**Biostatistics**

**Lab 2**

*Tasks*

Tour of R Studio, Commands, Assignment Operator, readr

*Introduction*

Yesterday we installed R, R Studio, and a number of basic packages to increase functionality. Today we are going to get the lay-of-the-land in R Studio and learn how to enter some basic commands.

*Tour of R Studio*

Before you start R Studio, create a new folder named Lab1. You can place this folder wherever you like on your computer. Download the data file for today and put it in this Lab1 folder. Once you have done this, start R Studio.

As we saw yesterday, when we start R Studio, the interface window has three panels. The biggest is on the left and is called the console. If we imagined just the console by itself, we can get an idea as to what base R looks like. It is really just a command line prompt with some options in a menu bar at the top.

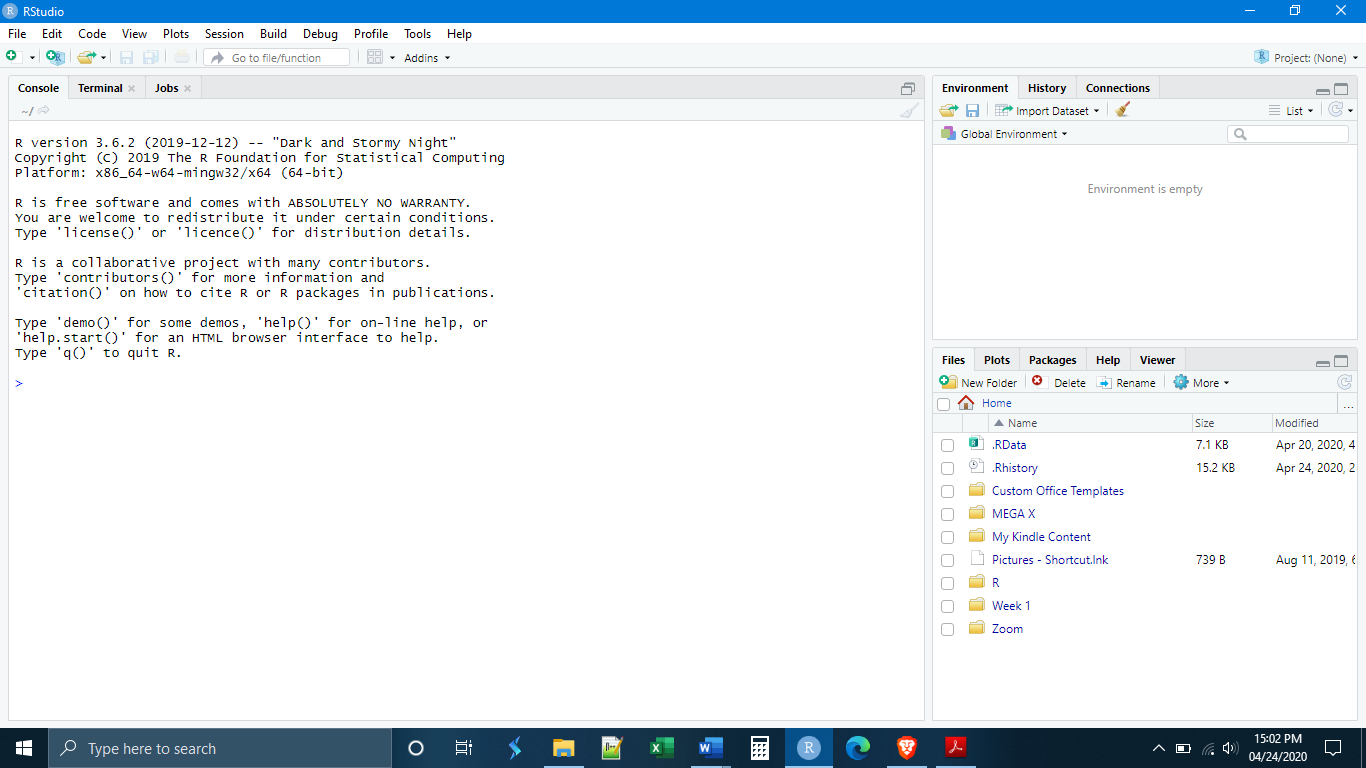
In the top right we see another panel. This panel contains information about the environment and history of your work in R. The environment includes what data have been imported, what functions are active, and what saved objects are active. The history is just what it sounds like, a history of the commands you have executed. You can toggle between the environment and the history through the tabs at the top of the panel.

The bottom right panel is a miscellaneous panel with a number of tabs granting access to a number of different things. The first tab is the “files” tab. Like other command line programs, unless told to do otherwise, R Studio will look for files in what is called the “working directory”. A “directory” is a word used in computer science to refer to a location where files are stored, in other words a folder. The working directory is the folder that you are currently “inside” of. Upon opening R Studio, the working directory will be set as the folder where R Studio has been installed. The files tab lists the files in the current working directory. We will see ways to change the working directory later. We will not worry about the “connections” tab for now.

The next tab is the “plots” tab. If we click on it, we will see that it is empty. Plots that are generated from the console will appear here.

After the plots tab there is a “packages” tab, which will show what packages are installed. Note that most of them have an unchecked box next to the name. We can understand what this means if we continue with our analogy from yesterday of comparing R packages to computer applications. In order to use a computer application, you have to open it first. To type in Word or browse the internet in Firefox, I first must start Word or Firefox. On the other hand, some applications on your computer are always on, such as the clock, and you do not need to explicitly open them in order to use them. Similarly, in R, many packages need to be explicitly loaded before you can use them (like the tidyverse). In other cases, however, the package is always active, so long as R is running (such as data.table). The unchecked boxes here indicate that the package has not been explicitly loaded during this session.

There are also “help” and “viewer” tabs that we will not worry about at this time.



**Files, Plots, Packages**

**&**

**Help**

**Environment**

**&**

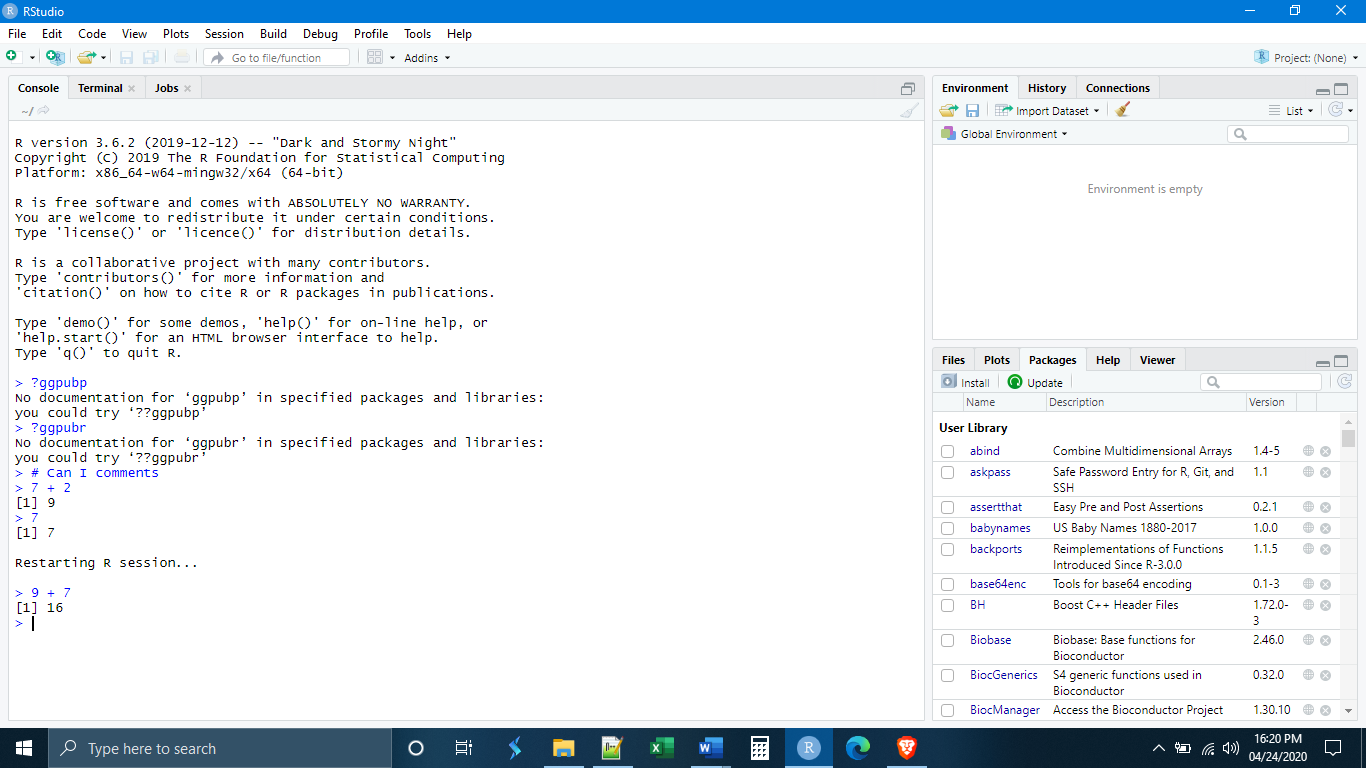
**History**

**Console**

Let’s start by entering some simple commands into the console. We can use R like a simple calculator. In the console, type

9 + 7

If we press enter, what do we see? It simply gives the sum as the output:



We can do this with all kinds of mathematical operations. Experiment with some.

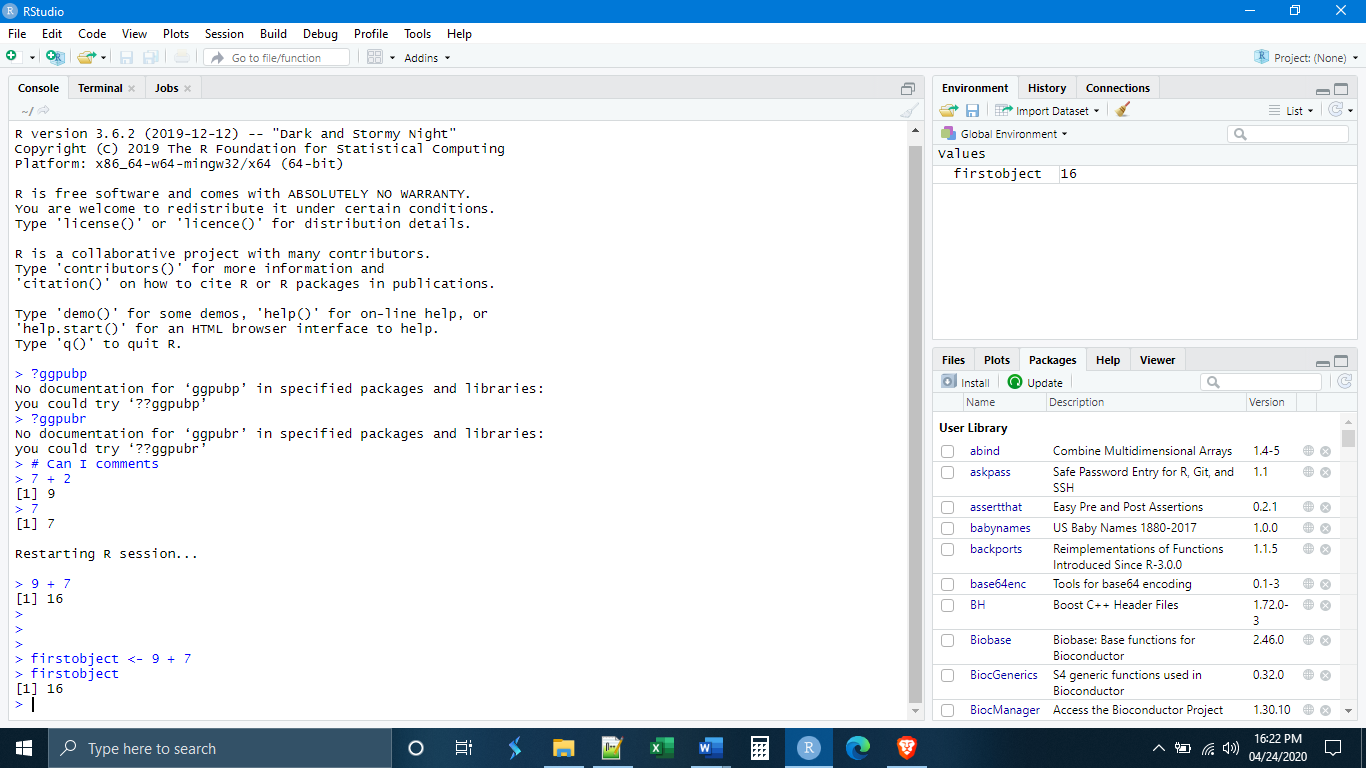
What if we wanted to save the answer of our operation for some other task later? The way things are saved in R is through what is called the “assignment operator”. This is used to create what are called “data objects” or simply “objects”. An object can be anything. It can be a number, a list of numbers, a table of numbers, a list of characters (letters), or a table of characters and numbers, a plot or graph, etc. Those different characters and numbers can also have a variety of different formats, which we will talk about later. It is important to realize that saving something and creating a data object are the same action. Every time I save something, I create a new object.

There are actually three ways to create objects, but Hadley Wickham has advocated for a specific convention that we will follow. This convention is to only use the “left assignment operator”, which has the form <-. In other words, the “is less than” symbol followed by a dash or minus sign. It is intended to resemble a left-pointing arrow (hence “left” assignment operator). The left assignment operator uses the following syntax:

NameOfSavedObject <- ObjectBeingSaved

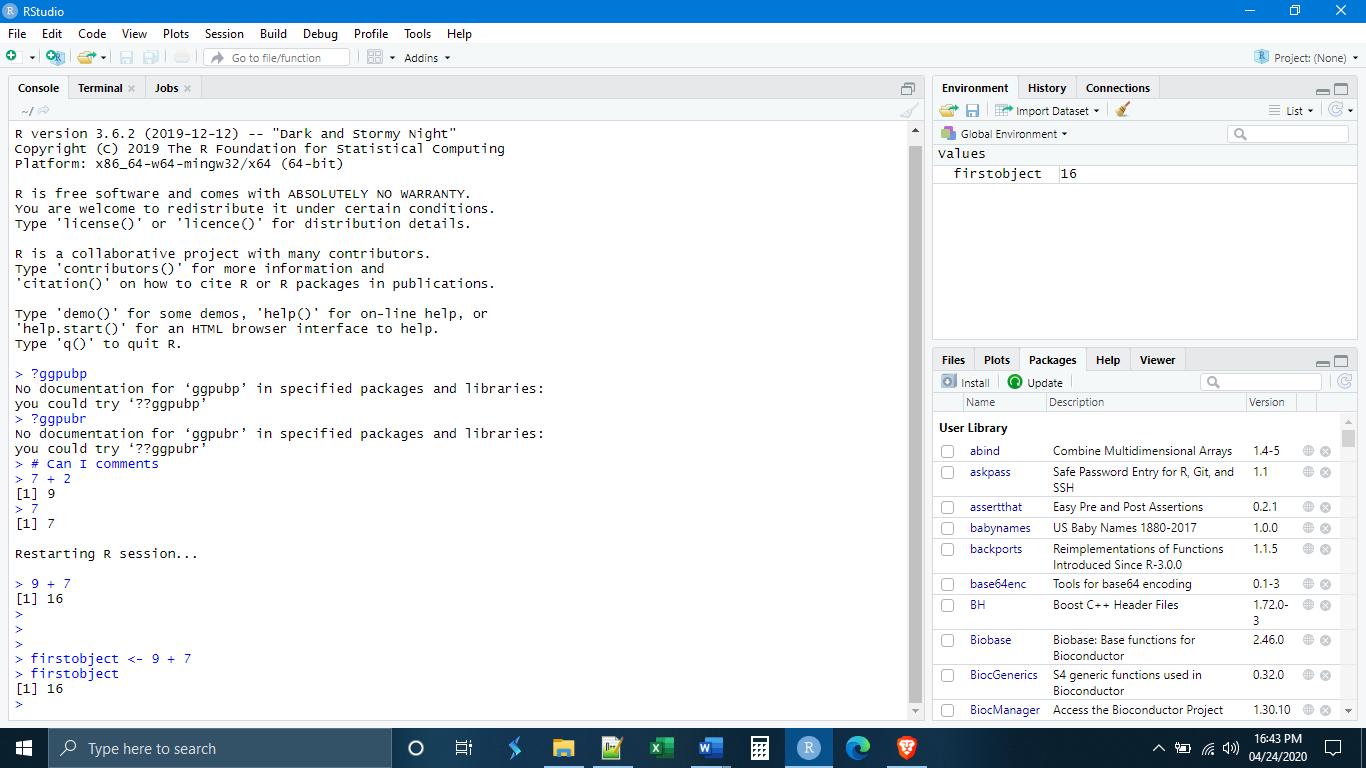
Whatever is to the left of the arrow is the new name of the object that you are creating, and whatever is to the right of the arrow is the content of what you want that object to be. Let’s make our first object:

firstobject <- 9 + 7

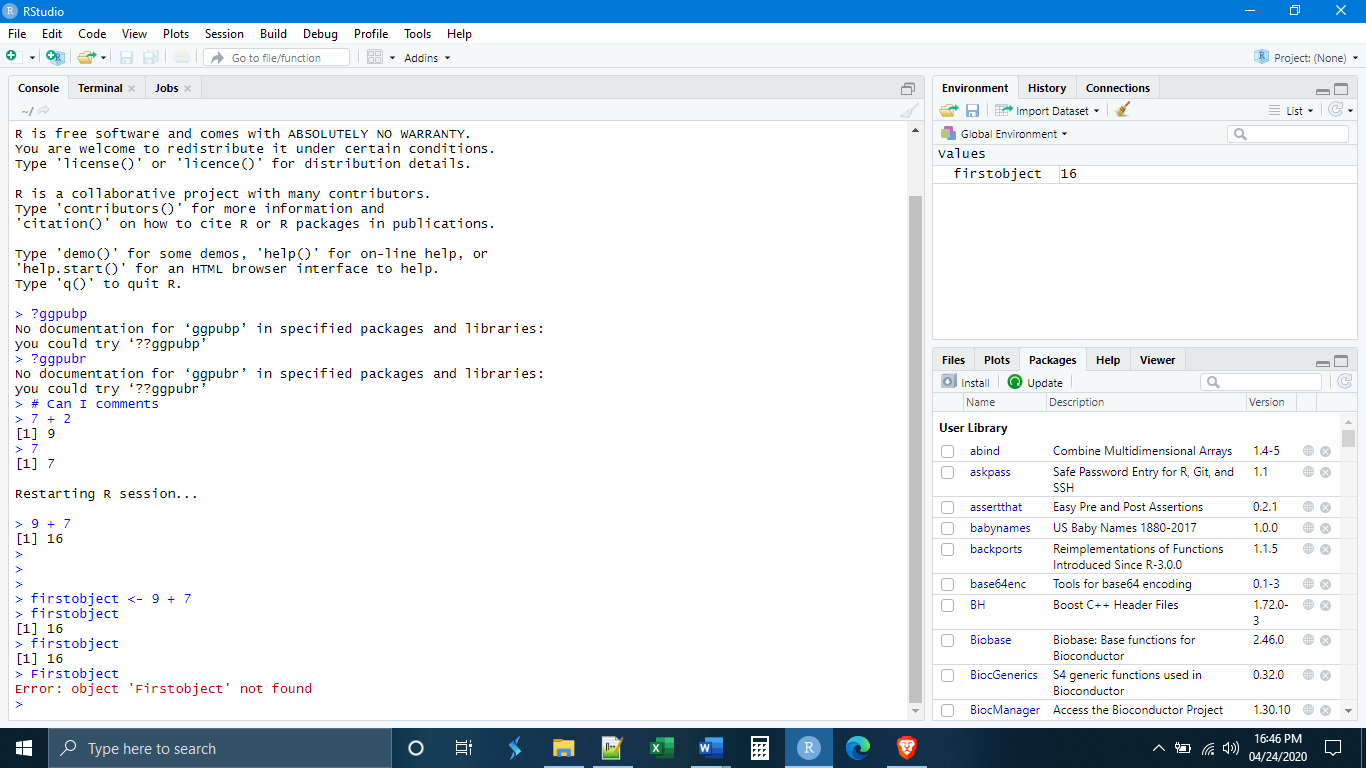


A few things need to be noted here. First, see that we now have something in our environment! We have the name of our object firstobject, followed by a description of that object, in this case just a single number, 16.

Second, note that R Studio did not give us an output in the console. This will be the case whenever we create an object. If we want, we can see our object by “calling” it, typing it in the console and pressing enter. What do we see when we do that?



The third thing that we want to note is that R did not save the object as “9 + 7”. R recognized that both were numbers and simply saved the sum. What happens if we try to call our object by typing: Firstobject?

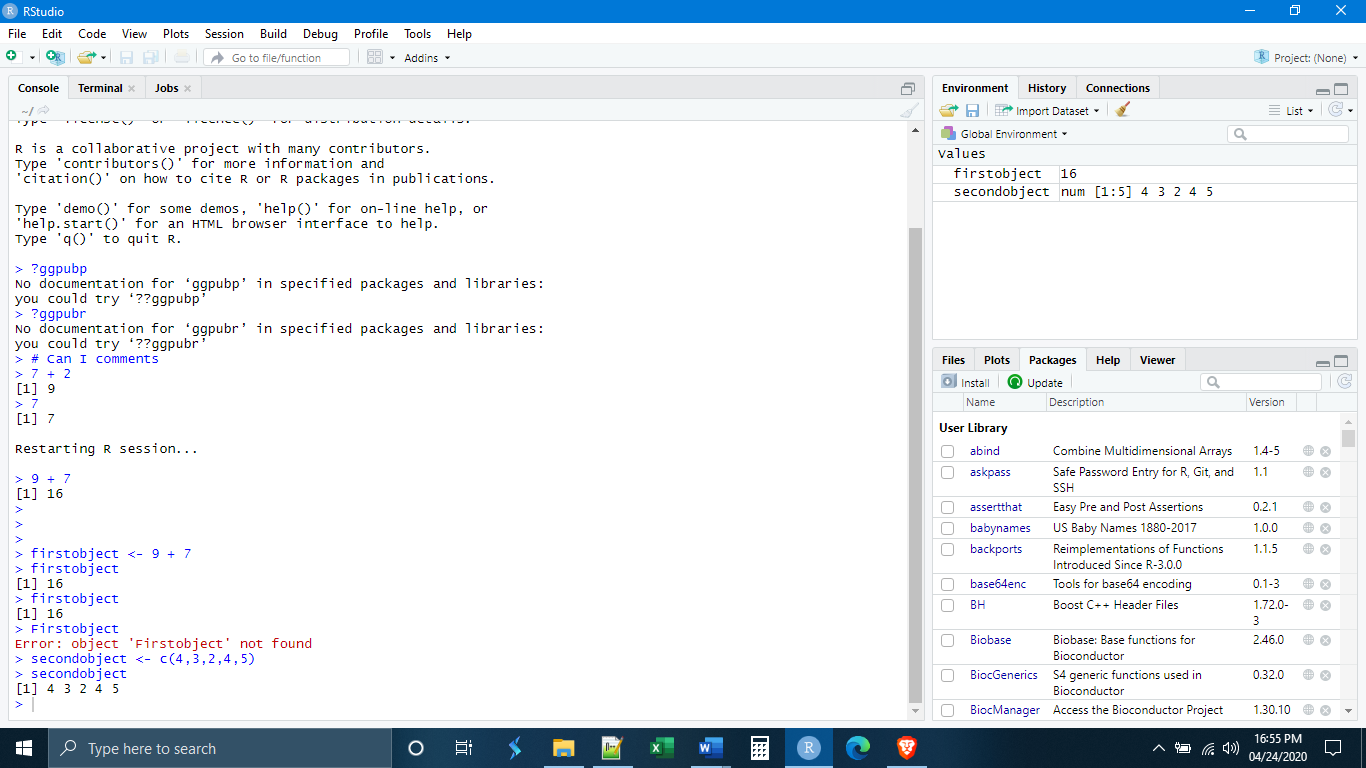


R is case-sensitive. If you try to call an object you must type it in exactly as you saved it. Thankfully, if you are working in R Studio, R Studio will suggest objects as you type by using autocomplete, helping to eliminate this problem.

Let’s make a second object, a vector.

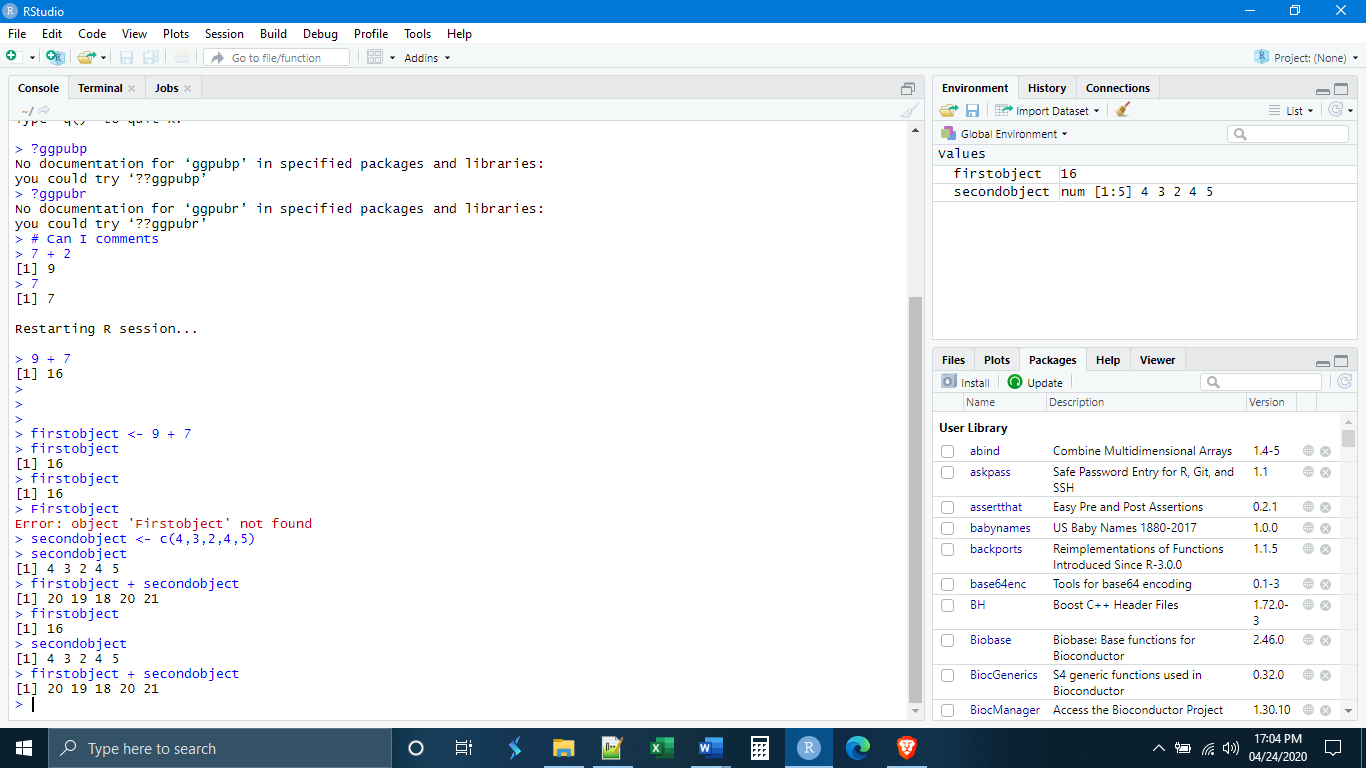
secondobject <- c(4,3,2,4,5)

What happens when we call this second object?



We will learn about different data types later, for now, a vector is simply a one-dimensional collection of data points, be they numbers or letters. The way that you indicate what is inside of the vector is with the function c(), meaning “concatenate”, where what is inside of the parentheses are the contents of the vector. We might be tempted to call this a list, but a list means something specific in R, so we should be careful not to use that word. Remember, if you want to save something you always need to use the assignment operator. Simply entering the vector will not save it.

How do our objects interact? What happens if we add firstobject to secondobject?



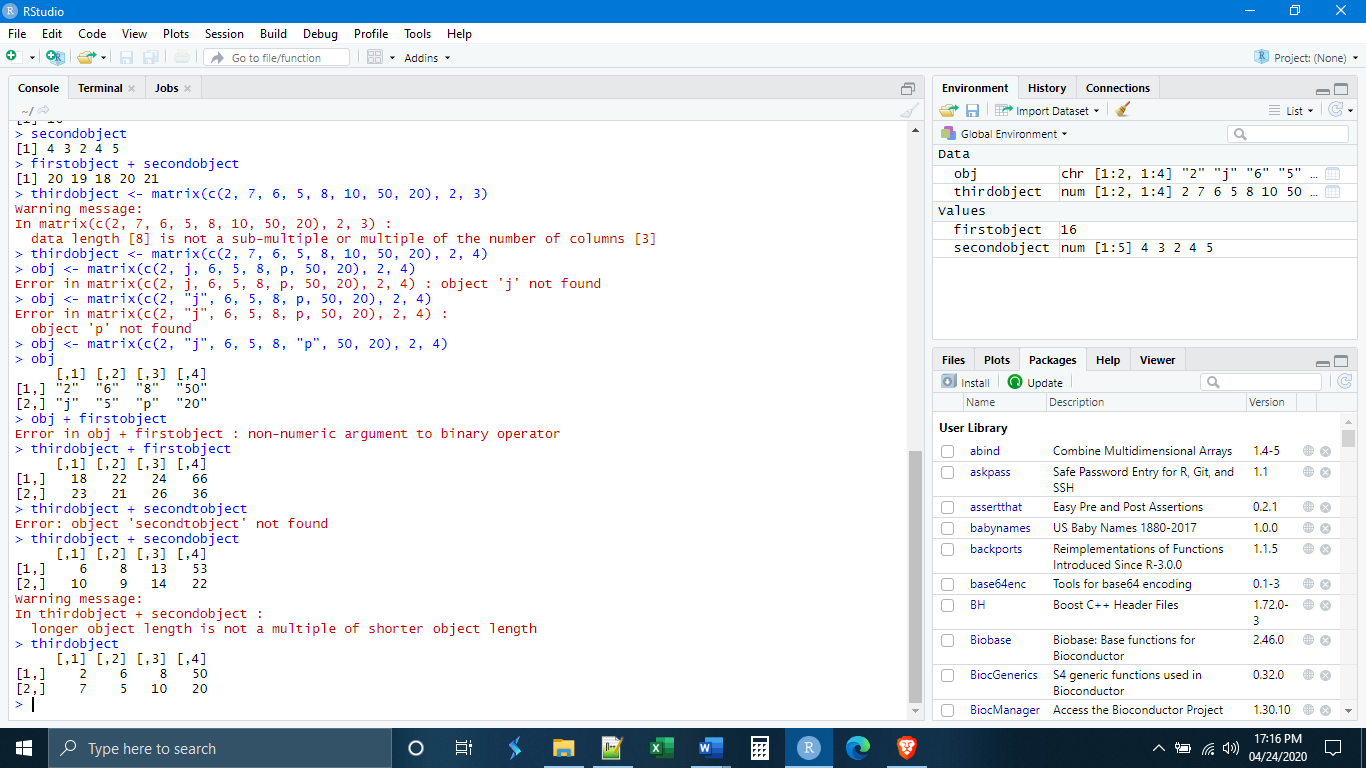
We see that firstobject was added to every element of secondobject. This is a very important point about vectors. Anything done to a vector is done to each element of that vector.

Before we move on to other ways to enter commands in R Studio, let’s make one more object, a matrix.

thirdobject <- matrix(c(2, 7, 6, 5, 8, 10, 50, 20), 2, 4)

The matrix() function will turn the input into a matrix. A matrix is a two-dimensional collection of data elements (again, be they letters or numbers). The syntax for the matrix() function is:

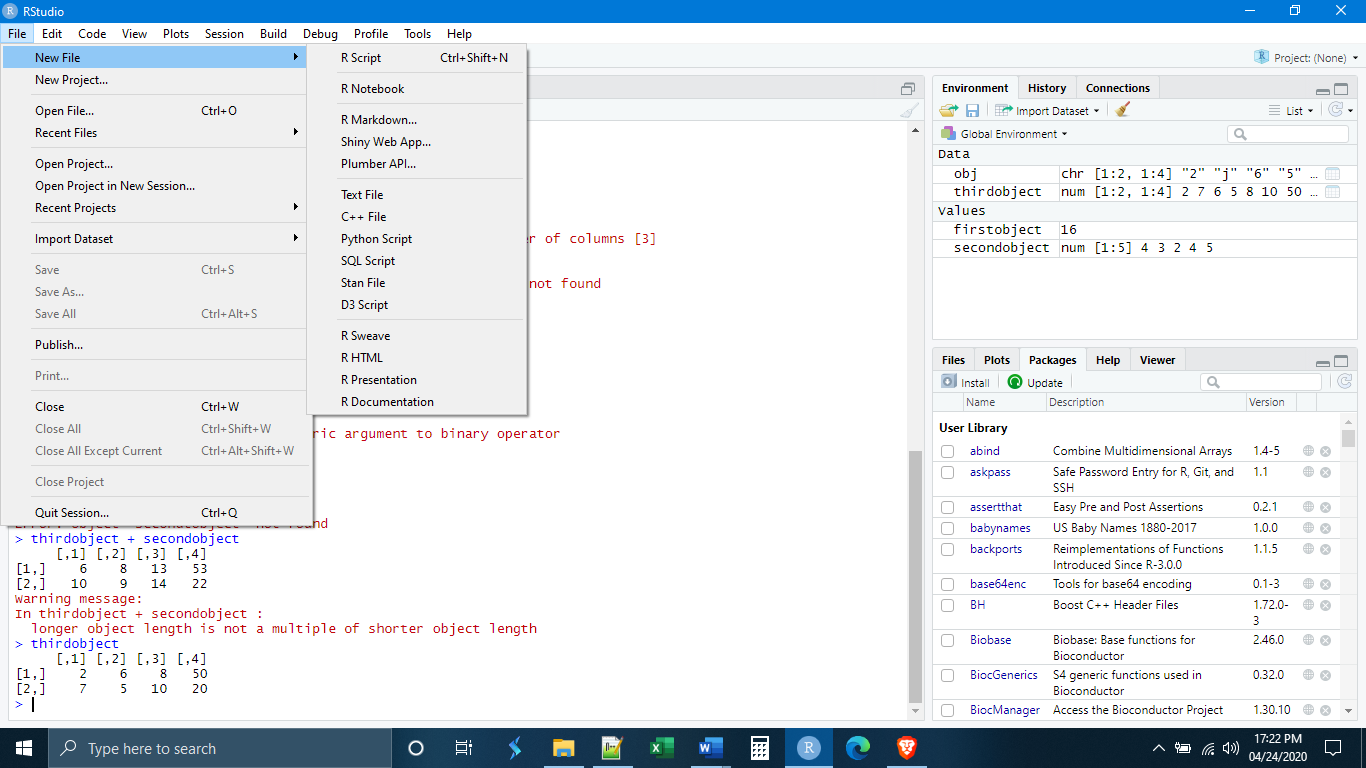
matrix(c(ListOfElementsInTheMatrix), NumberOfRows, NumberOfColumns)



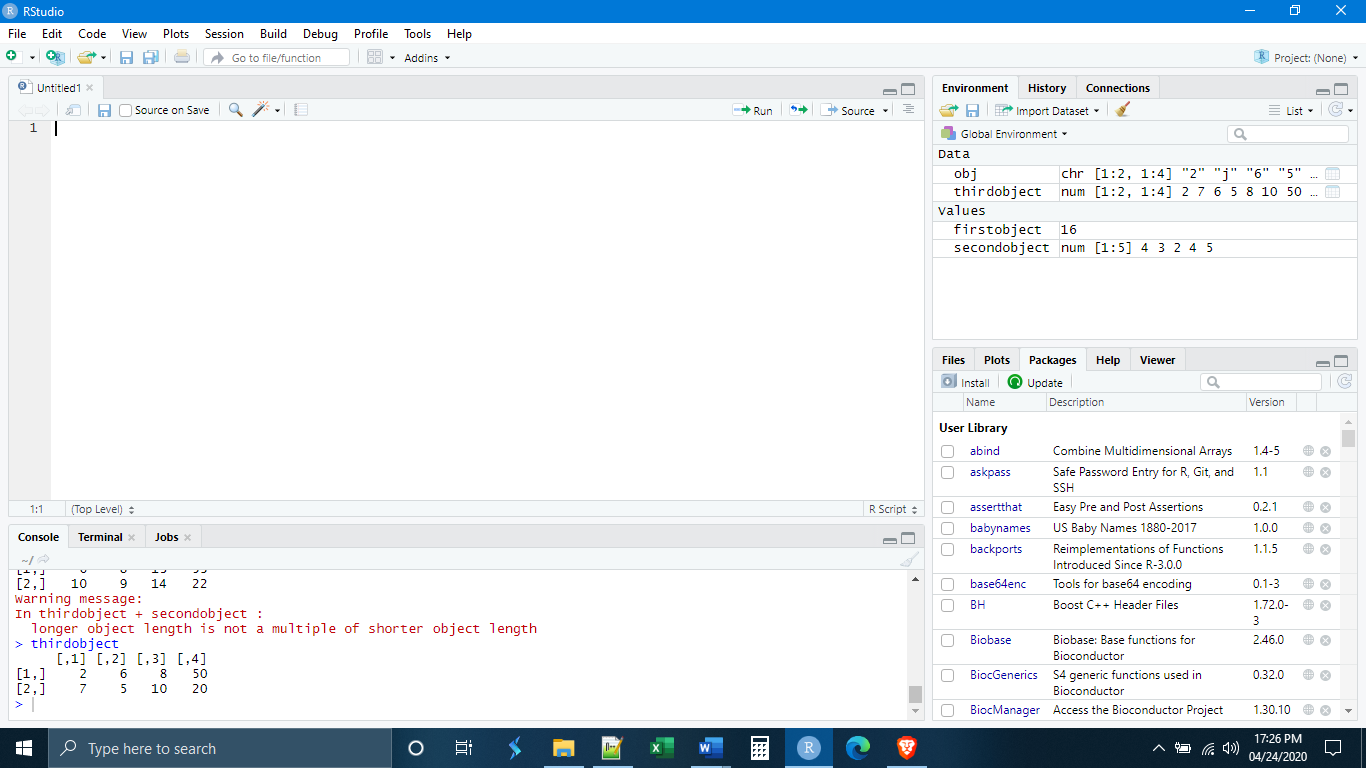
*Scripts, Notebooks, and Markdown Files*

The console is fine for doing simple calculations as we have been doing, but it has some serious drawbacks. First, you will notice that you can only feasibly enter commands one line at a time. Second, going back to previously entered commands can be a bit cumbersome. Third, there isn’t a really good way to save your work and come back to it.

All of these difficulties can be overcome by the other ways to enter commands in R Studio. If we go to file, new file. We see that there are a number of different file types that can be created in R Studio. We will focus on the first three, R Scripts, R Notebooks, and R Markdowns.

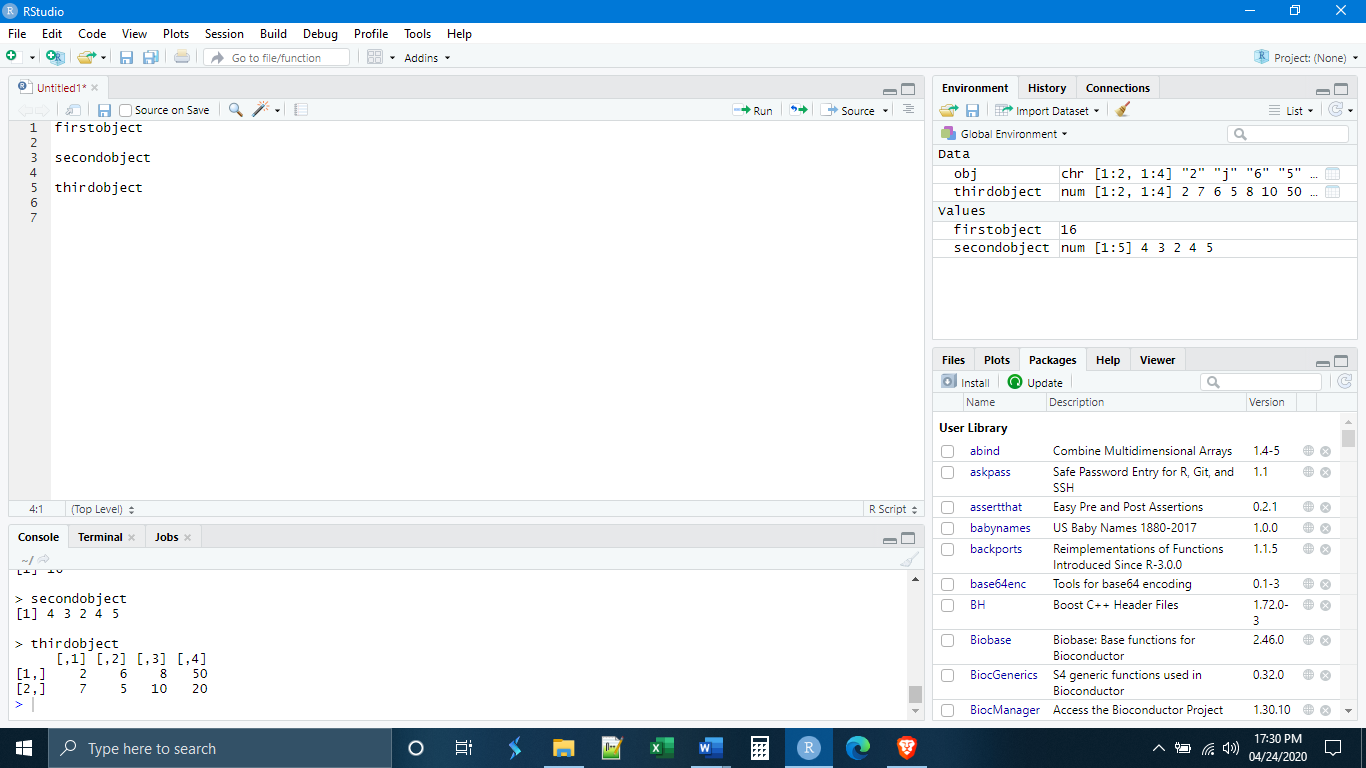


Begin by opening a new R script. Notice that immediately the layout of the panels in the R Studio window change.

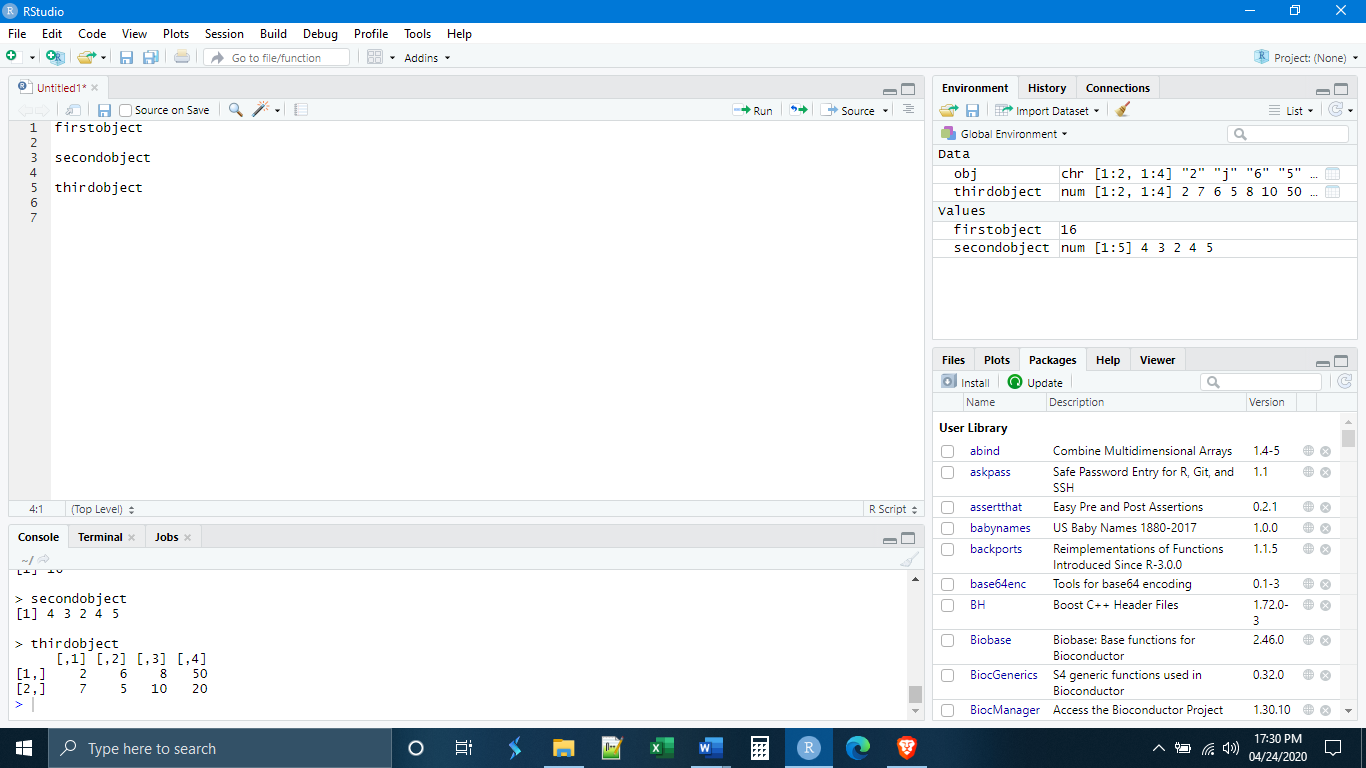


We now have a fourth panel, which is our script. The script editor works just like a simple text editor, with the difference that we can also execute R code from within it. The console panel has been made smaller and placed below our script panel.

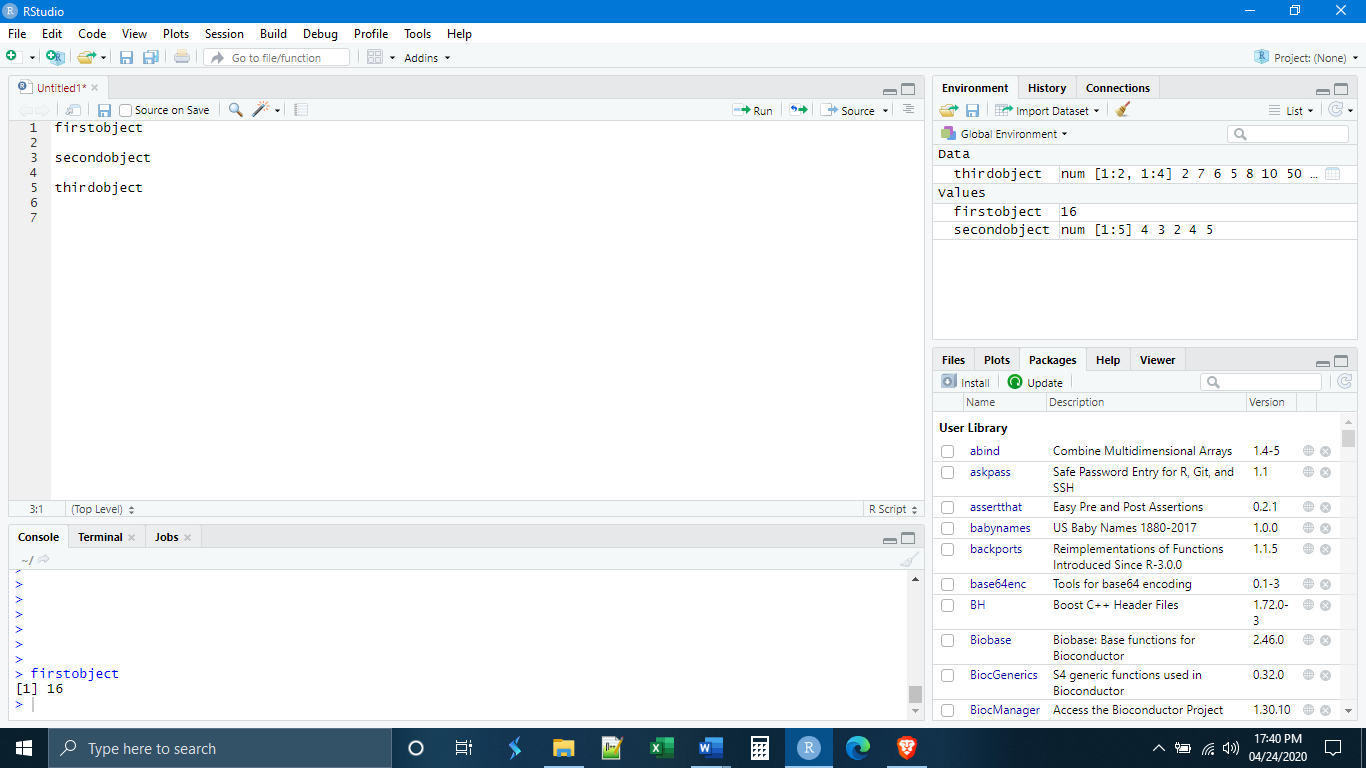
Let’s begin by entering our three objects with a space in between each of them.



In order to get the three objects on different lines, we had to use the enter key. Since this was the key we used to execute commands in the console, how do we execute commands in a script? There are two ways. The first way is to put your cursor at the beginning of the line you would like to execute and the use the keys ctrl + enter (cmd + return for Macs). The other way is to put the cursor at the beginning of the line you would like to run and then click the “run” button in the top right of the script window (see next page).

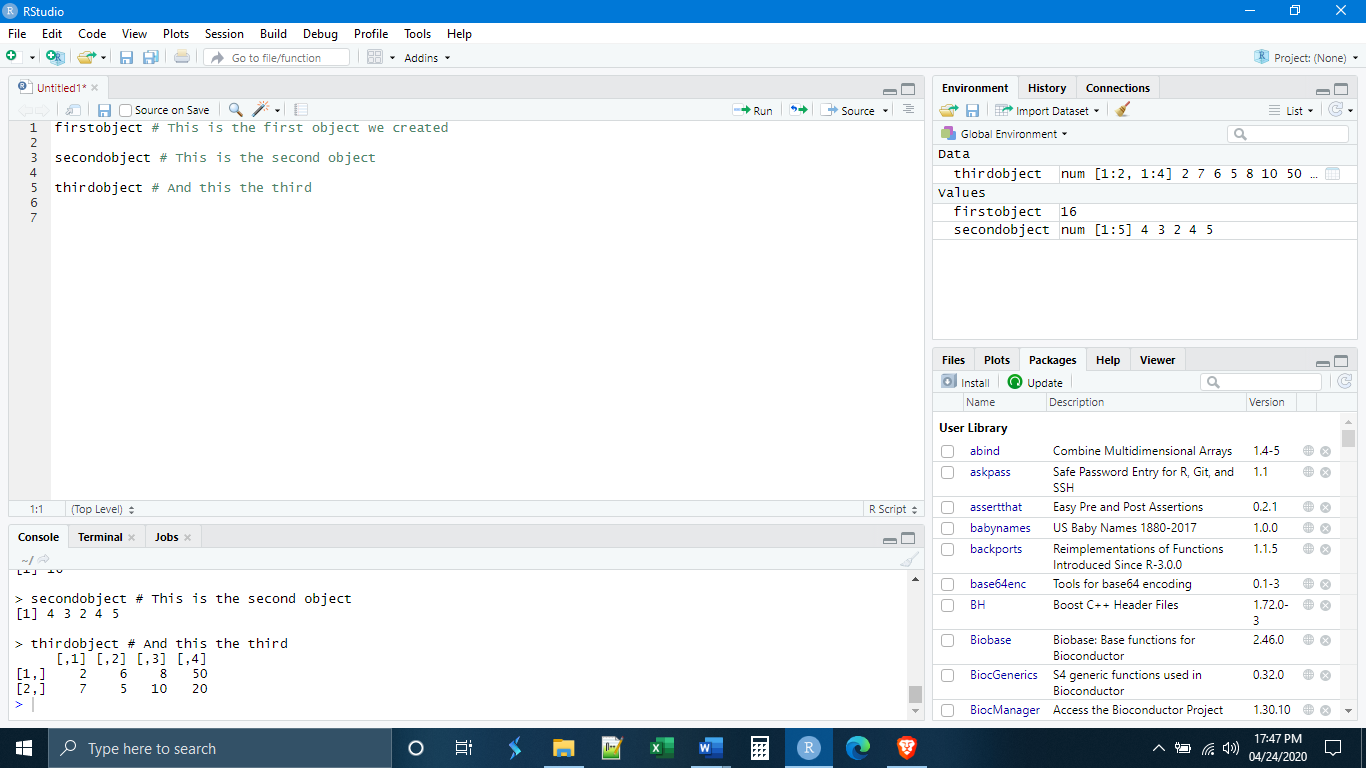


Let’s execute the first line. What do we see? The output appears in what remains of the console window below.

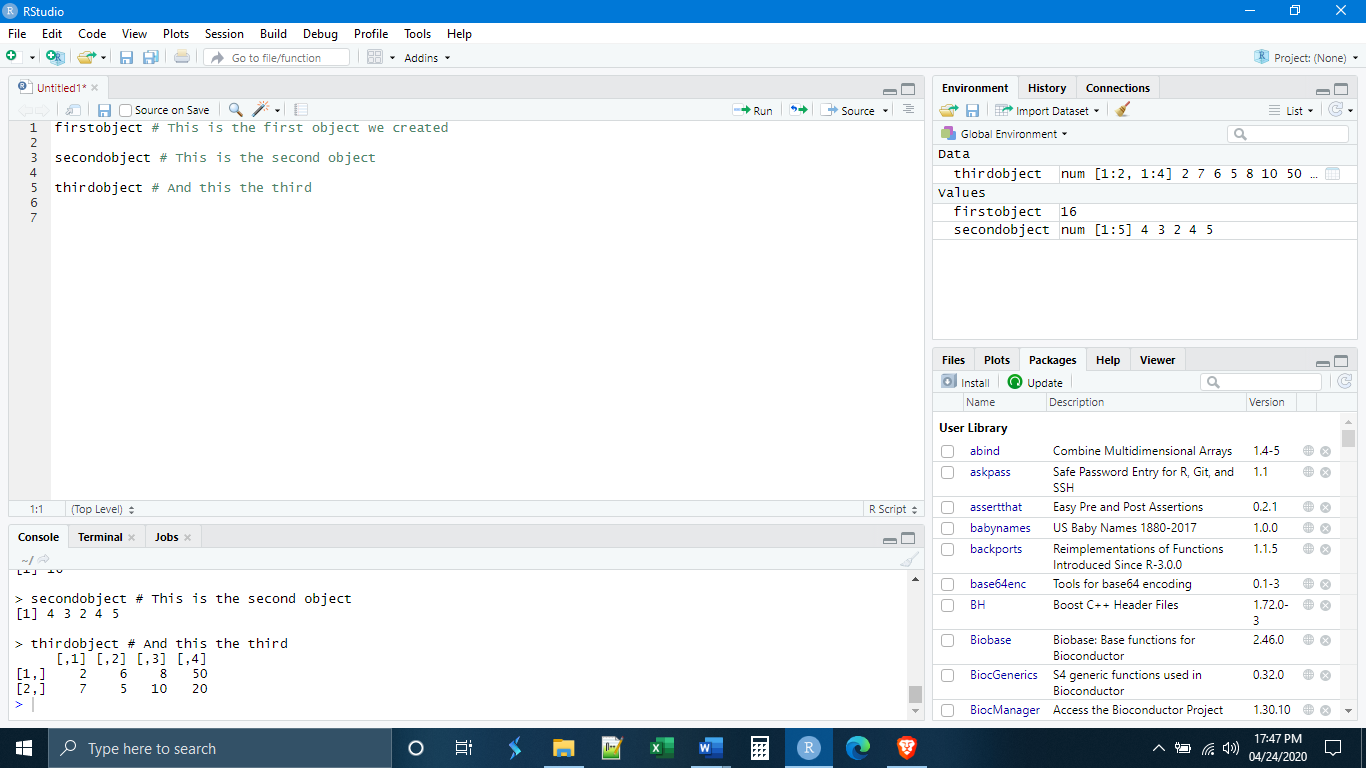


We can also execute everything in the script simultaneously by placing the cursor at the beginning of the script and using the keys ctrl+shift+enter (cmd+shift+return for Macs). Place your cursor on the first line and execute all of the code.

Using a script also allows us to make use of comments. R will ignore anything that comes after a hashtag/pound sign (#). Thus, we can put comments in our code to remind ourselves, or someone else, what we have done. When the code is executed, the comments will be ignored.

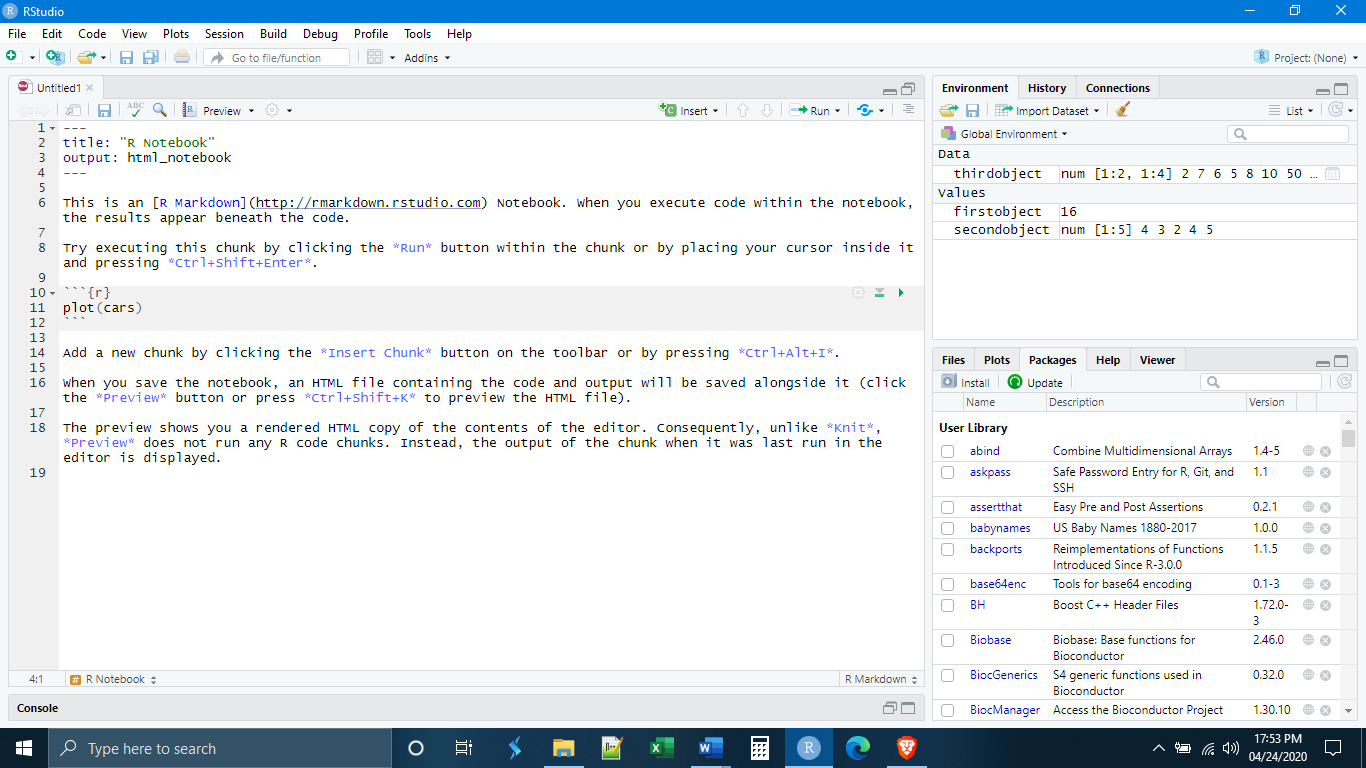


Lastly, notice that we can save our script and come back to it later using the save button at the top of the script panel. Save this script in the Lab1 folder you have created. It is VERY IMPORTANT to realize that, once you save the script, R Studio will immediately change the working directory to the folder where the script has been saved, in this case Lab1.

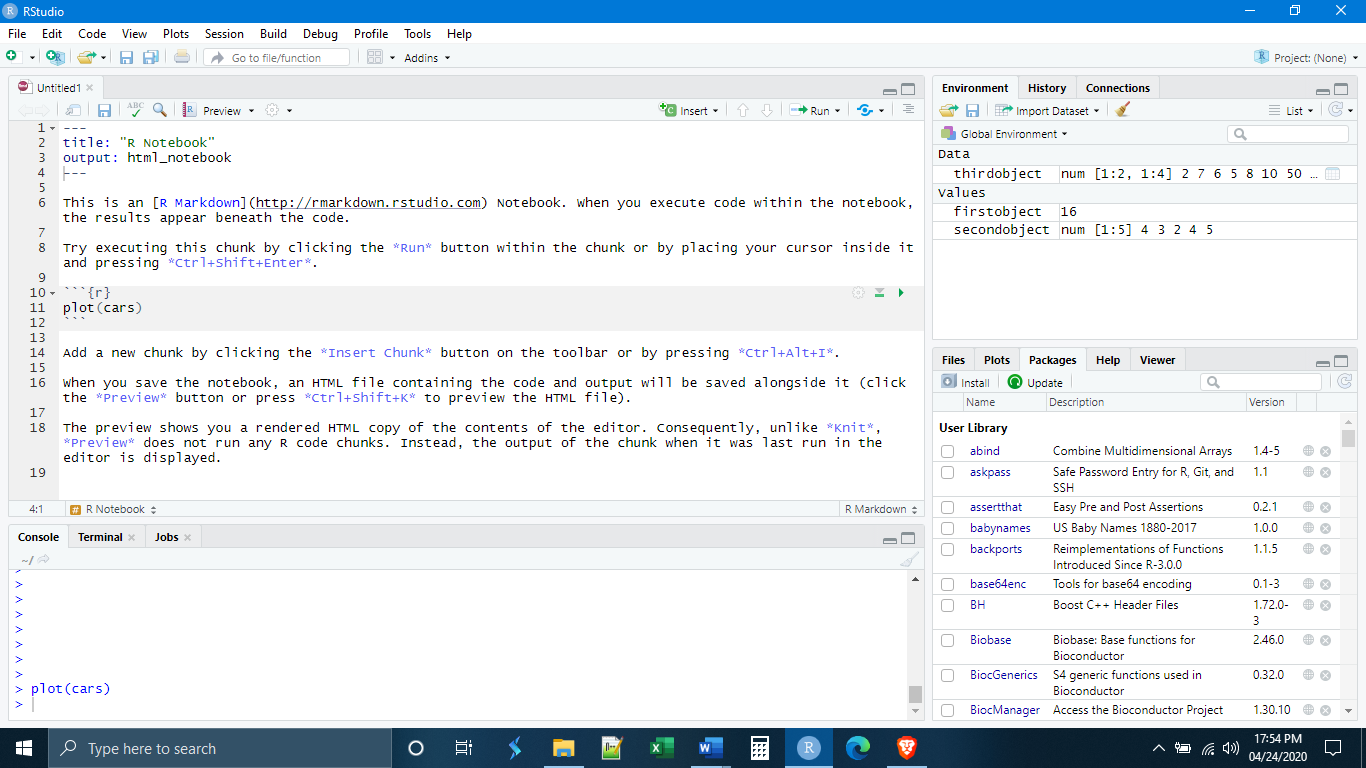


There are two other kinds of files in which you can execute R code: R Notebook and R Markdown. These are very similar. Both function as ways to organize your work in the form of a report and allow you to export your code, and the results of that code, as an .html, .docx, or .pdf.

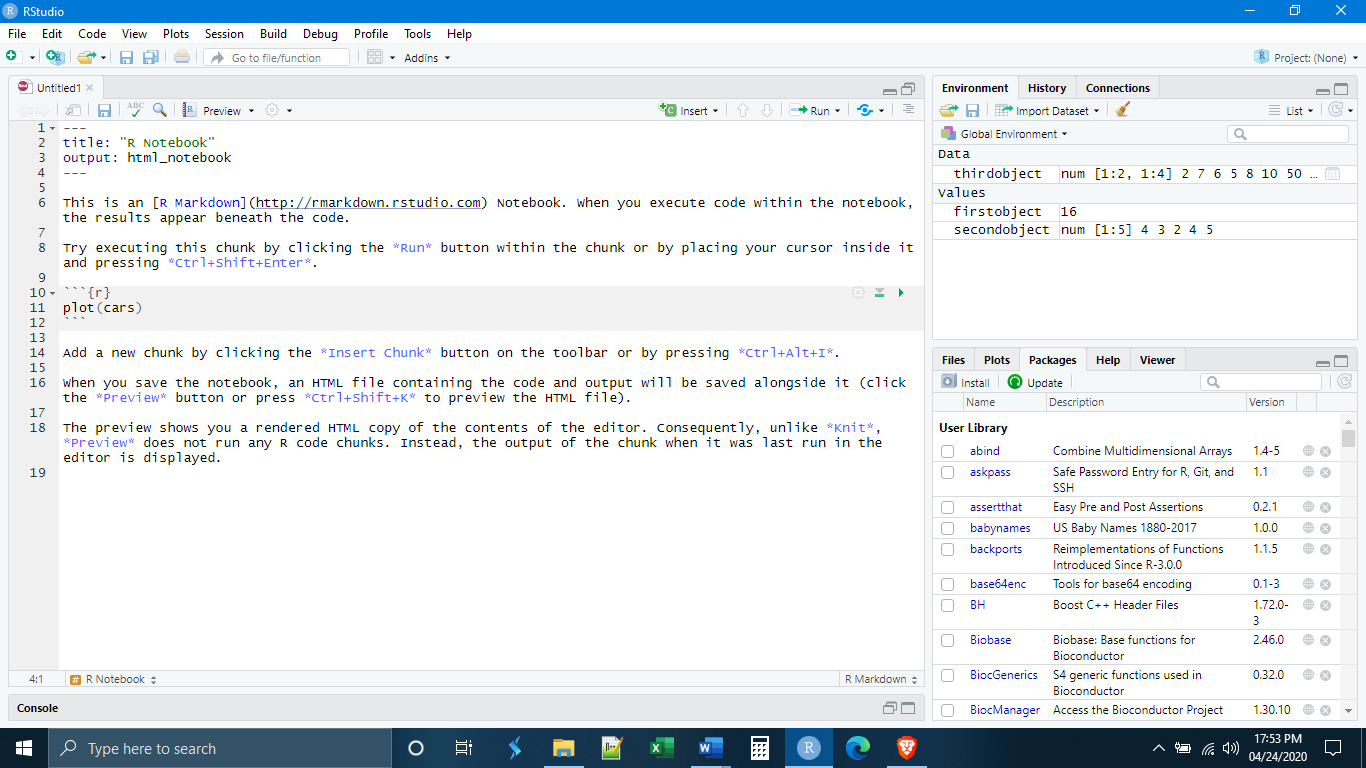
Let us begin by opening up an R Notebook. Notice that when we open the notebook, the console is completely collapsed to the bottom of the window and is no longer visible.



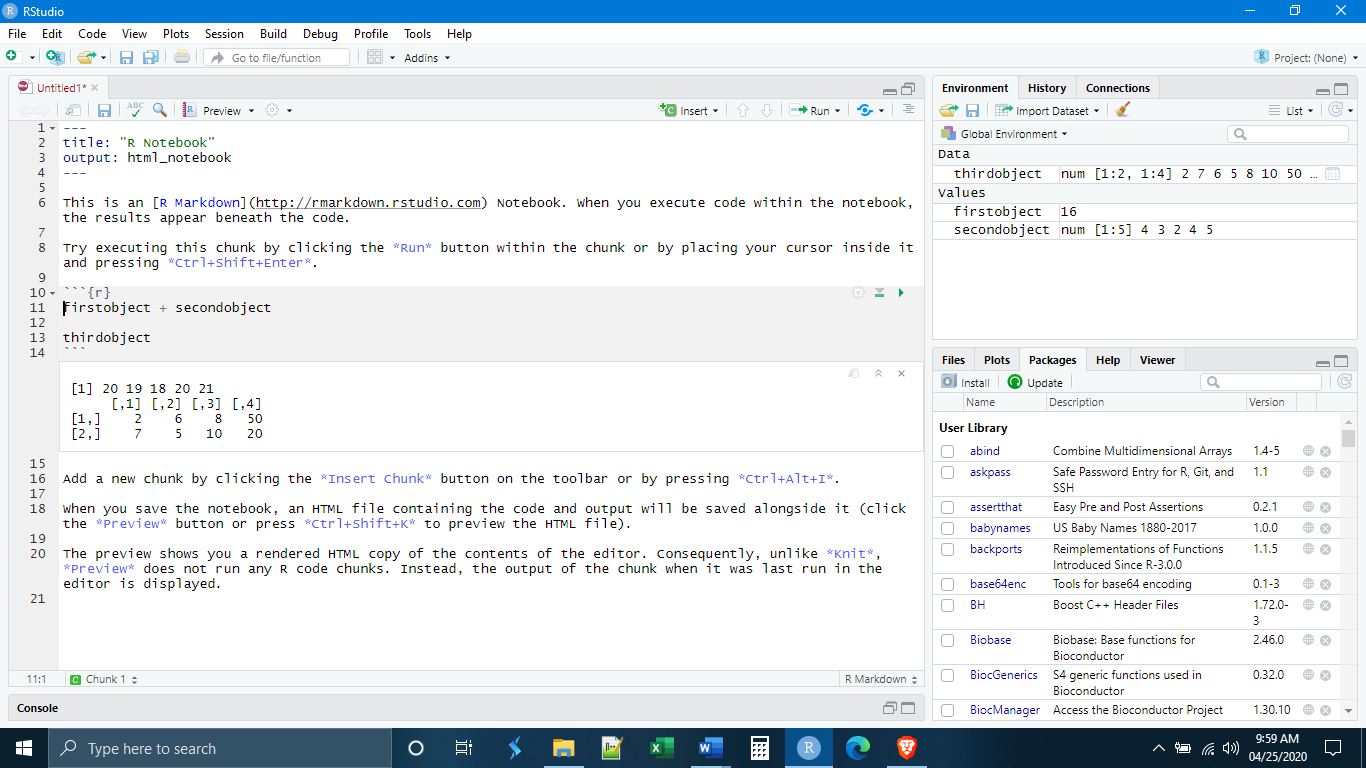
If we really want to have our console back, we can double click on it and it will pop up in the way that it did with the script. Part of the point of the notebook and markdown documents, however, is that they do not need the console, as we will see.



Unlike the console or the script, not all areas of a notebook or markdown document can execute R code. By default, the notebook or markdown document functions as a basic text editor. In order to execute R code, it must be put inside of a “chunk”. In the template that appears at the start of a new notebook or markdown document there is already one chunk present. The chunk is identified at the start by the sequence ```{r} and is ended by ```. The apostrophe-like symbol is not an apostrophe but a different character found just to the left of the number 1 key on your keyboard. Each chunk must be given a unique name at the beginning ```{r NameOfChunk}.



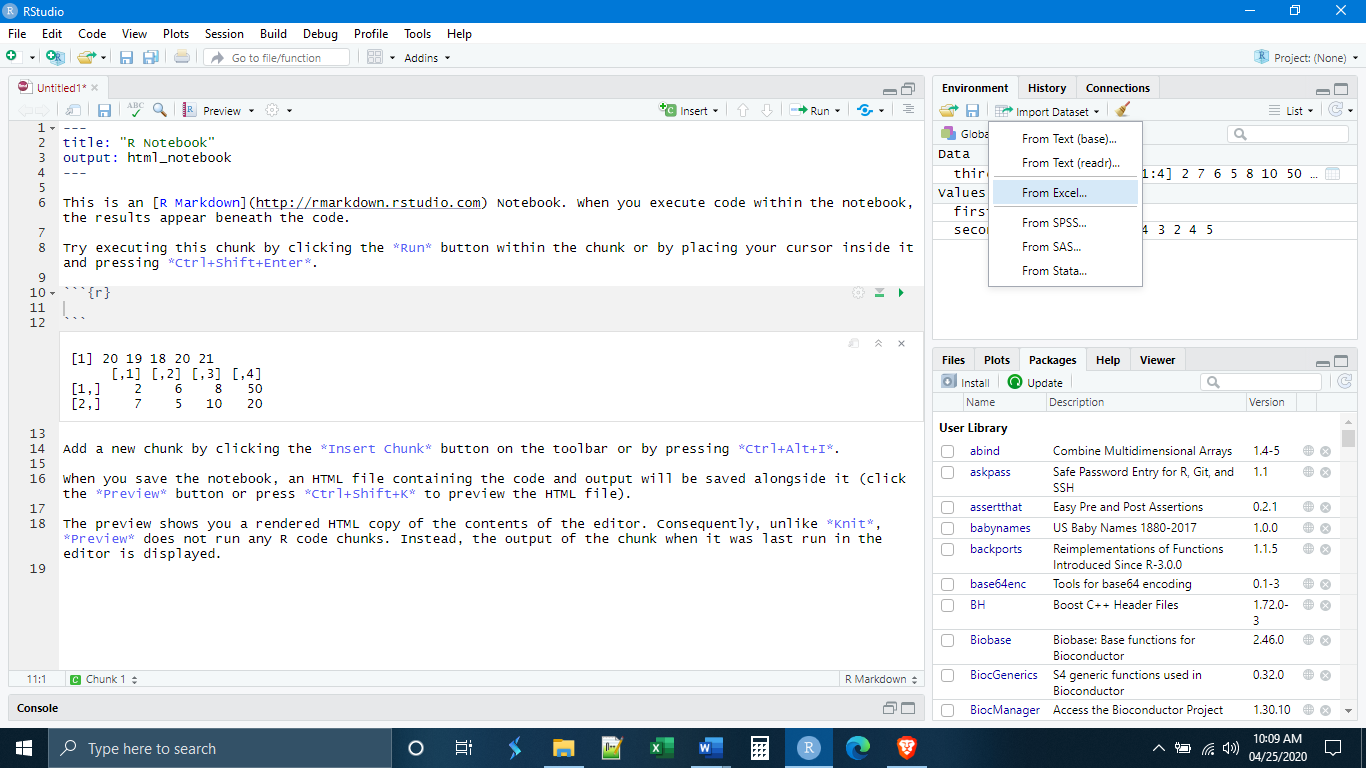
The results of any code executed inside the chunk will appear just below the chunk.



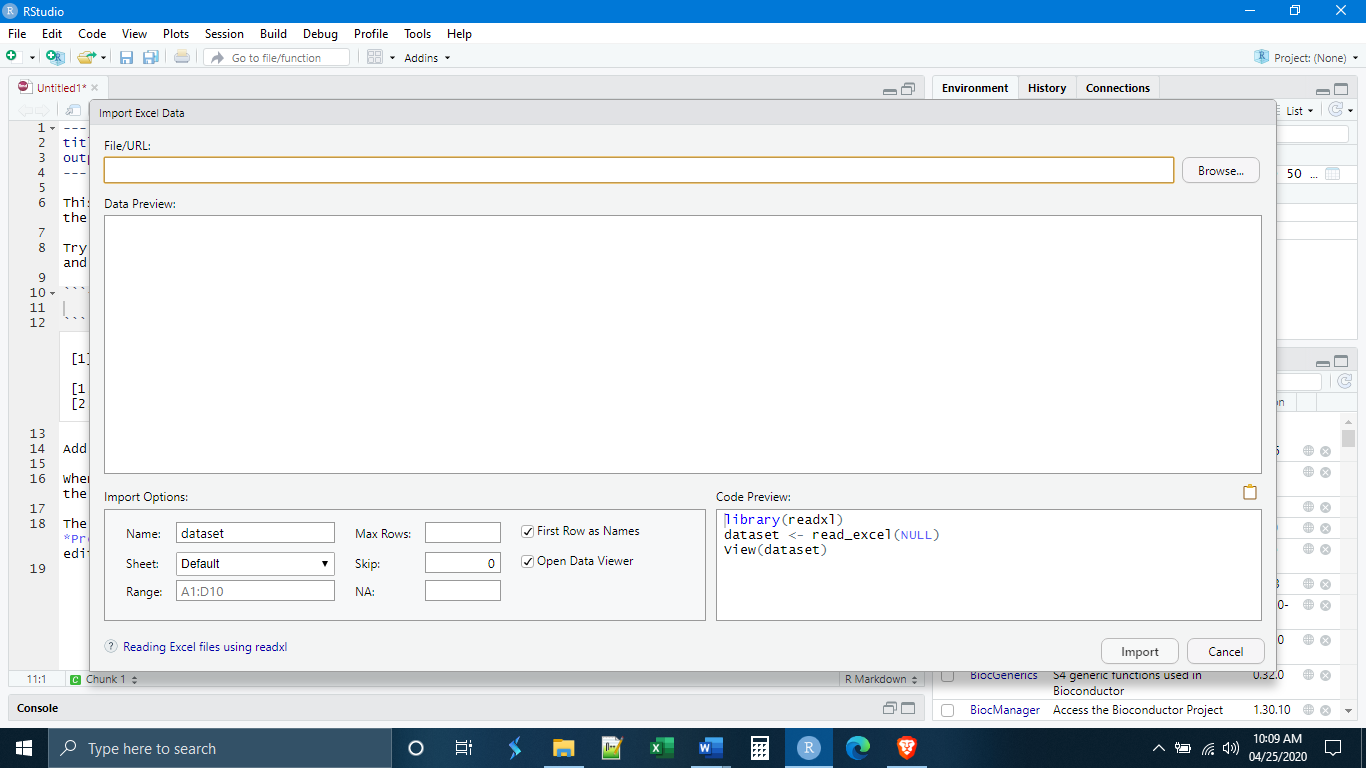
*Reading Files*

It is rare that you will enter data manually into R as we have been doing thus far. Most likely, your data will be stored in some other spreadsheet program, such as Excel, and you will want to import it, or “read” it, into R in order to work with it.

Within R Studio there are two ways that data can be read-in. The first way, which we will not be using, is through the environment panel. Click on the button that says “Import Dataset” and you will be prompted with the next steps.



Notice that this is really just a way for R to generate the code for you automatically, and R Studio shows you the code that will be entered.



The second way to read in data is by manually entering the code. Even here, there are a number of ways to do this. For now, we will focus on the read\_csv() function from the readr package in the tidyverse. I highly recommend that you take a look at the readr cheat sheet that is on blackboard. We can see that the basic syntax for read\_csv() is:

read\_csv(“filename.csv”, col\_names = TRUE/FALSE)

There are other options in read\_csv(), which we won’t worry about at the moment. Note that this will only read in files that are in comma-delimited .csv format, which is a standard format for data tables. To convert an Excel file to .csv, use Save As and change the file format to “CSV (Comma delimited)”. It is generally a bad idea to read in Excel files directly because Excel adds a lot of additional formatting that can cause problems with data importation.

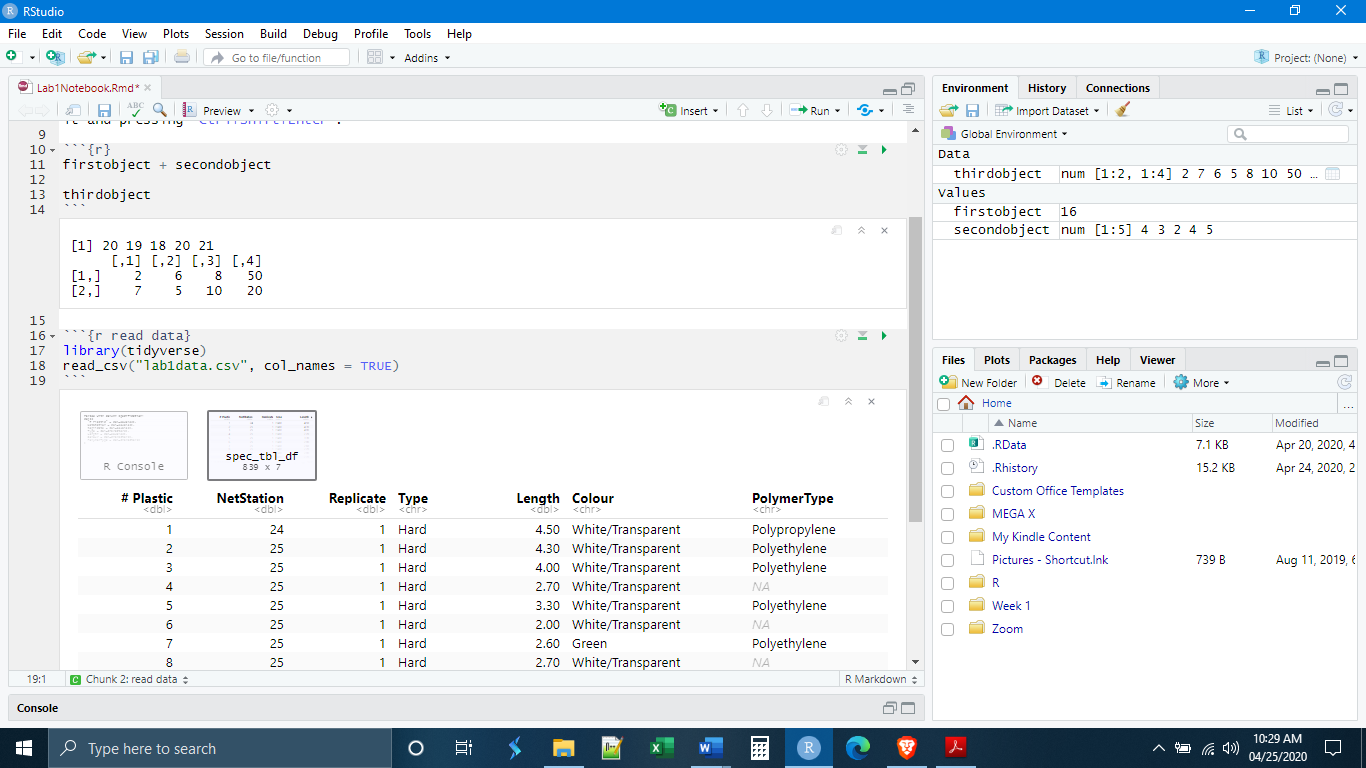
First, we want to save the notebook we are working with. Save it with whatever name you like, but it must be saved in the same Lab2 folder where you downloaded the data for today. This will be our working directory. Recall that, unless told to do otherwise, R will only look for data files within the working directory. By default, the working directory is the folder that the script/notebook/markdown is saved in.

When you have done this, make a new chunk and type the following code:

libraries(tidyverse) #readr is not part of base R, so we need to load it before use

read\_csv(“lab1data.csv”, col\_names = TRUE) # our .csv file includes column names

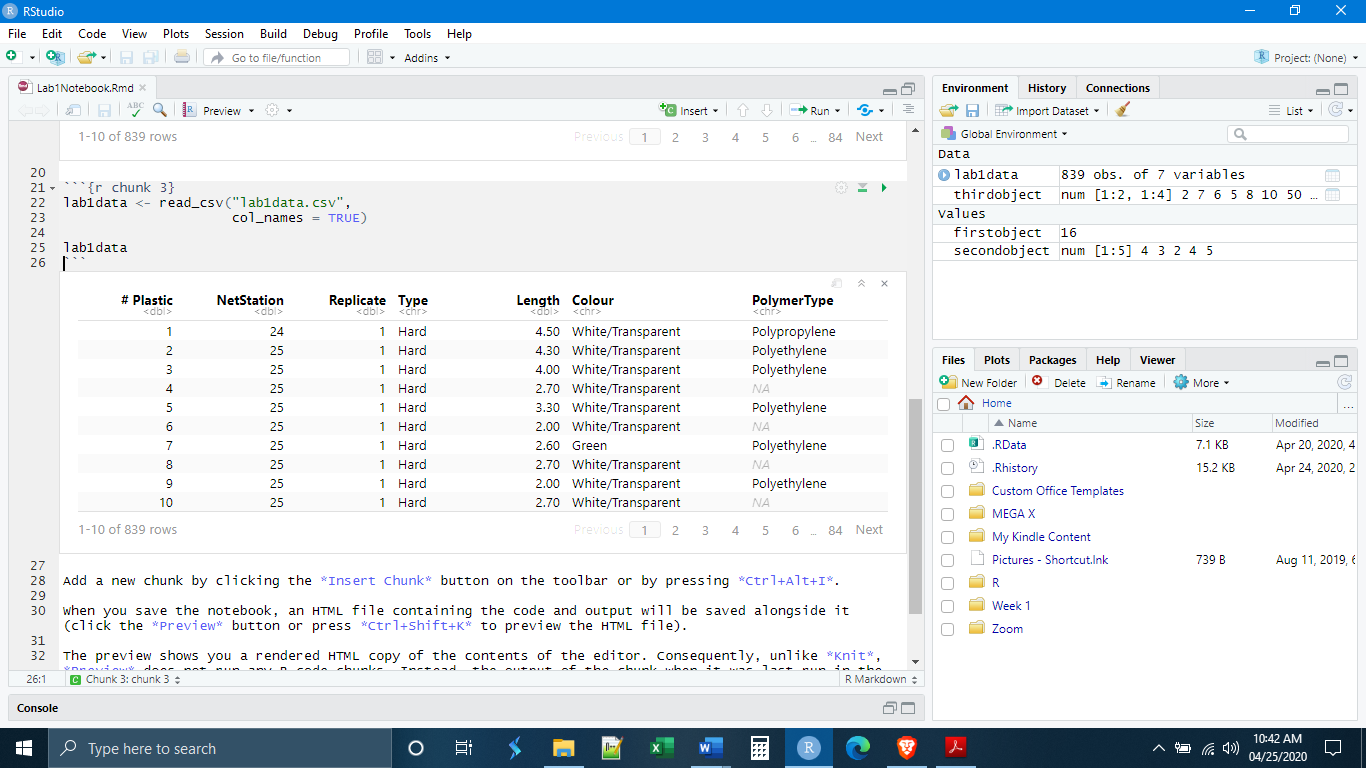
You can then execute the code line-by-line or all together.



You can see that the data appeared just below the chunk in an interactive data table. But notice that the data are not part of our environment. This is because we never turned them into an object using the assignment operator.

Let’s make a third chunk and this time save the imported data as an object. There is no need to load tidyverse from the library again. You only need to do this once per session.

Now we can see that our imported data appears in our environment as one of the objects.



*Individual Exercises*

**1.** Make a new chunk called “chunk 1” and write code that will complete the mathematical operation .

**2.** Make a new chunk called “chunk 2” and write the code that will create the data object “vector1” with the contents of the vector being the numbers 1, 2, 3, 4, 5.

**3.** Make a new chunk called “chunk 3” and write code that will multiply the result from exercise 1 (by the data object vector1 created in exercise 2. (Hint: you will need to create a new data object for the result of exercise 1)

**4.** Make a new chunk called “chunk 4” and write the code that will import the ToothGrowth data set (found on Blackboard), and save it as a new data object. (Hint: make sure you write a line of code with the library() command to load the tidyverse)

**5.** Make a new chunk called “chunk 5” and write the code that will import lab1data2.csv (found on Blackboard), and save it as a new data object. (Hint: do the data need/have column names? What does that mean for col\_names = ?)