**Biostatistics**

**Lab 14**

*Tasks*

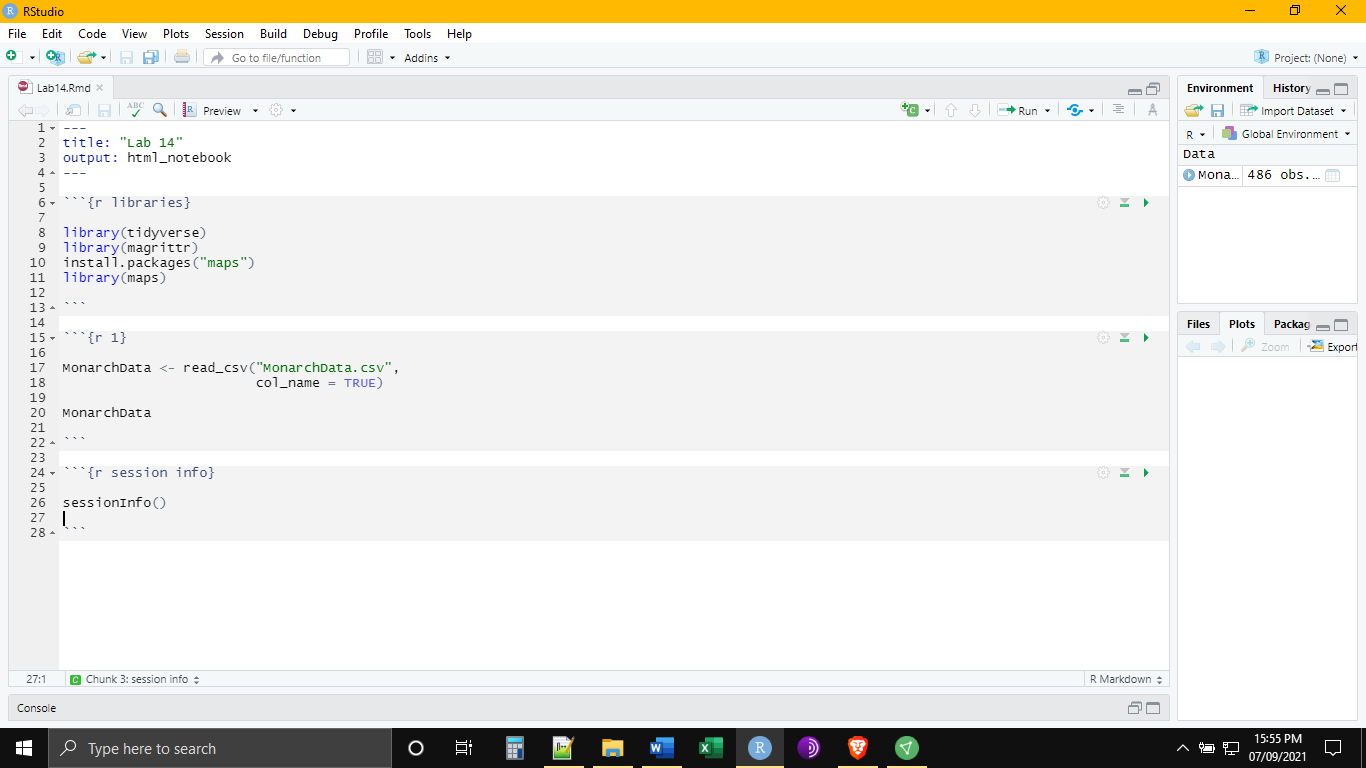
Programming, Scripting, and Reproducibility

*Introduction*

Today’s lab will cover some miscellaneous topics. These topics are intended to be a first step out of the beginner’s world of R and an introduction to some tools that can allow you to handle more complex projects. In the first case, we will learn how to write custom functions in R. Throughout this course we have used functions from simple ones such as c() all the way through more complicated ones such as pivot\_longer(). Yet, all of these functions were written by R developers, particularly those involved in developing base R and the tidyverse. Today we will learn how to write custom functions and programs in R to perform tasks that are not covered by preexisting functions. Following this we will talk in more detail about scripts. We discussed scripts in our introduction to R but then moved on quickly to R notebooks as our primary interface. Here we will talk more about scripts, how to write them and then how to use them effectively by calling them within other file types. This will lead us naturally into a discussion of reproducible workflows. Reproducibility, as we will see, is really about organizing your files and code in such a way that others can easily repeat your analysis.

*Set-Up and Data*

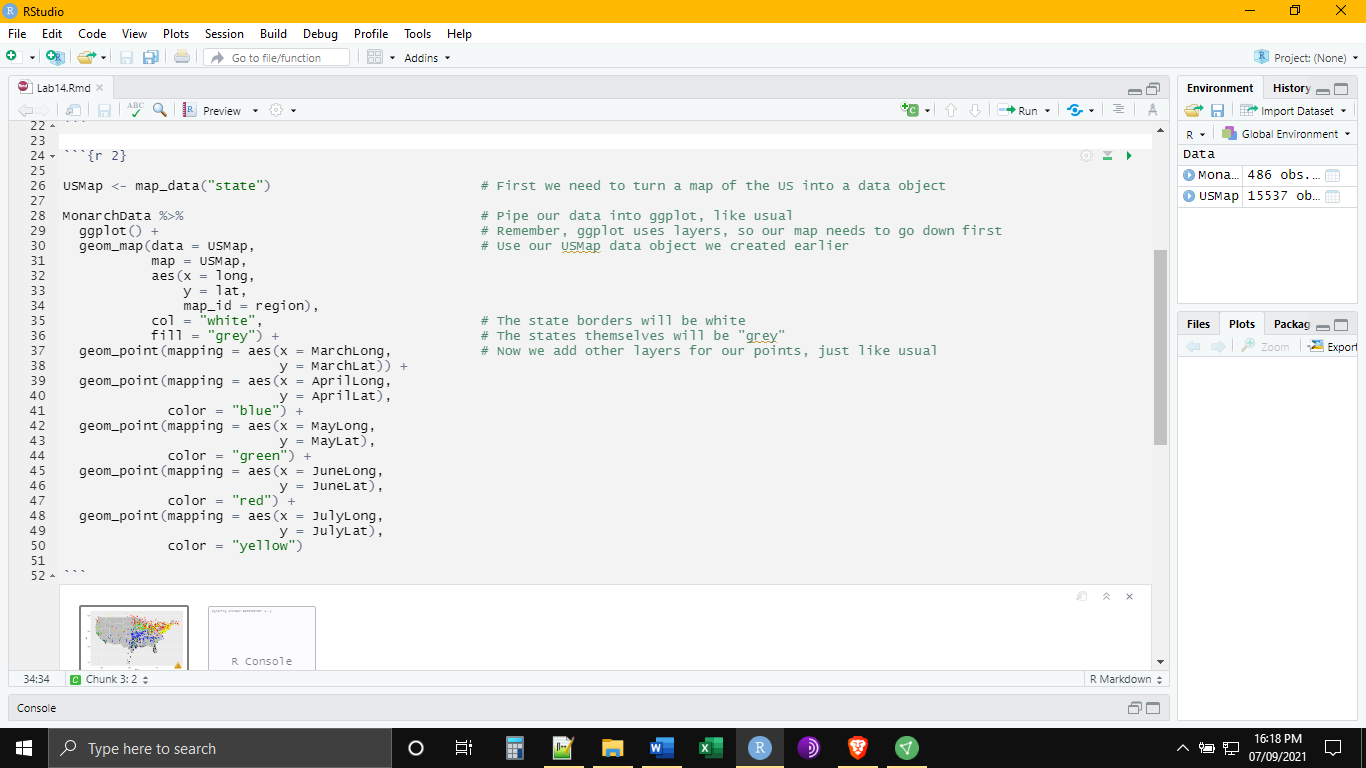
If you have not done so already, download all of the data files necessary for the lab this week and save them in the folder for today. The file that will be most important for today is the MonarchData.csv data. Start R Studio and open a new R Notebook file. Save this file in the folder where the data files are. Delete the text that comes in the notebook template, and make chunks to load libraries, import data, and give session information. Notice that we are installing a new package today “maps”. We will be plotting data onto a map today.

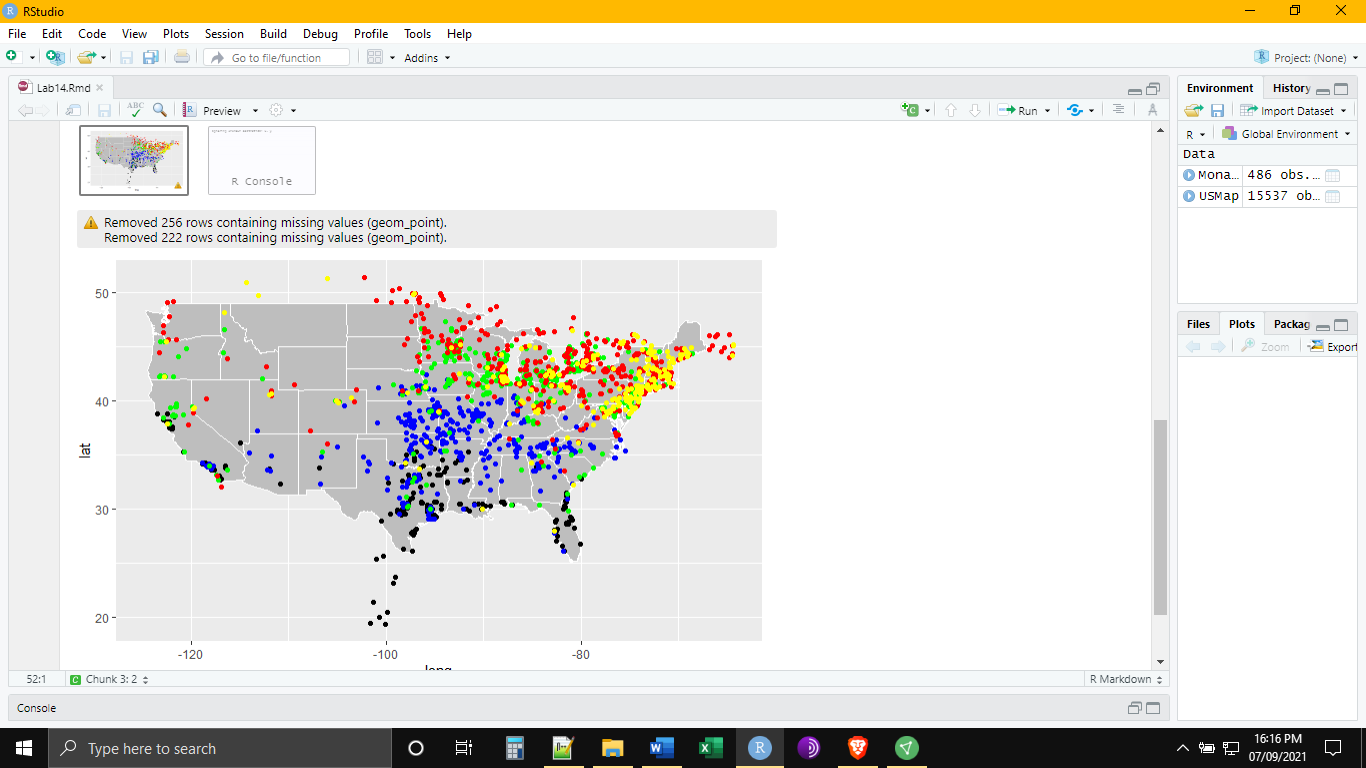


*Programming*

As the data file today suggests we will be working with data from the migration of monarch butterflies. Each year monarchs migrate from their wintering grounds in Central America to the Northern United States and Southern Canada. The data set that we have today are coordinates for the butterflies at different stages of their journey.

It might be helpful to visualize this migration We can do this through a scatter plot.





Let’s say we want to know is how far these butterflies are traveling each year. We can do this by measuring the distance between the most southernly and most northernly of these points. In a very simple way, we can do this using the Pythagorean theorem. Here, the travel distance is the straight light connecting the two points and completing the triangle produced by the respective differences in latitude and longitude between the two points. The Earth is elliptically shaped, so calculating the distance between two points is actually more complicated than this, but for the sake of keeping this example simple we will ignore that fact!

Determining the distance will require three steps: 1) finding the change in latitude and longitude between the points, converting that change into more familiar distance units, and then using the Pythagorean theorem to calculate the hypotenuse formed by those two distances. There are a number of ways we could do this in R, such as through a series of mutate() functions. Since our goal today is to learn how to program our own functions, we will take that approach. The basic syntax for defining a function is as follows:

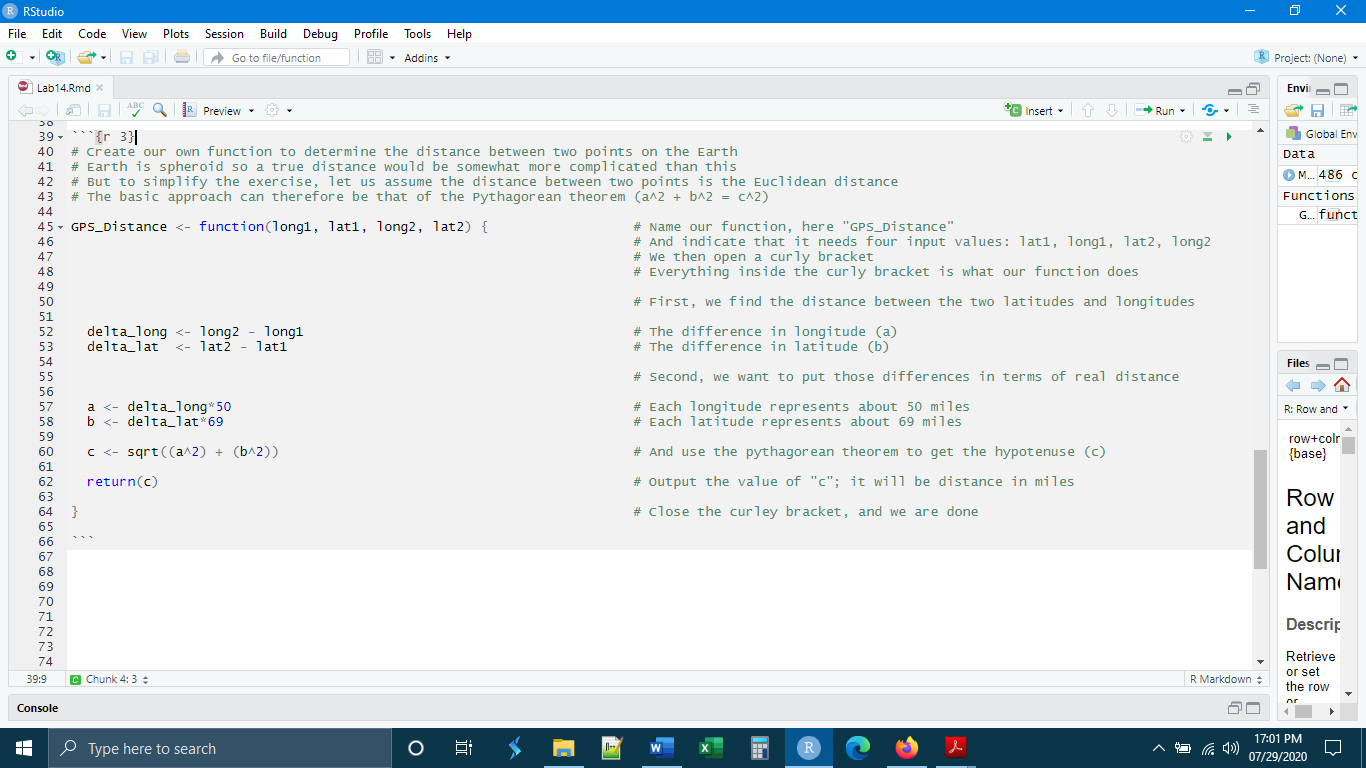
NameOfFunction <- function(Names, OfThe, DifferentInputs) {

Contents of the Function # i.e. what the function does to the input values

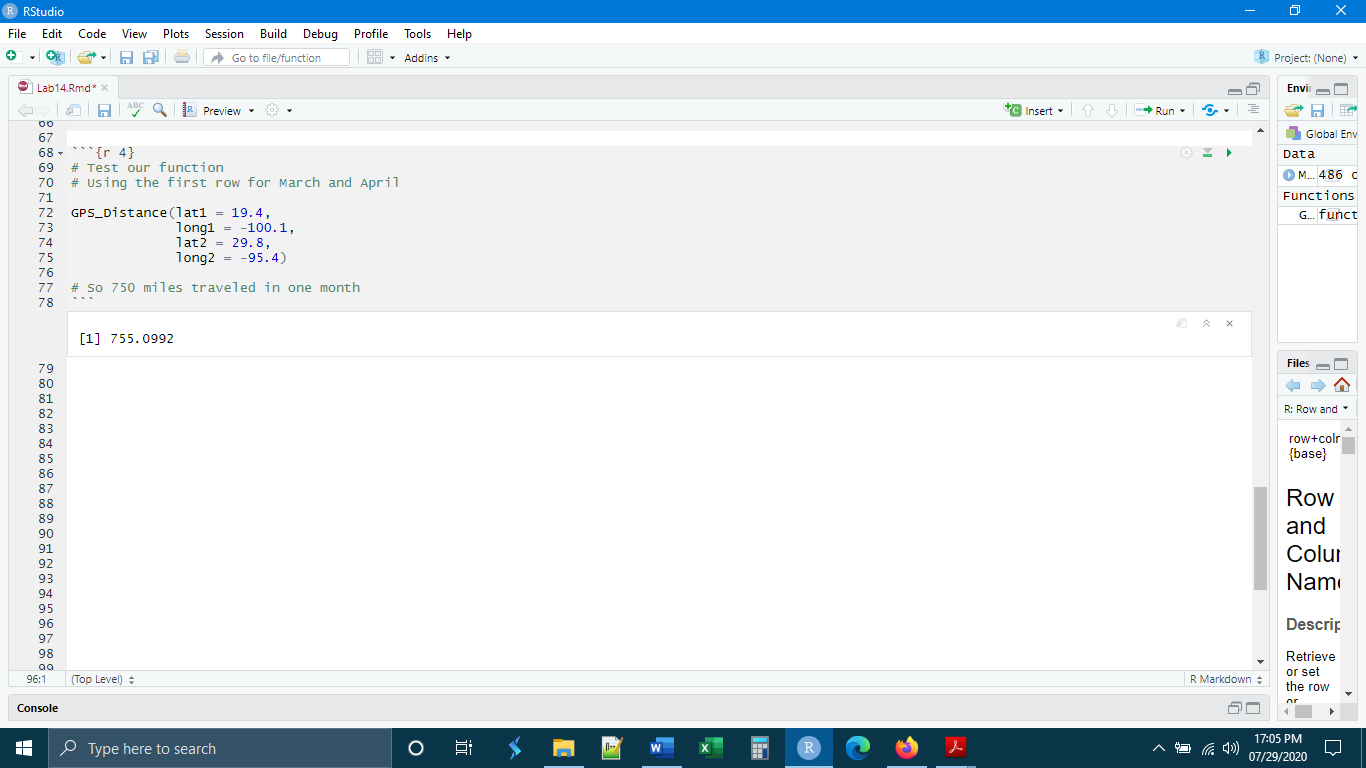
return(Variable, Names) # i.e. what values will be outputted by the function when it is finished

}

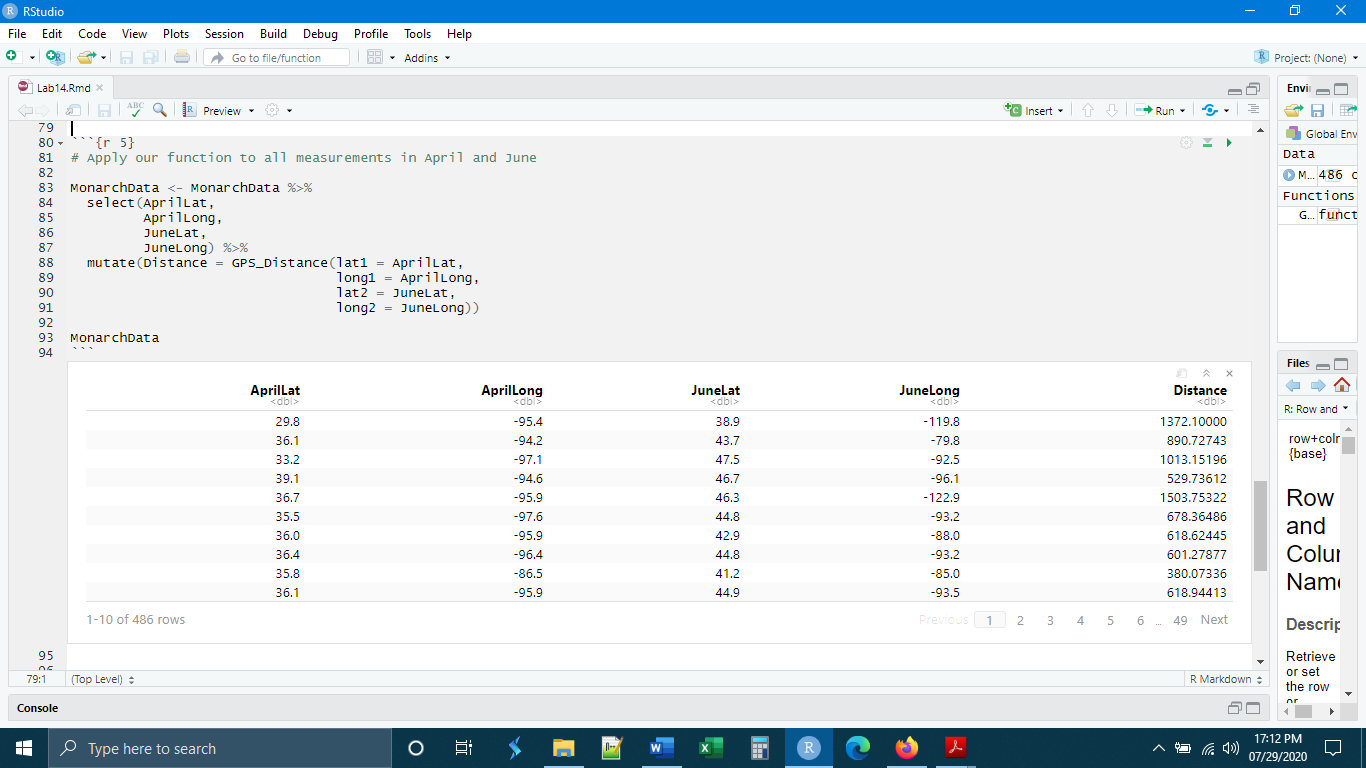
In some ways, this is something best learned by doing. Let’s then talk our way through writing out the function below.



Now that we have defined the function, let’s test it. To test it, let’s use the points in the first row from March and April, an arbitrary selection. We use our function in the same way that we would use any other function. We type out the name of the function and then supply information for each of the arguments (inputs) that are required, in our case four.



Initially, it seems that this is a lot of work, to write a whole new function simply to do calculations one at a time. The utility of defining a new function becomes apparent when we go on to apply it to a large number of operations. We can do that in our case by combining mutate() with our function GPS\_Distance(). Instead of supplying individual values for each of the four arguments, for each argument we now supply an entire vector and R will perform the function for each element of the vector.



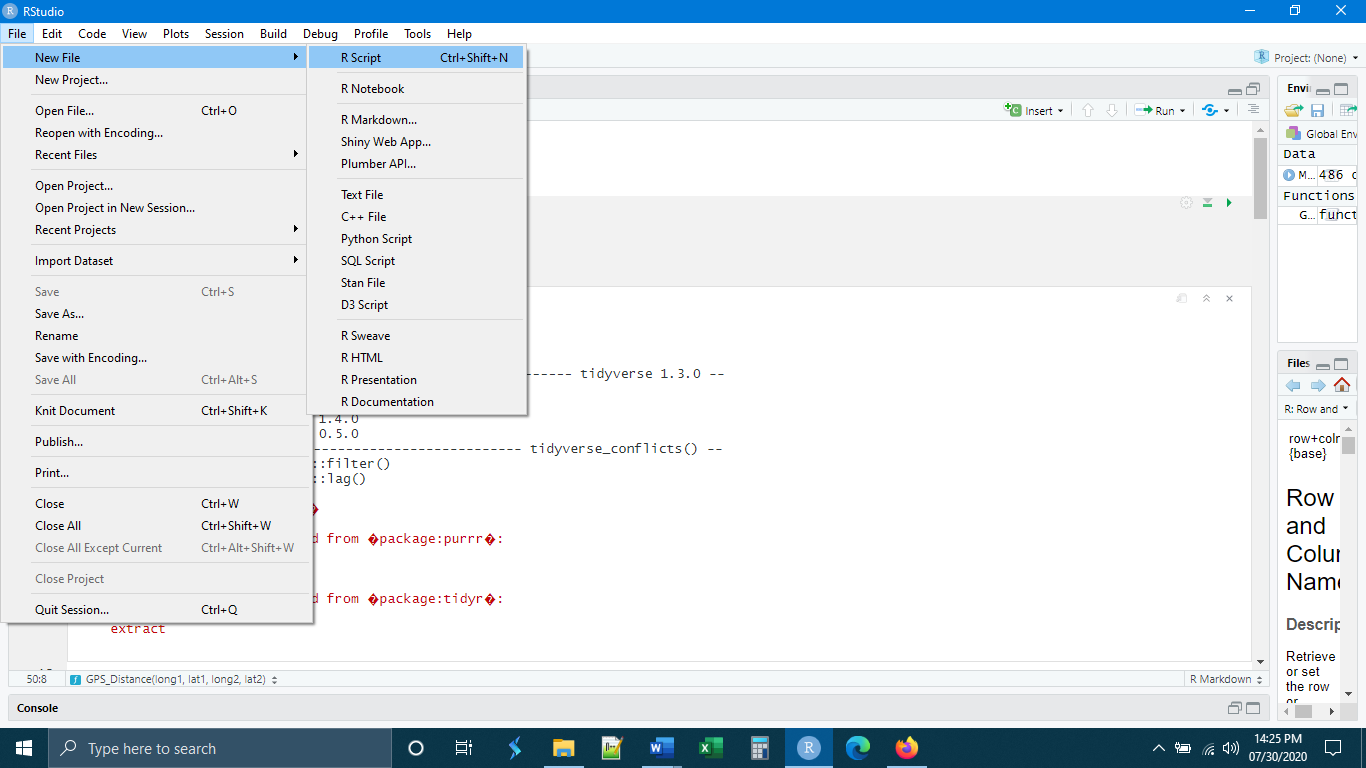
A final word on custom functions. Each time you restart R you will need to redefine the function explicitly before you use it. R will not remember your custom functions from one session to the next.

*Scripting*

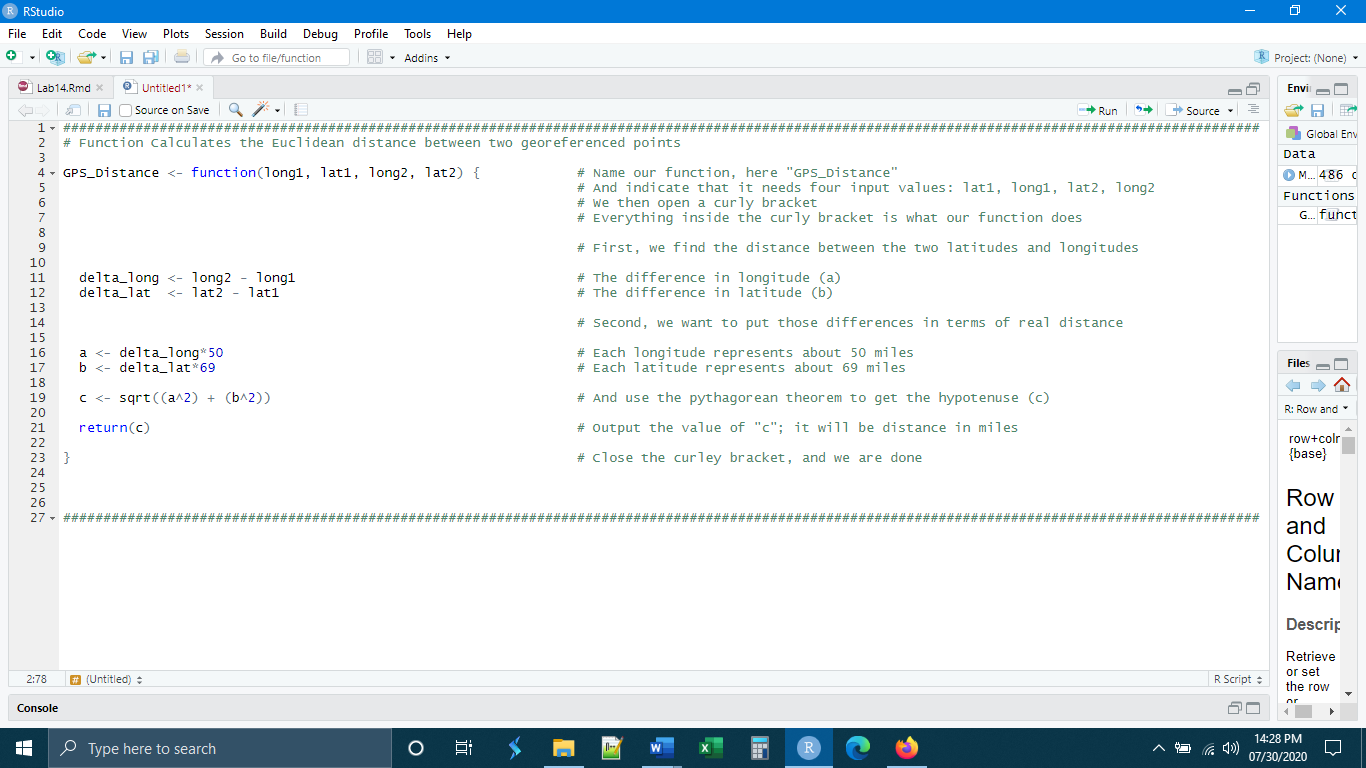
That leads us into the next section of this lab, scripting. We touched on scripts at the very beginning of this course but then moved away from them in favor of the notebook. As a quick recap, in the R Studio interface, the script appears simply as a free text editor where R code can be executed. The utility of the script, however, is that it can be used as a way to execute large amounts of frequently used, pre-written code.

For example, let’s say that I often do projects where I am dealing with georeferenced data and over time I have created a number of custom functions to help me analyze these data. It may be a good idea to put the definitions of all of these functions into a single script and then execute this script at the beginning of my R notebook, right after loading the libraries, that way I have all the functions that I will need, defined and ready to be used. Let’s practice this using the function that we defined.

First, while keeping our R notebook open, let’s open a new script.



And then copy and paste the function definition from the notebook into the script. You can see I have cleaned-up the result and added a bit of annotation. If there were more than one function being defined here, it would be a good idea to separate them by comment symbols, just to keep everything organized.

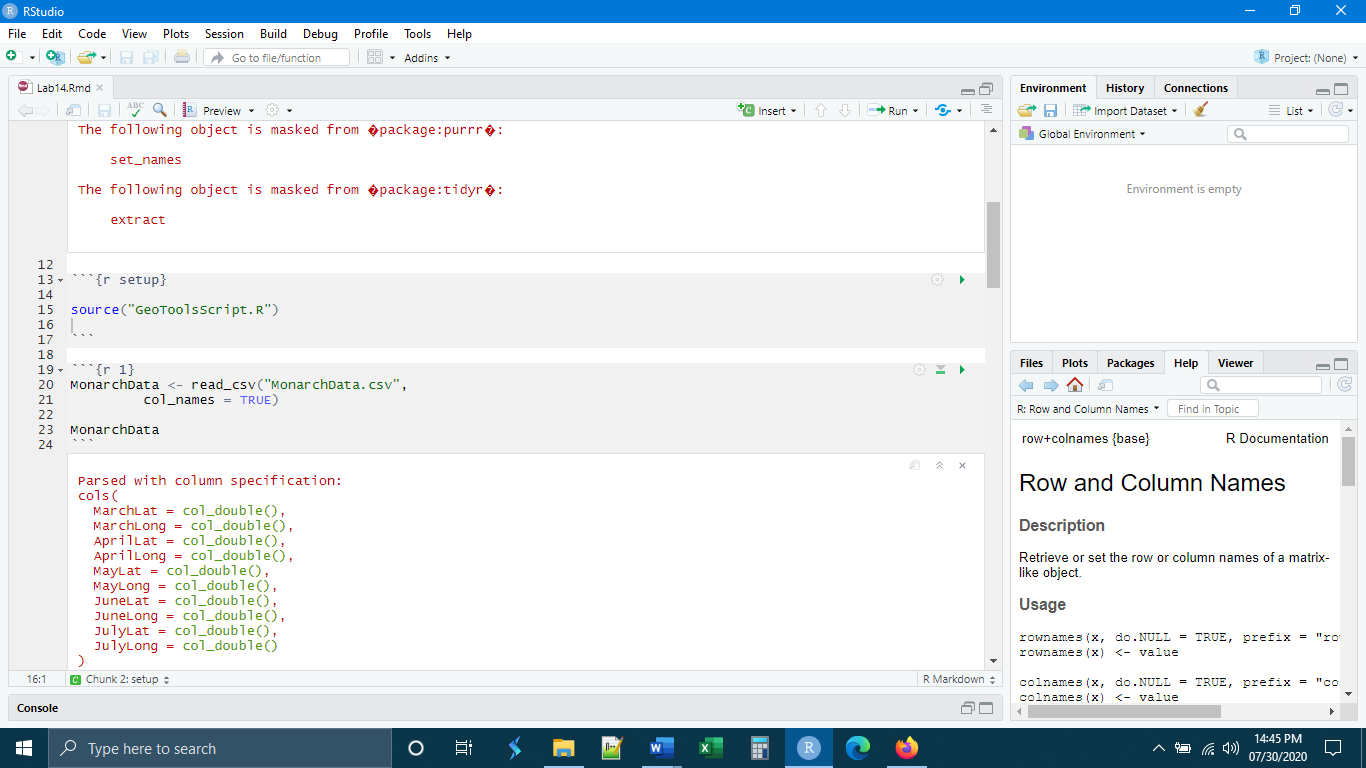


The next step is to save this script. As is the case with other files like our .csv data files, it is best to have a script stored either in the working directory itself, or within a subdirectory of the working directory (as we will talk about in the next section on reproducibility). For now, let us put it in the working directory (the same folder as the notebook file we are actively using).

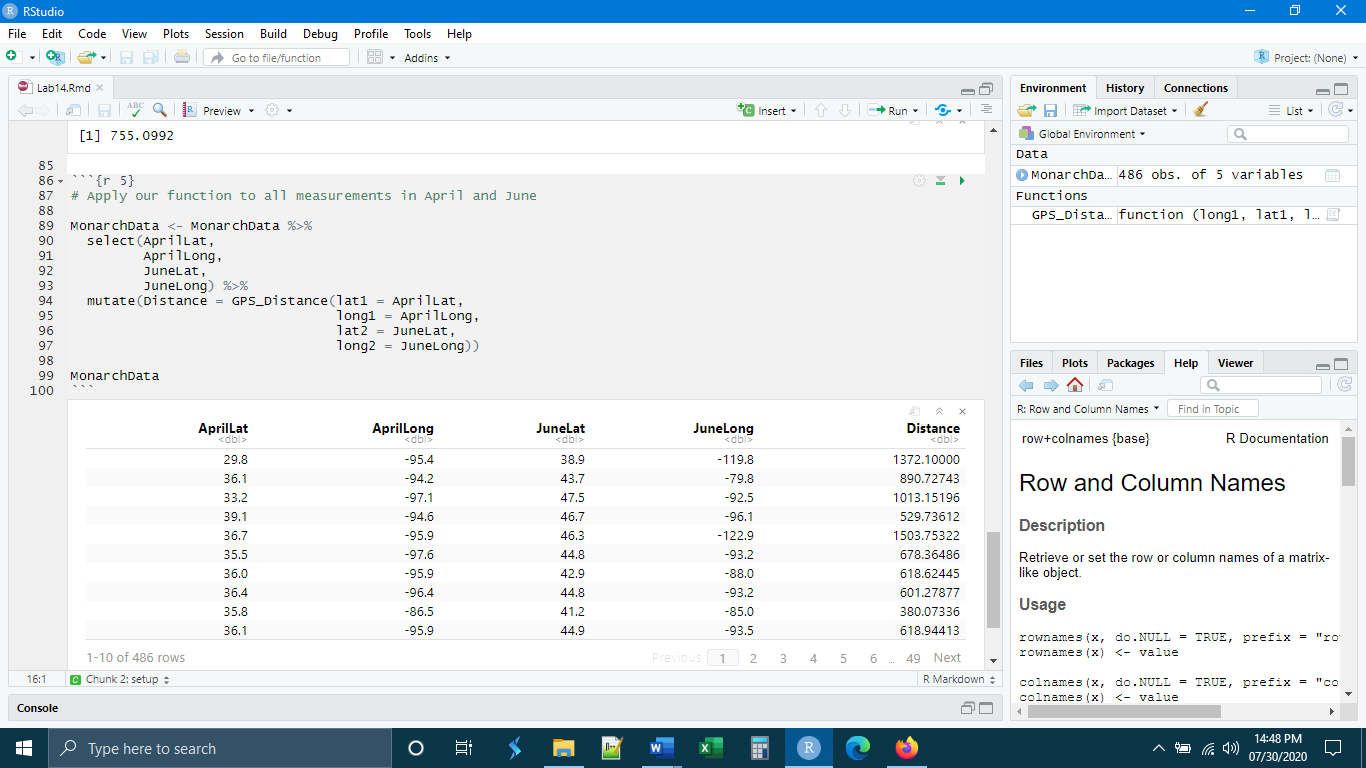
Once the script is saved, close the script file and return to the notebook file. Go to the top of the notebook and below the libraries chunk, make a new chunk called “setup”. We will use this chunk to call and run the script. The basic syntax for calling a script is:

Source(“NameOfScript.R”)

After this has been completed, clear your environment with the broom button. Clearing the environment makes R Studio forget everything we have done in this session, including the definition of our custom function. We want to make sure it is the script defining our function.



Now, let’s test to see that the script works. We want to run the libraries chunk, the setup chunk, chunk 1 (to load the data), and chunk 5 to apply the function to the data. If everything has been done correctly, chunk 5 should run with no problems. The script has defined the function in the setup chunk so R knows what to do when the function is asked for in chunk 5.



*Reproducibility*

A word that is often used in the scientific community is “reproducibility”. We want to conduct and document our work, especially our experimental work, in such a way that somebody else could do our work over again exactly as we did (to the extent that that is possible). When we talk about reproducibility, we often focus on the design of experiments and the documentation of their execution through the taking of notes as the experiment or study progresses. We will talk about some of these aspects of reproducibility in lecture.

Yet, there is another aspect of reproducibility, and this is reproducibility in the analysis workflow. The way that statistical analyses are conducted can influence the results of the study as much, or sometimes more, than variations in the experimental design and execution. In an ordinary project some things to consider are, did I process my raw data in any way before analyzing it? Did I exclude portions of the data? How did I decide what was excluded? Did I transform any of my data? How and why did I transform the data?

If I am working on a project that involves modeling, an additional set of concerns arises. For example, did I control my random number generation through explicit seeds? Did I document and retain a copy of my input data and parameters, especially if they are from an open source repository that may be updated periodically and therefore may be different now than when I first did my study?

If we choose to use R as the platform for our data analysis, there is a way to organize our work in such a way that our analysis can be repeated by anyone with just a few clicks. It is this that we will focus on in this last section.

I would like to begin by giving the general layout of the files. For R, a reproducible workflow will have all of the files necessary for the analysis contained within the project directory.

ProjectNotebook.Rmd

PlotOutput4.png

PlotOutput3.png

PlotOutput2.png

PlotOutput1.png

Script2.R

Script1.R

RawData3.csv

RawData2.csv

RawData1.csv

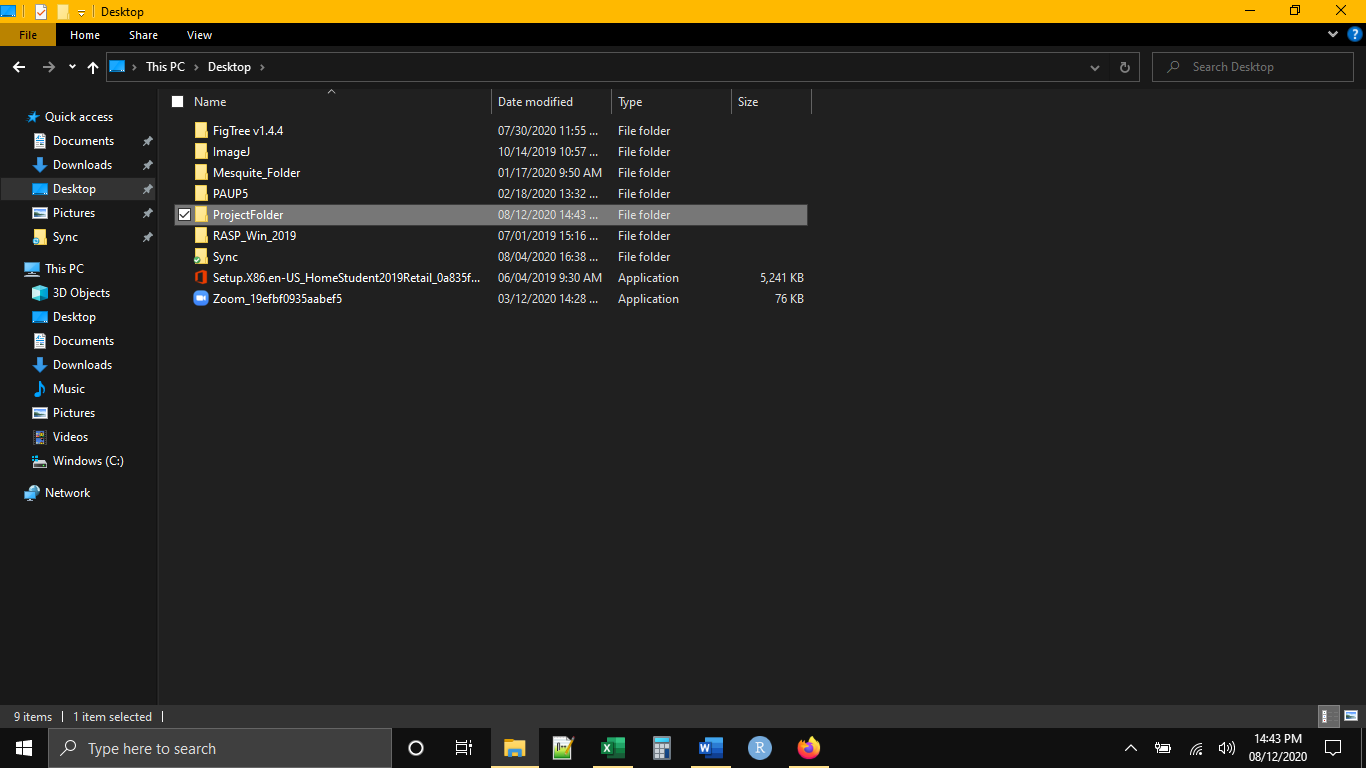
**Project Directory**

**Plots Subdirectory**

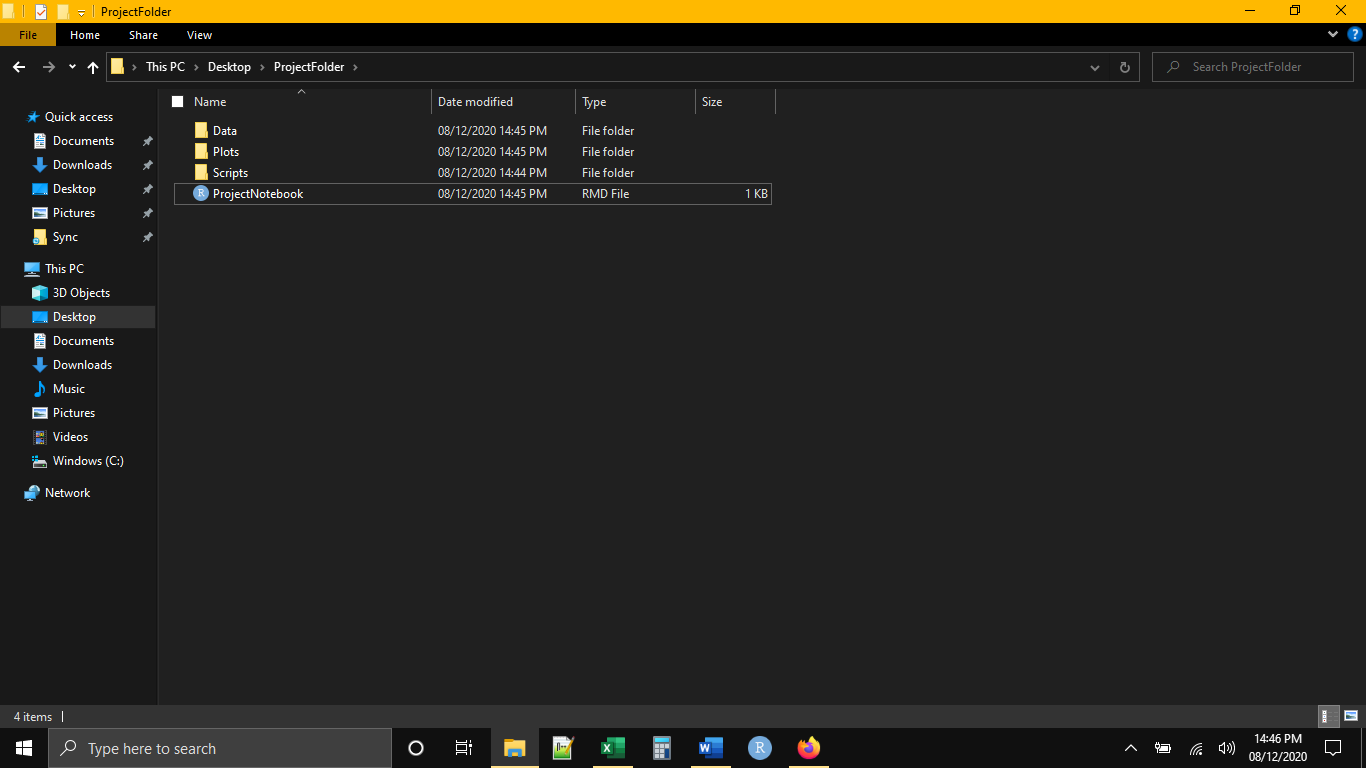
**Data Subdirectory**

**Scripts Subdirectory**

Note that the plot files will not be initially present but will only become present once the code is run. Yet, there needs to be a place to send the plots from the code. Just to emphasize what this layout looks like:



And within the project folder we will have this setup. The data files in .csv format will be in the Data folder and any scripts that we will be running will be in the Script folder in .R format.



Placing files inside of subdirectories of our working directory will change the way that we write our code. Up to this point we have been dumping all of our files directly in the working directory. This meant that we simply had to list file names and extensions, both when reading (importing) and writing (exporting) files from the notebook. Now that we have introduced subdirectories, however, we need to use file “paths” to let R know where our files are and will be stored. The “path” is simply the location of a particular file on your computer. Fundamentally, there are two kinds of paths: absolute paths and relative paths. An absolute path can be thought of as the “complete” location of a particular file, from the lowest possible drive or directory. As an example, consider the following path:

C:/Users/HP/Desktop/Work/Coursework/Undergrad/Stats/ProjectDirectory/Data/FileName.csv

Here we start in the C Drive of the local computer, and then we have a list of the folders or directories that we need to pass through in order to get to the file in question (FileName.csv). The advantage of using an absolute path is that there is no ambiguity about the file that we are referencing. The major disadvantage of an absolute path is that any code written using absolute paths cannot be shared and used by other people without first changing the paths. The organization of files and folders will undoubtably be different on another person’s computer. In fact, absolute paths will “break” if you move your R notebook from one folder to another or change the name of any of the directories in the path. As a result of these disadvantages, reproducible workflows do not use absolute paths.

Instead, a reproducible workflow will use relative paths. A relative path gives the location of a file in reference to some other directory, in our case, the working directory. In fact, we have already seen that R Studio is set-up to deal with relative paths. So far, we have simply listed file names when reading and writing files and R Studio assumed those files were in the working directory. Thus, any relative paths that we give will be relative to the working directory. Essentially, this means indicating the subdirectory. In practice, this will look like the very end of an absolute path:

Data/FileName.csv

Scripts/FileName.R

Plots/FileName.png

If we want to see what this looks like in terms of code, our data reading (import) code would become (differences from what we have been doing are highlighted):

DataObject <- read\_csv("Data/FileName.csv",

col\_names = TRUE)

or calling a script:

source("Scripts/FileName.R")

or writing an image file, which does not have the path indicated with the file name but in an argument call “path” where you would indicate the relative path:

ggsave(filename = "FileName.png",

plot = last\_plot(),

device = "png",

path = "Plots",

width = 6,

height = 4,

units = "in",

dpi = 500)

If all of the code is written using relative paths and the files are organized into clearly labeled subdirectories, it is very easy to share our analysis. All we would need to do is make a copy of the project directory and send that project directory to our collaborator. The collaborator would simply need to open the project directory and run the notebook file and all of the analyses would be accomplished in exactly the same way as on our own computer.

*Final Thoughts*

Three final thoughts. First, scripts will come to play an important role in your research. Writing and saving scripts to define programs or accomplish common tasks helps reduce error in your code because, once written, the text of the script code does not ever need to be edited. This makes it less likely that you will make mistakes when sharing your code with others.

Second, always make the last bit of your code a call for session information. If your analysis produces a different result on someone else’s computer, you can compare the session information between the two analyses to make sure that both of you are running the same version of R and the same versions of all of the packages used in the analysis.

Finally, in all of this, documentation is key. As with reproducibility in the experimental design and data collection, the more we document of the analysis workflow through writing out protocols ahead of time and taking notes along the way (including comments within our code), the easier we make it for others (and ourselves) to repeat what we did.