Lab 2

RScripts, data types, data structures and subsetting

[ENVS 193](index.html)

Angus Watters - based off material from Mike Johnson and input from Debra Perrone

# **Goals**

* Work within an RScript file (.R)
* Load libraries into an RScript (tidyverse & here)
* Read a .csv data file into an RScript
* Explore different data types and structures in R

# **Open Script from Pre-lab\_02**

* Open up your lab-02 Rproject from your Rprojects folder. At the top of your Rscript, you should have the #, followed by Lab-02, your name, and the date.
* Remind yourself what the hash symbol (i.e, #) does. The **hash symbol is used to “comment out”** text in your RScript. What does this mean? R does not run it as code. Note that if you kept your text, but removed the #, you would get an error message.
* It is good practice to use the # to comment on your code and describe your process. It helps you remember what code does, as well as helps you keep track of things like units. It also helps someone viewing your code to understand your processes.
* Note that under your intro comment, you should have two libraries ready to load with code that looks like the following

# Angus Watters  
# ENVS 193 - Professor Perrone  
# Lab 2  
# 1/4/2021

# **Load in libraries**

* In this lab, we will be using the tidyverse and here packages
* Load these libraries in by using the library() function

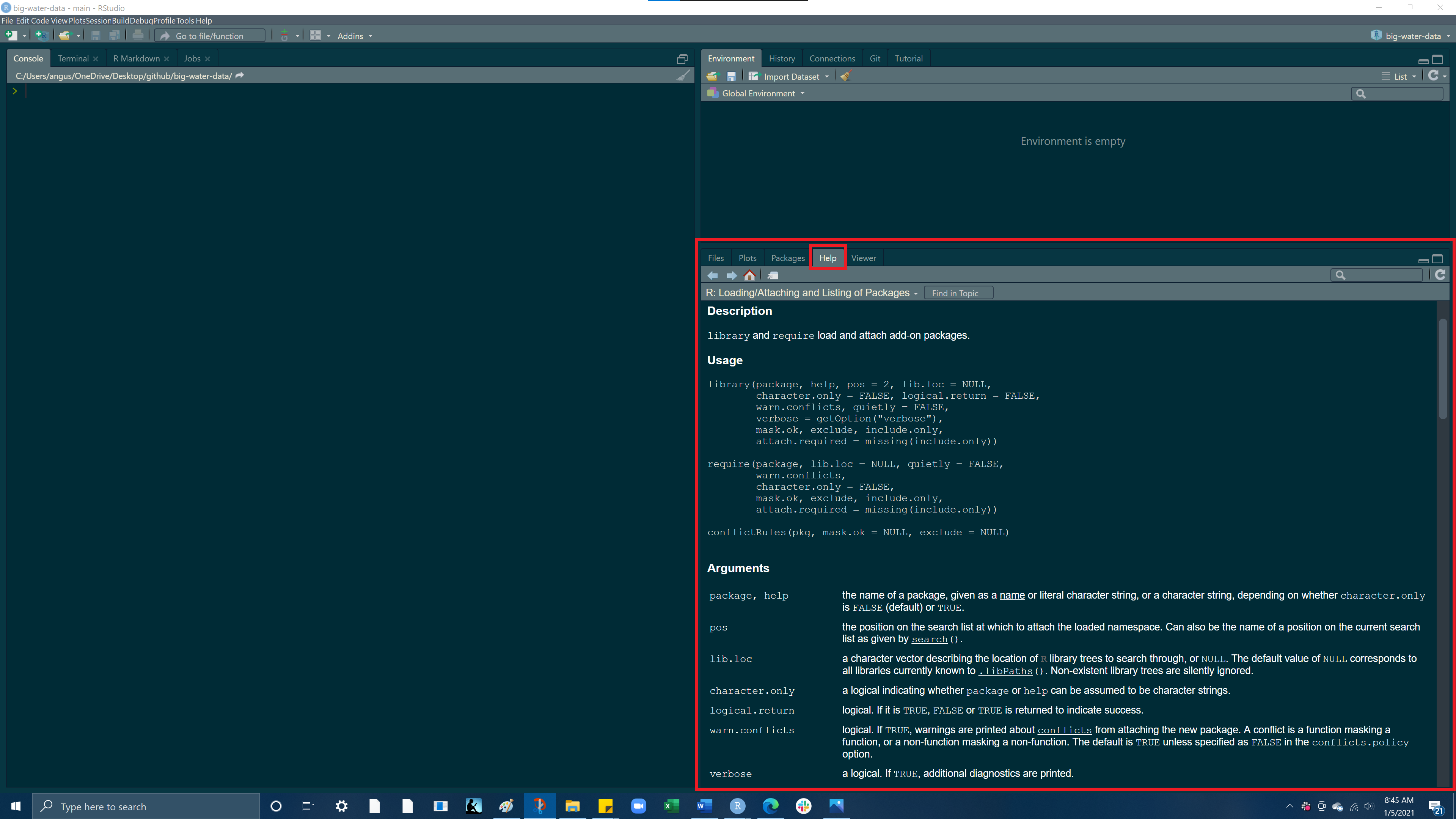
library(tidyverse)  
library(here)

## **help() function**

The help() function is a tool for understanding what a function does, what inputs it takes, and what outputs it gives. When you call the help function, the help page for the function you called will appear in the bottom right pane, a tab to the right of the Files tab.

The Help tab is great. At the bottom page you will find code examples for that function, you can run these yourself to get an idea of how to use a function! **As you go through this lab, use the help() function to get more information on the functions we will be using.** You can access the help for any function; for example, let’s use the help function to get information on the library function.

help(library) # calling help for the library() function



# **Read in a .csv file using here()**

* Click here to download a .csv file of water data ADD LINK TO CSV DATA
* Once the file downloads, save it into the data folder in your project directory.
* Now lets read that .csv file into our Rscript using the read\_csv() function and the here() function

- here() allows you to locate files relative to your project root. It finds files based on the current working directory at the time when the package is loaded. This means that if someone else would like to run your code, the file path will adjust to that persons current working directory, giving them a file path to the data. Using here() is good practice as it allows for a reproducible file path for whoever is using the code. ***Reproducible code and work flows allows you to cut down on extra coding in the future and saves time in the long run for you and anyone else who might want to use your code.***

My current working directory when I loaded the here package

here()

## [1] "C:/Users/angus/OneDrive/Desktop/github/big-water-data"

So to access the export.csv file in our data folder we need to get to **“C:/Users/angus/OneDrive/Desktop/github/big-water-data/data/export.csv”** here() will give us an easy way of finding this path

here("data/export.csv")

## [1] "C:/Users/angus/OneDrive/Desktop/github/big-water-data/data/export.csv"

read\_csv(here("data/export.csv"))

## # A tibble: 1 x 14  
## X1 `Public Supply` Domestic Industrial Irrigation `Irrigation-Cro~  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 1 49.9 0.52 6.36 56.0 53.0  
## # ... with 8 more variables: `Irrigation-Golf` <dbl>, Livestock <dbl>,  
## # Aquaculture <dbl>, Mining <dbl>, Thermo <dbl>, `Thermo: Once  
## # Through` <dbl>, `Thermo: Recirc` <dbl>, Total <dbl>

Run this code and you’ll notice the dataframe outputs to our console. How do we save this dataframe to a variable so we can conveniently access again.

# **Variables**

Lets create a variable named water\_df and assign it to our .csv file!

water\_df = read\_csv(here("data/export.csv"))

If you look at your environment (top right pane) you should see a variable called water\_df which has 1 observation of 14 different variables!

Now run water\_df in your Rscript or your console you should see this:

water\_df

## # A tibble: 1 x 14  
## X1 `Public Supply` Domestic Industrial Irrigation `Irrigation-Cro~  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 1 49.9 0.52 6.36 56.0 53.0  
## # ... with 8 more variables: `Irrigation-Golf` <dbl>, Livestock <dbl>,  
## # Aquaculture <dbl>, Mining <dbl>, Thermo <dbl>, `Thermo: Once  
## # Through` <dbl>, `Thermo: Recirc` <dbl>, Total <dbl>

Before getting into the data within our data frame lets just quickly look at different data structures within R

# **Data Structures**

## **Vectors**

* Vectors can be created using the c() (combine) function
* The length can be checked with length()

### **Character vector**

Let’s use c() to make a **character vector** of different water demand sectors

water\_demands = c("Public Supply", "Domestic", "Industrial")  
water\_demands

## [1] "Public Supply" "Domestic" "Industrial"

### **Numeric vector**

Now let’s use c() to make a **numeric vector** detailing the volume of water use for each of these sectors

water\_use = c(49.94, 0.52, 6.36)  
water\_use

## [1] 49.94 0.52 6.36

### **Adding elements to a vector**

If we wanted to add **Irrigation** to our water\_demands vector, we would need to surround **Irrigation** in quotes to create a character string, as to match the data type of the other elements in the water\_demands vector.

water\_demands = c(water\_demands, "Irrigation")  
water\_demands

## [1] "Public Supply" "Domestic" "Industrial" "Irrigation"

To add the volume of water use by **Irrigation** we can do the same thing but because this is a numeric vector we do not need to use quotation marks

water\_use = c(water\_use, 55.95)  
water\_use

## [1] 49.94 0.52 6.36 55.95

### **Subsetting a vector**

To subset a vector we use [], this is known as indexing. Each element in a vector has an index based on its order in the vector.

If we want to retrieve only the 3rd element we would put a 3 in the brackets to “index the 3rd element”

water\_demands[3]

## [1] "Industrial"

If you want to **remove an element from a list,** while keeping all the other elements, you can use the a **negative**. The following removes just the 3rd element from the list and returns a vector of the remaining elements

water\_demands[-3]

## [1] "Public Supply" "Domestic" "Irrigation"

If you want to get a range of elements from a vector use the :. The structure of this method is vector[startingElement:endingElement]

water\_demands[1:3]

## [1] "Public Supply" "Domestic" "Industrial"

## **Lists**

An ordered collection of objects (components). A list allows you to gather a variety of (possibly unrelated) objects under one name. Extends vectors and allows elements to be any type - the list() function lets you create a list - Let’s input the values we used to create our water\_demands and water\_use vectors into a list

list = list(  
 c("Public Supply", "Domestic", "Industrial"),   
 c(49.94, 0.52, 6.36))

But because we have already created vectors containing these values, a more efficient way of making this same list would be to call our vectors directly into list()

list2 = list(  
 water\_demands,   
 water\_use)

Use the class() function on our list2 variable, to check what type of data structure list2 is

class(list)

## [1] "list"

### **Subsetting a list**

The content of elements of a list can be retrieved by using double square brackets []

Here if we call list2[[1]] we will be returning the first list element, water\_demands

list2[[1]]

## [1] "Public Supply" "Domestic" "Industrial" "Irrigation"

Then if we want to get a certain element from within our first list element water\_demands we can use the same indexing we used with vectorsSo within our first list element we want the object in index [1]

list2[[1]][1]

## [1] "Public Supply"

We can do the same thing with the second list element, water\_use

list2[[2]]

## [1] 49.94 0.52 6.36 55.95

list2[[2]][2]

## [1] 0.52

If we wanted to **name the two elements of our list we can use the names() function.** To use the names() function you have to supply a vector with your desired names and this vector must have the same number of elements as the length of your list.To check our lists length we can use the length() function

length(list2)

## [1] 2

So we see our list has a length of 2, therefore if we want to assign names to our list we need to supply a vector with 2 names.

names(list2) = c('sector', 'water\_volume')  
list2

## $sector  
## [1] "Public Supply" "Domestic" "Industrial" "Irrigation"   
##   
## $water\_volume  
## [1] 49.94 0.52 6.36 55.95

Now we have a list with 2 elements named sector and water\_volume respectively. **What if we wanted to match the element by index?** That is we want the element in index 1 of the first vector to match with the element in index 1 of the second vector. The function bind\_rows() allows us to take the 2 vector sequences in list2 and bind the rows together, so index 1 in water\_demands becomes row 1, column1 and index 1 in water\_use becomes row 2, column1, thus matching.

binded\_row\_list = bind\_rows(list2)

If you assign bind\_rows(list2) to a new variable binded\_row\_list and hit run. You will notice we have a new object in our environment! Use the class() function on your binded\_row\_list object and what do you get?

## **Dataframes**

* a named list of vectors.
* data.frames are one of the biggest and most important ideas in R, and one of the things that make R different from other programming languages
* Unlike a regular list, in a data.frame, the length of each vector must be the same.
* typically dataframes are read into R via read\_csv, read\_table, or read\_xls, depending on the file type
* dataframes can also be built using the data.frame()function

### **Useful dataframe function**

* head() - shows first 6 rows
* tail() - shows last 6 rows
* dim() - returns the dimensions of data frame (i.e. number of rows and number of columns)
* nrow() - number of rows
* ncol() - number of columns
* str() - structure of data frame - name, type and preview of data in each column
* names() or colnames() - both show the names attribute for a data frame

***Lets now inspect different qualities of this dataframe***

### **Structure of a dataframe**

If we use the class() function on our dataframe we see that it is a tbl\_df, a **tibble** is very similar to a dataframe, but it also displays the type of data underneath each column heading name

class(water\_df)

## [1] "spec\_tbl\_df" "tbl\_df" "tbl" "data.frame"

And here is our dataframe. The data frame is 1 observation (rows) by 14 variables (columns)Notice the num classification next to all of the columns, this is indicating the type of data in each column

str(water\_df)

## tibble [1 x 14] (S3: spec\_tbl\_df/tbl\_df/tbl/data.frame)  
## $ X1 : num 1  
## $ Public Supply : num 49.9  
## $ Domestic : num 0.52  
## $ Industrial : num 6.36  
## $ Irrigation : num 56  
## $ Irrigation-Crops : num 53  
## $ Irrigation-Golf : num 3  
## $ Livestock : num 0.88  
## $ Aquaculture : num 0  
## $ Mining : num 0.04  
## $ Thermo : num 0  
## $ Thermo: Once Through: num 0  
## $ Thermo: Recirc : num 0  
## $ Total : num 114  
## - attr(\*, "spec")=  
## .. cols(  
## .. X1 = col\_double(),  
## .. `Public Supply` = col\_double(),  
## .. Domestic = col\_double(),  
## .. Industrial = col\_double(),  
## .. Irrigation = col\_double(),  
## .. `Irrigation-Crops` = col\_double(),  
## .. `Irrigation-Golf` = col\_double(),  
## .. Livestock = col\_double(),  
## .. Aquaculture = col\_double(),  
## .. Mining = col\_double(),  
## .. Thermo = col\_double(),  
## .. `Thermo: Once Through` = col\_double(),  
## .. `Thermo: Recirc` = col\_double(),  
## .. Total = col\_double()  
## .. )

### **Subsetting a dataframe**

* We can subset our dataframe and look at specific rows or columns by using [] or $
* To subset with brackets the structure is dataframe[row, column]
* To subset using the $ the structure is dataframe$column

So if we call water\_df[1,2] we will get the first row and second column of our dataframe

water\_df[1,2] # row 1, column 2

## # A tibble: 1 x 1  
## `Public Supply`  
## <dbl>  
## 1 49.9

If we call just water\_df[1,] we will get the entire first row with all the columns of our dataframe

water\_df[1,] # row 1, column 2

## # A tibble: 1 x 14  
## X1 `Public Supply` Domestic Industrial Irrigation `Irrigation-Cro~  
## <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 1 49.9 0.52 6.36 56.0 53.0  
## # ... with 8 more variables: `Irrigation-Golf` <dbl>, Livestock <dbl>,  
## # Aquaculture <dbl>, Mining <dbl>, Thermo <dbl>, `Thermo: Once  
## # Through` <dbl>, `Thermo: Recirc` <dbl>, Total <dbl>

You can use : to subset a range of columns, the structure to do this is dataframe[column1:column(n)]

water\_df[1:4]

## # A tibble: 1 x 4  
## X1 `Public Supply` Domestic Industrial  
## <dbl> <dbl> <dbl> <dbl>  
## 1 1 49.9 0.52 6.36

Let’s subset the first row and the second column using the $

water\_df$"Public Supply"[1]

## [1] 49.94

And then the fourth column and all the rows (we only have 1 row in our dataframe so we will only get the first row)

water\_df$Industrial

## [1] 6.36

You’ll notice when we wanted to subset the first row and second column, water\_df$"Public Supply"[1] we had to use quotations around the column name, but we didnt need to do this in the following example, water\_df$Industrial. This is because the “Public Supply” has column has a space between the two words.

**Try running the same code without the quotations and see what happens.**

**Did you receive an error?**

The reason for this is because the **“Public Supply”** column has a space between the two words, this space needs to be acknowledged by R, this can be accomplished by adding the quotations, indicating everything in the quotes is the column name. We didn’t need to do this with the water\_df$Industrial call as the column **“Industrial”** is all one word. This is one reason why it is important to be careful when naming data frame. Typically it is best to avoid spaces and instead it utlize the \_ character in place of a space.

# **Data types**

Values in R can be one of 6 different types :

1. **numeric** (e.g. 2, 2.15) - values w/ decimals2. **integer** (e.g. 2L) - values w/o decimals (create w/ uppercase ‘L’)3. **character** (e.g. “x”, “Welcome!”)4. **logical** (e.g. TRUE, FALSE)5. **raw** (e.g. holds bytes)6. **complex** (e.g. 1+4i) - *we are going to ignore*

For this class we will be focused on **numeric** and **character** data types

The class() function tells us what kind of object is it (high-level)The typeof() function can tell us the object’s data type (low-level)

## **Numerics**

***Let’s look at what data types are in our water\_df dataframe we created***

THe data type for the values in the Public Supply column

class(water\_df$"Public Supply")

## [1] "numeric"

And for the Industrial column

class(water\_df$Industrial)

## [1] "numeric"

All of our columns contain numeric data types but you will encounter situations where you may have a column with character data types that you wish to be numeric, and vice versa.

**The following example shows why this is important to know**

Lets say we had country information on domestic and agricultural water use in some objects:

country = "Mexico"  
domestic\_water\_use = 146  
agricultural\_water\_use = 74

If we wanted to we could add domestic\_water\_use and agricultural\_water\_use and get an answer

domestic\_water\_use + agricultural\_water\_use

## [1] 220

But what if we try and add domestic\_water\_use and country?

domestic\_water\_use + country

## Error in domestic\_water\_use + country: non-numeric argument to binary operator

This non-numeric argument error is saying that one of these objects is not a **numeric value** (country is a character value)

For whatever reasons, you will encounter data that has discrepencies like this and thus why it is good to look at the structure of whatever dataframe you’re working with.

**What’s the difference between 2012 (the number) and '2012' (the character)?**

class(2012) # numeric

## [1] "numeric"

class('2012') # character

## [1] "character"

When you surround a something in quotes (double or single), that element is coerced to the class of character, as shown above as the numeric 3 becomes a character once wrapped in quotes