Name: ___Jayla Worsley_____

Date: _____2/11/21_____

SPEND SOME TIME WITH R AND LEARN HOW IT 'THINKS'

Purpose:

1. **Learn how R *thinks***
2. **Begin to see how we can use R to ask questions about our data**

Overview:

To do this homework

a. Start R and create a new Project titled the Quant_stats_HW project-[your name]

   i. **Just to be clear when I say [your name] I mean to put *your own name* there. Like I would call it "Quant_stats_HW_Marc_Kissel." If I had a dollar for every time a student called a file "HW-[your name]" I could buy a fancy dinner.**

b. Open a new R script & title it [Firstname]_[Lastname]_HW1

c. Follow along with the info below. when it is time to do the coding simply type in the ***Source*** and then run the command. This way you have an easy way to keep track of what you did

d. To help you, I attached a draft of this homework file with the first few filled in. You can put it in your project folder to start this. But be sure to rename it!!! Remember you can use '#' to make comments

e. You can hand in this work in a few ways (I'm not sure the best way and happy to help if you are having trouble with this).9+3-52<,

   i. Copy and past your answers into this Word document and post it to the assignments section

   ii. Post your R code to the assignments section

   iii. Write your answers by hand and scan it into the assignments section

Overview on R:

While I think computer scientists would disagree with this claim, the way I think about R is as follows:

We create objects and then do things to these objects with functions that help us to understand the object we created.

So what do I mean by this? Well, we are often going to be taking data from outside of R and getting it into R. When we have it in R it is stored as an ***object***. So for example the following codes take a dataset of Christmas songs from online and stores it as an object (which I called ***christmas_songs***). I then use another function to plot the relationship between the song's peak position and the total weeks it was on the chart. Finally, we add a best fit line. Feel free to try it out in R! Note that this is a much more complex code that we normally will use, but I wanted you to have something that would work right away. Also remember that the '#' is a way to make a comment without having R read it, so I use that often to make notes etc

```
christmas_songs <- read.csv("https://raw.githubusercontent.com/rfordatascience/tidytuesday/master/data/2019/2019-12-24/christmas_songs.csv")


mean(christmas_songs$peak_position) # here, we use the mean function to get the mean peak position of the songs on the list


plot(x=christmas_songs$peak_position, y=christmas_songs$weeks_on_chart) #here, we use the built in plot function to make a scatter plot

abline(lm(christmas_songs$weeks_on_chart ~ christmas_songs$peak_position), col = "blue") # this abline function adds a regression line to out plot and colors it blue. How can we change the color?
```

Questions:

#1: introduction

The first thing we want to do is to learn how to run something in R. Take a look at the code below:

```
6 + 7
```

```
## [1] 13
```

the part in grey is the code that is entered into R in the ***Source*** panel's script. I think the best way to lean R is to retype the code so that you begin to get a sort of muscle memory for the functions.

a. How can you send a command from the *Source* pane to the *Console* Pane? If you are having trouble with this look for the 'run' button or see the *Installing R* handout from week one. (fun R fact: ***Source*** is actually the function!)
b. What would be the code to add 9 and 1? **9 + 1**
c. What would be the code to add 365 and 1986 **2351**

#2 - Math

Now let's learn how to do basic math. Using whatever sources you can think of, figure out how to add, subtract, divide, multiply, take an exponent (i.e. square a number), get the square root.

solve the following programmatically using R in your script. Be sure to save as you go.

a. 3-4 **-1**
b. 7 divided by 10 **0.7**
c. 6 times 89 **534**
d. 8 raised to the 7th power **2097152**
e. the square root of 52 **7.211103**

#3 - Assigning variables:

One of the key things you will do is assign a value to an object/variable. A variable is simply a label for information. I like to think of it as a name tag in a way.

to do this we use this symbol <-.

```
x <- 42
```

read the above like: "assign the value of 42 to the object x" or "x gets the value of 42" or "the variable X is going to represent the value 42."

There are some rules that

solve the following programmatically using R

a. How would you create a name ***y*** and assign it the value 334 **y <- 334**
b. how would you add the *x* and *y* together to get the sum of the two numbers?
   x + y
c. how would you create a new variable (*z*) that stores the result of x +y
   z <- x + y
d. now, change the value of *x* to be 500 and see what happens when you add *x* + *y* together? if you ask for the value of *z* now what do you get? why?
   - **Z ; and the value is 376**
e. what are the rules for what the name of an object in R can be? This can be found by searching online or using the R help function
   Names in R must start with a letter or a dot, they should contain only letters, numbers, underscore characters (_) and dots (.)

#4. types of data

When you assign a value to a variable/object, it is given a specific ***class***. the class is **VERY** important and is probably the #1 reason for having troubles with R. It is always good to check the class of an object. To do this, we can use a specific ***function*** in R called class().

Basic class types

1. integers - 'natural numbers': 5, 6, 987
2. numeric - decimal numbers: 4.5, 8.76666
3. characters - text (sometimes called strings): "hello", "goodbye"
4. logical - Boolean: True or False

5. factors - categorical data. This is different from characters because factors are given numbers (or *levels*) that are associated with that factor and then used for analysis…..we will come back to this later for part 6.

```
x <- 42
class(x)

## [1] "numeric"
```

Note that the class of X here is numeric rather than an integer (which might not be want you expected). why? well, it has to do with some inside stuff on computer languages but in general R is going to store everything as numeric unless you tell it differently. In order to make it be an integer we need to use a trick, which is to add the suffix L to the number

```
xx <- 42L
is.integer(xx)

## [1] TRUE
```

a. Assign *my_value* to be "hello". then check its class
   It's class is False/True or Logical

Below are some examples…copy the code into R and play around with this until you get a feel for the different classes

```
my_value <- 1+ 3
is.numeric(my_value)

## [1] TRUE

my_name <- "Marc Kissel"
is.numeric(my_nameis)

## [1] FALSE

is.character(my_name)

## [1] TRUE #and how!
```

below is some R code. guess what each type of object will be and then use R to find the answer

```
a <- 1.333- numeric
b <- TRUE – logical
```

```
c <-  "my name is" – character-text
d <- Sys.Date() # tricky  -factor
```

A class defines what kinds of operations can be implemented on an object & how a function will return a value (for example, it wouldn't make sense to get the mean of something that is stored as a character). It is important to keep track of the classes of your objects. Class mistakes are probably the most common kind of problem in R

Part 5: Vectors

So far we have only stored one value into an object. but most of the time we are going to have to work with a lot of data. Say you have a series of numbers and want to add 7 to each of them. It would be a pain to have to do that manually. We can use R to store a series of values (called a *vector*) by using the *c* function:

```
my_vector <- c(1,2,3,4,5)
my_new_vector <- c(6,7,8,9,10)
```

let's say you are doing some research and want to record the biological sex of the skeletons in your study. After analyzing them you decide that this is the correct designation

male male female male female female female

a. make a new object called **my_study** and make it a vector of the recorded biological sex *** < - this is harder than it might look.

Congrats! you now have a *vector*!

Let's say you want to figure out how what the sex of the 5th skeleton is. you could print the object and count, but that takes time and an get difficult. R makes things easier for people like me who are lazy and want the computer to do it all.

to get an *element* from a list we use square brackets

```
my_study[2]

## ["male"]
```

a. how would you get the sex of the third skeleton?
my_study[3]

c. can you figure out how, in one line of code, to get the sex of the 1st and 4th skeleton?
Yes, you type my_study[1];my_study[4]

Ok, but it is kind of confusing.. You know that they are skeletons 1-7, but maybe someone else doesn't. We can assign names to objects using a special function called *names*

```
names(my_study) <- c("one", "two", "three", "four", "five", "six", "seven")
```

d. print my_study now and see how it differs.

Note: now that they have names we can also get the values that way

```
my_study[2]

## two
##   "male"

my_study["two"]

## two
##  "male"

#these are the same

my_study[2] == my_study["two"] # the '==' asks R to tell you if the value on
the left and the value on the right are the same....

##  two
## TRUE
```

to be fair, most of the time there are easier ways to name things, but having a basic understanding of how R works helps a lot

Part 6: matrices

a vector is simply a list of numbers a matrix is a **rectangular** array of numbers

```
cx1980 <- c(7, 13, 8, 13, 5, 35, 9)
cx1988 <- c(9, 11, 15, 8, 9, 38, 0)
chimp <- cbind(cx1980, cx1988) # cbind binds the vectors together a columns
class(chimp)

## [1] "matrix"

chimp

##      cx1980 cx1988
## [1,]      7      9
## [2,]     13     11
## [3,]      8     15
## [4,]     13      8
## [5,]      5      9
## [6,]     35     38
## [7,]      9      0
```

one thing you want to learn is how to read a matrix and identify elements. Let's say you want to get a vector of just the first observation. Or you want the data from 1980. we can use the square brackets again but need to know a trick.

```
chimp[1,] #note the comma

## cx1980 cx1988
##      7      9

chimp[,1]

## [1]  7 13  8 13  5 35  9

#putting the number before the comma gets us the row. putting it after the co
mma gets column.
#one way to remember that Rows come first is the mnemonic Railway Cars
chimp[3,2] # third row, second column

## cx1988
##     15
```

a. how would you make a matrix by row rather than column (use google if need be)

putting the number before the comma or byrow

b. There are other ways to make a matrix. Look at the code below and figure out how it works

```
freq <- c(32,11,10,3,  38,50,25,15,  10,10,7,7,  3,30,5,8)
hair <- c("Black", "Brown", "Red", "Blond")
eyes <- c("Brown", "Blue", "Hazel", "Green")
freqmat <- matrix(freq, nr=4, nc=4, byrow=TRUE)
dimnames(freqmat)[[1]] <- hair
dimnames(freqmat)[[2]] <- eyes
freqmat

##       Brown Blue Hazel Green
## Black    32   11    10     3
## Brown    38   50    25    15
## Red      10   10     7     7
## Blond     3   30     5     8
```

c. now, create your own matrix with made up data…give the code you used to make this

Black White Blonde Brown

Boxer 32 11 10 3

Pitbull 38 50 25 15

German Shepard 10 10 7 7

Poodle 3 30 5 8

Part 7: data frame

a dataframe stores data! it can hold different kinds of classes so it is different from a matrix. You can think of it as a list of variables that are all the same length. In other words, it is shaped like a rectangle. You can't have one column that has 4 entries and another that has 5.

Data frames are probably the most common way we will work with R

```
bone <- c("humerus", "radius", "ulna", "femur", "tibia", "fibula")
size_inches <- c(14.4, 10.4, 11.1, 19.9, 16.9, 15.9)
injury <- sample(c("yes","no"),6,replace=TRUE)
sample_letter <- LETTERS[1:6]
my_sample <- data.frame(bone, size_inches, injury, sample_letter)
my_sample
```

```
##      bone size_inches injury sample_letter
## 1 humerus        14.4    yes             A
## 2  radius        10.4     no             B
## 3    ulna        11.1     no             C
## 4   femur        19.9     no             D
## 5   tibia        16.9    yes             E
## 6  fibula        15.9    yes             F
```

If you use the function View you can see a spreadsheet of the data frame you just made

```
View(my_sample)
```

you can view a specific column/vector using the $

```
my_sample$bone
```

```
## [1] humerus radius  ulna    femur   tibia   fibula
## Levels: femur fibula humerus radius tibia ulna
```

a. run the code below. it should show an error on step 4. Why? rewrite the code so it works!

b. There was an unexpected constant that shouldn't be there so once it's removed than the code works

```
num <- c(1,2,3,4,5)

food <- c("bread", "butter", "milk", "cheese","coffee", "tea")

quantity <-  c(1,1,3,5,7,1)

shopping <- data.frame(num, food, quantity)
```

c. what is the class type of the different vectors in the my_sample dataframe?

Class type is characters and numerical

Part 8: comparing values

often times we are going to want to compare things.

run the code below

```
a <- 5
b <- 9
c <- 7
d <- sqrt(49)
```

Now, figure out how to have R evaluate the following:

solve the following programmatically using R in your script. Be sure to save as you go.

a. is *a* bigger than *b*? **False**
b. is *c* equal to *d* (careful with this one..) **True**
c. is *c* less than or equal to *b*? **True**

d. make a new vector called *temp* with the values of 1,5,7,9,11,14,6,8. then write a single line of code that evaluates if 3 is greater than each of the values in the vector

e. how would you ask R if the 5th value in *temp* is larger than 5?

Temp[c(5)] > 5

Part 9: getting data into R. (Note this doesn't have any thing you need to do in R for the homework)

to be honest, while building a datatable from scratch is totes possible, for the most part you are going to be ***reading in*** data from outside. In the past this was not too easily and was the source of many headaches. Nowadays it is still a pain but there are a lot of functions that help you do this.

There are a few ways to do this. Here we will work through 3 different ways. I prefer to third but honestly it usually doesn't matter.

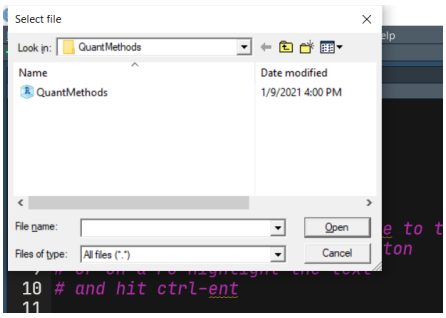
First way: using file.choose()

This lets you interact with the computer in a more 'easy' way then hardcoding. However, the drawback is that we lose some reproducibility

1. type the code into the Source and then run it

```
my_data <- read.csv(file.choose(), header=T)
```

2. once you run it you should see a small window pop up that looks something like this:



3. that's the ***file chooser*** window. You can simply select the file you want and then R will read it in
4. let's take a second and look what is going on here with this line of code.
    a. The ***Function*** read.csv is used to read in files that are *comma-separated values* (we will come back to this later). Remember that a function is simply a collection of commands that do something for us. In this case, it lets us read that data into R. It takes a few

***arguments*,** which is the stuff that goes between the parentheses and is what the function works on. In this case we are using a second function called file.choose which lets us interact with a mouse click to choose a file. We are also saying *header = TRUE*¸which tells R that our file's first row has names of the columns.

b. We are then using the assignment arrow to give this value to a new ***object*** called my_data

5. Now, see what happens if you run my_data in the console
6. You can also use the View() function to see the data in a spreadsheet form (note that unlike many other functions, this one is capitalized.

```
View(my_data)
```

Second way: using the "Import Dataset"

This method uses the icon in the Environment tab to import a dataset. It gives a lot more functionality than the file.choose and while it lets you interact with a mouse click it also reproduces the code used so you have a record. To be honest I tend to forget this exists but it is really useful!

Third way: using read_csv or read.csv and giving the location of the file name

This is probably the hardest of the 3. It pretty much is the same as number 2 but rather that using the file import feature you just write the code needed. I am not sure if there is any other difference. It does make it slightly easier to read data off the web, so we may be using this in the future to get data into R that lives online

Part 10: how did you do

1. What the most challenging part of this homework? **Typing the code instructions for vectors was very hard I messed up multiple times but eventually I figured it out**

2. What could be more clear? What info might be helpful to include? **Some of the instructions could be a little clearer, since my knowledge of coding is very limited it's very easy for me to get confused after I make a simple mistake.**
3. After sitting with this, do you think you have a better idea of what R is all about? **Absolutely**
4. Probably the most awful thing about R are the *classes*. If you had to explain what a *class* was how would you do that? **A category that groups like materials together**
5. Now that we have an idea of how R thinks, next week we are gonna talk a bit about how to store data. Based on what you now know, what might be good practices for recording data on a spreadsheet so that others can use them? **The last method since it's easier to reproduce what you did in R**