

Title

Name: _____

Date: _____

GOAL: GET UP AND RUNNING WITH R!

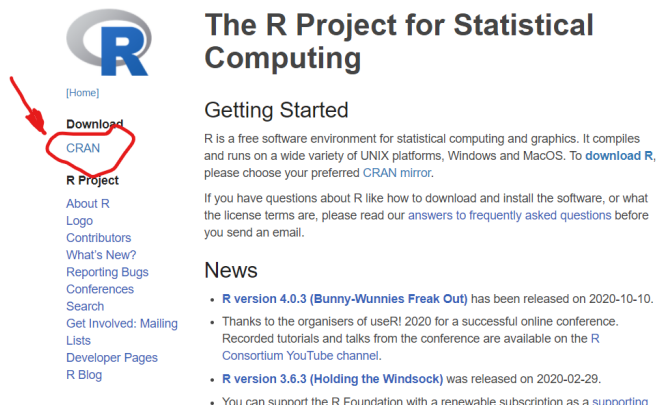
Purpose:

1. Install R
2. Install RStudio
3. Begin to get a handle on how the program works and the different things it can do
4. Follow along with code to see how R 'thinks'

Tasks:

1. Install R

- Open an internet browser and go to www.r-project.org.
- Click the "CRAN" link at the top of the menu on the left (see image)



- Select a CRAN location (a site that stores the code and documentation for R) and click the corresponding link. Probably best to choose one nearest to you but I don't know if it really matters
- Click on the "Download R for Windows" or "Download R for (Mac) OS X" link at the top of the page depending upon what computer you have

The Comprehensive R Archive Network

Download and Install R

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- [Download R for Linux](#)
- [Download R for \(Mac\) OS X](#)
- [Download R for Windows](#)

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Source Code for all Platforms

- Click on the “base” if you are on windows or the “pkg” if you are on mac (either way it will be on the top of the screen) and follow the directions. I don’t have a Mac but from what I read it should download and then you can just click on the icon to install it. If you run into trouble let me know!

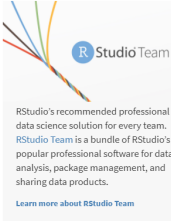
2. [install R Studio](#)

- Go to <https://rstudio.com/products/rstudio/download/> and Make sure to choose the free version:

Choose Your Version

The RStudio IDE is a set of integrated tools designed to help you be more productive with R and Python. It includes a console, syntax-highlighting editor that supports direct code execution, and a variety of robust tools for plotting, viewing history, debugging and managing your workspace.

[LEARN MORE ABOUT RSTUDIO FEATURES](#)



RStudio's recommended professional data science solution for every team. RStudio Team is a bundle of RStudio's popular professional software for data analysis, package management, and sharing data products. [Learn more about RStudio Team](#)

RStudio Desktop Open Source License	RStudio Desktop Pro Commercial License	RStudio Server Open Source License	RStudio Server Pro Commercial License
Free	\$995 /year	Free	\$4,975 /year (5 Named Users)
DOWNLOAD <small>Learn more</small>	BUY <small>Learn more</small>	DOWNLOAD <small>Learn more</small>	BUY <small>Evaluation Learn more</small>

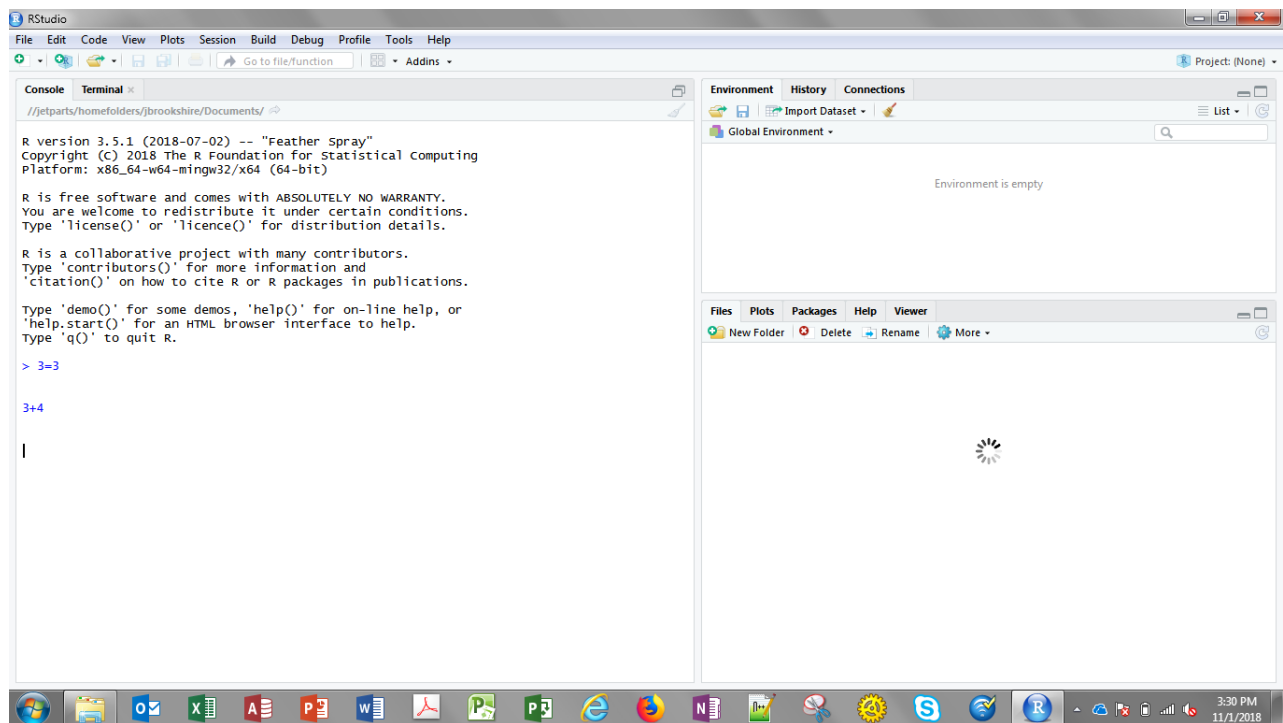
- Click on "Download RStudio Desktop."

- Click on the version recommended for your system. For the Mac version, save the .dmg file on your computer, double-click it to open, and then drag and drop it to your applications folder.

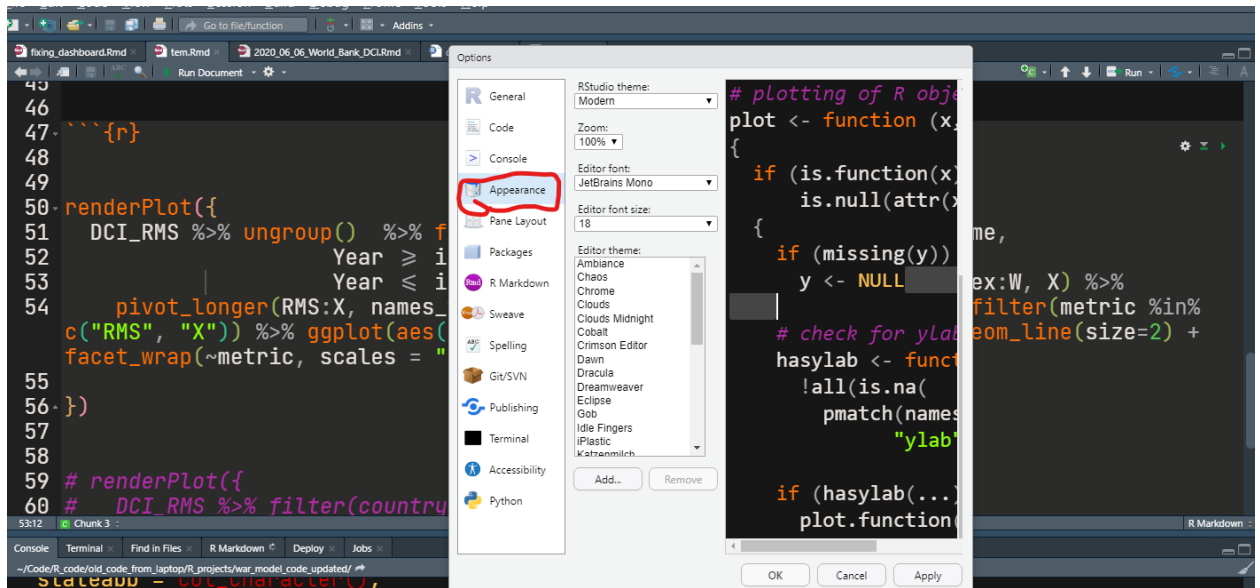
3. opening R

Now that you have both R and RStudio on your computer, you can begin using R! Open RStudio just as you would any program, by clicking on its icon or by typing “RStudio” at the Run prompt.

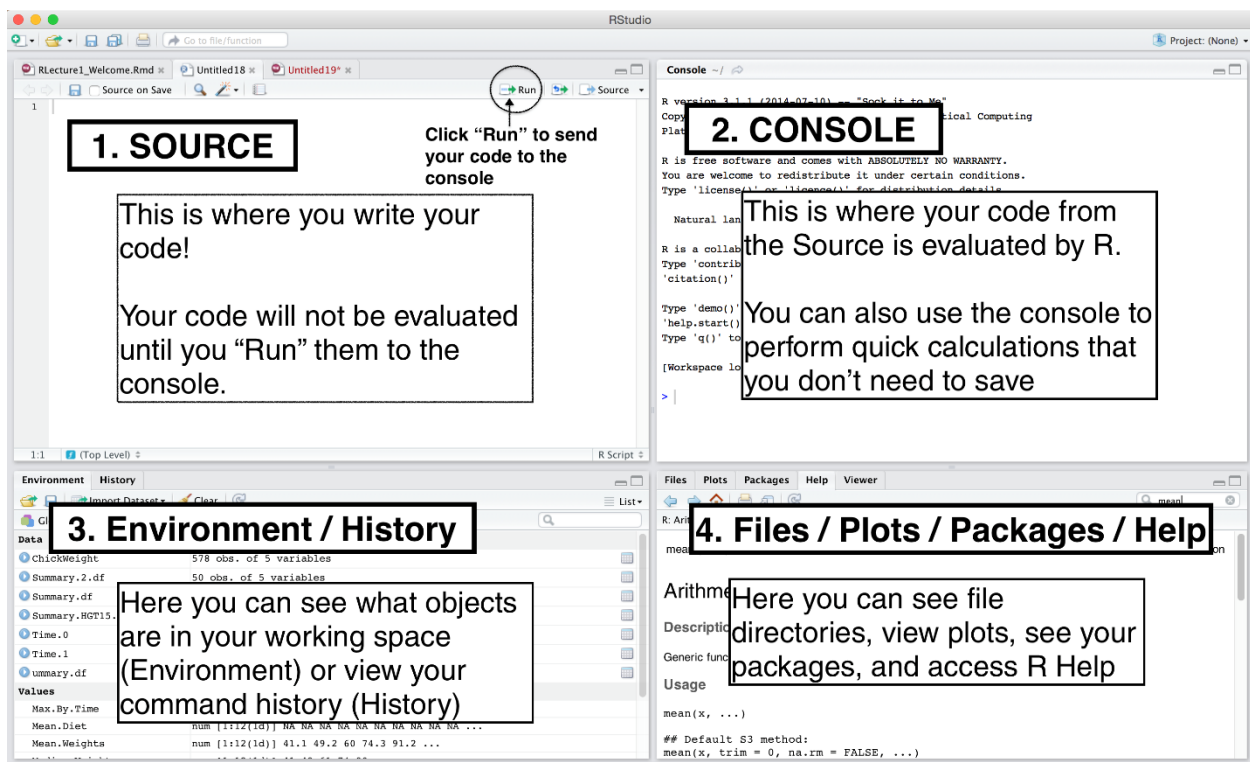
- **When you open R-Studio you will see something like this (it may look different):**



- The first thing a lot of people want to do is change the appearance. For example, if you do work at night you may want to change to 'dark mode' so it is easier on the eyes
 - To do this go to Tools > Global options > Preferences > Appearance to explore the many options available. The window lets you preview each theme. Your computer might have different fonts available so play around to choose one that looks good to you



- R-Studio has a lot of different windows/panes...many are for high-tech stuff that us mere mortals might never use. The image below gives the main names of the major panels we might be using



- **The Console**

- This is where code is run....it is also what you see if you open R *without* R-Studio. You enter code in a line and then hit 'enter/return' to run it
- It also shows some details such as the version of R you have. For example, you can see below that I am running version 4.0.2 and it is called "Taking Off Again" (you will probably have a newer version. I need to update mine!)

```
R version 4.0.2 (2020-06-22) -- "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (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.

Natural language support but running in an English locale

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.
```

- If you look below all the text you will see a little arrow (sometimes called a carrot)

```
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.

>
```

- That's where you type stuff. For example, you can write $5 + 3$ like I did and hit enter to see the answer:

```
> 5+3
[1] 8
> |
```

Congratulations! You just turned your expensive computer into a \$4 calculator 😊

- Important R fact:

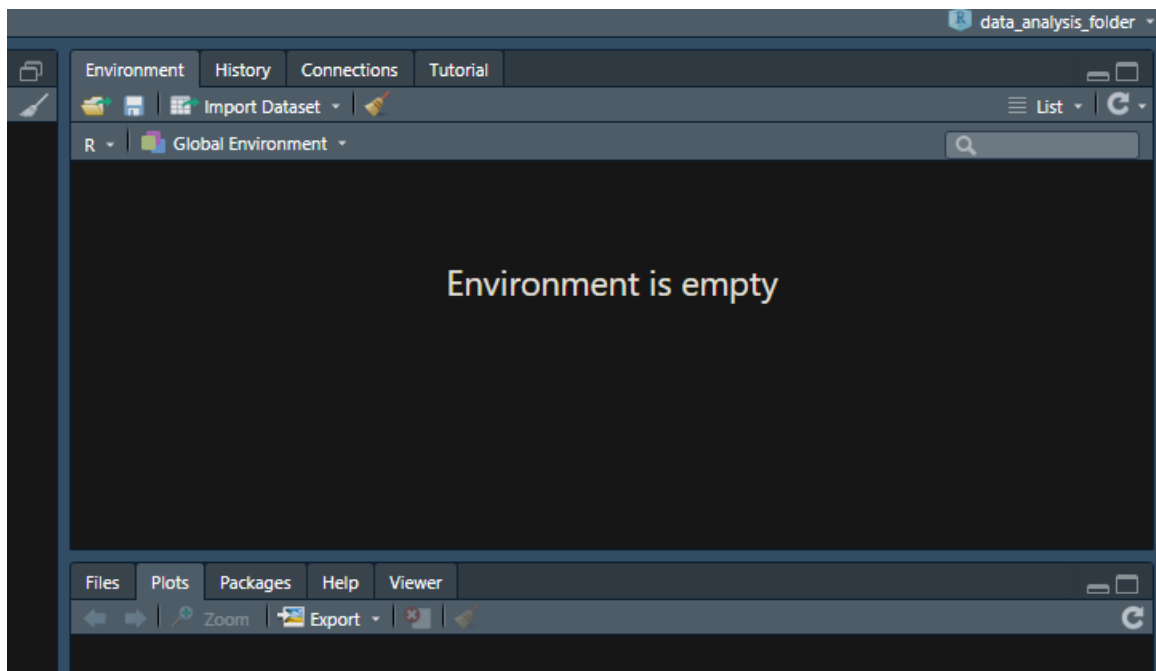
When writing in the console hitting the enter key will run your code. It will then let you know it is ready to do more stuff by showing the prompt again (>). But this happens **only** if R thinks the code is complete. Otherwise the '>' turns into a '+'

Notice in the picture below that R is confused about the second line I tried to run

```
> my_data1 ← c(1,2,3,4)
> my_data2 ← c(1,2,3,4
+
+
+ |
```

We will figure out why this happened later. But for now if you get stuck at the + sign hit the escape key to cancel the code!

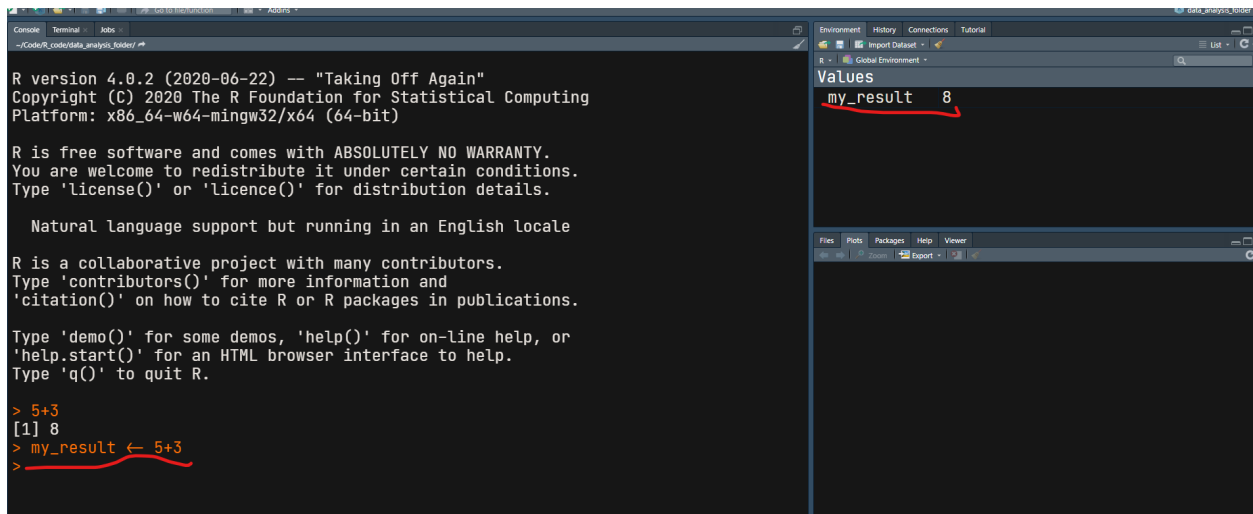
- The Environment: this is sort complex but in general it tells us what R knows about what you are working with....you can see now that it is empty!



- This is a happy thing....R doesn't know *anything* since you haven't told it anything yet
- Go back to the console and type this:

```
my_result <- 5+3
```

- (if you are having trouble making that arrow, it is the left “<” and then the dash “-”
 - On a Mac OS X: Option and - is the shortcut key for this arrow
 - On a Windows/Linux: Alt and - is the shortcut key for this arrow
- Now, look back at the environment panel and you should see something there!



```

R version 4.0.2 (2020-06-22) — "Taking Off Again"
Copyright (C) 2020 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (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.

Natural language support but running in an English locale

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.

> 5+3
[1] 8
> my_result <- 5+3
>

```

Environment

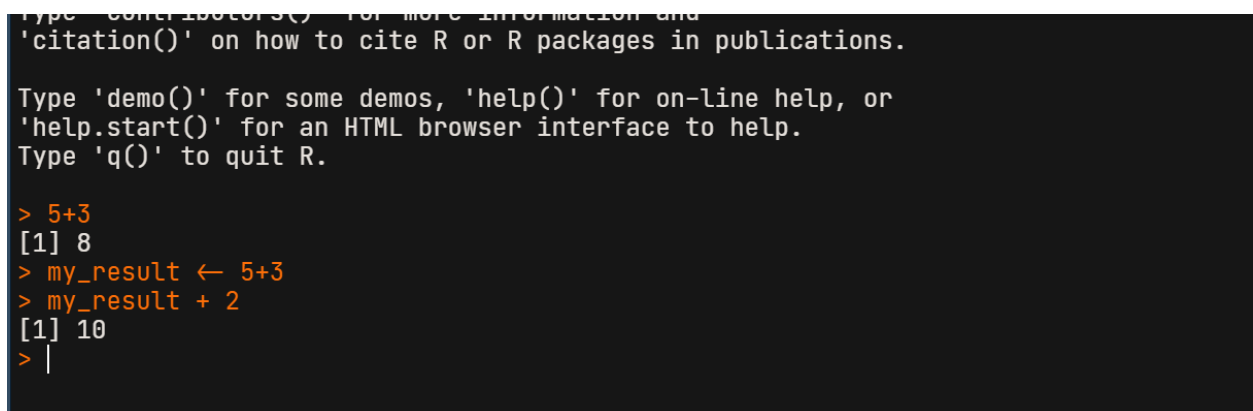
Global Environment

Values

my_result 8

So what is going on here? Well by using that “<-”, called the assignment arrow, we are giving the value of 5+3 to the **object** called **my_result**. R now *knows* that you want **my_result** to store the value you gave it.

- To show this, type `my_result + 2` into the console. If all went well, you should see the same thing I did:



```

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.

> 5+3
[1] 8
> my_result <- 5+3
> my_result + 2
[1] 10
> |

```

I'll say this again but I think the trick to learning R is to remember that we assign values to objects (in this case, we called the object `my_result`) and then do things to that object to learn something new.

4. More R goodness

On the class AsULearn site I have a document under week 1 called [r4beginners v3](#). This is a nice overview of R. it is a bit long but if you have the time reading it might help. To be honest I wish it were available when I started learning this stuff back in the early Holocene 😊

5. Installing packages

R comes with a lot of stuff that we can use. This is called **Base R**. There are things called **Packages** which give us more things we can do. All a package really is a bunch of code someone wrote to make R even better!

Some folks swear by only using base R when possible. However, it is a bit tricky at times and R is at its core a programming language, so it is designed for people with computer science background. But fortunately, folks have worked to make R easier. One group of packages have become known as the **Tidyverse**. We will be using this often since I think it is easier to learn and make more sense (though it is still tricky and will take some time

- To install a package we need to enter some code into the **console**:

```
install.packages("tidyverse")
```

- What is this doing? Well **install.packages** is what we call a **function**. A function for the most part does something to the thing inside the parenthesis, called an **argument**. In this special case, the function is telling R to go online and install this package. Here is what I saw when I installed it (might be different based on the operating system you are using)

```
>
>
> install.packages("tidyverse")
Installing package into 'C:/Users/marck/Documents/R/win-library/4.0'
(as 'lib' is unspecified)
trying URL 'https://cran.rstudio.com/bin/windows/contrib/4.0/tidyverse_1.3.0.zip'
Content type 'application/zip' length 439960 bytes (429 KB)
downloaded 429 KB

package 'tidyverse' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
  C:\Users\marck\AppData\Local\Temp\RtmpmLVX7\downloaded_packages
> |
```


- Once you install a package you need to load it into R's memory/environment. To do this we will use the **library** function:

```
library(tidyverse)
```

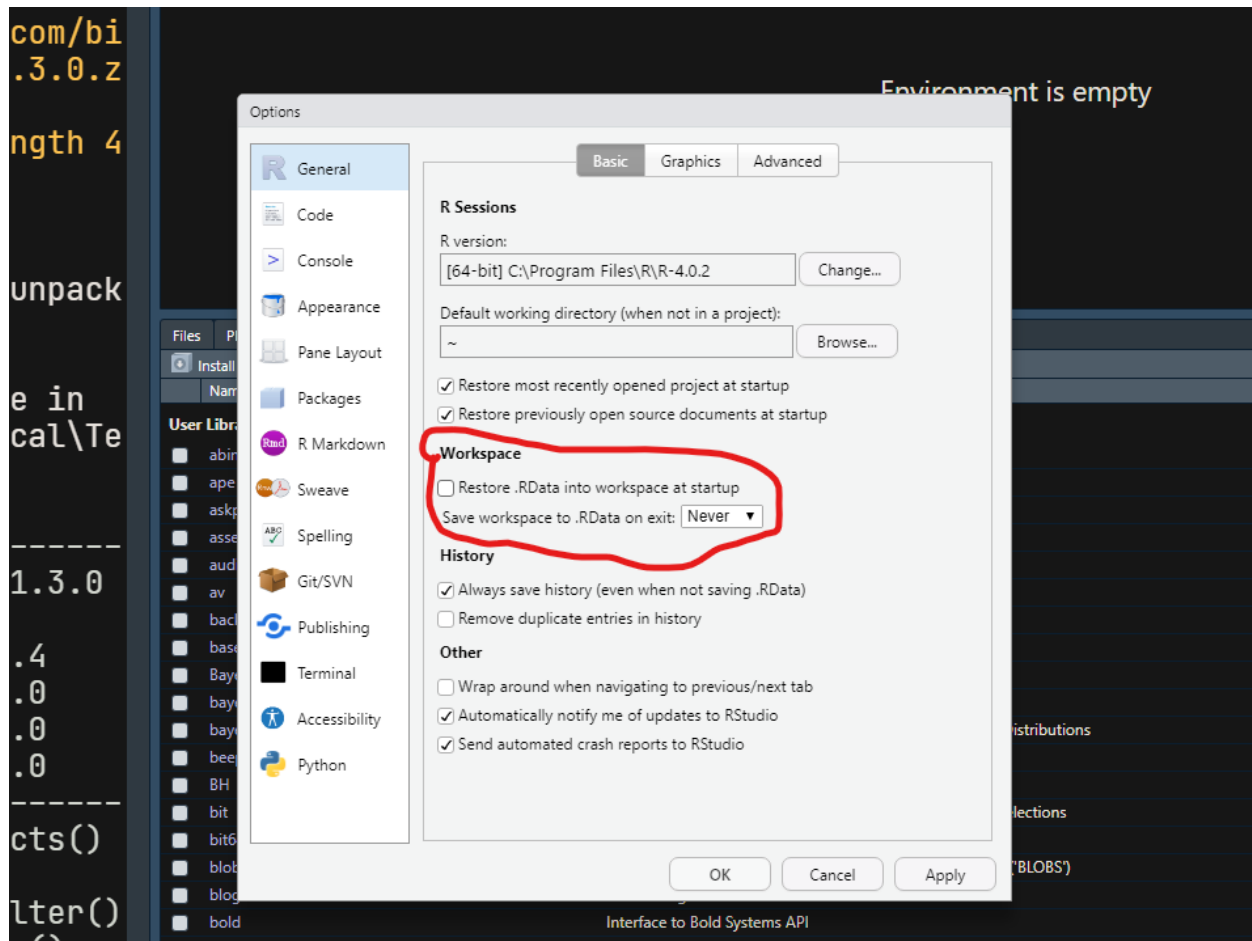
- Important: As with the example above, R doesn't remember much so you will have to do this part every time. **You only have to install a package once, but you will need to load it every time you restart R**
 - NOTE: forgetting to load a package with the library function is in my top 10 of reasons why my R code fails...Also in my top 10 is the fact that I always misspell length 😊

```

> library(tidyverse)
-- Attaching packages ----- tidyverse 1.3.0 --
v ggplot2 3.3.2      v purrr  0.3.4
v tibble  3.0.3      v dplyr  1.0.0
v tidyr   1.1.0      v stringr 1.4.0
v readr   1.3.1      v forcats 0.5.0
-- Conflicts ----- tidyverse_conflicts() --
x dplyr::filter() masks stats::filter()
x dplyr::lag()    masks stats::lag()
Warning message:
package 'tidyverse' was built under R version 4.0.3
>

```

- You can ignore the messages that you see. It is telling you what packages are being **attached**, which means that their functions are now in R's working environment. **Conflicts** notes any function names included in the package that we just loaded that share the same name as a function already loaded into memory. This normally is not an issue but sometimes good to pay attention to. For example, we are being told the package dplyr (which is part of the *Tidyverse*) has a function called **lag** and that that same name is used for a function from the **stats** package which come preloaded into R.
- Fun fact: you can see the packages that are loaded by going to the **packages** tab in R-Studio



- Under **Workspace** make sure to unselect the “restore .RData into workspace at startup” & that “Save workspace to .RData on exit” is set to “Never”
 - Why? Because this way you don’t have to worry about recreating the environment. Instead, we will make sure our code works on any machine. Experts note that this short-term pain will save you long-term agony because it forces you to capture *all* of the important interactions in your code.
- **Projects:**
 - One of the hardest things about learning to use R is figuring out where your work ‘lives.’ In other words, you are going to run into trouble trying to load in data because it is not always easy to figure out where R is looking for that data. For example, if you were to download a dataset on the bones found at an archaeological site in Italy and put it in your download folder, you need to find a way to tell R where to look!
 - This leads to the very helpful notion of a **working directory** which is basically where on the computer R looks for files.
 - If you want to see where R is looking for files you can use the function `getwd()`

```
getwd()
```

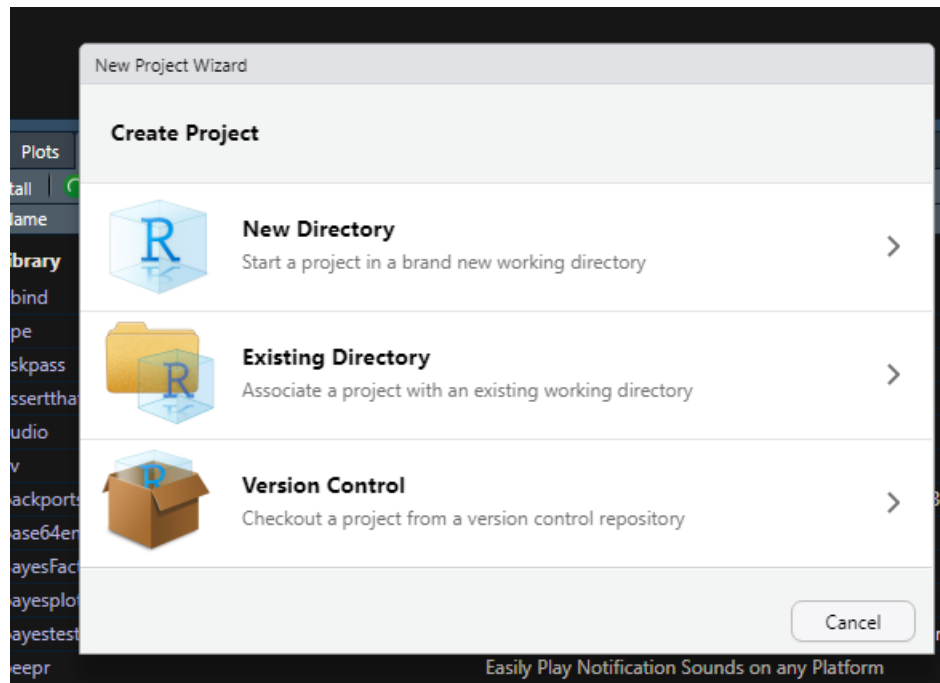
- RStudio 'defaults' to using your home directory on your computer. You can change this by going to

Tools > Global options > General > Basic.

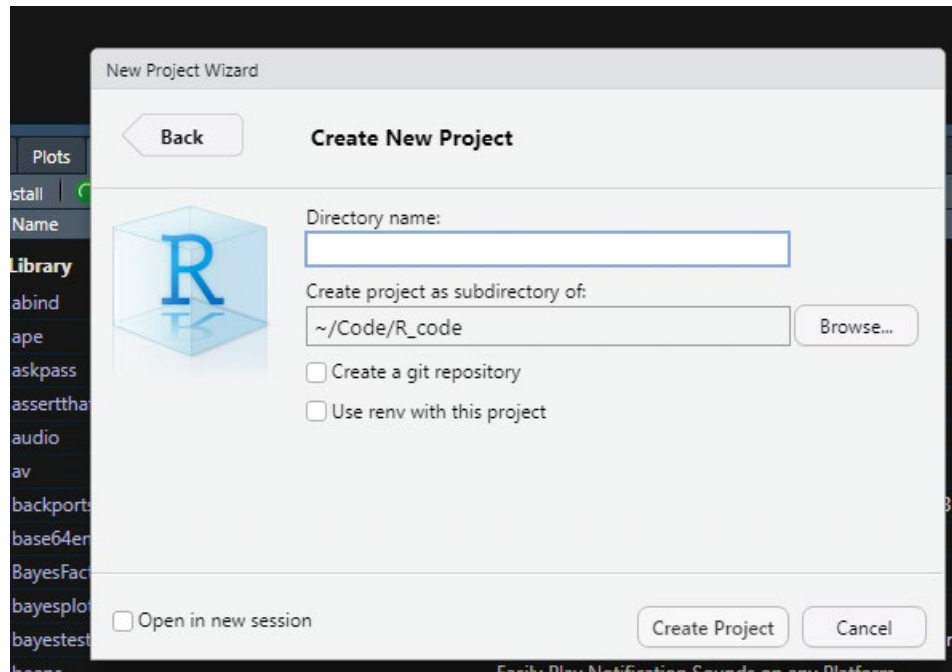
And then clicking on the “browse” button next to “default working directory”

- I highly recommend R-studio's **Projects** feature. Basically, this will not only create a new folder for you within the home directory, but it will see that new folder as your projects working directory. Whenever I do a new analysis, I make a new project folder and then put all the data I am going to use in that folder. This way I can send the final code etc to a colleague and everything should work on their machine
- To make a new project

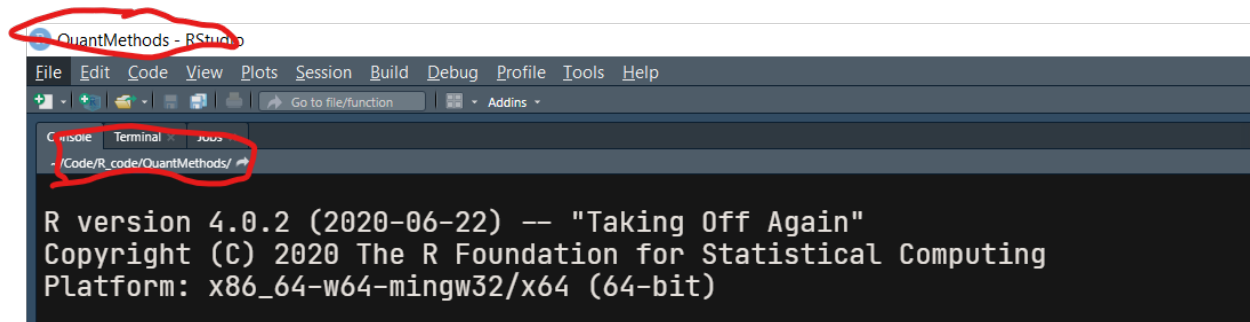
File > New Project.



- Click on the “new directory” option and then choose “New Project”



- give the project a name. for example “QuantMethods” and hit “create project.” R will then restart (note: if you click the “open in new session” it will leave the previous working RStudio session up and then open a whole new one. This is sometimes useful.)
- also note the part that says “create project as subdirectory of”. This tell us where it will live on your computer. **When you download a dataset to use, you want to make sure to put it in that new folder so you can access it easily. We will go over this later but just keep this in mind for now.**



Notice how QuantMethods shows up on the top tab? That means that is the project you are in. The function **getwd** tells you where R is currently looking for stuff

```
getwd()
```

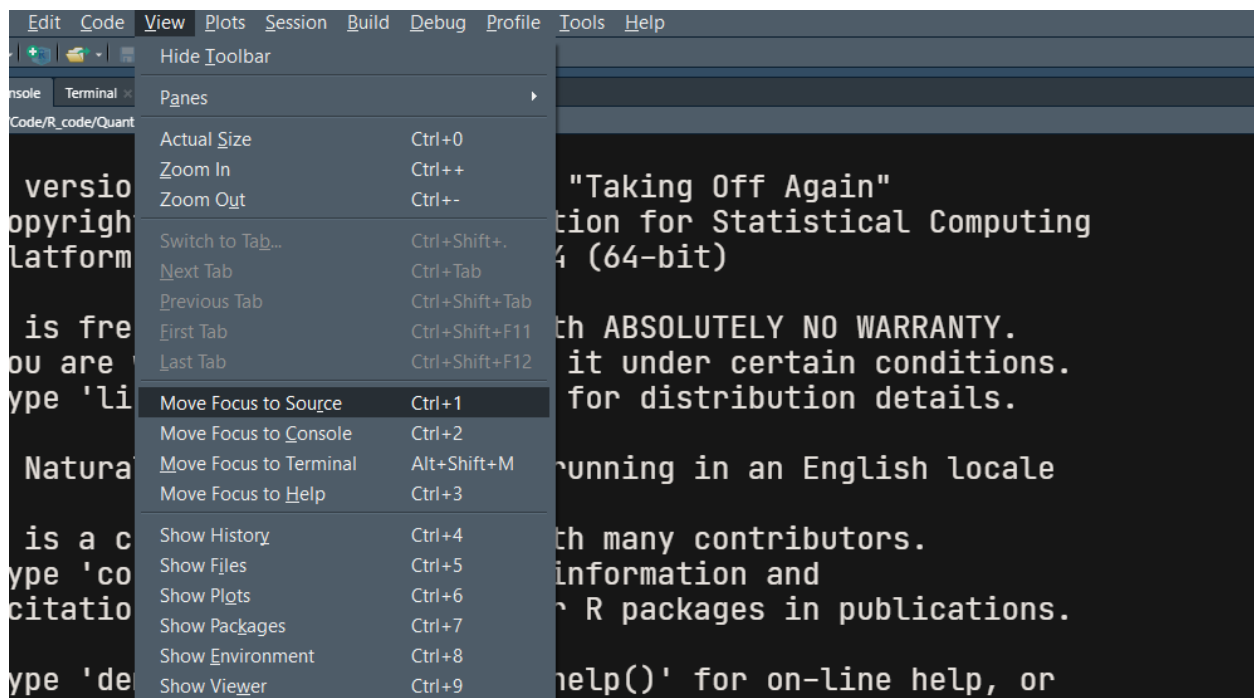
```
> getwd()
[1] "C:/Users/marck/Documents/Code/R_code/QuantMethods"
> |
```

So if I have a dataset to examine I would make sure to save it in the folder QuantMethods...

7. [Code](#)

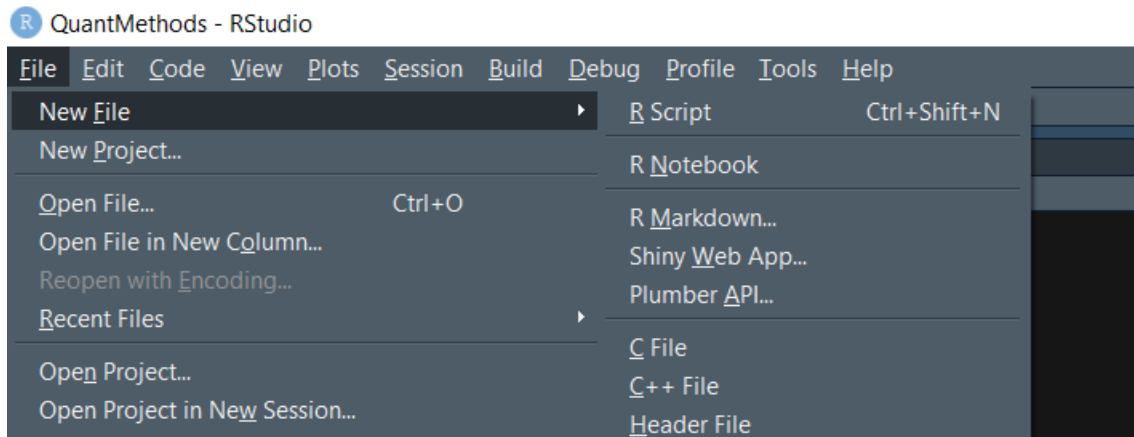
Ok, so you now know how to get going with R...but it still seems kinda silly. For example, writing in the **console** isn't very easy and you have to go line by line. The solution to this is to use the **source**, which is where you can write and then run code! Depending on how RStudio is setup you might not see the source when you open the program.

- Usually the **source** is in the upper left of the screen. If you don't see the source then you can go to the View tab and select "move Focus to Source" like I am doing below:

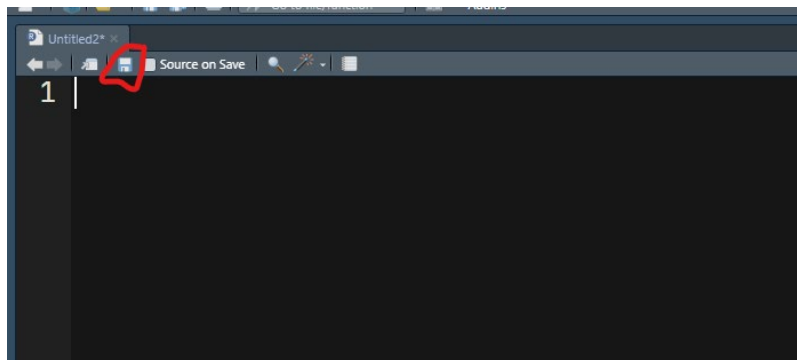


- The **source** is sort of a text editor. You write code in it (often called a **script**) and then 'send' it to the Console. Happily we can also save the text in the **Source** so we can rerun it later. To create a new script all you have to do is navigate to :

File>New File>New Script

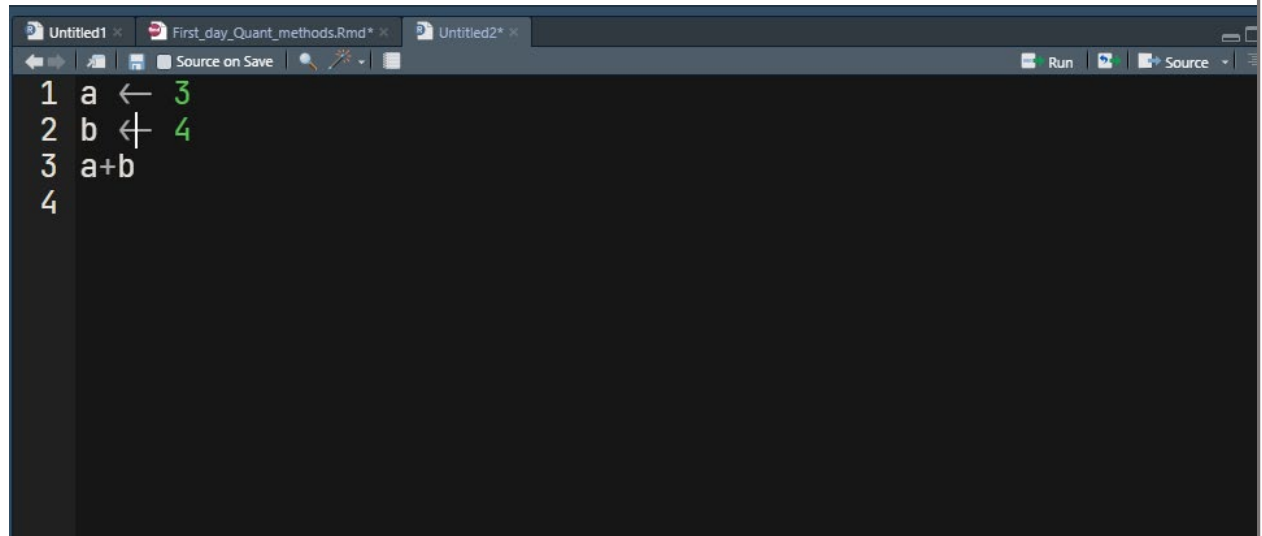


- To save the script you created click on the little disk icon on the top of the **source** pane (side note: how many of you have seen a disc like that in real life? Seems like a relic from another age of computers). A script written in R will have the file extension .R.



- Note how there is a little "*" next to the file name? that tells us that the file has new info that hasn't been saved yet. Nice feature to keep track of work.
- Once you have a new script window you can start to do some fun things. Below is a screen shot of some code. I also copied it below.
- In the script you now have open type the following code:

```
a <- 3
b <- 4
a + b
```

A screenshot of an RStudio editor window. The window has three tabs: 'Untitled1', 'First_day_Quant_methods.Rmd', and 'Untitled2*'. The 'First_day_Quant_methods.Rmd' tab is active. The editor shows four lines of R code:

```
1 a ← 3
2 b ← 4
3 a+b
4
```

The numbers 1, 2, 3, and 4 are in the left margin. The code is in a dark theme. The RStudio toolbar is visible at the top, showing buttons for 'Run' and 'Source'.

- To run the code highlight the whole section and click the run button (instead of the run button you can also use the keyboard shortcuts (on a PC it is *ctrl+enter* and on a mac it is *cmd+enter*)
 - o Fun R fact: if you have the cursor at the end of the line and hit the run button it will run that line of code!
- Note: the '#' lets you **comment out** the rest of the line of code. This is helpful when you are writing something since it tells you what a line is doing

Code Example:


```

1 install.packages("plotly") #this installs the packages we need
2 install.packages("gapminder") #this installs the packages we need
3 library(plotly) #load a package that lets us make an interactive graph
4 library(gapminder) # load a data package
5
6 p <- gapminder %>%
7   plot_ly(
8     x = ~gdpPercap,
9     y = ~lifeExp,
10    size = ~pop,
11    color = ~continent,
12    frame = ~year,
13    text = ~country,
14    hoverinfo = "text",
15    type = 'scatter',
16    mode = 'markers'
17  ) %>%
18  layout(
19    xaxis = list(
20      type = "log"
21    )
22  )|
23  ggplotly(p)
24

```

```

install.packages("plotly") #this installs the packages we need
install.packages("gapminder") #this installs the packages we need
library(plotly) #load a package that lets us make an interactive graph
library(gapminder) # load a data package

```

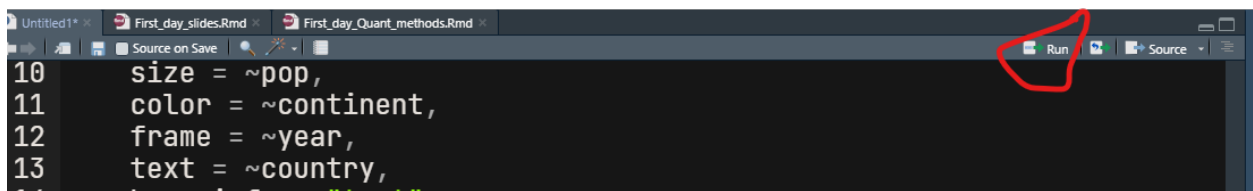
```

p <- gapminder %>%
  plot_ly(
    x = ~gdpPercap,
    y = ~lifeExp,
    size = ~pop,
    color = ~continent,
    frame = ~year,
    text = ~country,

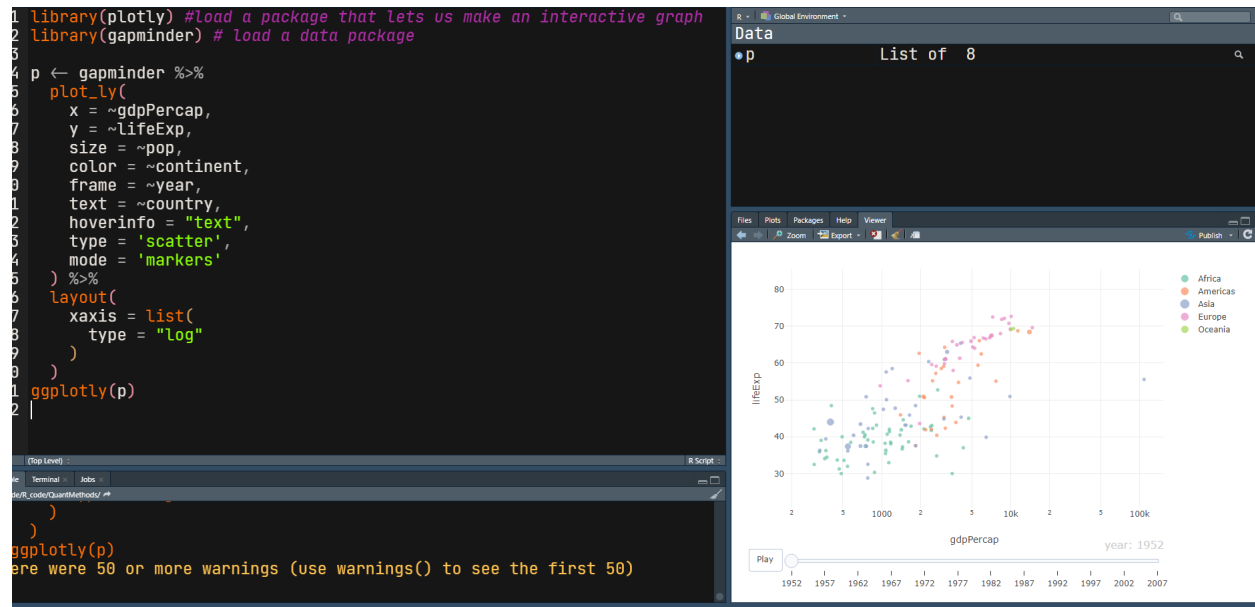
```

```
    hoverinfo = "text",  
    type = 'scatter',  
    mode = 'markers'  
  ) %>%  
  layout(  
    xaxis = list(  
      type = "log"  
    )  
  )  
  ggplotly(p)
```

- This is a complicated code with a lot going on. But also note how with only a few lines of code you can create really impressive visualizations.
-
- copy and past the code above into your **Source**. Note that you will need to install 2 packages (don't worry. These are pretty cool ones...).
- Once you have copied it, click on the run icon on the upper right of the **Source**



- You should see the result pop up in the Viewer panel (usually on the lower right of the screen).



- Also note that the code you wrote in the script is 'sent to' the console and then run.

Here is one more code you can check out:

```

install.packages("leaflet")

library(leaflet)

leaflet() %>% addTiles() %>% setView(-81.680244, 36.214232, zoom =17)

```

If you have gotten this far, congrats! This was a lot to go through. Don't worry if you have this intimidating or hard. It took me **years** to get a handle on this stuff and I still find myself lost at times. The point of this course is to introduce you to these ideas and give you some practice. No one learns this stuff overnight. We will build and come back to these concepts throughout the semester.

