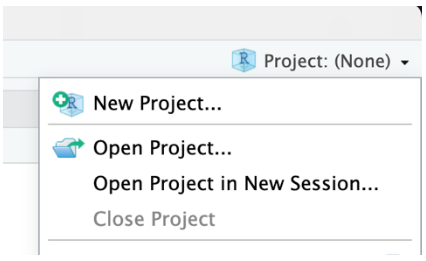
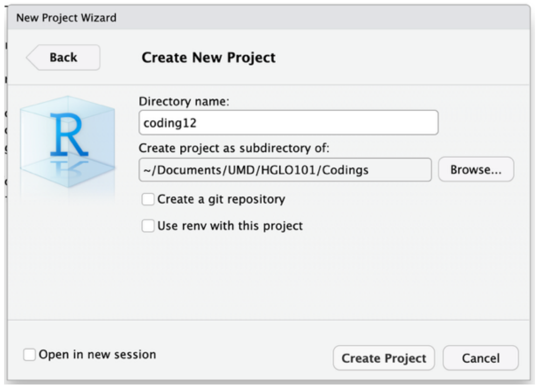
How to code in R

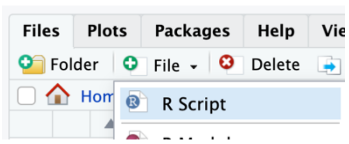
Intro: Hello, my name is Kate Kerllenevich and at the time of writing this, I am a freshman at the end of my first semester in the HGLO program. I am a Computer Science major, and have spent most of this semester teaching myself and others how to code in R. Now, Dr. Croco and your TA will likely do some explanation of coding in R – but I have found their ways to be largely unintuitive for people who do not know how to code. So, I designed my own way of coding R, and I have decided to share it here with you. For all of my examples, I have written code. You can find it here: https://github.com/katekerllenevich/hglo-r-example.

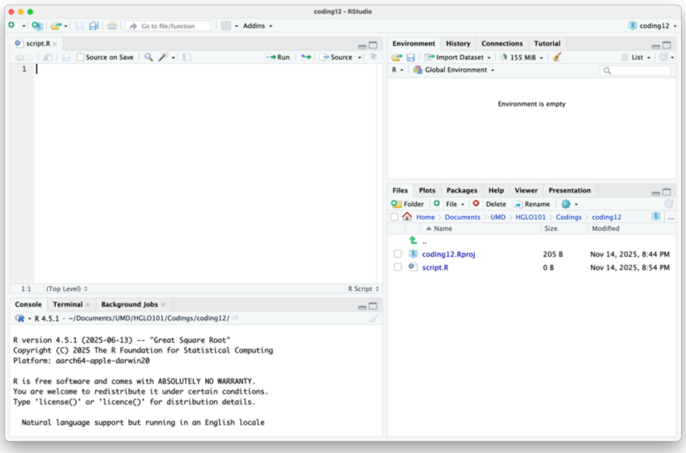
Creating a new project: The first and most important step when beginning a new R project is to create a new project in RStudio. This can be found in a button the top right. As seen the image, when clicking on the "Project: (None)" text you should open a dropdown. From there click "New Project," a larger window will appear, click "New Directory" then "New Project." A larger window will open where you will name the project as well as choose a folder in which the project will live. For example, if configured the same way that I have it in the second image, then a new folder will be created called "coding12" which will live in "Documents/UMD/HGLO101/Codings."





Project setup: To do statistical analysis in R, we first need to have some data. So, download whatever data you are analyzing and put it into the folder your project exists in. Then you will want to create a script file. To do this, look at the bottom right panel, click on "Files," then on the "File" button with a green plus. A dropdown will open, click on "R Script" and name it whatever you want. Then you will want to open the file where you will see four distinct panels. On the top left is where you will open files and write your code;





```
Name              Size   Modified
..
coding12.Rproj    205 B  Nov 14, 2025, 8:44 PM
script.R          0 B    Nov 14, 2025, 8:54 PM
```

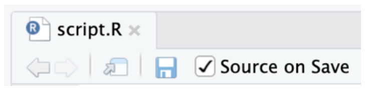
```
R version 4.5.1 (2025-06-13) -- "Great Square Root"
Copyright (C) 2025 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20

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.

  Natural language support but running in an English locale
```

the bottom left is the console (we'll get to this later); the top right will show all of your variables (also later); and the bottom right will show your files as well as all graphs that you create.

Before writing any code, at the top of the code editor panel (top left), there is a checkbox next to a button that says "Source on Save." You should always check this so that your code automatically updates whenever you save the file. Then, whenever you write code and you want it to update, you just save the file (Control+S, ⌘+S, or even the blue floppy disk icon next to check box you just checked).



Now for every project you will have the same general boilerplate code at the top of the file. Note that for all code blocks in this document, line numbers will be placed at each line of code. You do not need to copy these, they are just to simply explanation of code. In fact, copying them will prevent your code from working. This boilerplate does the following:

```
1. rm(list=ls())

2. setwd("~/Documents/UMD/HGLO101/Codings/coding12")


3. library(ggplot2)

4. library(dplyr)

5. library(readr)
```

1. When saving the code file, clears old variables as to not clutter code

2. Sets the current *working directory*. This is the folder that R will use as its base when searching for other files.

3-5. Here you will load all of your *packages* (code that someone else has written that you will use). The packages that you load, as well as the number of them, will vary project to project depending on what you need to do. *ggplot2* is the graphing utility, *dplyr* is used to filter out data, and *readr* is used to read data from files. These are the only libraries that we will be using in this example.

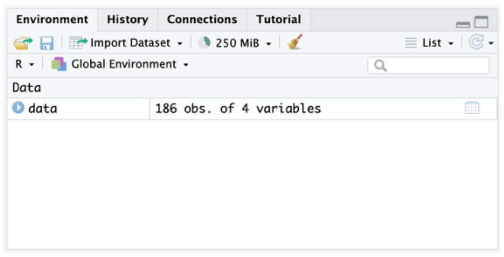
Now you should read the data that you downloaded earlier. I'm using a mock dataset that contains a country code, year, and GDP. To do this, you should use the following code:

```
1. data <- read_csv("data.csv")
```

This line of code creates a new *variable* (where data is stored) named "data." The *assignment operator* (<-) tells the variable to store what is on the right side. In this case, it stores the data from the "read_csv" *function* (a set of code that gets run – think like in math f(x)), which reads data from a csv file of certain name – in this case "data.csv." It is important to note that you should use the name of the file for the dataset which you downloaded.

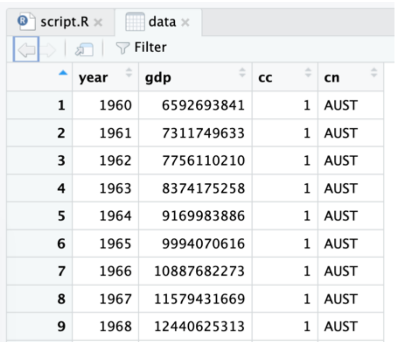
This ends the setup process for a new project. Note, sometimes your projects will be more complicated. Look at some of the example code (with caution), and never ever be afraid to use Google to help you answer a question.

Examining and using data:



Data
data 186 obs. of 4 variables

The core principle of R is to be a tool that makes data analysis easy, so there are several built in tools to simply our tasks. In the previous section we created the variable "data," to see and examining available variables, we look towards our top right panel on the "Environment" tab. To be able look a variable, click on its name – not the blue 'play' button next to it (this will not work). When you do so, it should open a new tab on your code editor (top left). When you look at it, there should be rows of data. Mine looks like this; yours will look differently.

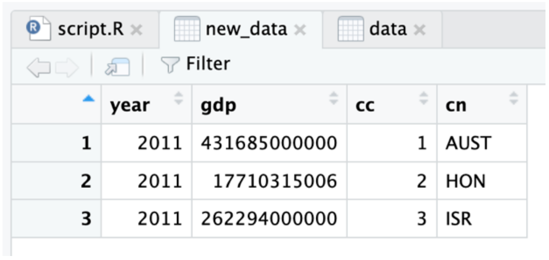


	year	gdp	cc	cn
1	1960	6592693841	1	AUST
2	1961	7311749633	1	AUST
3	1962	7756110210	1	AUST
4	1963	8374175258	1	AUST
5	1964	9169983886	1	AUST
6	1965	9994070616	1	AUST
7	1966	10887682273	1	AUST
8	1967	11579431669	1	AUST
9	1968	12440625313	1	AUST

There are lots of types of variables in R – for this class we only care about two: *factor* and *numeric*. Data that is a *factor* is categorical data, for example: colors. Data that is *numeric* could be any number, for example: number of students in HGLO. These types of data are useful for different things, as will become apparent.

One of the most important things that we do in R is data filtering and mutation. For the purposes of this guide, we will not be discussing data mutation – you will be able to learn this later in the future. Data filtering works by taking a large data set and paring it down to a smaller subset of that data. For example, if you wanted to filter for only data in the year 2011, then you would write code that looks like this. Let's walk through this code. We are creating a new *variable* called "new_data" and assigning it to "data" passed through what is called the *pipe operator* (%>%). This is a difficult concept to understand, think of it as an assembly line. One person starts with the raw materials and does x to it, while the next person does y to it, and so on and so forth. The *pipe operator* acts like that – "data" is the raw materials and "filter" is the first step in the assembly line (technically each of the "raw materials" is a row in the dataset); in this case we only have one step. The "filter" function only selects rows of data where the column year is 2011. This is done using the *equals operator* (==), which compares equality between two different items. When we take a look at "new_data," we can that it only contains data from the year 2011.

```
new_data <- data %>%
    filter(year == "2011")
```

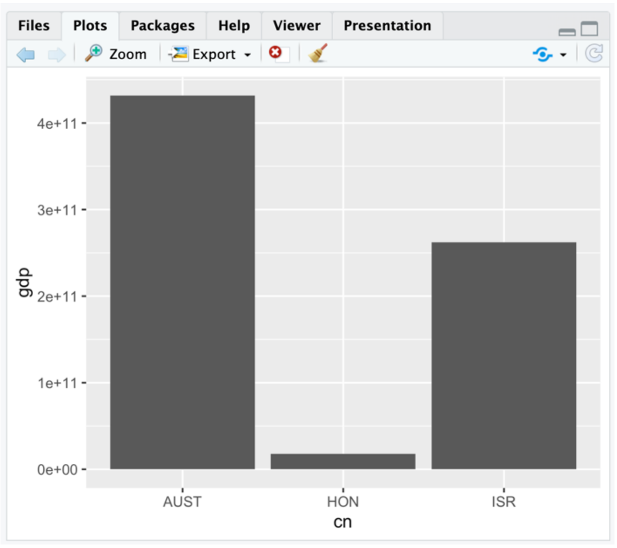


	year	gdp	cc	cn
1	2011	431685000000	1	AUST
2	2011	17710315006	2	HON
3	2011	262294000000	3	ISR

Making a bar chart: The main thing that we do in this class is make graphs. This document examines three common types of graphs: *bar charts*, *line graphs*, and *scatterplots*. We will start by looking at the *bar chart*. We are going to make a graph that compares the GDP in 2011 across all 3 countries in the dataset (conveniently we can reuse the "new_data" variable that we previously created). We are going to go step by step, building the graph and adding new components to it as we expand the graph. The code for the first graph is very simple. We use the *package* "ggplot2" to make graphs, which contains a *function* "ggplot" that we use in line one. The output of this function is assigned to the *variable* "graph1." In R, *functions* take *parameters* (like the x in f(x)). The "ggplot" *function* always takes two *parameters*, the first is the data set to make the graph from, and the second is the graphs 'aesthetics.' We create our aesthetics from the output of the *function* "aes," which can take several *parameters*, there is no defined amount. The aesthetics of a graph actually determine where data is displayed (x-axis, y-axis, color, shape, etc.). So, in this graph we are putting on the x-axis the country "cn" (taken from the dataset), and on the y-axis the country's gdp.

```
1. graph1 <- ggplot(new_data, aes(x=cn, y=gdp)) +
2.     geom_col()
```

After retrieving the output of the "ggplot" *function*, we use the *layer operator* (+) to add different rendering layers onto the graph. In this case we are adding a layer using the *function* "geom_col," which creates simple *bar graphs*. It doesn't need to take any parameters right now. The output graph looks like this. For you to see the output graph in RStudio, you have to go to the console (bottom left panel) and type the name of the *variable* which you want it to display; in this case it is "graph1." Then you press the enter key, and the graph is displayed in the bottom right panel, under the Plots tab. This graph is pretty ugly, there is a lot we can do to improve it.



The first thing we can do is to apply simple labels to the graph by adding a new layer, using the function "labs." This will allow us to change are labels on the x-axis, y-axis, title, subtitle, and so on. This code adds a new layer using the *layer operator* to set the x-axis label to "Country," the y-axis label to "GDP," and the title to "GDP by Country in 2011." Now, if you look below each bar, it labels which country the bar represents (AUST, HON, ISR). If we wanted to edit these, then we would need to *mutate* the data in the dataset (I know I said we wouldn't do this, but data mutation gets far more complicated than this).

```
1. graph1 <- ggplot(new_data, aes(x=cn, y=gdp)) +
2.     geom_col() +
3.     labs(
4.             x = "Country",
5.             y = "GDP",
6.             title = "GDP by Country in 2011"
7.     )
```

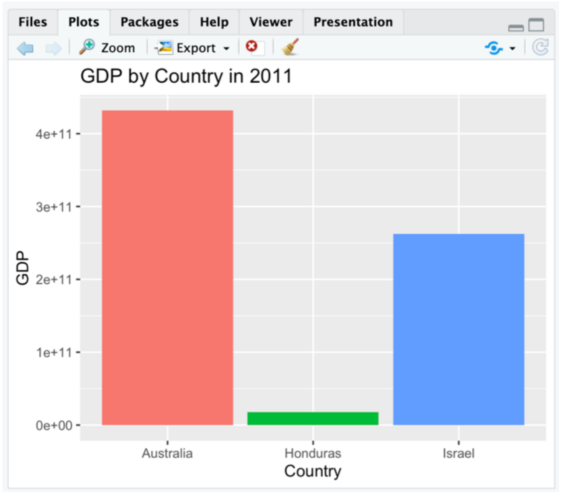
Remembering from earlier, data that is a *factor* is categorical data. We need to adjust the "cn" column on the "new_data" dataset to set it to be a *factor*. We do this using the "factor" *function*. When modifying data, we use the *list-extractor operator* (`$`) to select the column of data. Then we adjust it using the *assignment operator*. On the next page, take a look at the code.

```
new_data$cn <- factor(new_data$cn,
    labels = c("Australia", "Honduras", "Israel")
)
```

Inside of the "factor" *function* we pass to *parameters*, the dataset that we are turning into a *factor* and the labels that we want to use to refer to the different countries. "labels" has to be a list, and the way we make these is by using the "c" *function*, which takes unlimited *parameters*. We write them in this order as, if we look at the graph, that is the order that they show up in, and hence are processed in.

```
graph1 <- ggplot(new_data, aes(x=cn, y=gdp, fill=cn)) +
    geom_col() +
    labs(
            x = "Country",
            y = "GDP",
            title = "GDP by Country in 2011"
    ) +
    guides(fill = "none")
```

The last thing we will change for this graph is the color, which can be done by adding a "fill" *parameter* to the aesthetics. The other thing that we have to do, which is not too important for this context, is adding a layer that comes from the "guides" function that removes the key on shows the color. We don't need this because the name of the countries is already shown below the bars.



Now that all of this code has been shown, we can look at the final product of the graph. As we can see, the graph now has a title and labels for the axes. It also says the name of the countries below the bars, as well as each country using a different color. The only thing that would be ideal to change would be the scale on the x-axis, but that is more complicated to do than is in the scope of this graph (also I cannot think of how to do this easily).

Now we did it! I hope you understand the basics of making graphs, as I will explain significantly less in the next two graphs that we are going to make.

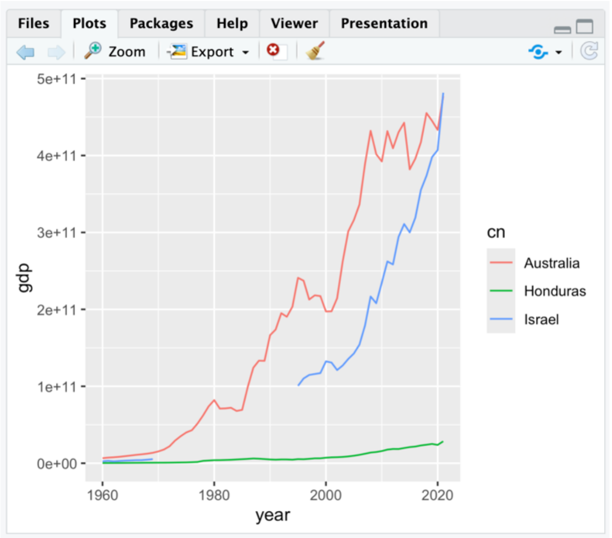
Making a line graph: Making a *line graph* is a lot like making a *bar chart* – pretty much all graphs are made in the same way. Our *line graph* will display each's countries GDP against time as a separately colored line. For this graph, we do not need to *filter* any data, as we are displaying all of the data within the dataset. We do, however, need to create a *factor* for the country names just like last time.

```
data$cn <- factor(data$cn,
    labels = c("Australia", "Honduras", "Israel")
)
```

The code for this graph is relatively simple. First, we will create the *factor* in the same way as last time. To allow for ease of coding, I will put the code here again. This time I am adjusting the main data set, because we do not need to *filter* any data.

```
graph2 <- ggplot(data, aes(x=year, y=gdp, color=cn) +
    geom_line()
```

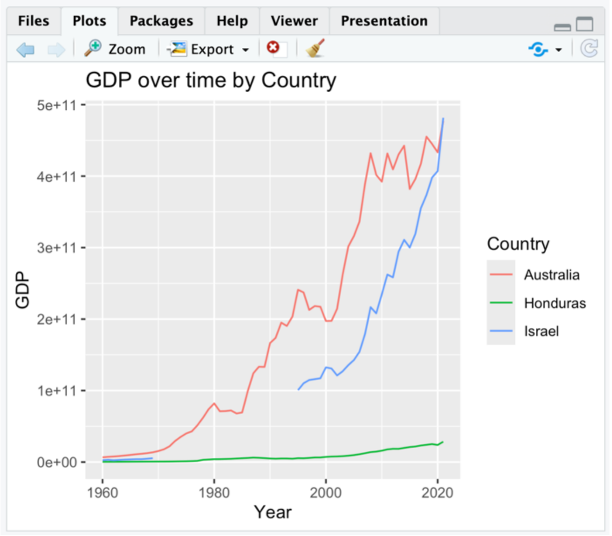
Next, we will make the basic *line graph*. We will put onto the year onto the x-axis, GDP onto the y-axis, and the lines will be colored by country. After running this code, we get the follow graph.



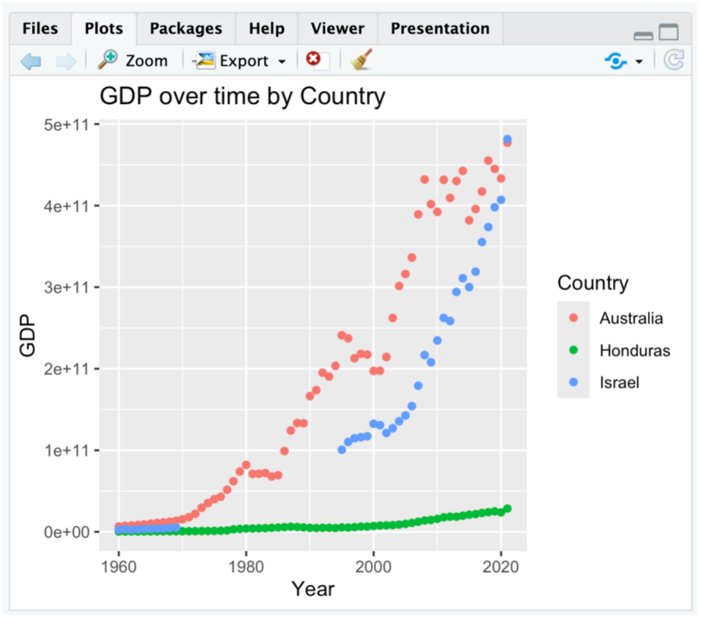
From the technical perspective, we are creating the base graph (the data representation) using the "ggplot" *function*, which is then rendered by adding the "geom_line" *layer*.

As you can see, the graph is mostly complete. Just like the bar graph, *ggplot2* automatically splits the countries by different colors and places a key on the side. We just need to add label *layer* to the graph. The code for the "labs" *function* to create the final graph looks mostly like the "labs" *function* from the *bar chart* code. We have "x," "y," "title," and unlike last time, "color" as *parameters*. The first three titles are self-explanatory, but the "color" parameter adjusts the title on the color key. It will change where it says "cn" to whatever text of your choosing.

If you do it all right, it should look something like the following. If you are confused about anything at this point, I suggest re-reading the *bar chart* section (it goes far more in-depth) or looking at the example code.



Making a scatter plot: Just like the *line graph*, our *scatterplot* will be examining a country GDP data over time, and the dots will be colored differently by country. The code is so remarkably similar, that you should be able to take the code that you wrote for the line chart section, and instead of adding the "geom_line" *layer*, adding the "geom_point" *layer* (which is what is used for *scatterplots*). Doing that, the graph should turn out something like this.



You should note, most of the time that you will not be making *scatterplots* where you graph against time. This data I am using is, however, very minimal, and there is no other way to create a *scatterplot*. Instead, you will likely be comparing the effect of one thing on the other. For example, from one of my projects: health care quality vs health care expenditure.

Conclusion: I hope you've found my guide helpful and feel like you have a more solid foundation on coding than most people this year have ever had. If you're confused about something, do not be afraid to look it up, ask a friend, hell, ask ChatGPT. Or, if you think that you really need it, get help from me. Ask around, someone is bound to have my phone number – I'm more than happy to help. I hope you have a good time in the HGLO program, and good luck on your future endeavors.