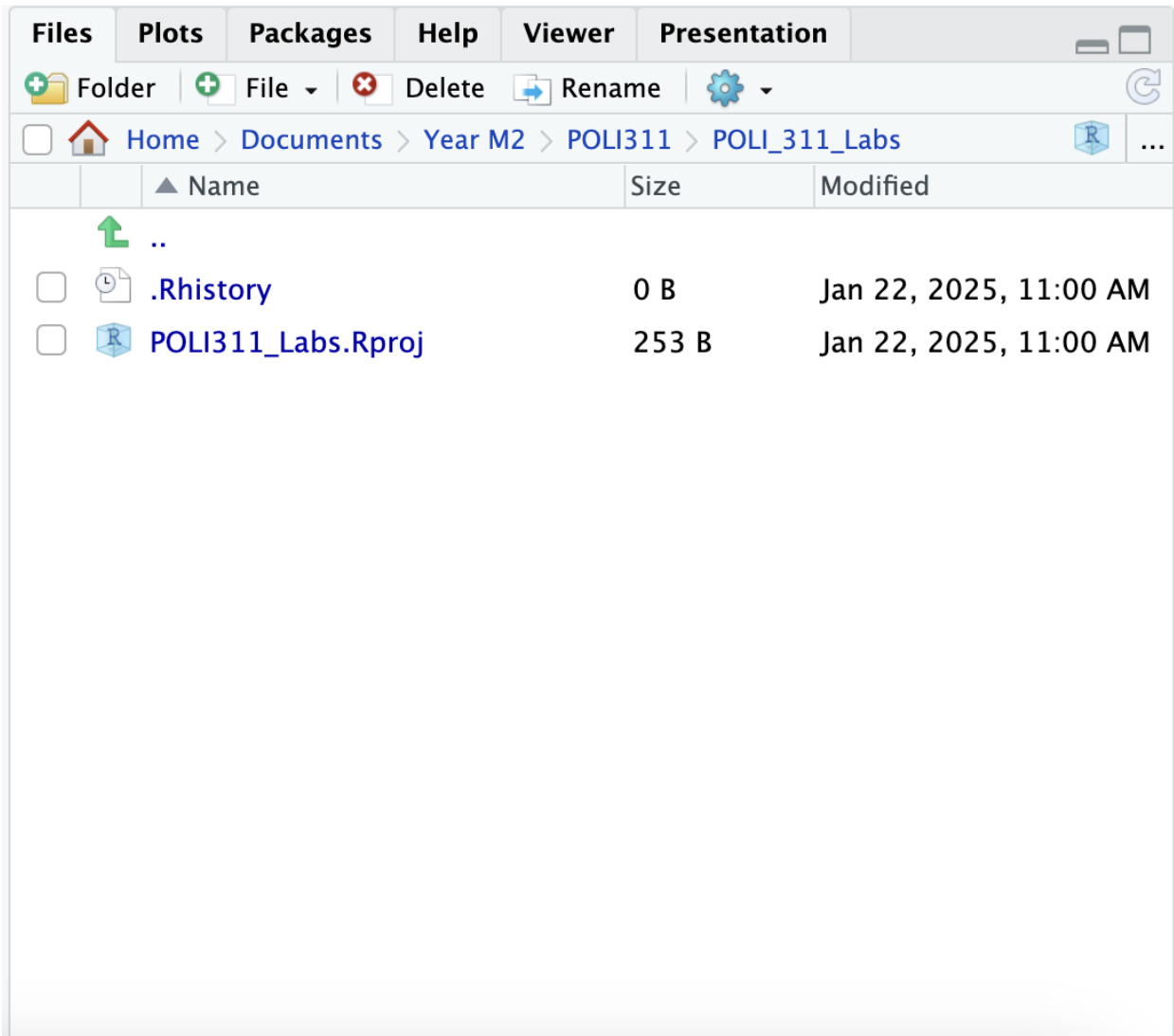
Create Project

Cancel

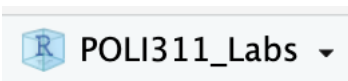
In the **Directory Name** box you can give your project a name, for now let's call our project **POLI311_Labs**. Now R Studio will create a new project that you can use for your labs in this class (I would suggest creating a separate R Project for your final assignment for this class).

Practice Problem 3: Create a new R Project in your POLI311 class folder called **POLI311_Final_Assignment**.

Once you have created a new R Project you will notice a new file in the **Files** panel in the bottom right of R Studio. This is your **.RProj** file which is how you will open your R Project in the future.



Additionally, in the top right corner of R Studio is name of your R Project (**NOTE** if the name of the R Project in the top right corner does not match the .RProj file you see in the **Files** panel you are not in the right R Project and the files you see in the file viewer cannot be opened in R).



Importing Data

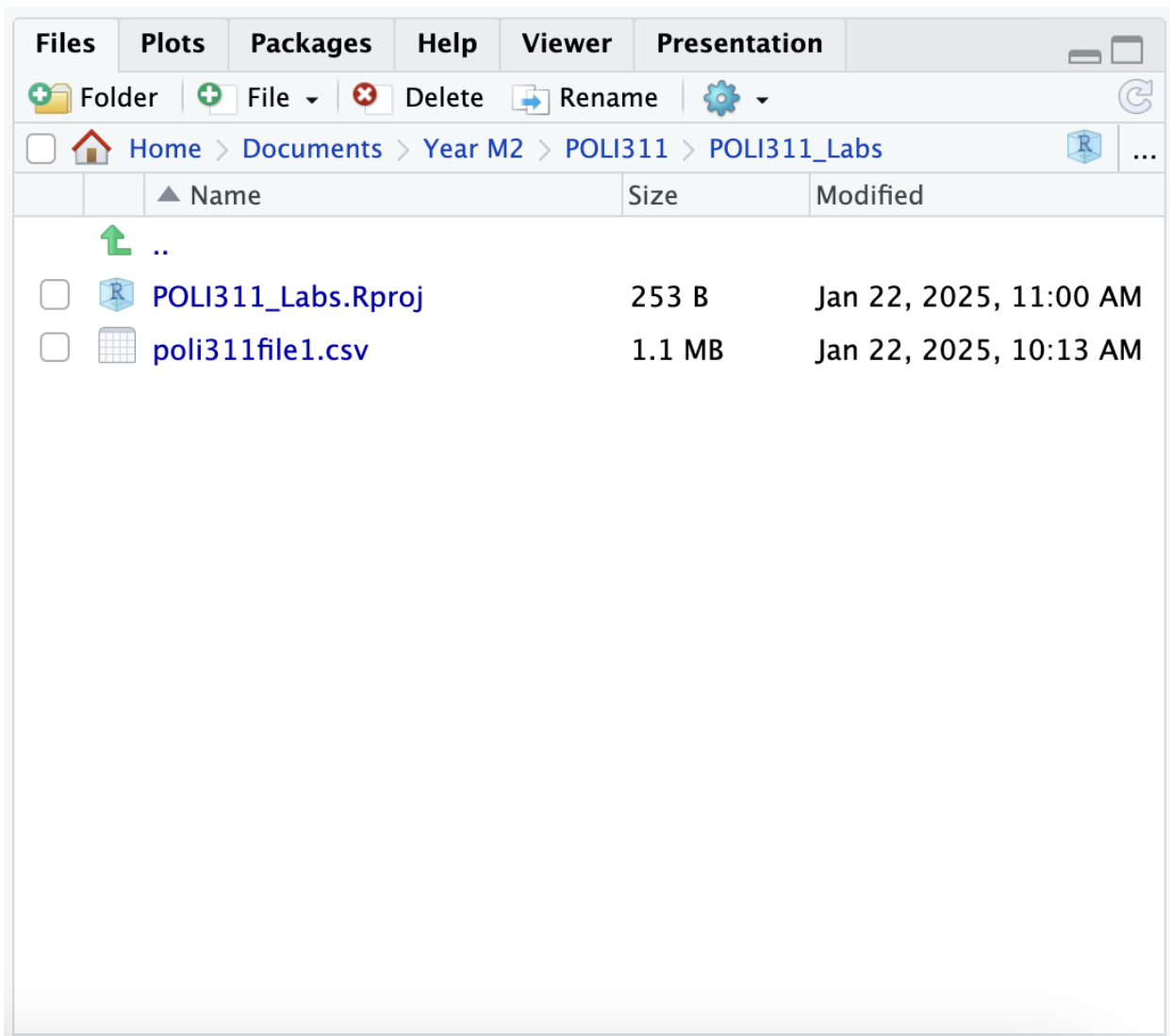
Now that we have created an R Project we can upload data into R. For this lab will be using the `poli311file1.csv` from *mycourses*. Download this file and then locate it in your downloads folder (which is usually found through **Finder/Macintosh HD** on Mac and **File Explorer** on Windows). Next copy `poli311file1.csv` and paste it into your `Poli311_Labs` folder. If you need help locating your `Poli311_Labs` folder you can use the `getwd()` function in R Studio.

```
getwd()
```

```
## [1] "/Users/rafaelc-g/Documents/Year M2/POLI311/POLI311_lab_guides"
```

Which will give you the file path of your R Project folder (**NOTE:** If you are R Studio Cloud the [following](#)

[video](#) explains how to upload files to the cloud). Once you have placed the data file into your project's directory you can return to R Studio. You will now notice that there are two files in the **Files** panel.



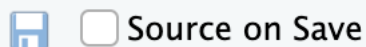
R Scripts

Now that we have added our data to our directory let's create an R script to save our code. To create an R script select **File > New File > R Script**. Now that you have created an R script you can write your code in the R script like we wrote it in the console. However, you will notice that when you press **Enter** R Studio creates a new line and does not run the current line. In order to, run the current line press **CMD/CNTRL + Enter** with your cursor on the line you want to run. If you want to run multiple lines at once you can highlight all the lines you want to run and then press **CMD/CNTRL + Enter**.

In an R script if you start a line with **#** you can write comments. R will not run these lines as code. Commenting your code is useful for when you need to return to your code later and might not remember what each line of code was for and for submitting assignments in POLI311.

```
# Creating a vector of student names
names <- c("William", "Paris", "Ali", "Natasha", "Maria")
```

In order to save your R script you can press the floppy disk icon below the name of the R script (**Untitled1**).



Let's call our R script for today **Lab1**. You will now notice that in the **Files** panel there is now a file called **Lab1.R**.

Dataframes

One of the most important object types in R is the **dataframe** (or **tibble** when using the **tidyverse**). Dataframes allow you to store two dimensional data (think of a spread sheet) of multiple classes. Dataframes have columns and rows where each row represents a unique observation and each column represents a variable. Unlike a matrix where each column needs to contain the same object class, dataframes can contain multiple classes. To import a **.csv** file into our R environment we can use the **read.csv()** function.

```
df <- read.csv("poli311file1.csv")
```

The **read.csv()** function requires you to write the file name (and path) in quotation marks. (**NOTE** spelling and spacing matter, your file name in quotation marks needs to match exactly with the file name in the folder. To make this process easier R Studio lets you process the files in your working directory by pressing the **tab** key with your cursor in the quotation marks. You can then use your mouse or the arrow keys and the **return/enter** key to select the data file you want to import.) Make sure you also save your data frame as an object in R. In this case we called the object **df** for data frame.

Now that we have a data file imported we can inspect the data frame with the **str()** function.

```
str(df)
```

```
## 'data.frame':    10078 obs. of  14 variables:
## $ year          : int  1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 ...
## $ logpop         : num  0.902 0.903 0.904 0.904 0.905 ...
## $ loggdp         : num  NA NA NA NA NA NA NA NA NA NA ...
## $ cowcode        : int  2 2 2 2 2 2 2 2 2 2 ...
## $ region         : int  NA NA NA NA NA NA NA NA NA NA ...
## $ v2jupoatck    : num  0.809 0.809 0.809 0.809 0.809 0.809 ...
## $ system         : chr  "" "" "" "" ...
## $ frac           : num  NA NA NA NA NA NA NA NA NA NA ...
## $ polity         : num  0.95 0.95 0.95 0.95 0.95 ...
## $ durable        : num  0.559 0.565 0.571 0.576 0.582 ...
## $ constitution_age : num  0.734 0.738 0.742 0.747 0.751 ...
## $ constitution_scope: num  0.605 0.605 0.605 0.605 0.605 ...
## $ amendment_rate  : num  0.0674 0.0674 0.0674 0.0674 0.0674 ...
## $ stabs_rate       : num  0.182 0.182 0.182 0.182 0.182 ...
```

This function displays the class of our dataframe (**data.frame**), the number of observations in our dataset (10078 **obs.**), the number of variables in our data frame (14 **variables**), and a preview of each variable. The variable previews contain the name of the variable (e.g., **year**, **logpop**, **loggdp**, etc.), variable class (e.g., **int**, **num**, **chr**, etc.) and the first 10 values of each variable.

Now that we know more about our dataframe we can begin using our data frame by indexing it. There are two ways to index a dataframe. We can use square brackets (**[]**) or the **\$** selection operator. When using **[]** brackets the first argument in the brackets is the row(s) and the second argument is the column(s).

For example the following code selects the 3rd observation (row 3) in the **polity** (column 9) variable.

```
df[3, 9]
```

```
## [1] 0.95
```

We can also select multiple rows/columns using the `:` operator or the `c()` concatenate function. The concatenate function is one of the most used functions in R and allows you to supply multiple values to a single argument in a given function.

To select multiple consecutive rows/columns we can use the `:` operator:

```
df[4:16, 1]
```

```
## [1] 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975
```

and to select non-consecutive rows/columns we can use the `c()` function.

```
df[5, c(1, 4, 5, 10)]
```

```
##   year cowcode region   durable
## 5 1964      2      NA 0.5823529
```

We can also combine both methods to select a mix of consecutive and non-consecutive rows/columns.

```
df[3:12, c(1:3, 6)]
```

```
##   year   logpop loggdp v2jupoatck
## 3  1962 0.9037685    NA  0.8091911
## 4  1963 0.9044514    NA  0.8091911
## 5  1964 0.9051106    NA  0.8091911
## 6  1965 0.9057038    NA  0.8091911
## 7  1966 0.9062520    NA  0.8091911
## 8  1967 0.9067687    NA  0.8091911
## 9  1968 0.9072425    NA  0.8091911
## 10 1969 0.9077064    NA  0.6952432
## 11 1970 0.9082592    NA  0.8583714
## 12 1971 0.9088592    NA  0.8583714
```

Using square brackets we can also index datasets by variable name. In order, to index by name we can supply a character vector in the square brackets using single `'` or double `"` quotation marks. You can supply either a single variable name or multiple variable names using the `c()` function.

```
df[1:5, "year"]
```

```
## [1] 1960 1961 1962 1963 1964
```

```
df[1:5, c("year", "cowcode")]
```

```
##   year cowcode
## 1 1960      2
## 2 1961      2
## 3 1962      2
## 4 1963      2
## 5 1964      2
```

Finally, if you want to display all the rows/columns for a given set of rows/columns you can leave the other argument blank in the square brackets.

```
df[1:5, ]
```

```
##   year   logpop loggdp cowcode region v2jupoatck system frac polity   durable
## 1 1960 0.9022518    NA      2     NA  0.8091911      NA  0.95 0.5588235
## 2 1961 0.9030386    NA      2     NA  0.8091911      NA  0.95 0.5647059
## 3 1962 0.9037685    NA      2     NA  0.8091911      NA  0.95 0.5705882
## 4 1963 0.9044514    NA      2     NA  0.8091911      NA  0.95 0.5764706
## 5 1964 0.9051106    NA      2     NA  0.8091911      NA  0.95 0.5823529
```

```
## constitution_age constitution_scope amendment_rate stabs_rate
## 1 0.7339056 0.6049383 0.06737801 0.181598
## 2 0.7381975 0.6049383 0.06737801 0.181598
## 3 0.7424893 0.6049383 0.06737801 0.181598
## 4 0.7467811 0.6049383 0.06737801 0.181598
## 5 0.7510729 0.6049383 0.06737801 0.181598

# df[, "year"] NOTE this line is commented out because it will output 10,078 oberseervations.
```

The other method of indexing a dataframe involves using the `$` operator and the name of a specific variable. This method outputs all the values of a given variable as a vector and is generally used when supplying variables to functions. For example to select the `year` variable from our `df` we would run the following line of code: `df$year`.

Another function we can use for inspecting our data is the `table()` function. This function allows us to create a table that displays the number of observations in each value of a given variable.

```
table(df$year)
```

```
##
## 1960 1961 1962 1963 1964 1965 1966 1967 1968 1969 1970 1971 1972 1973 1974 1975
## 145 146 146 146 146 146 146 146 146 146 146 148 148 148 148 148
## 1976 1977 1978 1979 1980 1981 1982 1983 1984 1985 1986 1987 1988 1989 1990 1991
## 148 148 148 148 148 148 148 148 148 148 148 148 148 149 164 167
## 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007
## 168 170 170 170 170 170 171 172 172 172 172 172 172 172 172 172
## 2008 2009 2010 2011 2012 2013 2014 2015 2016 2017 2018 2019 2020 2021 2022
## 172 172 172 173 173 173 173 173 173 173 173 173 173 173 173
```

In this table the number of top indicates the year (e.g., 1960) and the number below indicates the number of observations in a given year (e.g., 145).

Practice Problem 4: How many countries were surveyed in 2011?

We can also provide two arguments to the `table()` function which creates a cross table that displays the number of observations taking on both values in the dataset.

```
table(df$region, df$system)
```

```
##
##           Assembly-Elected President Parliamentary Presidential
## 0      0              33              692              66
## 1 112              68              281              469
## 2  37             168              389              297
## 3   2              60              74              491
## 4  22             113             129             1266
## 5   0              68             131             626
```

In this table there are 491 observations that are presidential systems in region 3, with the columns representing the political systems and the rows representing the region.

Practice Problem 5: How many observations are parliamentary systems in region 2?

Tips and Tricks for R

Think of learning R like learning a new language. Many of the rules will seem arbitrary at first but once you understand the underlying logic it becomes easier to to trouble shoot your problems in R. In the mean time I below I have provided some useful tips for troubleshooting issues in R.

1. Use the built in help function in R! If you proceed the name of a function with a question mark it will open the built in help file in the R Studio Help panel (e.g. `?table()`).
2. Google error codes. Whenever, there is a problem with your code R will give you an error in the console. However, when you are first starting out this codes are often confusing. Therefore, you can usually Google errors and find someone else who has already had the same issue.
3. Use Generative AI. ChatGPT is really good at explaining error codes and how to fix them. You can usually paste the error code into ChatGPT and receive an explanation and steps to solve the error. I find prompting ChatGPT in the following way is the most useful: I am coding in R using the **BLANK** package and received the following error: **ERROR**.⁴ ChatGPT will generally provide a number of tips to help you fix the error.
4. Come to office hours and ask your TAs for help. We are more than happy to help you understand any content from the labs during office hours.

Practice Problem solutions

1. The number [1] indicates that the first element in that row is the first element in the vector. If the vector outputted onto multiple lines then each line would start with a different number.

2.

```
sqrt((14 * 54^2)/((55 - 14) + 98 * 2))
```

```
## [1] 13.12453
```

3. Ask your TA to double check your answer if you are unsure.
4. 173
5. 389

⁴**NOTE:** ChatGPT is a useful tool for diagnosing errors in R but requires some base knowledge of R to be used properly. Therefore, learning R in a classroom environment is useful even with the availability of Generative AI programs like ChatGPT. Additionally, ChatGPT can be used to help you diagnose error but using GenAI to write the code for your assignments is **considered plagiarism** and will be dealt with according to the plagiarism policy outlined in the syllabus.