**Biostatistics**

**Lab 11**

*Tasks*

Data Manipulation and Cleaning Interlude, tidyr

*Introduction*

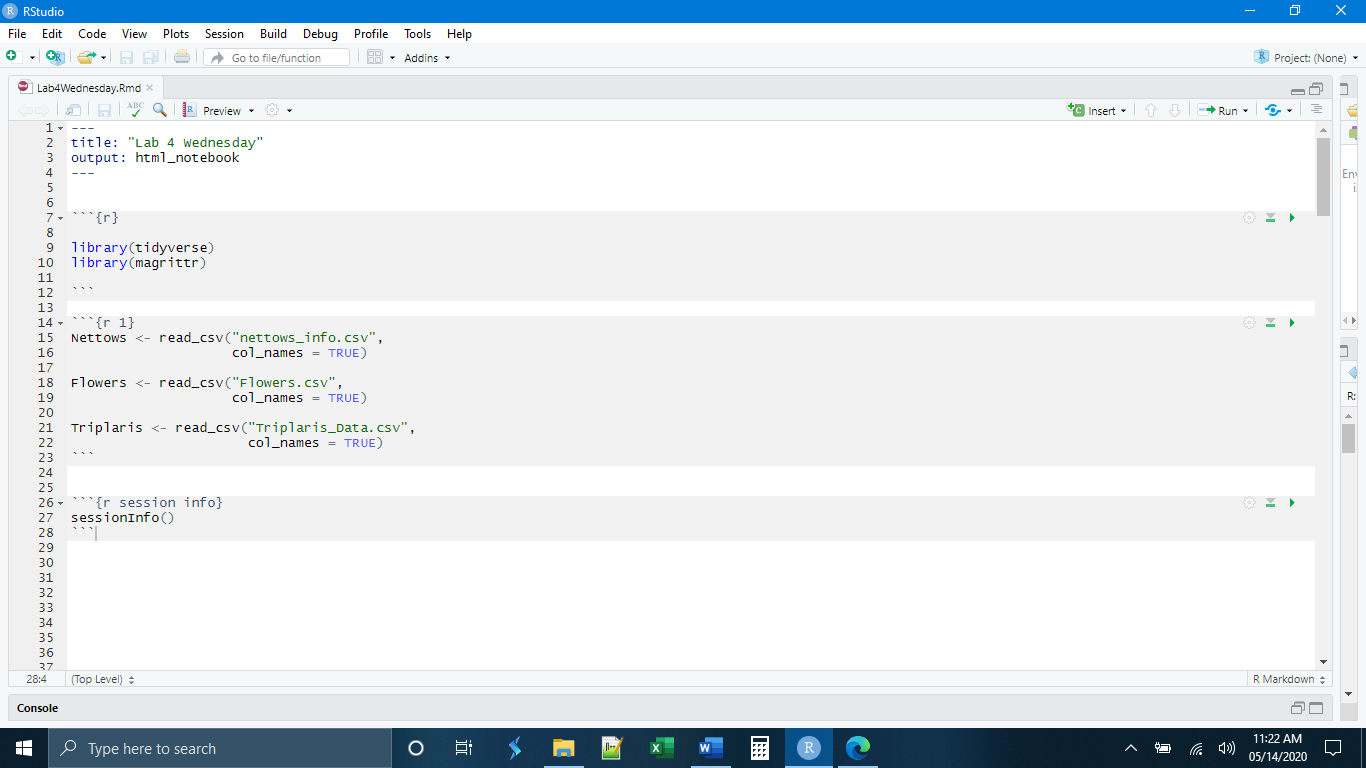
The first labs we did in R were all about data importation and manipulation. During our time together, I have tried to emphasize that this will be one of the things you will spend the majority of your time on in data analysis projects. This is especially true if you are working on a project with other group members and they are the ones who collected and recorded most of the data, but it is your job to analyze those data. Moreover, as we have experienced in our use of the t-test() and aov() functions, the analysis of your data may require it to be in a very specific form.

Today we are going to cover a few functions that can help with major data rearrangements. If you find yourself having a hard time conceptualizing and understanding how the data are being changed, be sure to read the R for Data Science textbook online (chapter 12). The chapter has some very helpful figures and illustrations. You may also consider taking a look at the tidyr cheat sheet. Unfortunately, the names of the functions we are covering today have recently changed and you will want to look for the functions gather() and spread() instead of pivot\_longer() and pivot\_wider().

*Set-Up and Data*

If you have not done so already, download from Blackboard all of the data files necessary for the lab this week and save them in the Lab4 folder you made Monday. There are three files that will be important for today nettows\_info.csv, Flowers.csv, and Triplaris\_Data.csv.

Start R Studio and open a new R Notebook file. Save this file in the Lab4 folder where the data files are. Delete the text that comes in the notebook template, and use the first chunk to load libraries. It is good practice to also present your session info, make a last chunk to do this. Make a chunk 1 to read in the data and save them as new data objects.



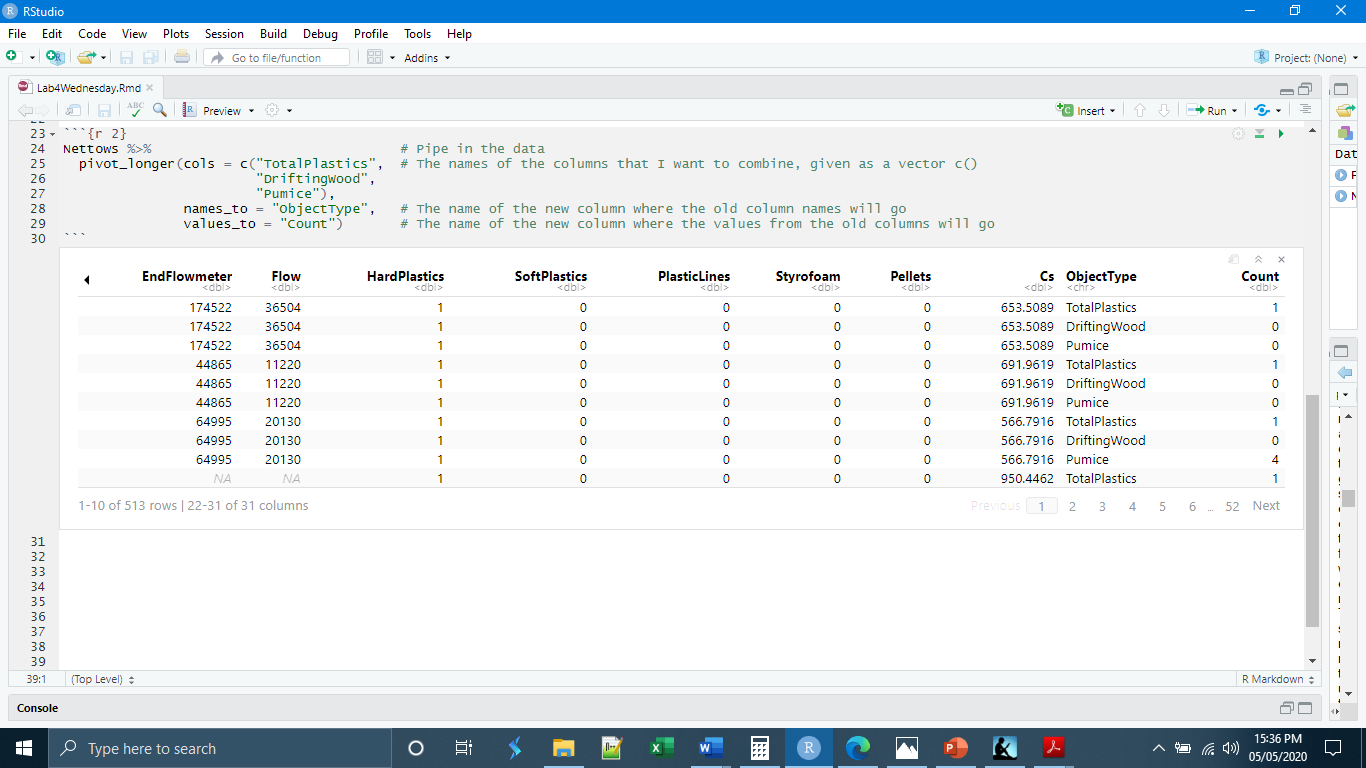
*pivot\_longer()*

Sometimes data have been stored in such a way that information we would like to be in a single column have been recorded in a number of columns. We can see that in our Nettows data. If we go to the last group of variables, we see that plastics, driftwood, and pumice collections have been placed in separate columns. But in some ways, these are all really measures of a single variable, objects caught in our net. The pivot\_longer() (called “gather()” in the tidyr cheat sheet) provides a way to put the data from these three columns into two columns, and reorganize the rest of the data frame in the process. The basic syntax of pivot\_longer() is

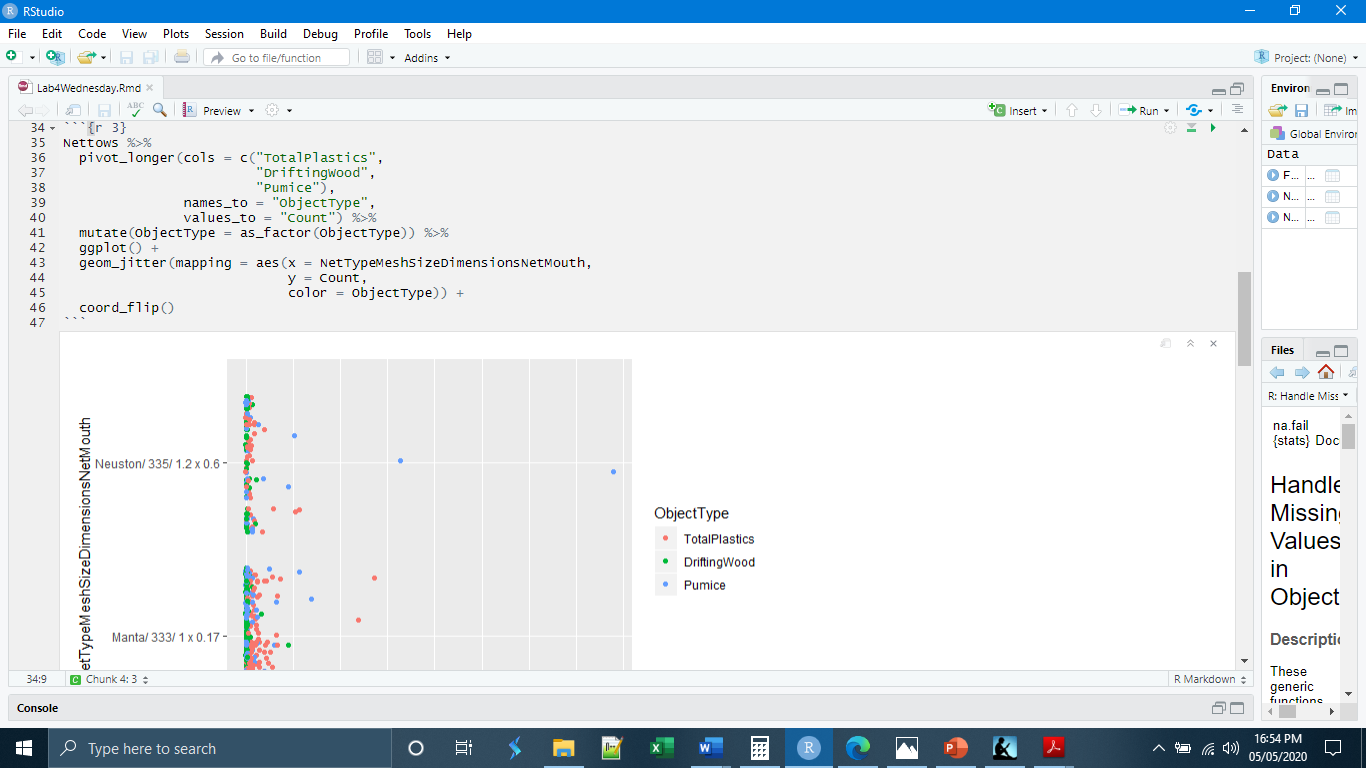
pivot\_longer(cols = c(“names”, “ofthecolumns”, “iwanttocombine”),

names\_to = “NewColumnWhereNamesGo”,

values\_to = “NewColumnWhereObservationsGo”)



To underline the utility of rearranging the data this way, we are now able to make a jitter plot of all of the items collected in the nets and color those items by their type (plastic, wood, or pumice). This is possible because that information is in two columns of our data frame. Similarly, it would be much easier to feed these data into an ANOVA or t-Test.



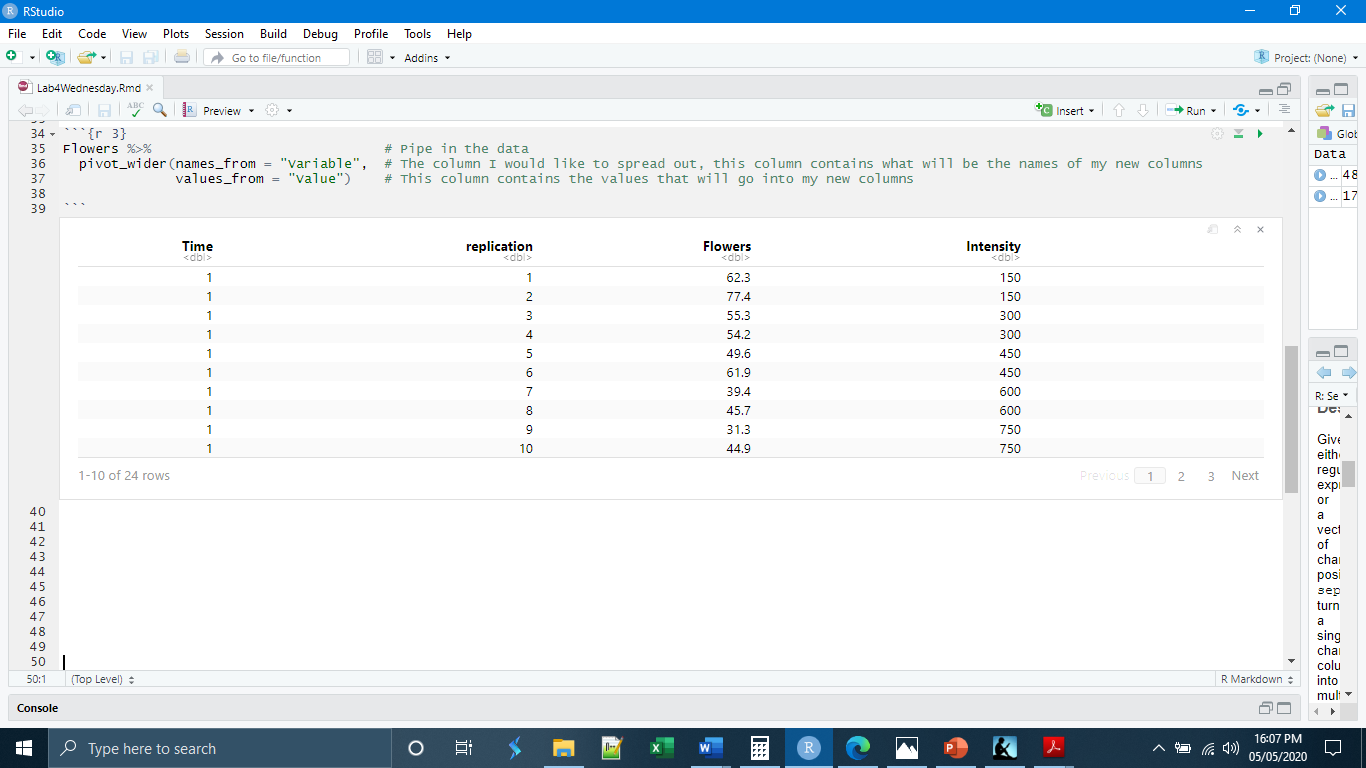
*pivot\_wider()*

In some cases, the opposite of pivot\_longer() is necessary. Here, the data have been stored such that different kinds of information are found in the same column. If we look at the flowers data, we see that the Variable column has information on two unrelated things, Flowers and Intensity. We can move these observations into their own columns and reorganize the data frame as necessary with the pivot\_wider() command (called “spread()” in the tidyr cheatsheet). The basic syntax for pivot\_wider() is:

pivot\_wider(names\_from = “ColumnIWantToSplitUp”,

values\_from = “ColumnThatHasValuesForMyNewColumns”)

R will automatically generate as many new columns as there are groupings in the column you designated in names\_from =



*separate()*

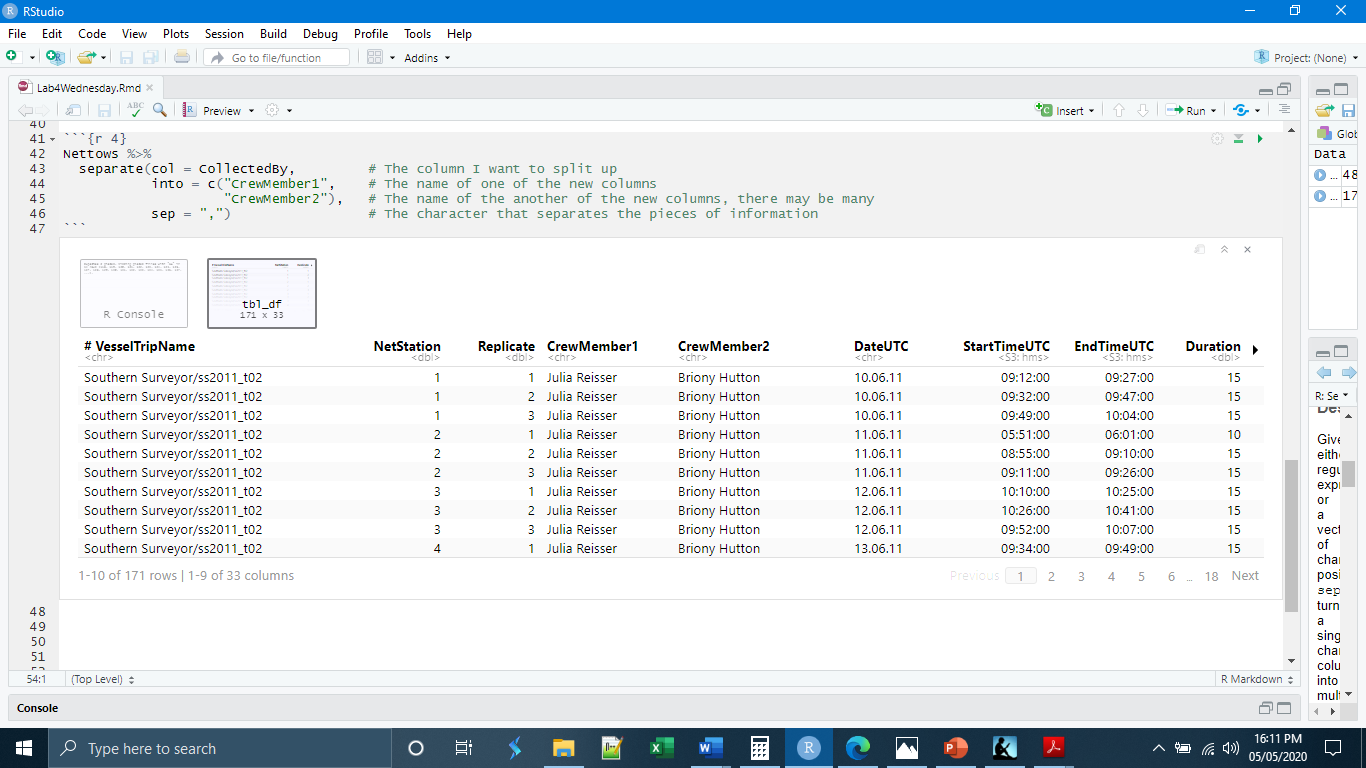
Sometimes a single column contains multiple pieces of information separated by commas, dashes, slashes, etc. and we want to put each of those pieces of information into its own column. If we look at our Nettows data, we see that the column CollectedBy could be an instance of this. Here we have the crew members listed and it may be helpful to have them in their own columns. This is the job of the separate() command. The basic syntax of this command is

Separate(col = “NameOfColumnWithMultipleInformation”

Into = c(“vectorof”, “namesofthe”, “newcolumnsIwant”),

Sep = “thesymboltheinformationisseparatedby”

Note that you need to provide separate() with the exact number of new columns you will be getting out of the command. There are other functions in stringr that can generate an unspecified number of new columns, but we will not be covering them in this course.

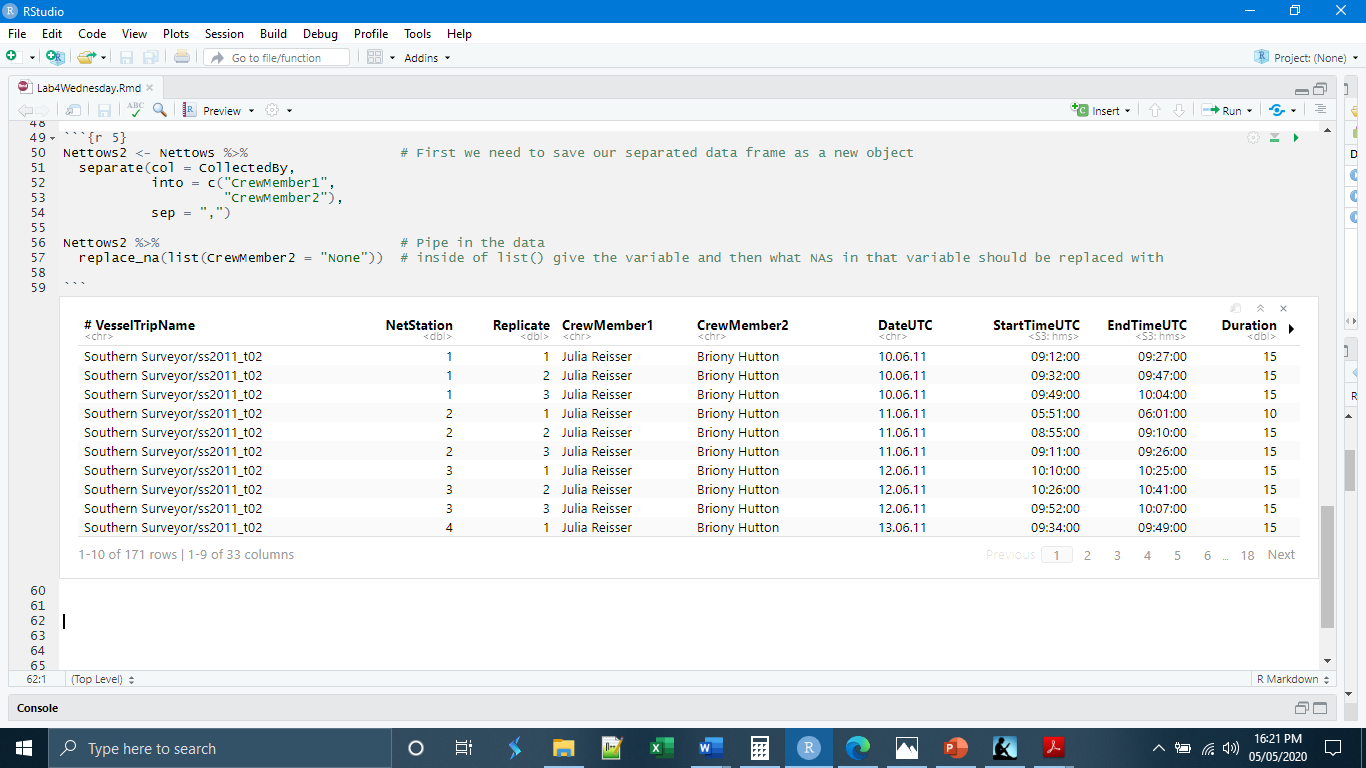


*replace\_na()*

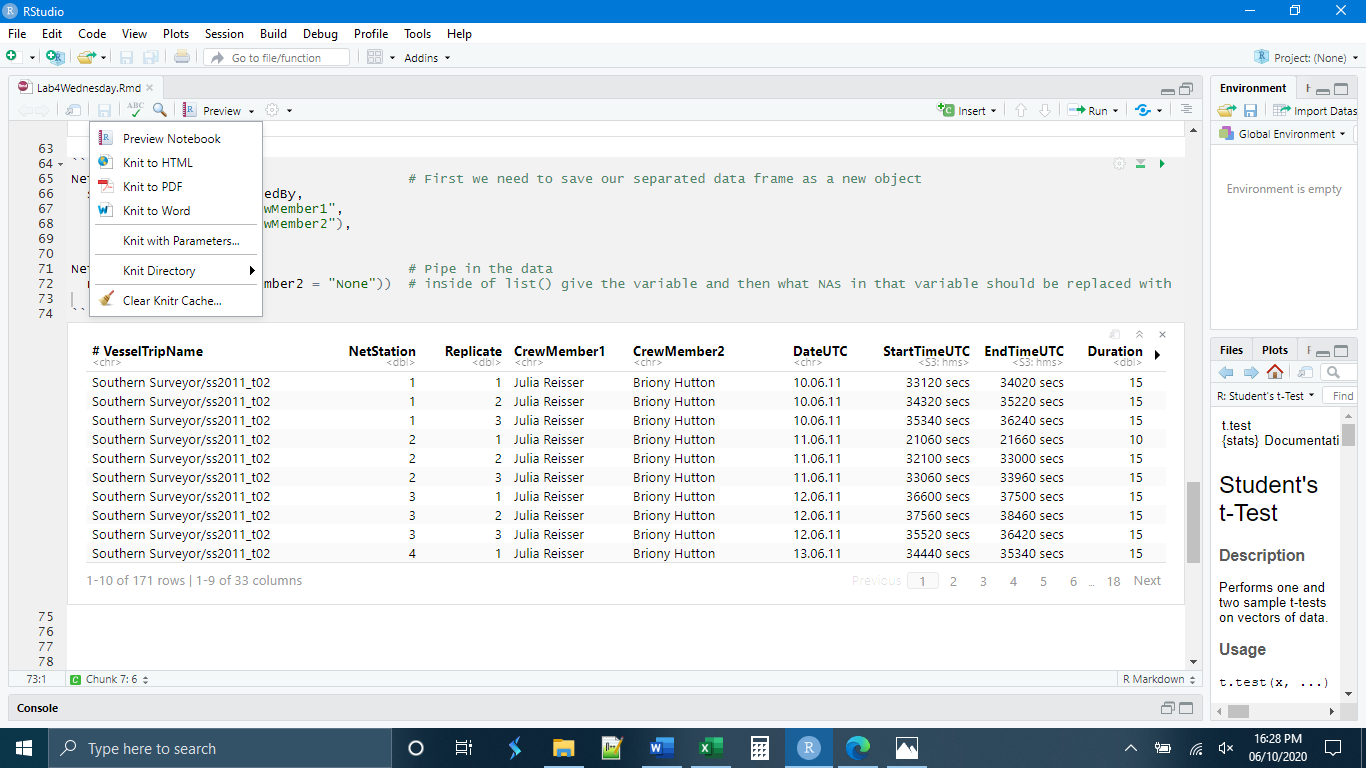
The last function we will be discussing is the replace\_na(). Missing data is inevitable in any project but the degree to which it is tolerable in the final analysis depends on the kind of project and the analysis methods being used. One way to deal with missing data is to use the na.omit() function, which removes any observation (row) that has a missing data point for any variable (column). Alternately, you can use is.na() and filter(). If you are missing data sporadically throughout the data set, however, these methods can result in the loss of a lot of data. One way to help with this problem is not to omit the data but to replace it with some other value that can be used in the analysis yet still expresses uncertainty. This is where replace\_na() comes in.

Notice that the separation we did in the previous section produced a number of NAs, missing data in the CrewMember2 column. These are cases where there was only one person on the ship collecting data. We can use replace\_na() to replace these NAs with something reasonable, like “None”. The basic syntax for replace\_na() is

Replace\_na(list(“ColumnNameToReplaceNAsIn” = “WhatValueToGiveNAs”))



The final thing we will do today is learn how to “knit” our document. Knitting is the term used for a special printing of R Notebooks and R Markdown documents. We have said that the advantages of using an R Notebook are that the output of code written in chunks appears directly below the chunk, and that we can use the space between the chunks as a primitive text editor. What knitting does is print all of those aspects together: the text, the code in the chunks, and the results of the code. You can knit to a pdf, a Word document, or to an html document. The knitted document will be saved to the working directory.



*Independent Exercises*

**1.** Make a new chunk called “chunk 5” and write code to

- Read in the Triplaris\_Data.csv data (from Blackboard) and save it as a new data object

- Use pivot\_longer() to move the columns titled PetioleLength1, PetioleLength2, and PetioleLength3 into columns titled “Replicate” and “Length”.

**2.** Make a new chunk called “chunk 6” and, using the Triplaris\_Data.csv write code to

- Use replace\_na() to replace all of the NAs in the “Habit” column with “Not Recorded”