**Biostatistics**

**Lab 4**

*Tasks*

dplyr, Filtering, Grouping, Summarizing, Arranging, Selecting

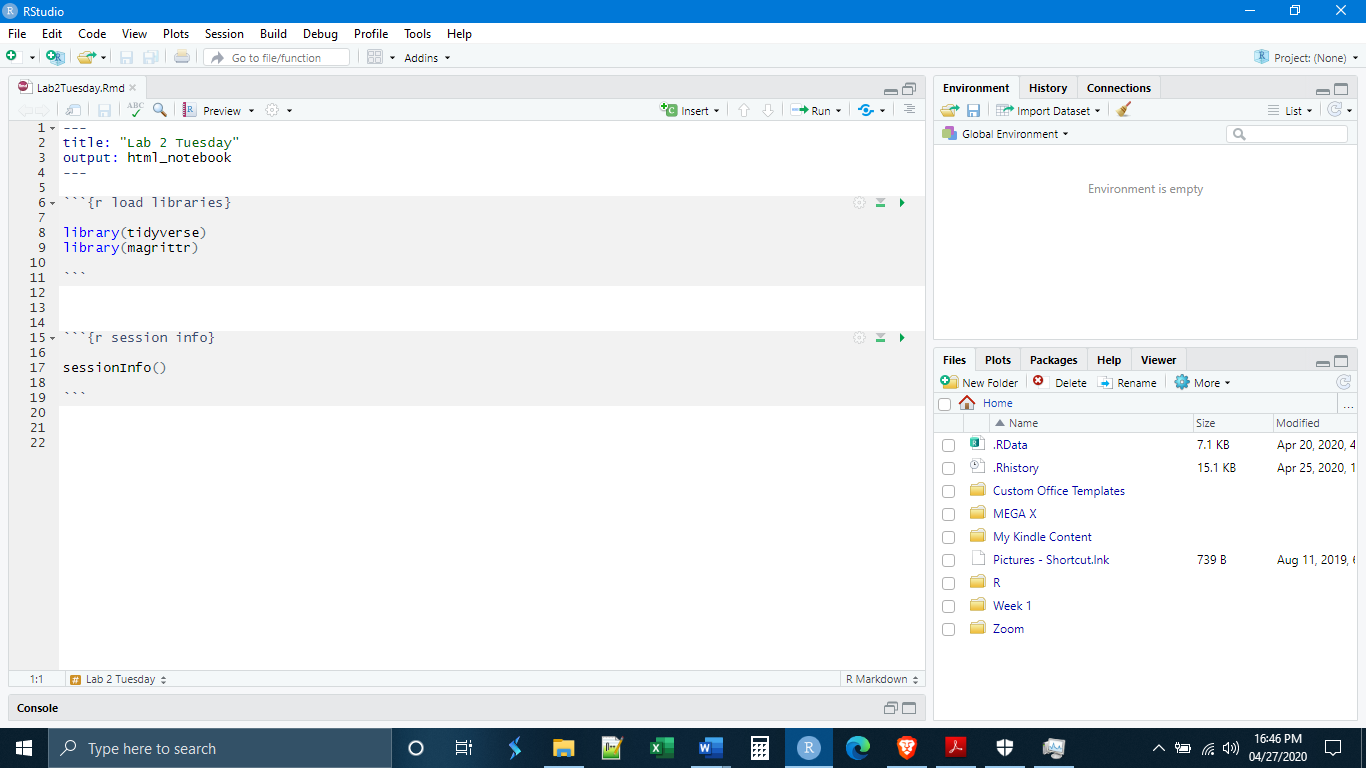
*Introduction*

Today we continue our adventures in dplyr and its data manipulation tools. You may be asking yourself, when are we going to get to the statistics? We will get to statistics. Yet, it is important to realize that, when it comes to statistical programming, most of your time and work is not spent in actually calculating the statistics. Computers do this work very quickly and often with only a few lines of code. Much of your coding work will be spent in importing and tidying your data and putting it in the proper form such that it can be analyzed. Doing these exercises in dplyr has the added advantage of familiarizing you with reading and writing R code, skills which will be necessary both in the remainder of this course, and if you use R for your own projects.

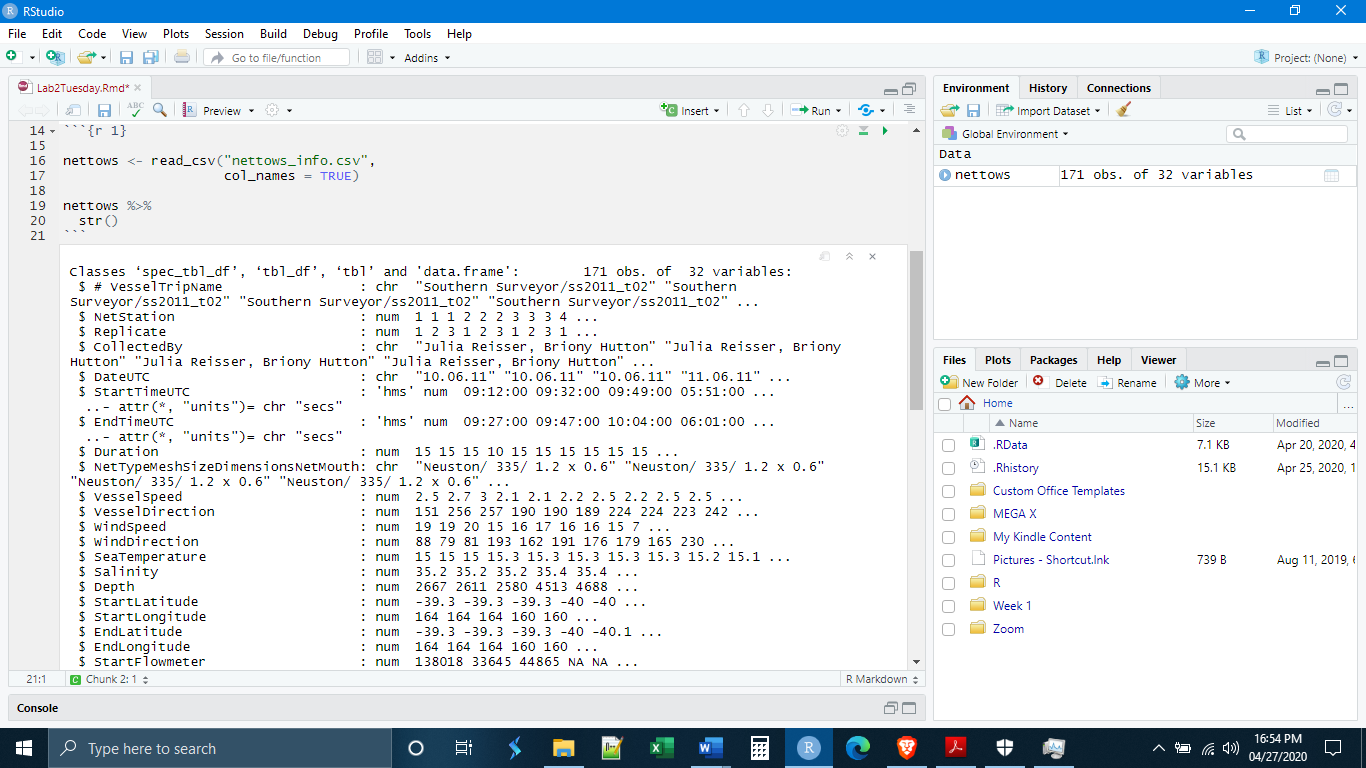
*Filtering and Selecting*

Begin by creating a new folder “Lab2” which we will use for all our files for this week. Second, download all of the data files necessary for the lab this week and save them in this same folder. The file that will be most important for today is the nettows\_info.csv.

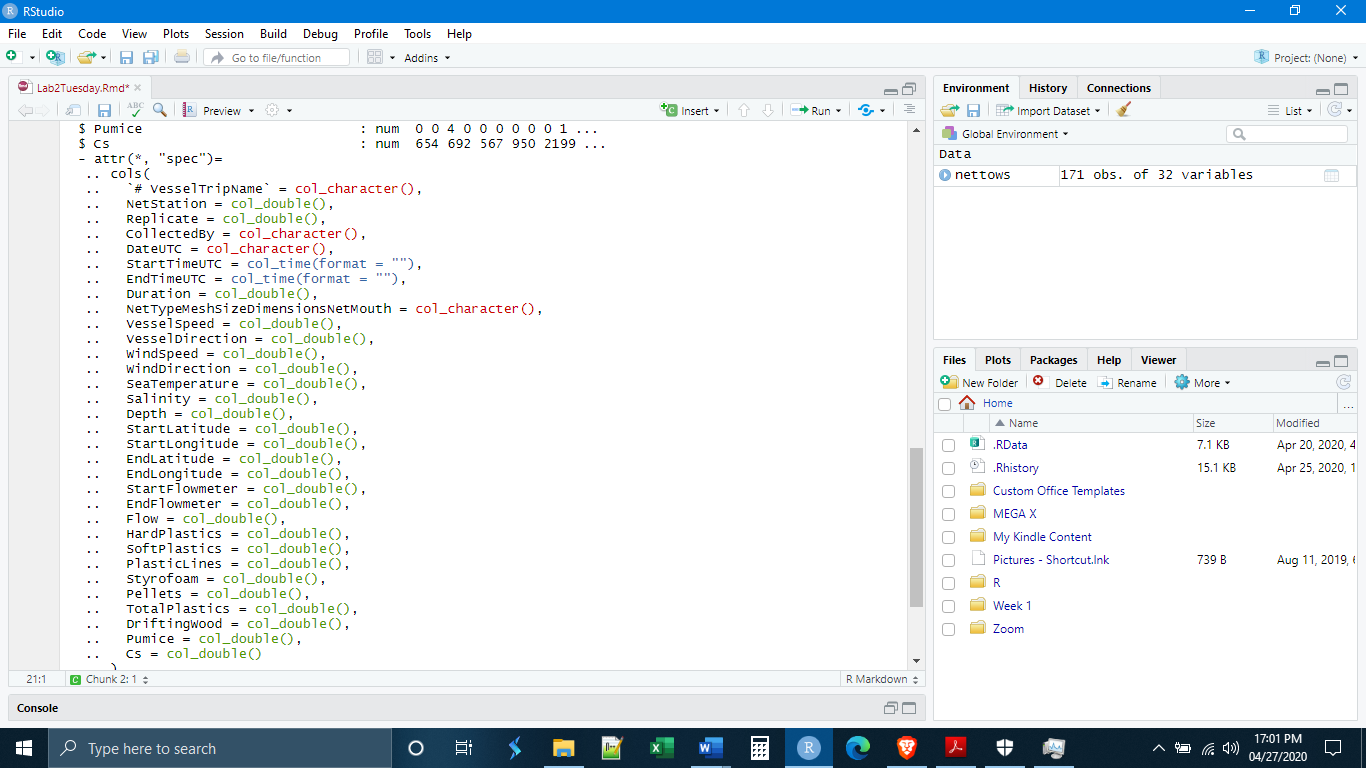
Start R Studio and open a new R Notebook file. Save this file in the “Lab2” folder where the data files are. Delete the text that comes in the notebook template, and use the first chunk to load libraries. It is good practice to also present your session info, make a last chunk to do this.



We now need to read in our data and have a look at it to see what we are dealing with. This will take two lines of code.

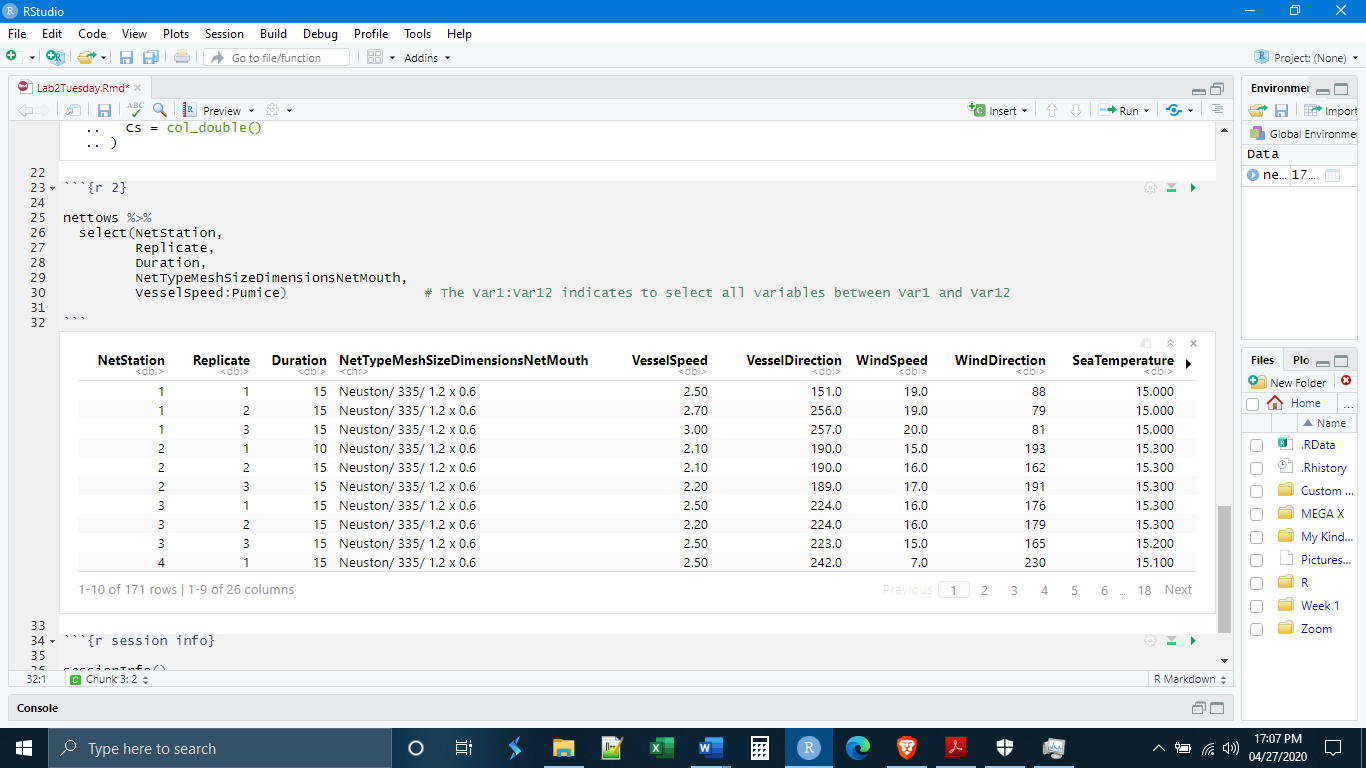


These data are from a pollution study off the Australian coast. Nets were dragged behind boats and plastics were collected. We can see that there are 171 observations for 32 variables and that there are some data types we have not seen before and have not introduced. For example, some of the variables are formatted as times, which R can interpret and do operations on in a tidyverse package called lubridate.

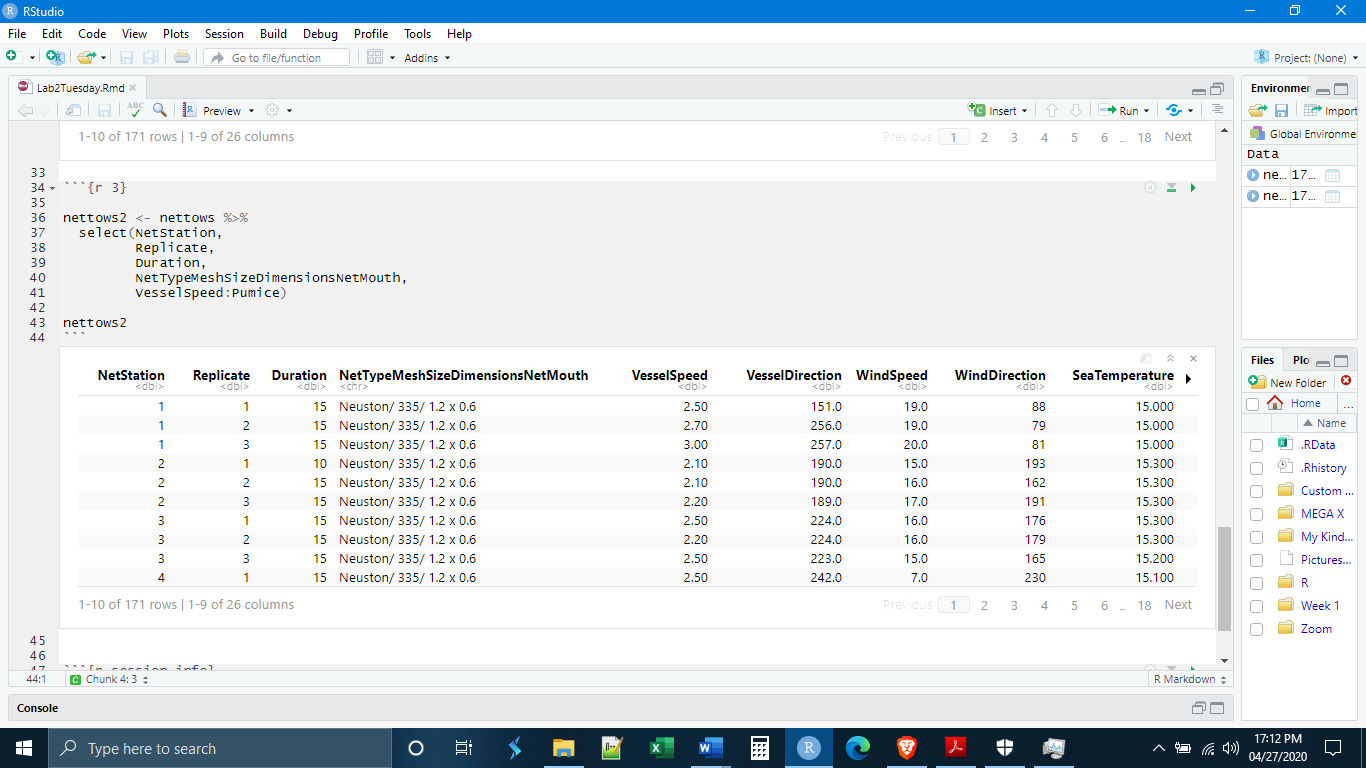


There are more variables here than are interesting to us, and our first task will be to use the select() function to take only those vectors (columns) that we want. Selected variables will be given in the order they are listed in select(). The syntax is simple, it is:

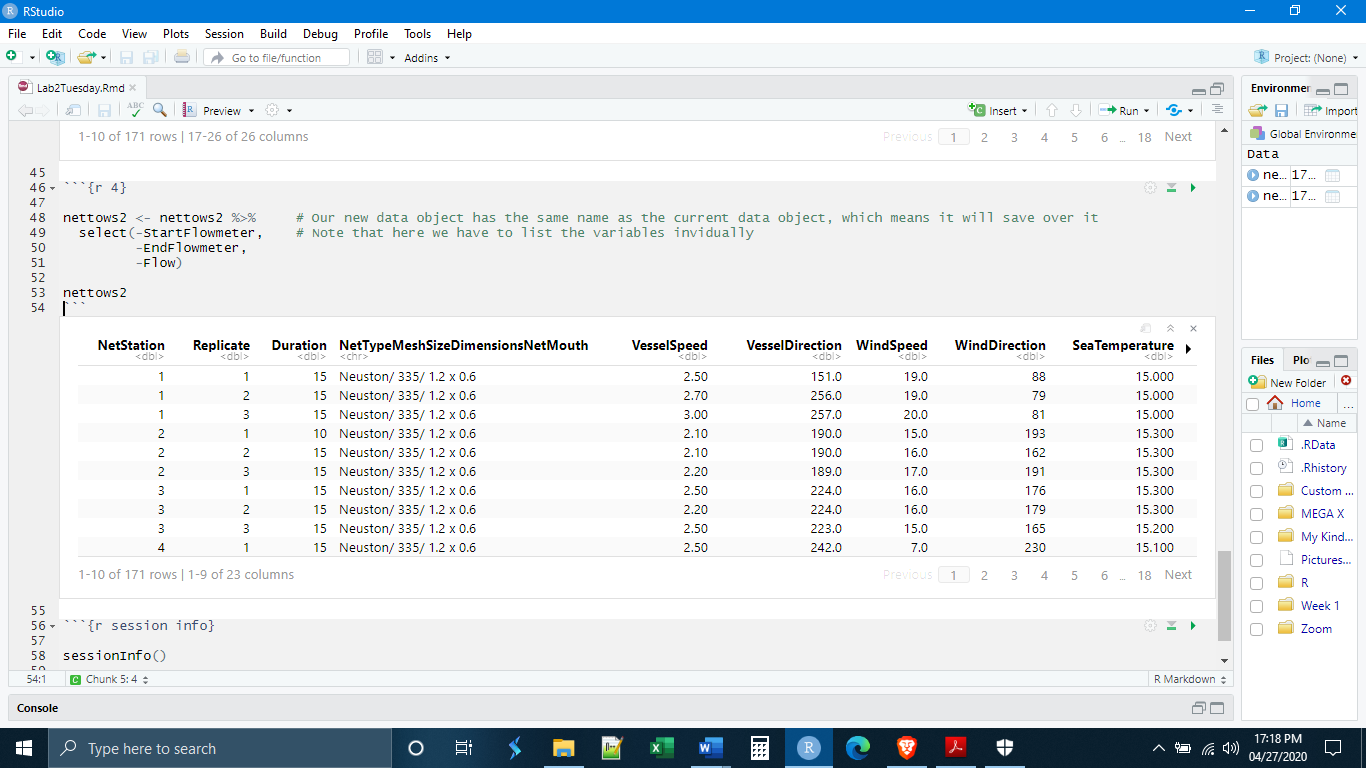
select(VariableName1, VariableName2, VariableName3)



Remember, the imported data set has not been changed because we have not created a new data object. All of the variables that we left behind still exist with all of their data in the nettows object. Let’s write the code to save what we have done as a new object.



Let’s say that we made a mistake, and some of the variables in nettows2 we actually did not want to select, for example, the variables about the flowmeter. An easy way to remove them, without having to list all the variables before and after them, is to use the negative select. By putting a minus (-) sign in front of the variable names, select() will keep everything except for the variables listed. And we will create a new data object to keep the changes.



Up to this point we have used the select() function to remove vectors (columns), but what if we wanted to remove some of the observations (rows)? This is accomplished by the filter() function. The filter() function allows us to select rows based on a combination of logical and/or mathematical expressions. You can find these on the middle of the first page of the dplyr cheat sheet. The most important are:

== is equal to (for both numbers and text)

< is less than

> is greater than

<= is less than or equal to

>= is greater than or equal to

is.na(VariableName) is missing (finds all missing observations in a variable)

!is.na(VariableName) is not missing (finds all not-missing observations in a variable)

! is not

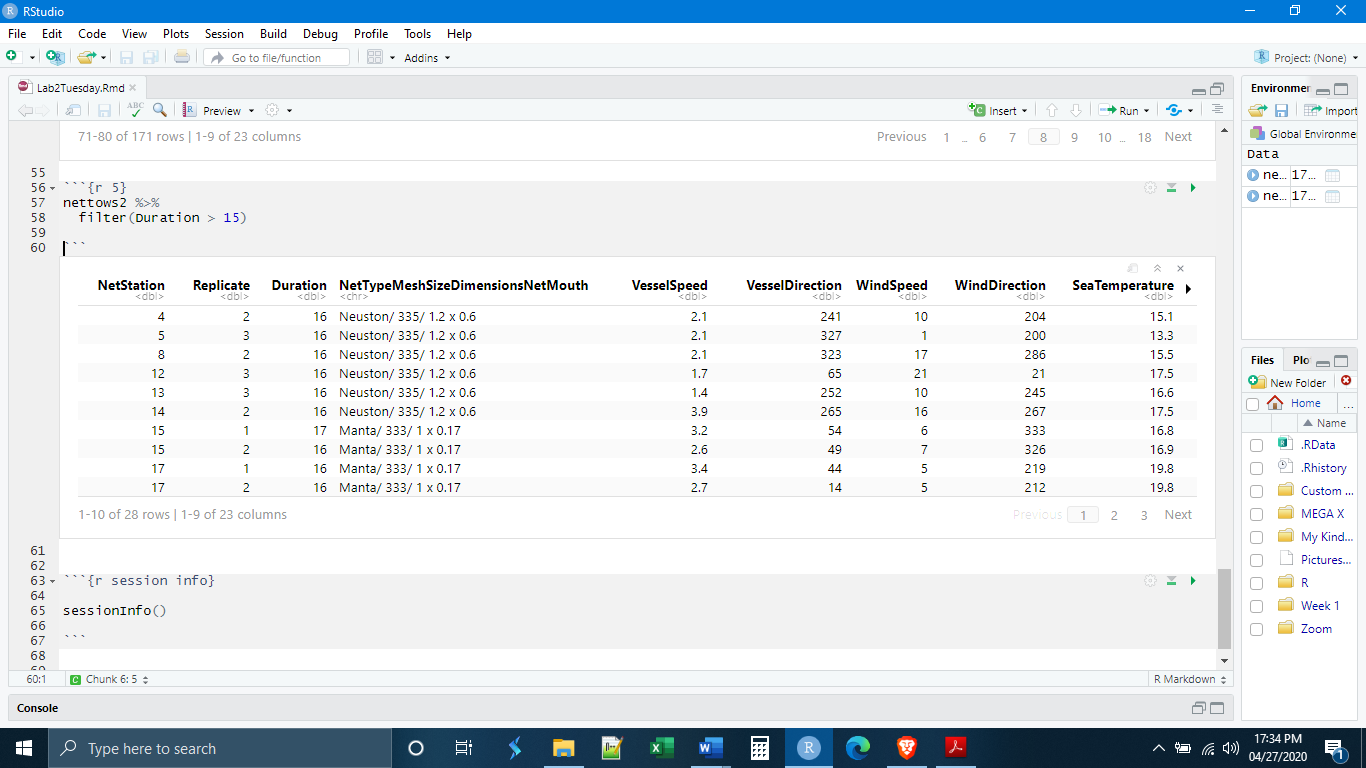
| or (this symbol is just below the backspace key)

& and

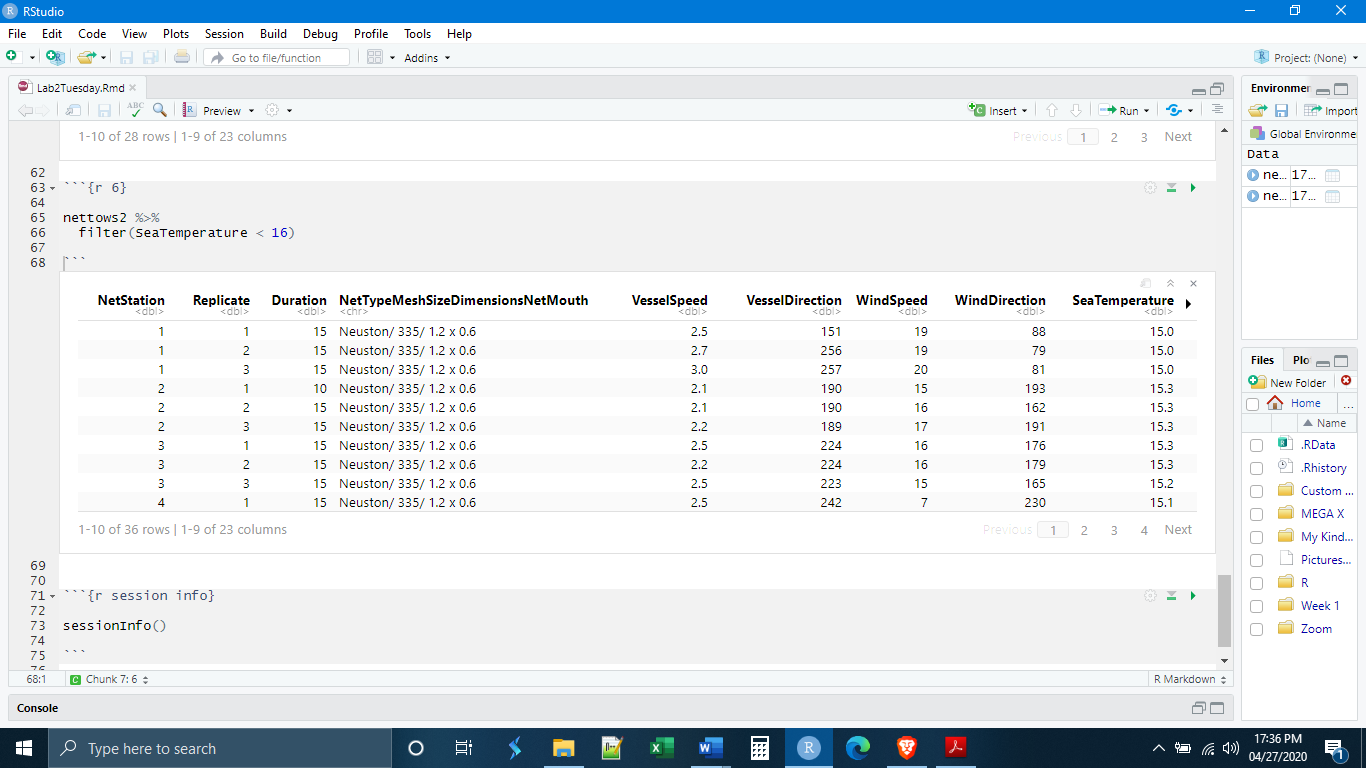
Filter is learned more easily by experimentation, but the syntax for filter() is roughly

filter(VariableName Expression Condition)

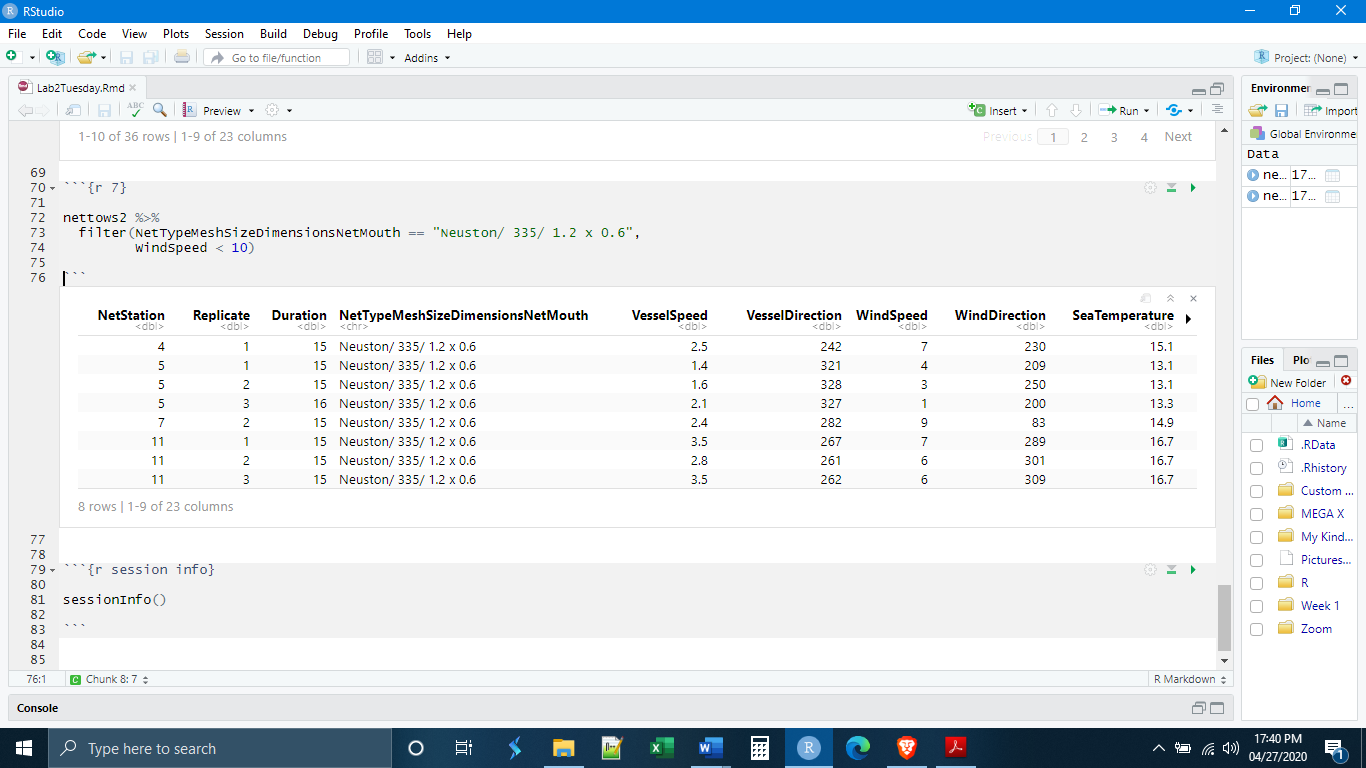
The filter() function can be used to select observations based either on text or numeric qualities. In other words, you can look for all observations greater than a particular number, or all observations that are a particular name. As a first example, let’s look at only those observations where the time spent collecting was more than 15 minutes.



Or where the sea temperature was less than 16 degrees.

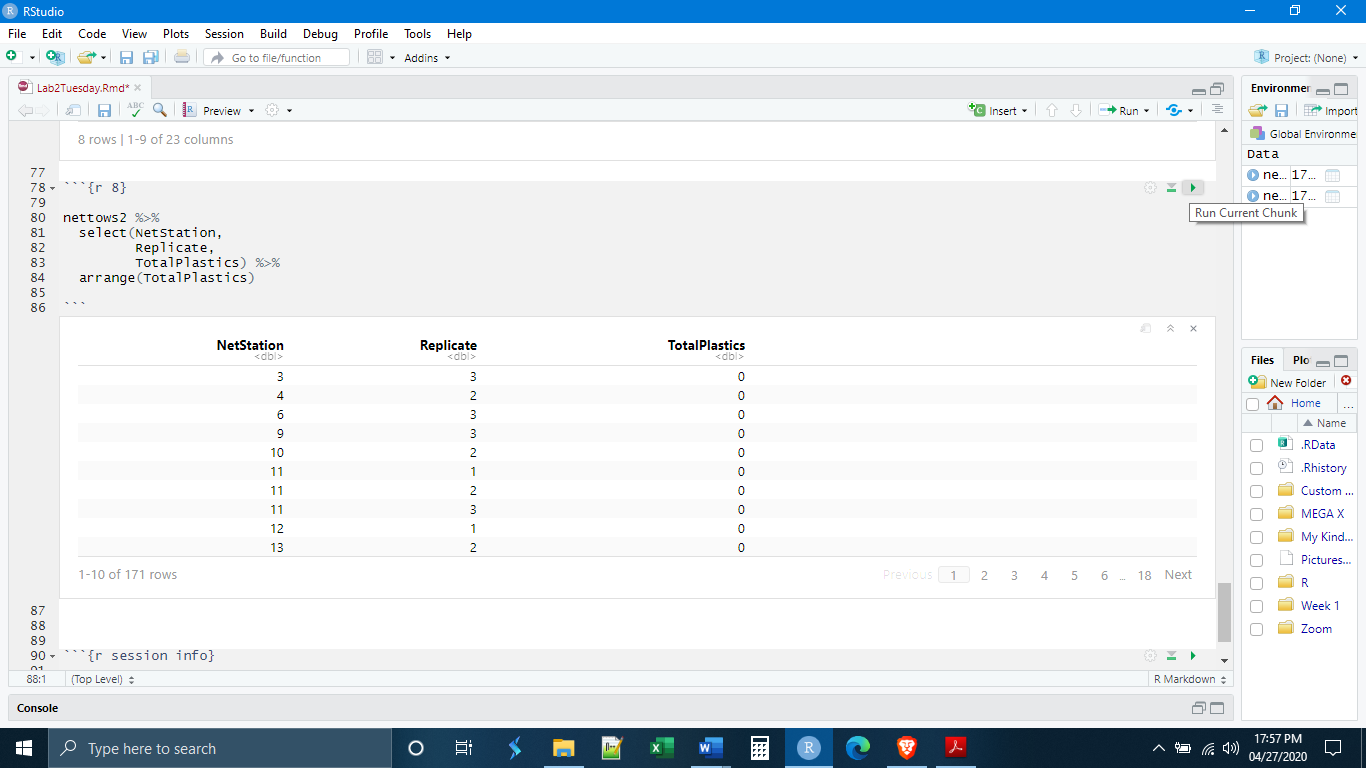


It is also possible to filter on multiple criteria. For example, let’s look for observations that both used the “Neuston/ 335/ 1.2 x 0.6” net and where the wind speed was less than 10 km/h. Notice the double equal sign. R will give you an error if you use a single one because technically this is a logical, not a mathematical expression.

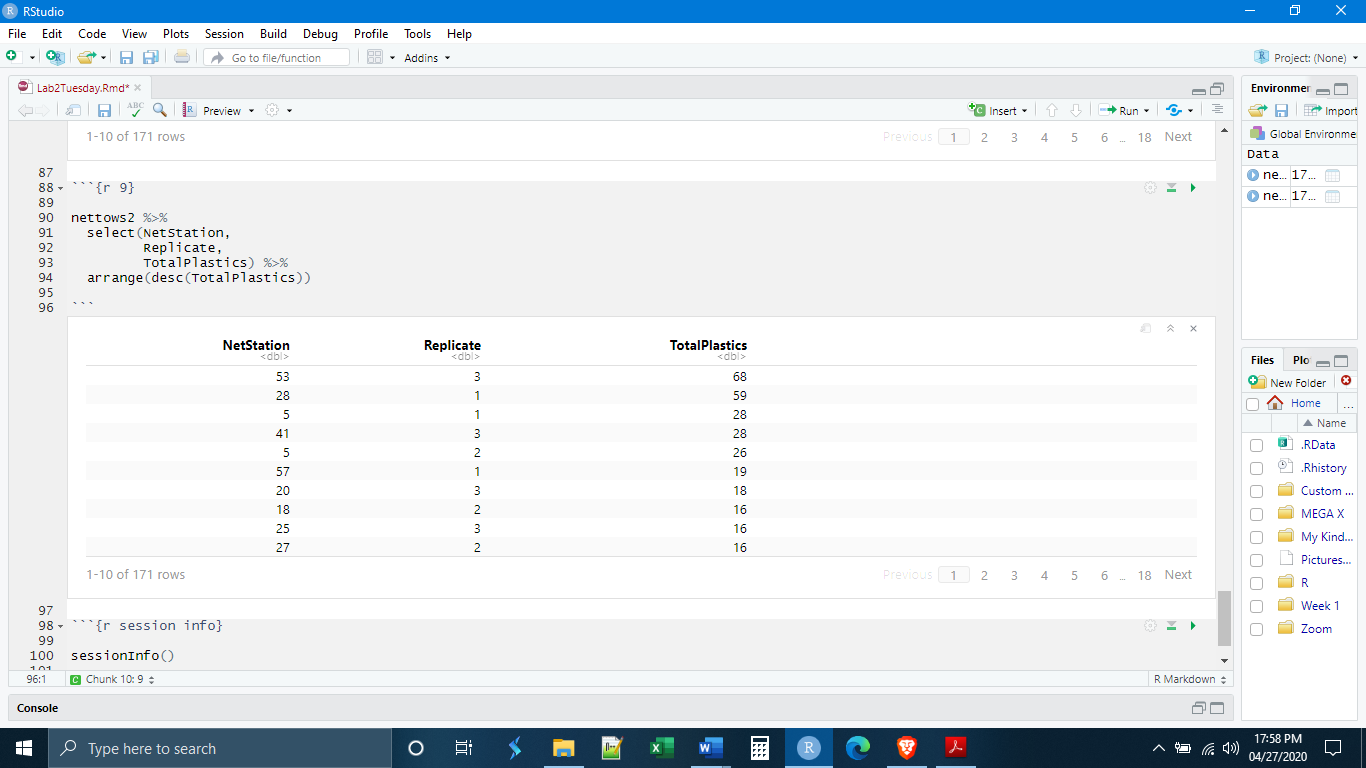


*Arranging*

The arrange() function reorders the observations (rows), alphabetically if a character variable, or numerically if a numeric variable. The syntax is the list of variables you would like to reorder the data by. There is also the option desc() that can be used within arrange() and will reorder in descending order. Let’s try ordering by the number of plastics collected.

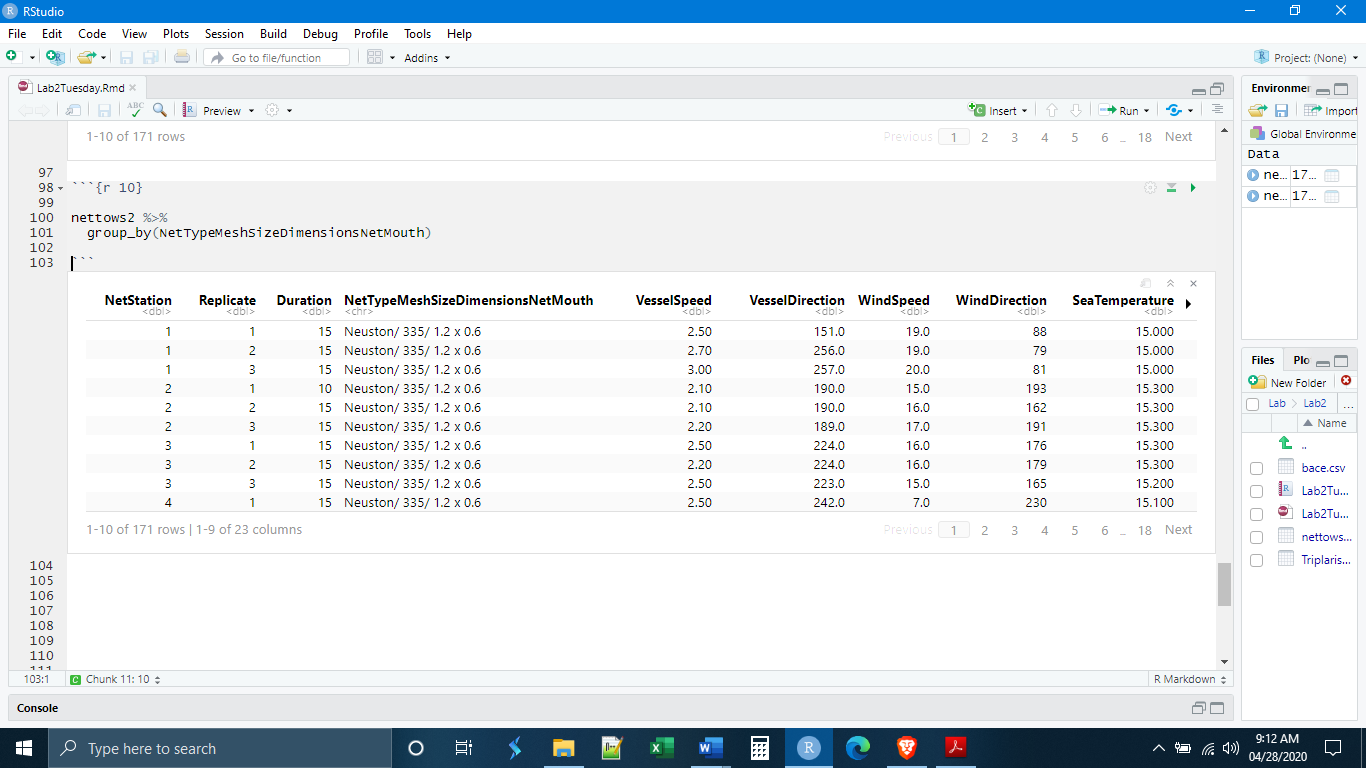


Now, if we use the desc() option we will get the observations listed by number of plastics from most to least.



*Grouping and Summarizing*

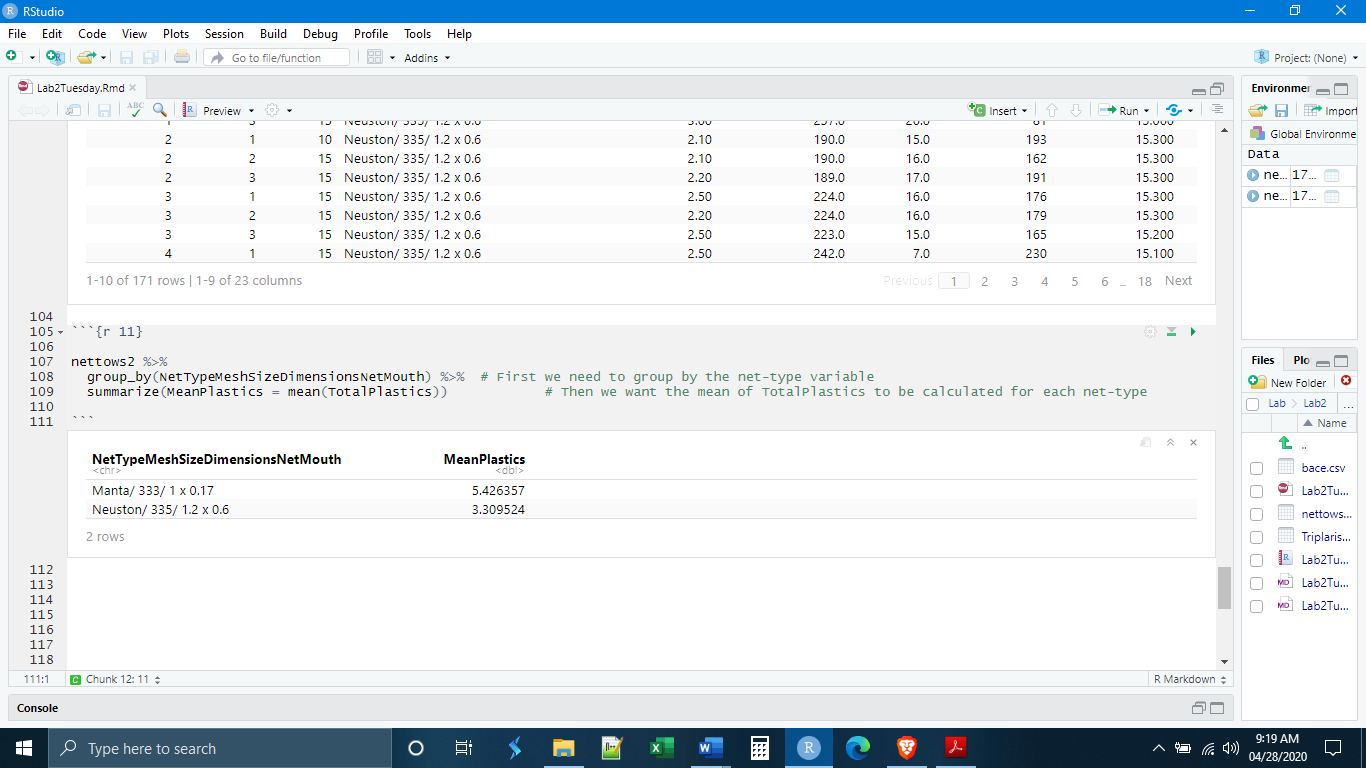
Conceptually, filter(), select(), and arrange() are reasonably straightforward. The functions group\_by() and summarize() are a bit more challenging, but just as useful. The function group\_by() links observations so that R will perform functions on the groups of observations rather on individual observations. For example, we see in our data that two different kinds of nets were used in the sampling, we can group by the net type to do operations on each subset of the data.



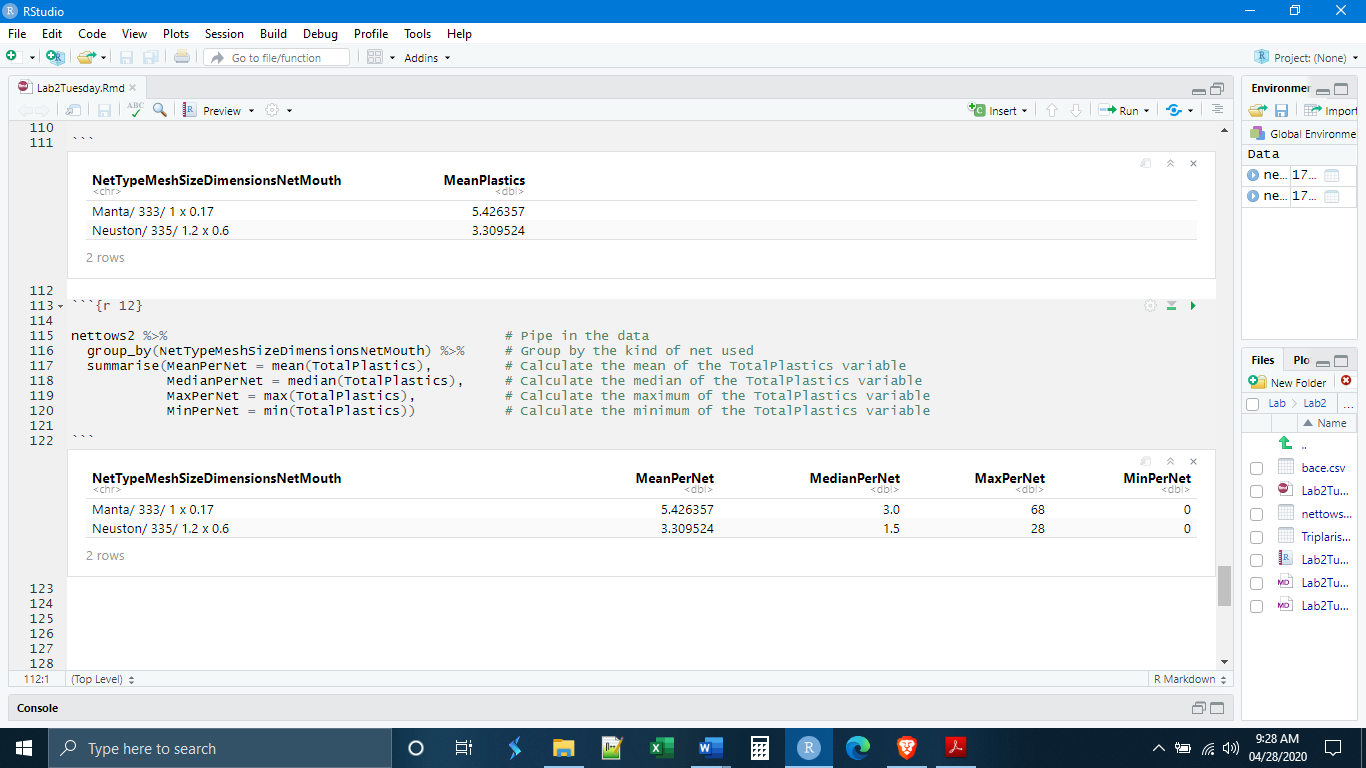
Nothing about the data actually changed; group\_by() does all of its work “behind the scenes”. The effects of group\_by() will become apparent only when we make calculations.

group\_by() is best paired with the summarize() function. This function allows us to make calculations across the grouping variable. Because summarize() will reduce the number of observations and variables in its output, you need to define a new variable. The syntax for the summarize() is as follows

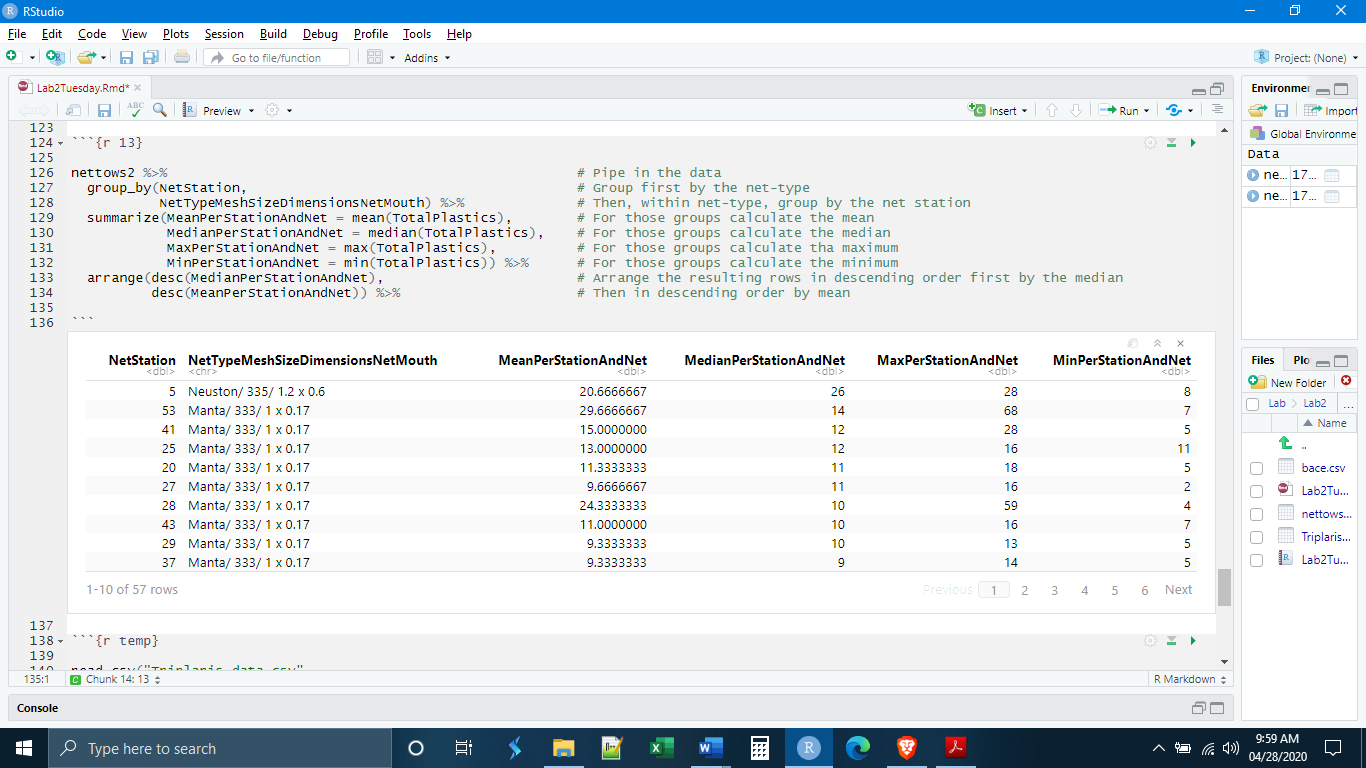
summarize(NewVariableName = Function(ExistingVariableName))



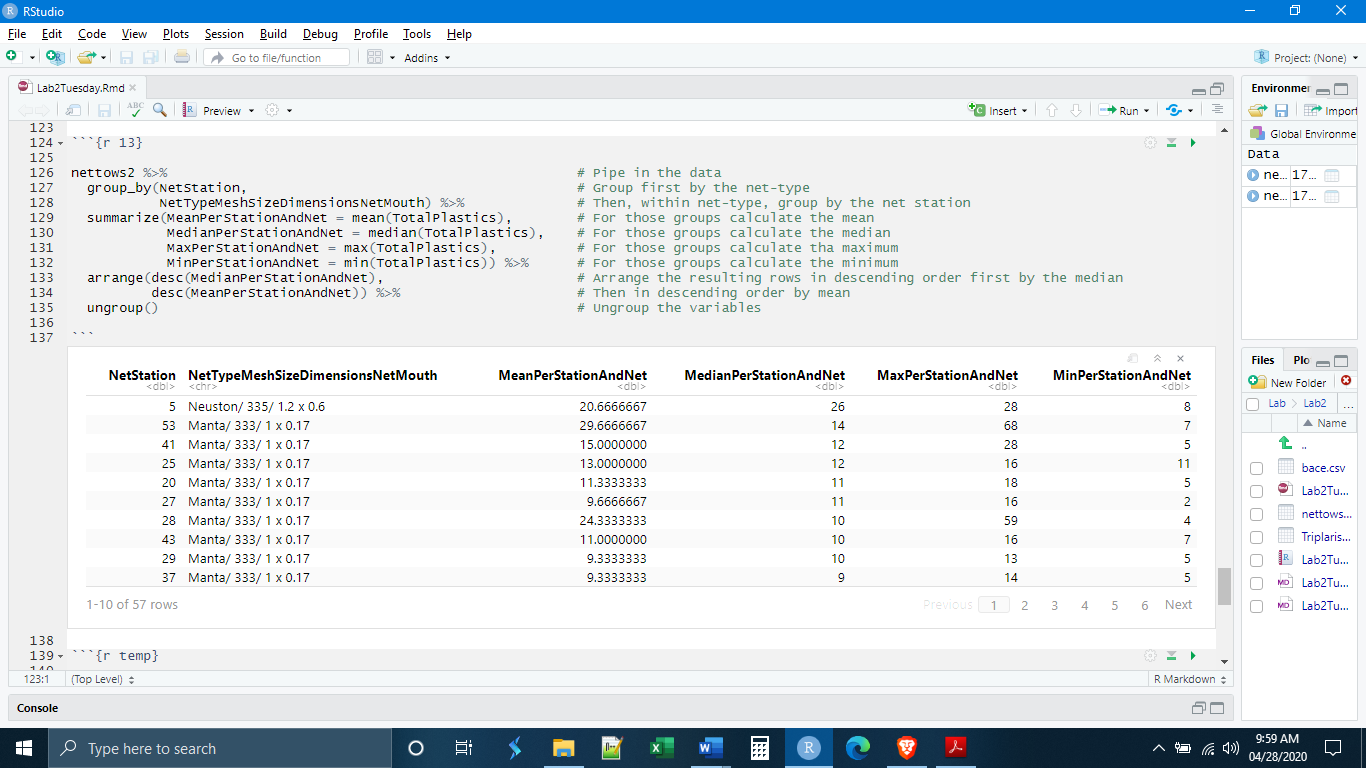
It is possible to ask summarize to do more than one thing, so long as each metric is separated by a comma.



Grouping can be a powerful way to generate summary statistics. You can group by multiple variables and generate multiple metrics.



Finally, particularly if you are working within a script, you will always want to ungroup your data using ungroup(). Grouping will not carry over between chunks but it can carry over the course of a single script. Using ungroup() prevents later calculations being made on the groups.



*Individual Exercises*

**1.** Make a new chunk called “chunk 1” and write the code for a dplyr pipe that will

- Read in the data set nettows\_info.csv (found on Blackboard)

- Select only the variables “NetStation”, “WindSpeed”, “SeaTemperature”, “Salinity”, and “Depth”

- Filter out (remove) NetStation numbers greater than 14

- Save this as a new data object “nettows\_HW”

(Hint: Remember that you need to load your libraries before you do anything else)

**2.** Make a new chunk called “chunk 2” and, using nettows\_HW and the group\_by(), summarize(), and arrange() functions, find the net station that has the lowest average (mean) WindSpeed.

**3.** Make a new chunk called “chunk 3” and write the code for a dplyr pipe that will

- Read in the data set Triplaris\_Data.csv (found on Blackboard)

- Select the variables “ID”, “AcceptedName”, “Country”, and “PetioleLength1”

- Rename variable “PetioleLength1” to “PetioleLength”, and “AcceptedName” to “SpeciesName”

- Filter out the missing data using the PetioleLength variable

- Save this as a new data object “TriplarisData”

(Hint: See page four for filtering out missing observations)

**4.** Make a new chunk called “chunk 4” and, using TriplarisData and the group\_by(), summarize(), and arrange() functions, find the country and species that has the tree with the largest (maximum) PetioleLength.

(Hint: This will require you to use two variables in the group\_by() function)