**Biostatistics**

**Lab 3**

*Tasks*

Data Types, Conversion, and Manipulation; The Pipe, dplyr

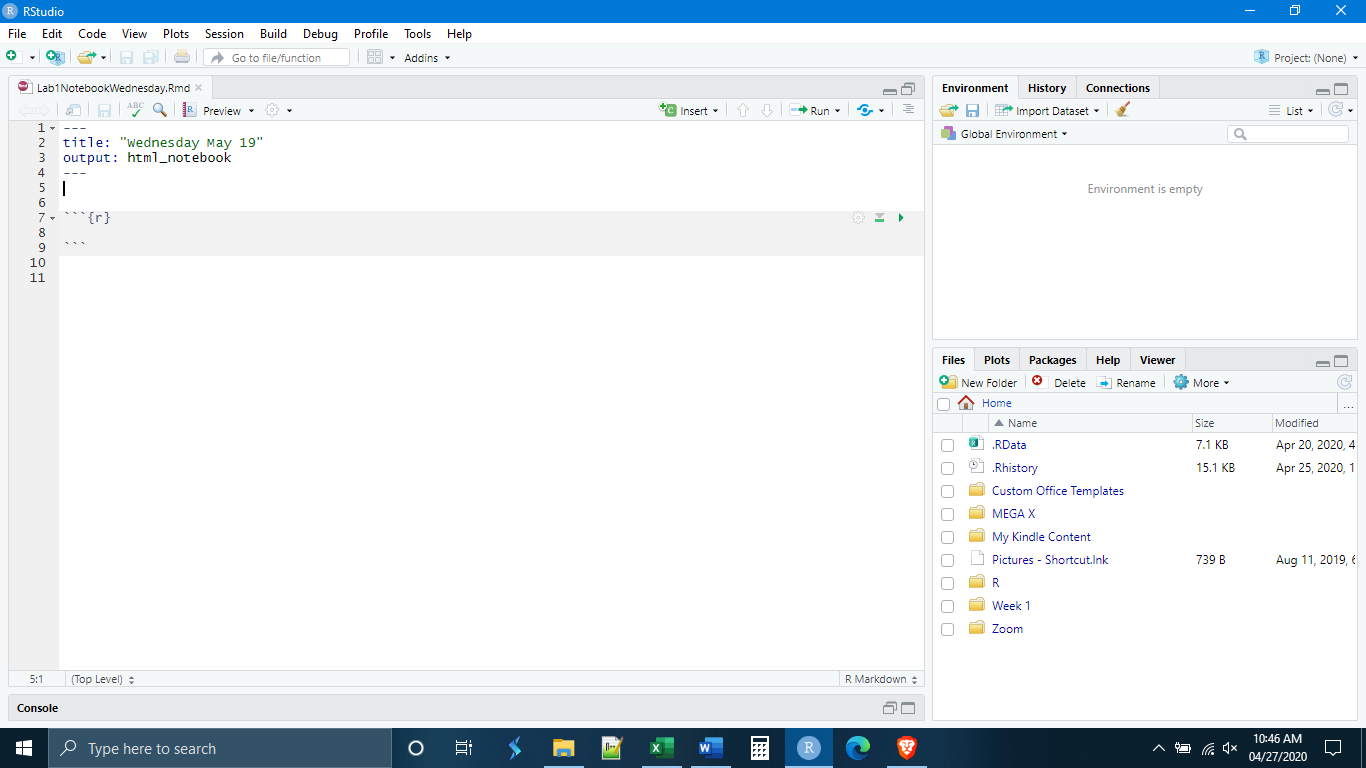
*Introduction*

R stores data in different ways. When you read a file into R, it looks at your data, makes an assumption about how you have recorded your data, and codes it accordingly. In most cases, R does a pretty good job. In other cases, however, you may need to tell R to change the way it understands your data. The categorization of data can be subtle, and there are a number of different options. Today, we will go through some of the basic ways that R will store and interpret your data as well as how to change those data types.

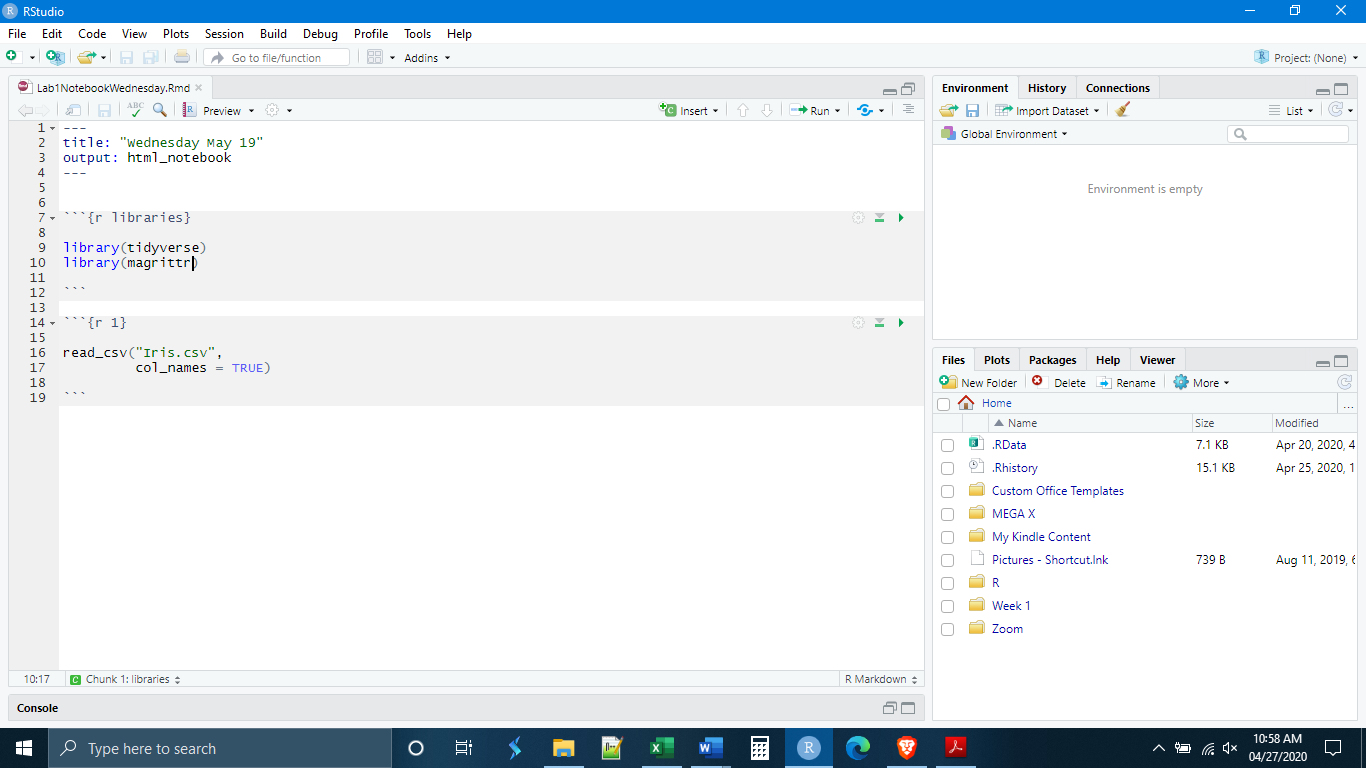
The other thing that we will cover today is what is called the “pipe operator”. The usefulness of this bit of code may not be apparent now, but it will become apparent later and I want to introduce it now so that we can use it as we go along. Pipe operators exist in other programming languages as well, and their function is to provide a way to write more complex code, code that involves multiple functions. In R, the pipe operator is also designed to make complex code easier to read and interpret.

*Read in Data*

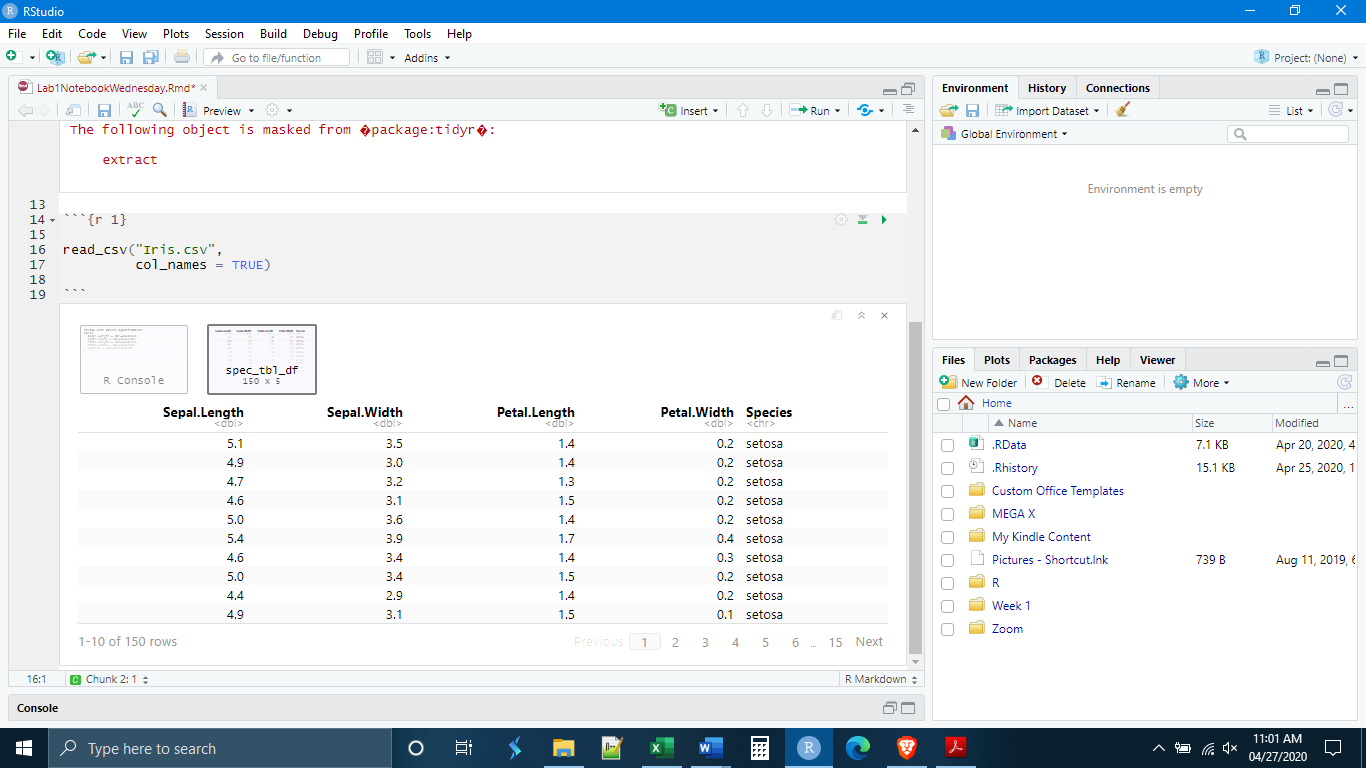
The first thing that we want to do is start R Studio and open a new notebook file. Save it in your Lab1 folder because we will be using some of the same data from yesterday and we want this to be our working directory. Also, you can delete all of the notes that the template begins with so that we are just left with our chunk. Second, make sure you have downloaded the Iris.csv data set and saved it in the same Lab1 folder that this new notebook is saved in.



Recall from yesterday that we need to explicitly load some of the functions that we plan to use. We do this by loading libraries. A good way to do this is to make the first chunk of the document devoted to the libraries. We will also want to load the library for the pipe operator, as we will use it later. Then, the second chunk can be for reading in data. Let’s execute the code in both chunks and read in the Iris data set.



Success! Let’s look at what R has given us. We see that we have five variables which are given in bold. If we look right below these column headings, however, we also see some light gray letters. These indicate the type of data in that column. In this case, we see <dbl> and <chr>.



It is important to know how R stores data because different types of data and different types of storage methods (different types of objects) are capable of different things. Some functions will not work with certain data types or data objects. R has six data types, six ways of classifying what kind of thing a particular piece of information is.

Data Types in R

Logical: TRUE, FALSE, AND, OR, etc.

Numeric: Numbers in the way that we usually understand them, including partial numbers.

Integer: Whole numbers only.

Complex: Numbers but including things like imaginary numbers.

Character: Data involving letters, or letters and numbers together.

Raw: Information stored as raw computer bytes information.

In this course, and for most biological work, we are interested in Logical, Numeric, Integer, and Character data types. You will only be expected to know what these are.

These different data types can be stored in different data objects. We know some things about data objects having created vectors and matrices yesterday. A vector is a one-dimensional collection of data elements. A useful way to think about this is that a vector is a like a column in a table. In contrast to this, a matrix is a two-dimensional collection of data elements, and an array is a multi-dimensional collection of data elements.

Some Object Types in R

Vector: One-dimensional collection of data elements (think of a single column in a table).

List: Similar to a vector but can store other things like functions and other lists.

Matrix: Two-dimensional collection of data elements.

Array: Multi-dimensional collection of data elements.

Factors: These are modified vectors where the data elements represent certain classes.

Data Frames: Think of this like a table, it is a collection of vectors (columns), with each vector being composed of data elements (observations/rows) of a single data type.

Tibble: A tibble is a special king of data frame unique to R.

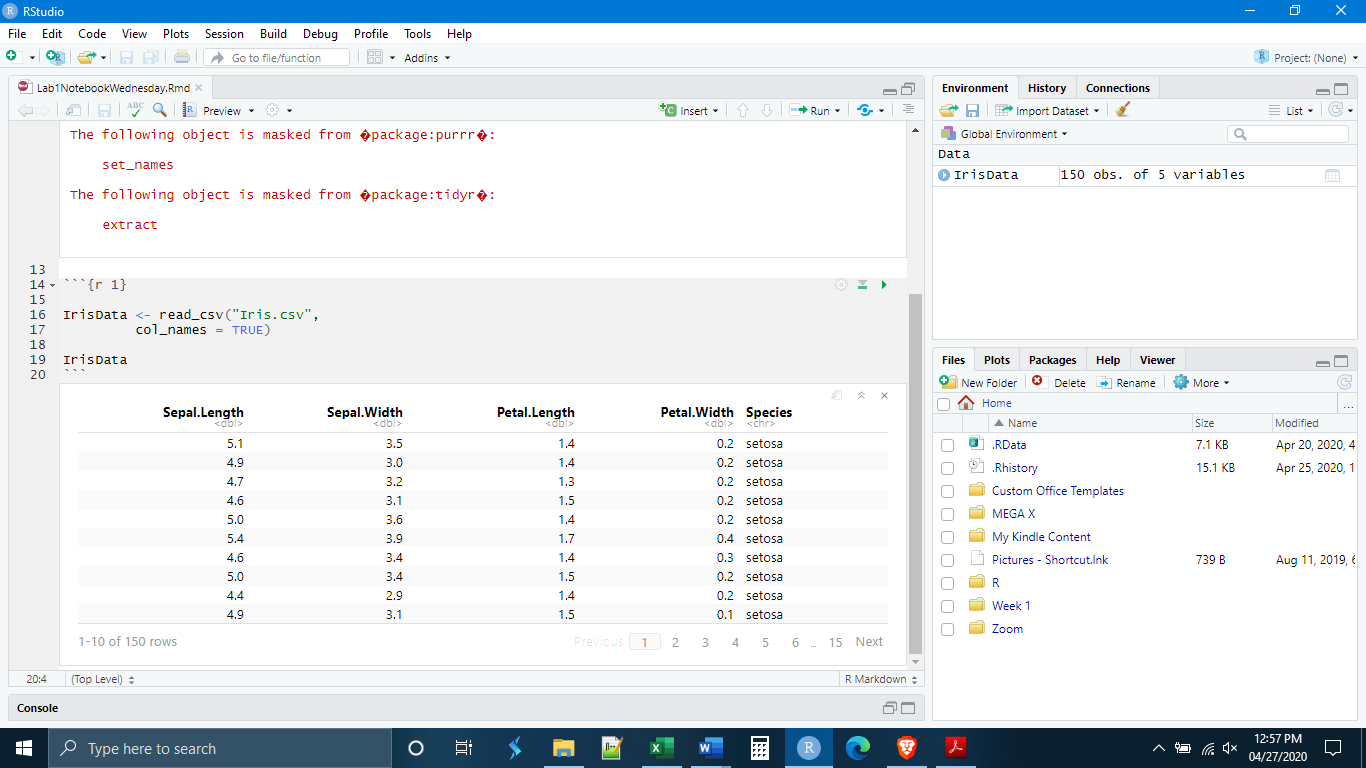
The data objects that you will need to know about are vectors, factors, data frames, and Tibbles. Of these four, the vector is the most important. The other three object types are built out of vectors. Factors are modified vectors, where each of the data elements of the vector belongs to a certain class or category. For example, our vector may be c(A, B, B, D, C, A, B), and if turned into a factor, it would have four categories or classes: A, B, C, and D.

For the purposes of this course, we can think of a data frame as a table of data. Really it is a collection of vectors. Each column is a vector where all of the data elements have the same data type. All of the vectors are related by sharing the same observations (rows). The Tibble is a special kind of data frame that was invented specifically for R and follows certain rules associated with the tidyverse, which makes it work better inside of the tidyverse. The read\_csv() function creates Tibbles, so most of our work will technically be with Tibbles, but it is fine to think of them as data frames.

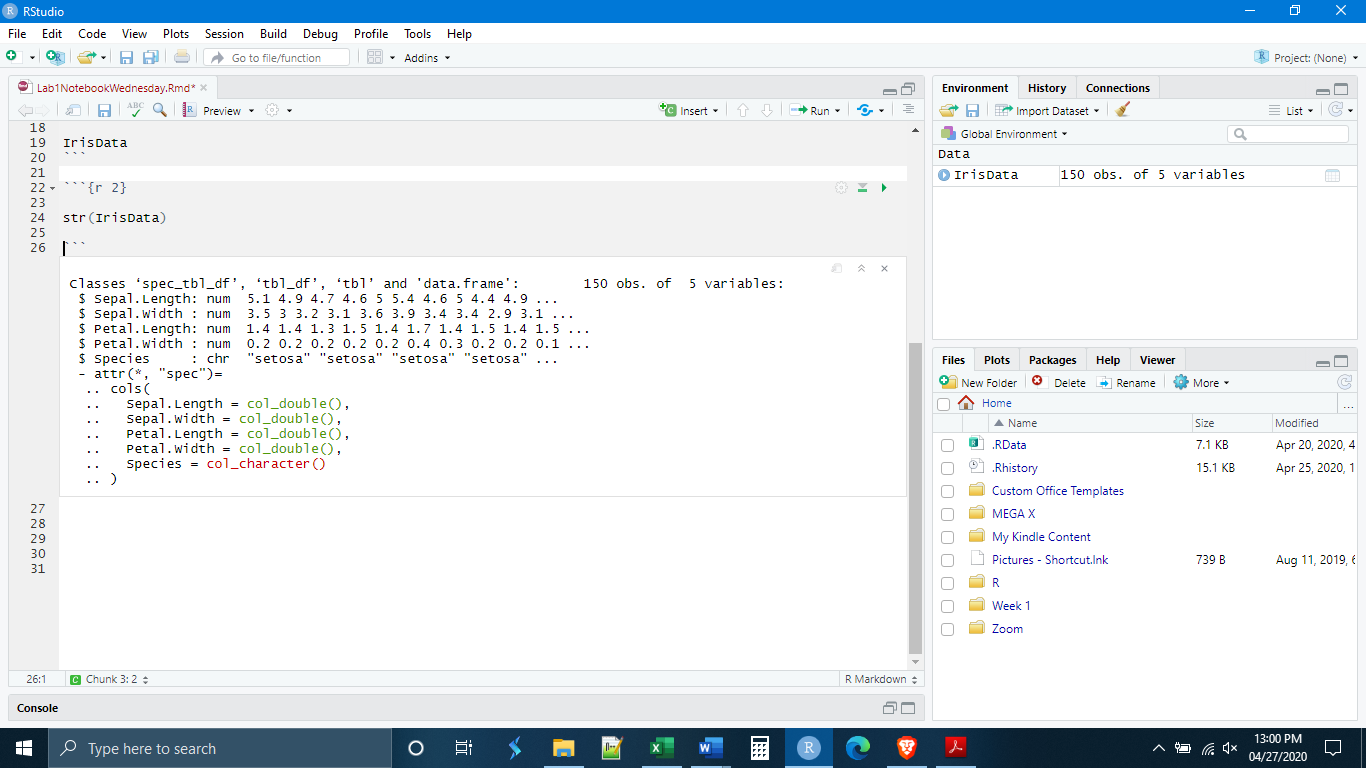
*Explore Data*

Given what was just outlined, what can we say about the Iris data that we have just read-in? Well, we know that the object type is a Tibble, which is a kind of data frame. What about each of the columns, what data type are they? We saw that they were listed under the column headings as <dbl> and <chr>. The <chr> stands for “character”. Data in that column are stored as a character data type. If we look at the data elements this makes sense. The <dbl> stands for “double”, which is the term that R uses for numeric data types. Looking at the data elements in these columns, this also makes sense. Another way to explore our data is through the commands str() and summary().

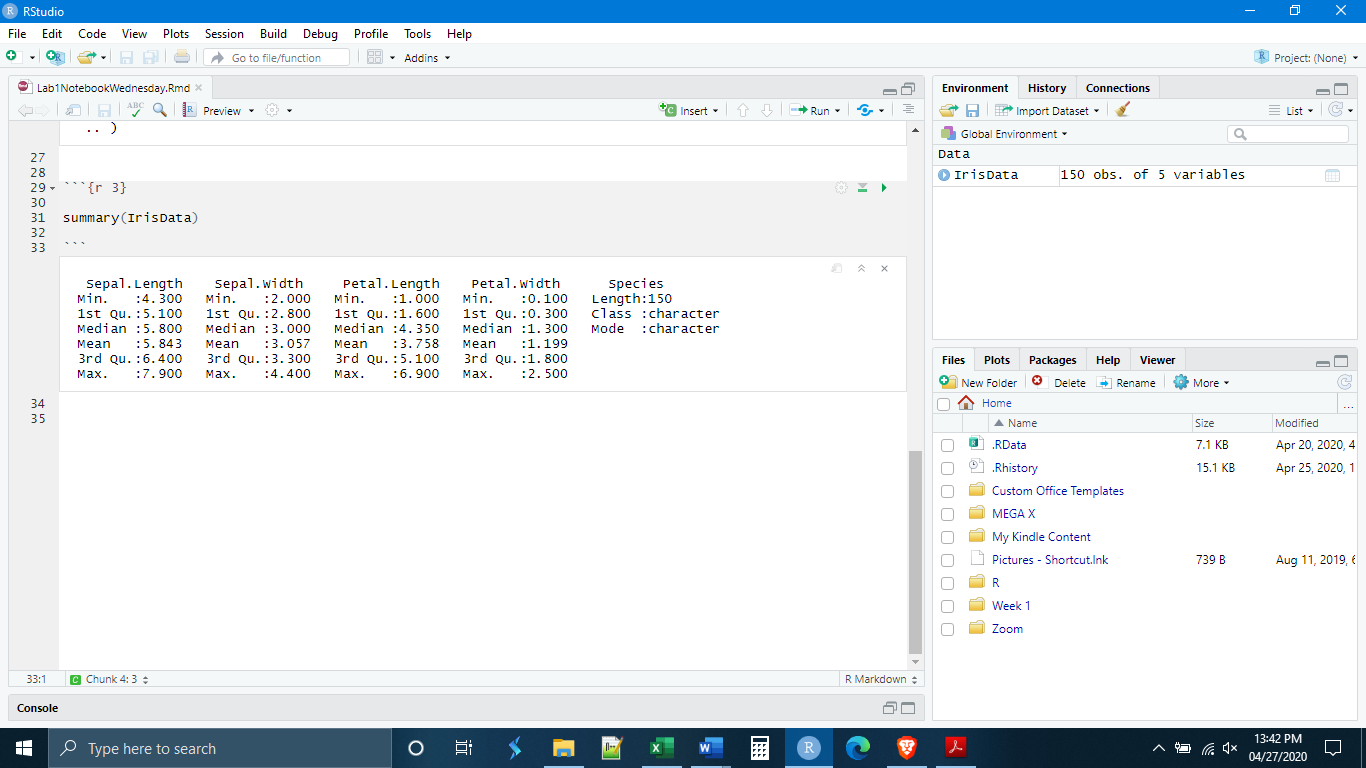
First, we need to rewrite our code to save the imported data as an object (notice it is still not in our environment).



Now, start a new chunk and use the str() command to get information about the data set itself. We get a good amount of information. We see information on the kind of data object that it is, both a Tibble and a data frame. This makes sense because a Tibble is a special kind of data frame. We also see that the data set contains five variables and 150 observations. Then we see two sets of what amounts to the same information. Each of the variables is listed with its name (the column heading) and then what kind of variable it is. As we saw in the table, all of them are classed as numeric (double) except the last one which is a character variable.



In lecture we have been going over what can broadly be called “summary statistics”, basic information about the data such as center and spread. There is an easy way to generate this in R, that will also give us more information about the data set, this is the command summary(). Begin a new chunk and execute the summary() command on the Iris data set.

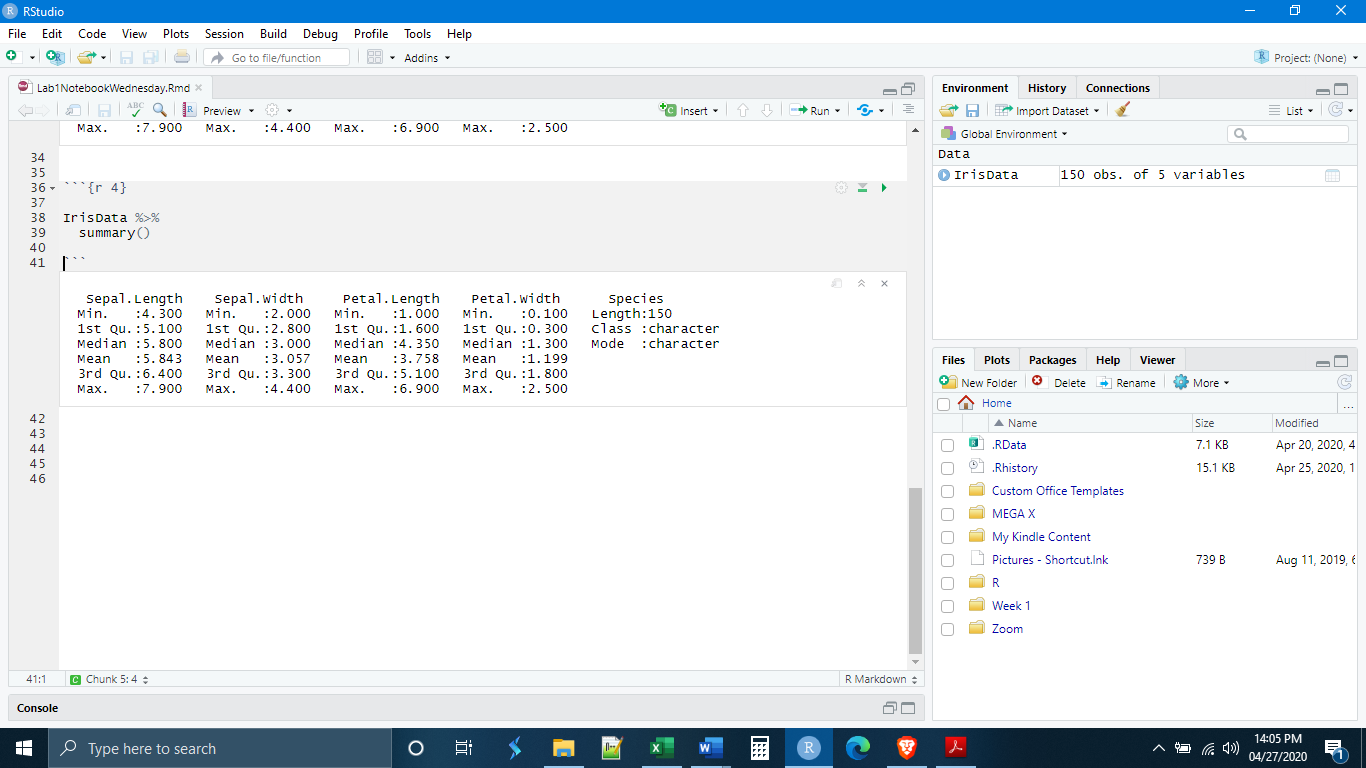


*The Pipe Operator*

Up to this time, we have written code as single lines, each line doing one or two operations or functions. But sometimes we want to do more than one function, or we want to take the output of one function and put it directly into another function. To do this in the way that we have been writing code, we would need to save each intermediary step as a new data object in order to enter it into the next function. In even a moderately complex analysis this would become tedious very quickly.

The pipe operator comes to rescue us from this problem. The function of the pipe operator is to take the output of one function and put that output directly into another function. In R, the pipe operator is written as %>%. And it is always written at the end of a line of code, indicating that the output or the result of that line of code should be fed into the next line of code. Because the pipe operator specifies what the data are going to be, you need not enter the data explicitly (there will be exceptions to this, but we will go over them as they arise).

Let’s do a simple example. How could we write the code to get the summary statistics using the pipe operator? (see Below) Notice that, when we hit the return key the next line is indented. This is a customary way of writing code which indicates that this second line is really part of the code that comes above it. R Studio knows this and automatically indents the code for you.



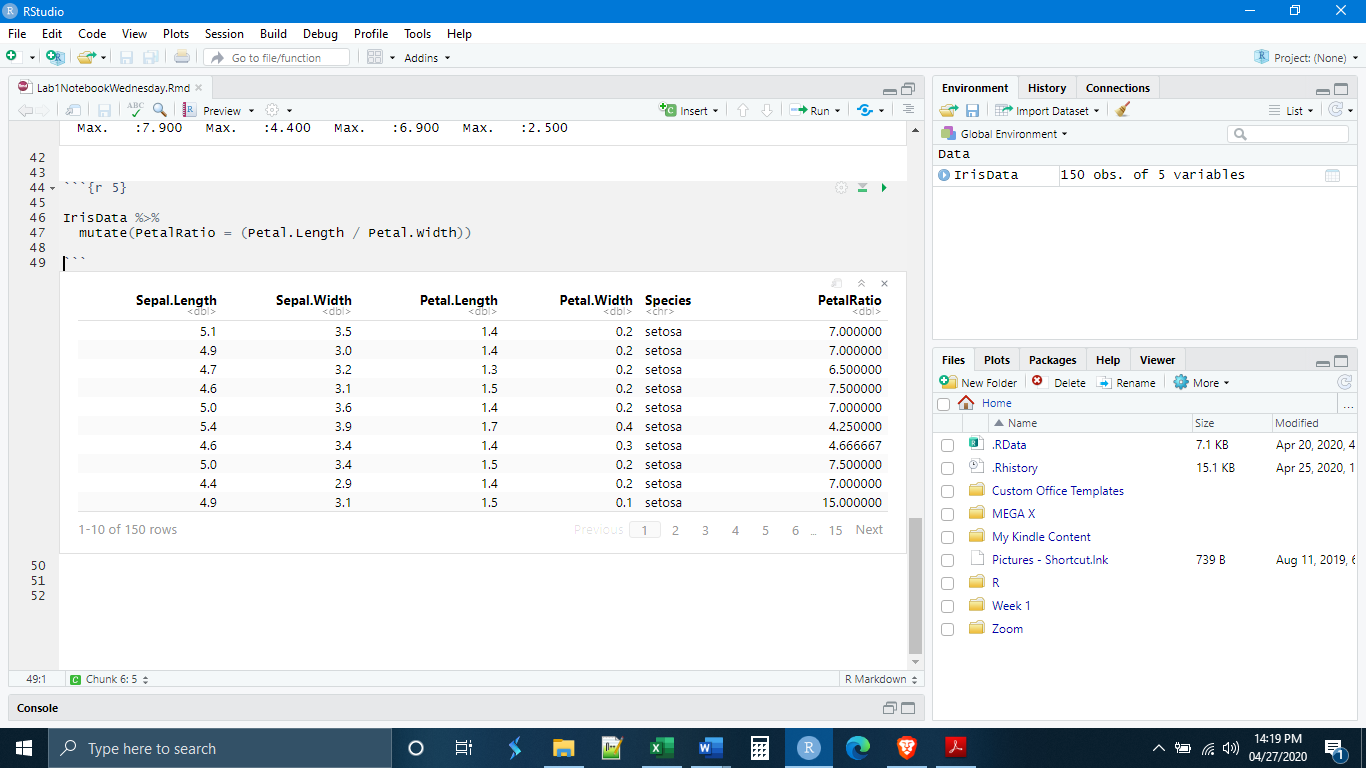
IMPORTANT: As we have seen with other functions, operations done through the pipe operator do not change the original data set unless we create a new data object.

*Manipulate Data Types*

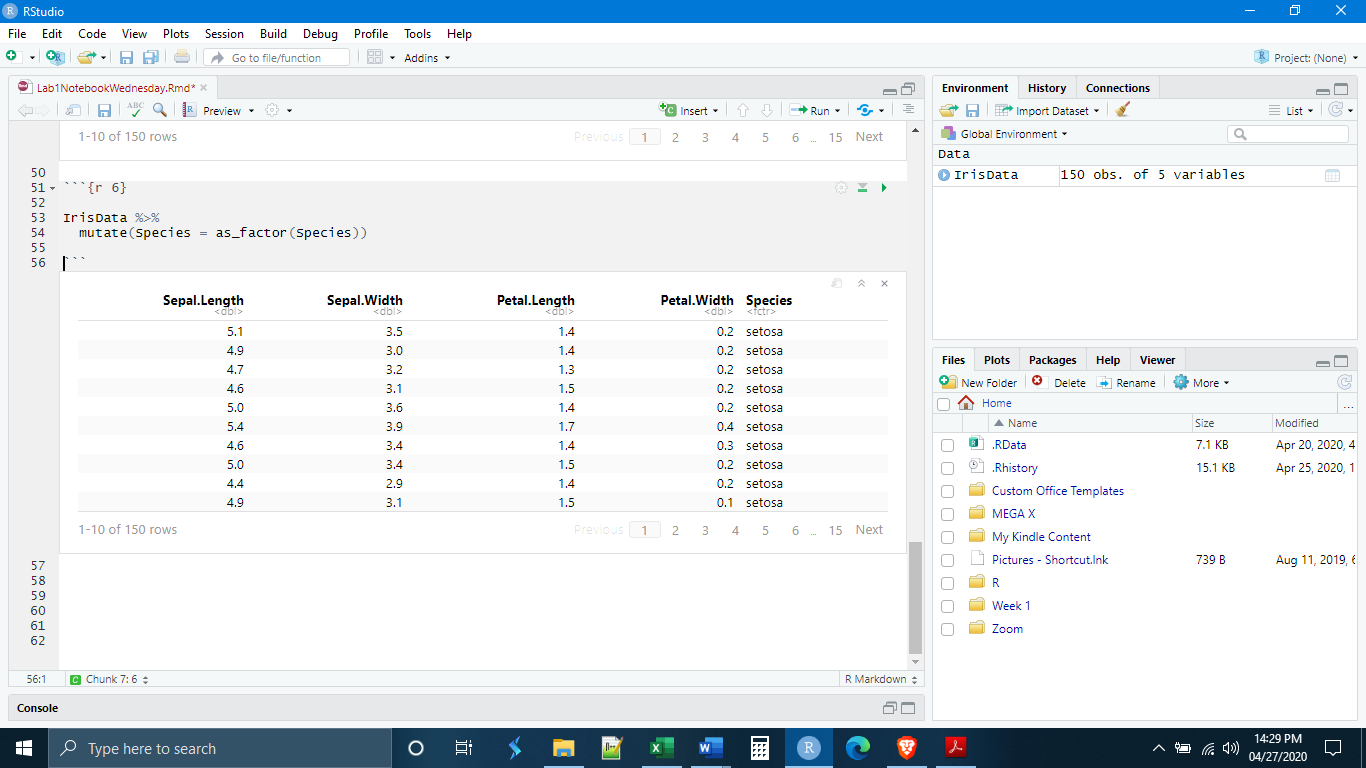
The last thing that we will do today is learn some of the basics of manipulating data in R. The section of the tidyverse that deals with the manipulation of Tibbles is dplyr. We will learn more about dplyr next week, but this will serve as a very brief introduction. Today we will learn about the dplyr functions mutate() and rename(). mutate() creates new vectors (columns) in a Tibble. The syntax for mutate() is:

mutate(NewVectorName = ContentsOfNewVector)

We can do a lot with mutate and it is perhaps best to simply show some examples. Our Iris data set includes the variables flowers petal length and flower petal width. What if we wanted to make a new column that was the ratio of petal length : petal width? This could be done with the code below. Notice that, because we piped the data into the mutate function, R already knows what vectors (columns) exist in that data set, and we do not need to specify it a second time.

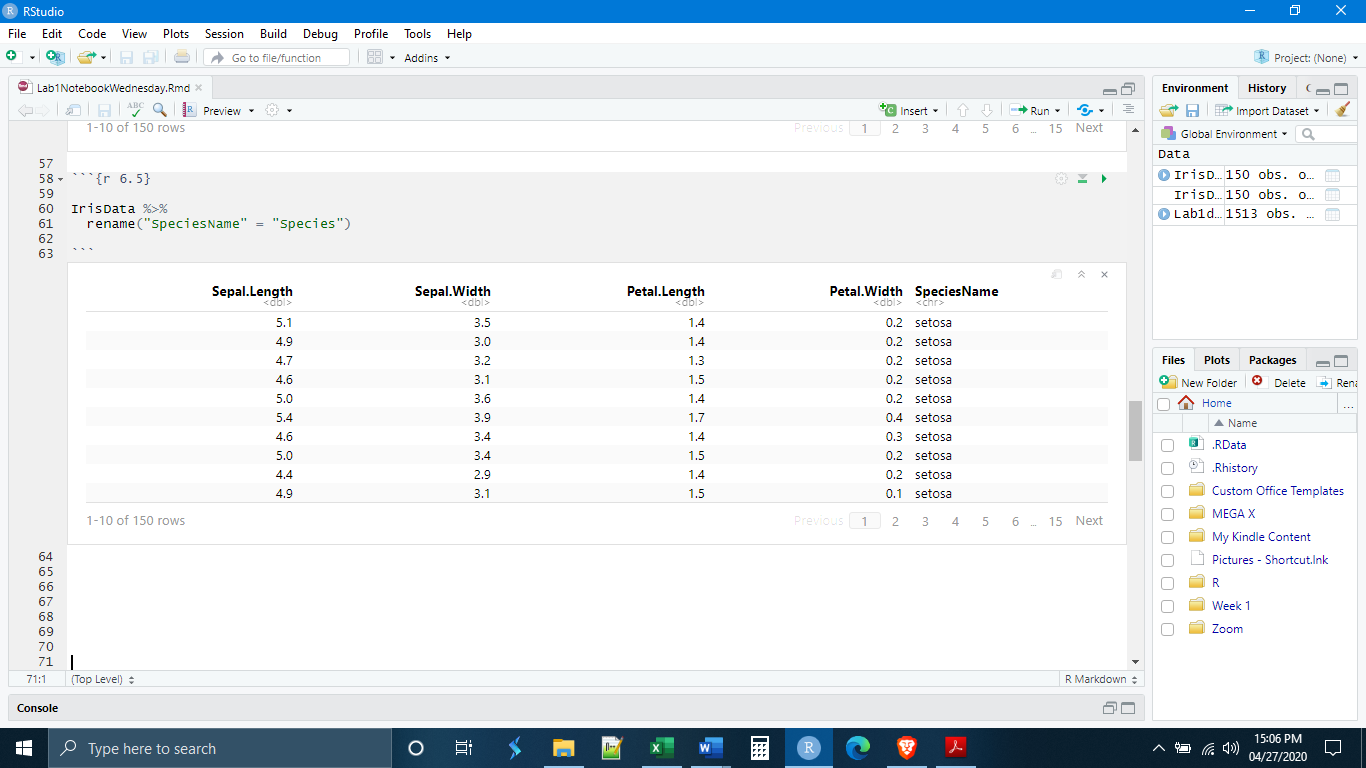


A second example. The fifth column of our Iris data set has a variable named “species”. R interpreted this variable as a character data type (also known as “string” data). But really those strings represent different categories (the different species). We can use mutate to change the data type to reflect this by using the function as\_factor(). (See Below) Now that column is labeled as <fctr> data type. Really, the Species vector in our Tibble is now a factor object. Sometimes R will not recognize numbers as numbers and will code a vector (column) of numbers as characters or factors. A function that can undo this is as\_numeric().

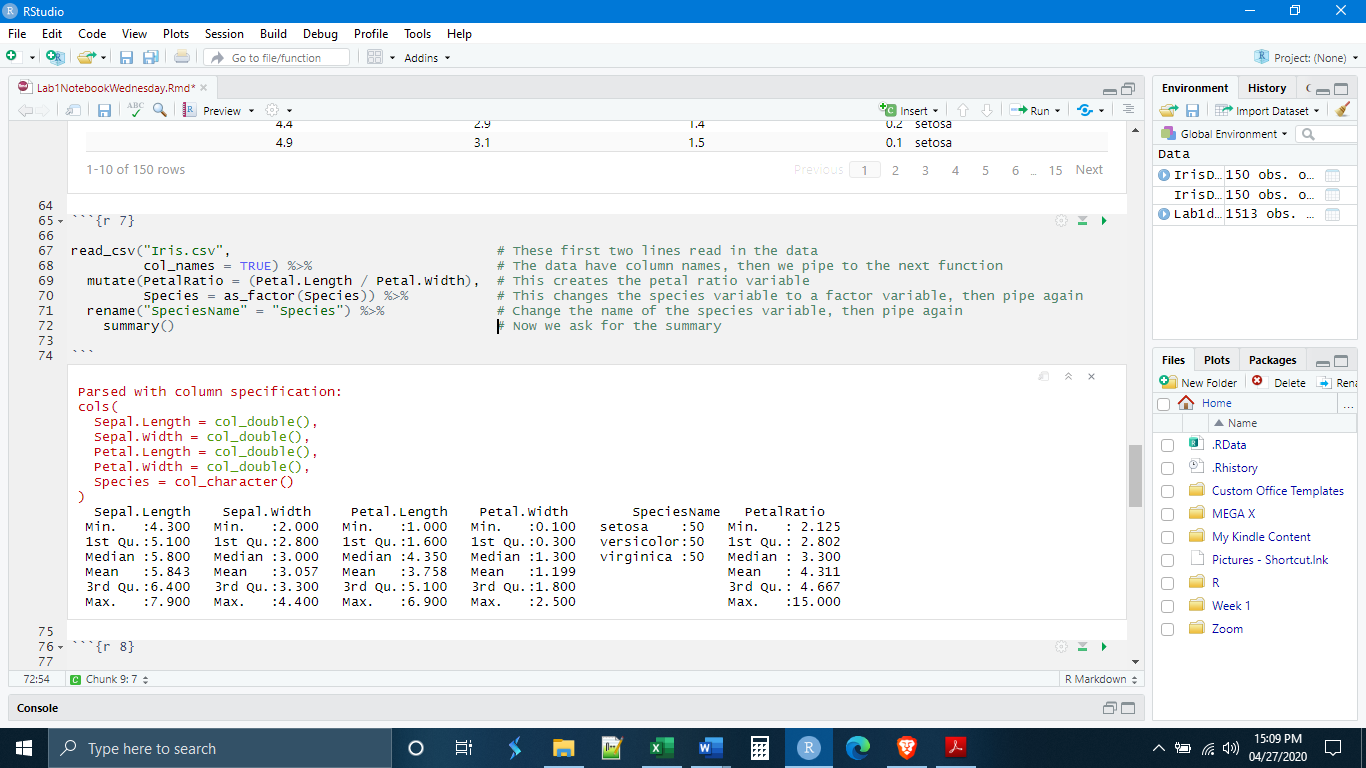


It is also possible to rename variables using the rename() function, which uses the syntax:

rename(“NewName” = “CurrentName”).

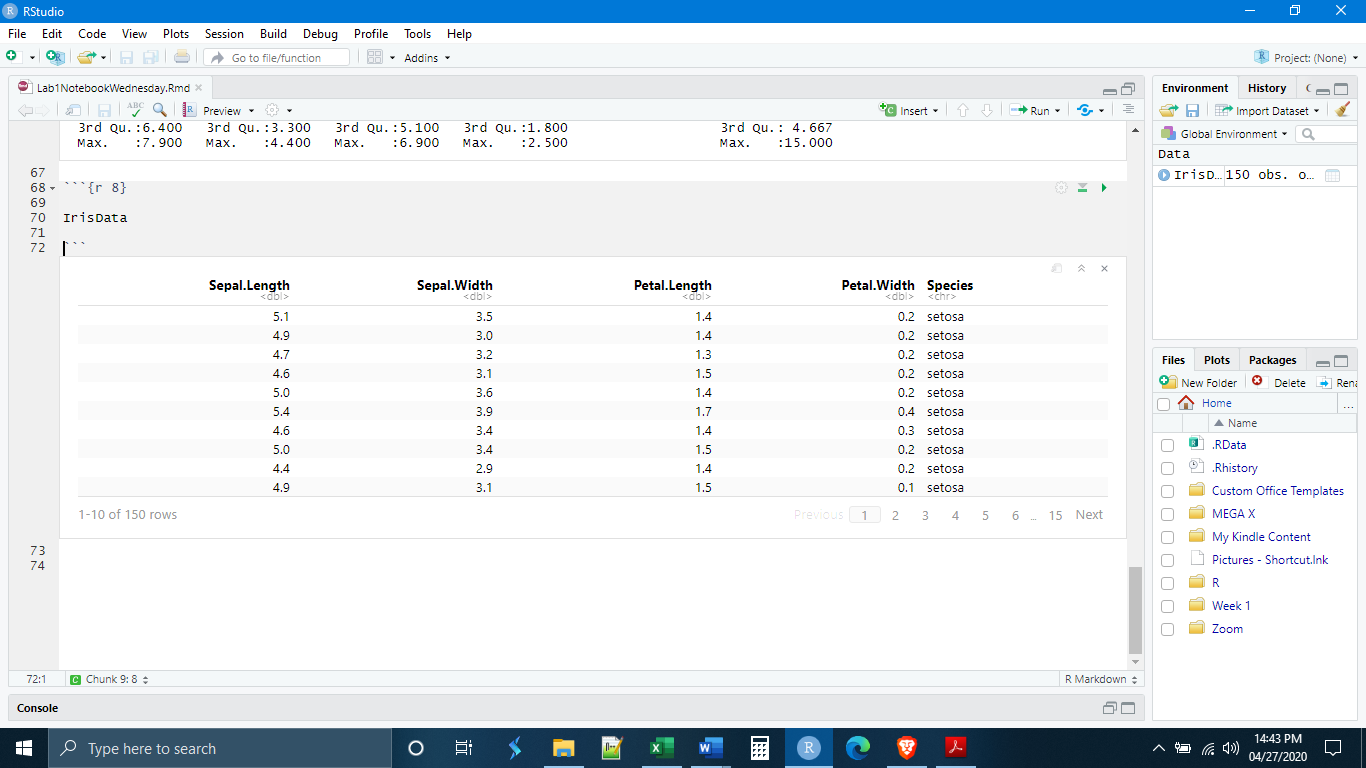


Let’s write our first dplyr pipe. All of the things that we have done today can be accomplished in one piped series of commands. (See Below) Notice, we can do as many mutations as we like, as long as we separate them by a comma. I put them on separate lines because it is easier to read.

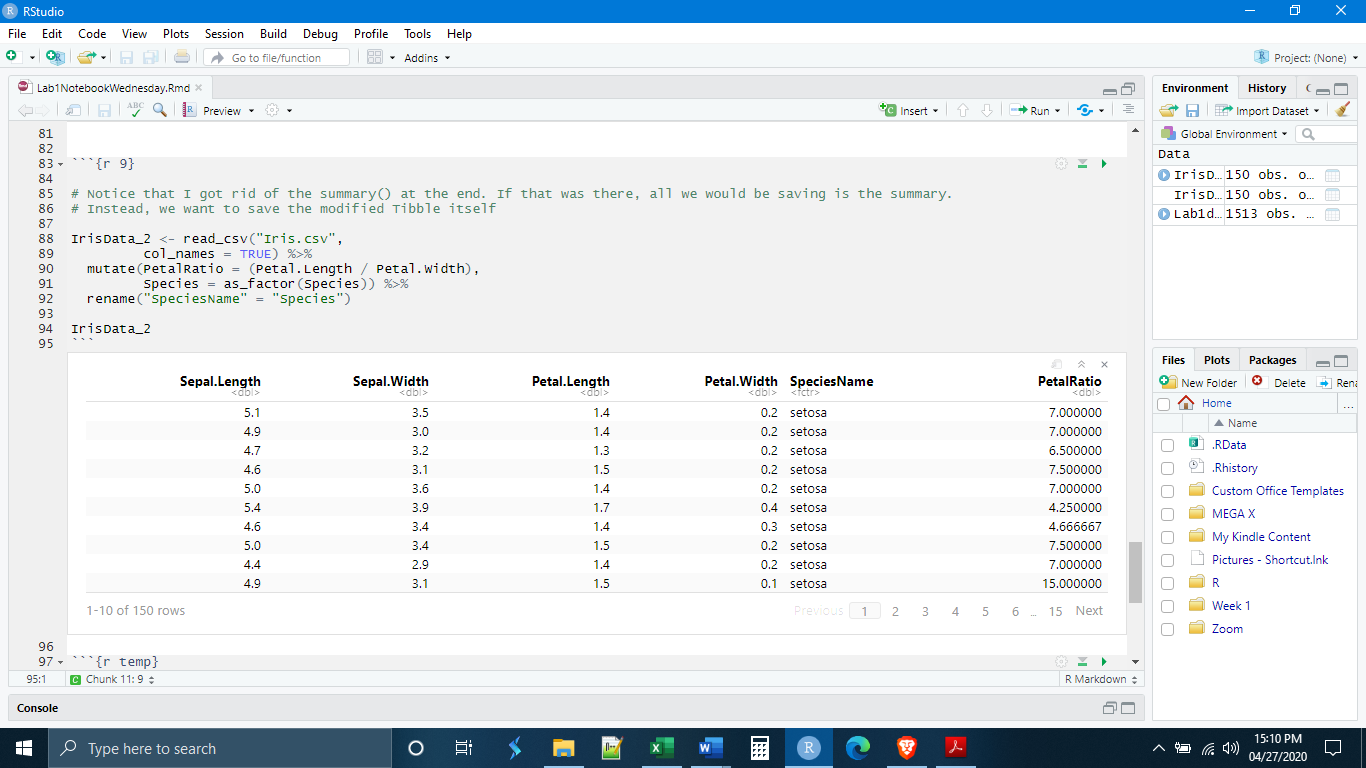


Our summary is different from the first time that we did it. Notice that it now lists all of the factor “levels” in our Species variable, in this case the three species in the data set. We also have information on the new variable that we created about petal length : width ratios.

What happens if we recall the original IrisData object? They are unchanged. Why? We did not create a new data object with the changes.



If we want the changes to stick, we need to create a new data object.



*Exercises*

**1.** Make a new chunk called “chunk 6” and write code that reads in the Lab1data2.csv and uses the str() function to determine the number of variables (columns) in the Tibble and the number of observations (rows). How many are there of each? Record the answer as a comment in the chunk.

**2.** Make a new chunk called “chunk 7” and write code that reads in the ToothGrowth.csv data and mutates the “supp” variable to a factor variable.

**3.** Make a new chunk called “chunk 8” and write code that reads in the ToothGrowth.csv data, mutates the “supp” variable to a factor variable, and then gets a summary of the data. How many “levels” are in the factor variable “supp”? What are the names of the levels? Record the answers as a comment in the chunk.

**4.** Make a new chunk called “chunk 9” and, without saving any new data objects, write a single dplyr pipe that:

- Reads in the data set Lab1data2.csv

- Mutates variables X2 and X3 to factor variables

- Renames variables X1 to “ID Number”, X2 to “Class”, and X3 to “Model”

- Summarizes the data

In the factor variable “Model”, what are the names of the factor “levels”? How many observations are in each level? Record the answers as a comment in the chunk.