Interactive Assignment 1: Introduction to R and R Notebooks

Learning objectives

In this lab, you will review how to do the following: 1. How to set up an R Notebook and how to organize R Notebooks on your computer 2. How to install packages 3. How to enter R code in **chunks** 4. How to enter notes to yourself in **annotations** 5. How to do basic R commands 6. How to load data into R using the `read.csv()` command 7. How to use the `summary()` and `aggregate()` command to calculate basic statistics about a dataframe

Introduction

This class is the second stats class and builds on Stats 1. However, the first few labs are some review to get you back to speed in doing data analysis. I understand that you may not spend your summer break analyzing data. Surprising, right?

The first thing we will do or review is R notebooks and reproducible data analysis. As touched on in class, one of the biggest issues in data analysis is reproducibility. It is one thing if a researcher has trouble reproducing another researcher's results. But what if a person can't reproduce their own results?

In grad school, I would do analyses and get results, and then have no idea how I could get those same results again. Or I had no way to know how I got from my raw data to the finalized data after cleaning and editing it. I thought I was following the same steps, and doing the same things, but I didn't know what I was doing.

Another issue is that I could have been making mistakes and not knowing it. I made several mistakes that I caught, but what if there were others? I took good notes, but what if I clicked the wrong button and didn't know it.

One of the strenghts of R is that you can create a text-based log of all your analyses through R Notebooks. RStudio makes this easy, allowing you to combine your R commands and your notes for your commands into a single document that you can export.

This assignment

For this assignment, I want you to create a basic R notebook, place some code in chunks and some notes in annotations, practice loading data into R and then to knit the notebook.

Organizing Assignments and Creating an R notebook

The way I suggest you organize your assignments is to create a folder for this class on your computer, and then create a subfolder inside that folder that holds each assignment. R notebooks create additional files when they are knitted and the folder for each assignment will hold these folders in an organized place. In addition, you can place any data files required for each assignment in that folder.

Go ahead and create a folder for this class if you have not already done so and in that folder, create a subfolder that is named 'InteractiveAssignment1'.

Now we will create an R notebook. Go ahead and open R Studio. Open a new notebook file by clicking File -> New File -> New Notebook. When you do this, RStudio creates a template file with a lot of information there. Delete everything except the first few lines with the front matter. This is the portion that is between the triple dashes '—'.

Now, save this new notebook on your computer in the folder you created for this assignment by clicking File -> Save As.

In the new notebook that opens, change the title in the front space to a title which is appropriate for this assignment

Installing Packages

The first chunk I have in any R Notebook is where I load packages. A package (or library) is an additional add-on to R which allows R to have more commands. We will use the `tidyverse` package in other assignments. However, we will go over how to load package in this assignment.

Before you use a library, you must install it first. You can do this by clicking the packages tab and then choosing "install". Then this will pop up a dialog box where you can install the package. Type the name "tidyverse" in the box that appears and then choose install. There will be a lot of message that appear in your console. If there are any messages that ask you to make any choices, just follow the prompts and accept them.

You only have to install a package once. However you will have to load the packages you need each time you do an R notebook.

Before entering a code chunk, create an annotation. This is just typing text that says what you are doing. You should type something like "loading packages". All the text that is not in chunks will not be run and only exists as notes to yourself.

Now go ahead and create a code chunk if you have not done so. Make a code chunk by going to Code -> Insert Chunk (or pressing "Control-Alt-I" (Windows) or "Command-Option-I" on Mac). In that code chunk, type `library(tidyverse)` just like indicated below.

```
library(tidyverse)
```

```
## Warning: package 'purrr' was built under R version 3.6.2
```

Once you type the code chunk, you can run the chunk by pushing the tiny green play arrow in the top-right corner of the chunk. This will give some messages. If you run a code chunk, you will either get an output, an error, or nothing.

R Commands in Chunks

Now we will go over doing some basic R commands. When I give you the R commands that are highlighted below, please place them in the chunks and run them. Answer the questions by typing annotations in between the code chunks. You should always annotate your code so that you know what each chunk is doing.

Type the following command as a code chunk. What do you get when you run it?

```
1
```

```
3+4
```

```
x
```

For the first two commands, you should get an output. If you input a number, you get that number as an output. The last command should give you an error. That's because R doesn't know what `x` means. We'll fix that in a minute.

Go ahead and delete the code chunk that gives you an error. When we knit the notebook, any code chunks that give errors will cause the knitting to fail.

When you see output, you may see a number in brackets, like if you type the output below:

```
3*5
```

```
## [1] 15
```

For now, ignore the number in brackets. This is a way of telling which element is output. The number in brackets helps us if we have a long list of numbers, like if I type the following which lists the numbers 1 through 100.

```
1:100
```

```
##   [1]   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17
##  [18]  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34
##  [35]  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51
##  [52]  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68
##  [69]  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85
##  [86]  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
```

The number in brackets tells me which value in the list is to the right of the number in brackets. So when I see, [18], I know that the number to the right is the 18th item in the list.

Doing basic math in R

You can use R like a calculator, using math notation. To add, use the plus key. To subtract, use the minus key. To multiply, use the asterisk key, and to divide, use the forward slash key. Exponents are used by the caret key. See below:

```
3+7
4-11
4*21
18/3
2^2
```

Go ahead and type code chunks to do the following:

Now in the space below, create chunks to how you would do the following operations:

9 multipled by 14

32 divided by 3

11 minus 4

Finally, R follows the basic order of operations, with information in parentheses done first, then exponents, then multiplication and division, followed last by addition and subtraction. So the following are different. Try typing each of these as code chunks and seeing the output.

```
3 + 11*5
```

```
## [1] 58
```

```
(3+11)*5
```

```
## [1] 70
```

Objects

The power of R though is that you can do a lot more than that. You can use things called **objects** which are like containers for data. An object can be anything from a single number to thousands and thousands of rows of data. Depending on what an object is holding, it can have different types.

The simplest type of object is a variable. It is simply a letter or word that stands for a number. To make a variable you use the following (replacing the bracketed number with an actual number (like 42).

To assign a value to an object, we simply use equals, with the object on the left and the value to assign on the right.

```
x = 1
```

When we assign a value to an object, we don't get an output, since there is nothing for R to say. To see what an object says, we simply type the object, or look at the environment in the top-right window of the RStudio console.

```
x
```

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

The name of objects can be any text that starts with letters. This can contain letters with numbers or dots in the middle. Objects should not contain spaces. This can be done but is very complex. Objects also can hold different types of data, such as letters, which are called strings. See the examples below. You do not have to type these, but you should at least try to understand what they are doing:

```
number = 4
name = "Michael"
day.of.week = "Tuesday"
question20 = 5
```

We can also do math on objects, just like numbers. Go ahead and type the following.

```
x = 3
y = 7
x*y
```

```
## [1] 21
```

An important thing to note is that objects will get replaced if you reuse them. If I type the following, I am only using one object, and just replacing the value stored there:

```
t = 3
t = 5
t = 11
```

Do the following in R.

1. Create an object that is labeled as your first name and have that object equal your age. Write down what you would type.
2. Make a second object that is the same as the name of the person sitting beside you, and have it equal to their age. If you're not doing this assignment with another person, just pick another person you know and use their name and age.

3. Add the two objects together. What would you type?

The power in R is that objects can have many different values, called **lists**. To make a list, we can use something called a **function**. This is simply a command that tells R that we are making a list, or an object with more than one number. Every function in R has the following format:

```
name(inputs)
```

The function name is immediately followed by a left-parentheses, and in the parentheses is where you put the input or inputs for the function. The simplest function in R is the concatenate function, or c(). It joins all the **arguments** into one object, called a list. Here are a few examples. Type the following.

```
x = c(1,3,5,7)
ages = c(19,22,37,41,18)
IQ = c(97, 132, 88, 101, 94, 107)
presidents = c("George Washington","John Adams","Thomas Jefferson")
```

Once you type these, notice these objects are stored in the Environment tab of RStudio on the top right. You can click on these to view the objects.

With lists, we can do functions which need more than one number, such as taking the sum of a list, the mean, and so forth. Try the functions below, which give the mean, the sum, and the standard deviation:

```
mean(IQ)
```

```
## [1] 103.1667
```

```
sum(x)
```

```
## [1] 16
```

```
sd(ages)
```

```
## [1] 10.78425
```

4. Create an object named `scores` with the values 23, 47, 39, 36, 41, 32, and 41. Calculate the mean and standard deviation of `scores`.

Creating data frames

Variables are nice, but sometimes we want to hold a lot of organized data together, like a table. In R, these are called dataframes, and they have variables as separate columns with each row being an observation.

Now we are going to make a dataframe. For example, I may want to make a dataframe with how many hours they slept and their number of items they got correct two different 25 question memory tasks.

Name	Sex	Sleep	Memory Test 1	Memory Test 2
Zoe	Female	9	22	19
Yannick	Male	10	17	20
Xavier	Male	6	15	23
Wendy	Female	5	13	18
Vance	Male	5	10	8

To do this, I have to make a list for each column and then combine them using the data.frame() command into the dataframe called "results". To do this, type the following (note I keep all the variable names in lower case; this is my convention but make sure you are consistent with capitalization):

```
name = c("Zoe", "Yannick", "Xavier", "Wendy", "Vance")
sex = c("Female","Male","Male","Female","Male")
```

```
sleep = c(9,10,6,5,5)
memory1 = c(22,17,15,13,10)
memory2 = c(19,20,23,18,8)
results = data.frame(name,sex,sleep,memory1,memory2)
```

If you get an error here, it might be because you didn't type each of the lists correctly. For example, if you miss a value, then one of the lists will have 4 values, instead of 5. Then when you put the values into a dataframe, R won't know what to do.

To see the dataframe, I can click on the data frame in the Environment tab. Click on the little icon to the right of where it says `results` that looks like a table. This will bring up another window that shows the data.

If we want to look at a single column or variable in a data frame, we would select it by first listing the data frame name, then a dollar sign, then the column name. To select the column for names, I would type:

```
results$name
```

```
## [1] Zoe     Yannick Xavier  Wendy   Vance
## Levels: Vance Wendy Xavier Yannick Zoe
```

This is important because we can do math on an entire column at once.

We can do math to each of the columns. For instance, if I want to create a new column which is the sum of each of the three tests, I would type:

```
results$memory.total = results$memory1 + results$memory2
```

Now I have created a new column called memory.total. Notice that I used a period in the middle of the column name to separate the words. This is something you can do to avoid using spaces. However, the column name shouldn't start or end with a period.

5. What would you type to select the memory.total column? Add this as a chunk and make an annotation to answer the question.

6. How would you construct a column called memory.mean, which is the mean memory score for the two memory tests. Remember, you could just create this column by dividing the memory.total column by 2.

##Loading data into R

Seventy-five percent of the questions I get about R involve putting data into R and getting the data to load correctly. This lab covers this valuable skill. It will seem repetitive but this lab is very important, since every other lab will require you to load data from outside sources.

Normally, we will input data from other sources. I use a spreadsheet program to enter data and then import the completed data into R. As we covered in class, there is a very important way to enter data into R. A spreadsheet should be set up so that variables in columns and each observation as a row. The first row will contain the column names and each of the other rows will contain the data.

Getting Set Up

You should already have the R notebook saved on your computer in a folder. Go ahead and download the data for this lab from Canvas the data for this lab. This data should be available on Canvas as IA1.csv. Download the data onto your computer. When you download the file, make sure you select "Save File As" or "Save Target As" and save the file in the same folder that contains your R Notebook. Do not open this file after it downloads.

The most important part is to make sure your R notebook and your data files are in the same folder. If this does not happen, you will get an error that says:

```
Error in file(file, "rt") : cannot open the connection
```

When you see this error, that means you do not have the data in the same file folder, or you mistyped the name of the data frame.

Now we will create a code chunk which loads the data into R. The code chunk will use the `read.csv` command and it will save the data as a data frame named `d`. Typing `d` is shorter than `data`, but you can name your dataframe anything you want. Type the following and run the code chunk:

```
d = read.csv('IA1.csv')
```

Note that you have to type the file name exactly correct for R to load the data. If you misspell it, you will get the error that happens above. One common misspelling is to type cvs instead of csv.

If you loaded the data correctly, you should see an IA1 dataframe in your environment. Click on the icon on the far right to view that dataframe.

Whenever I give you data to enter into R, I will also give you a key that tells you about each of the variables. When you make your own data (for a research methods project) it is a good idea to make your own key for when you share it with other people. Here is the key below.

Variables: * Person: person's name * Education: highest level of education completed: none, HS * for high school, college for college, and graduate for graduate work * Sex: F for female, M for male * Test: type of test: Math or English * Score: score on a 50 question test

Whenever we create a new object, we may want to check what kind of object it is. We can use the `class()` command to do so. Type the following:

```
class(d)
```

```
## [1] "data.frame"
```

This tells us that IA1 is a data frame. So we know we input the data correctly.

If we look at IA1 in the environment, we can also see the number of variables and observations.

1. How many observations are in x? How many variables? Please include this information in your annotation in your R Notebook.

This step is important because I will usually tell you how many rows and columns should be in the data frame. If you have typos in your spreadsheet, this can cause errors. That is why it is important to make sure all your data is present and view your data.

You can also find out how many rows and columns are in x by using the `nrow(x)` and `ncol(x)` command. Another command to find out the names of the columns is the `colnames(x)` command.

2. Type a code chunk to list the column names. Are they the same as you expect based on the guide above?

An interesting thing about colnames is that you can use it to change the column names as well. Let's say we want to change the name of the Person column to subject. Go ahead and type the following.

```
colnames(d)[1] = "Subject"
```

This command changes the column name of the column specified with the brackets. Now look at your data by clicking on the dataframe `d` in the environment. Did the column name change?

4. Now let's change the name of column back from "Subject" to "Name". Write a code chunk to do this. Make sure you note as an annotation what question you are answering with your code chunk.

Selecting specific observations

Above, we described how to select variables. Sometimes in R, we would like to select specific observations or rows Sometimes, we want to do statistics only on a smaller subset of the data. In R, we have a command that can do that, called the subset command.

This is an example of what you would type, if your dataframe x had a variable named variable and you wanted to select all the values of x$variable that equaled value. `$y = subset(x, x$variable == value)$`

Try the following. In your annotation, write what part of the data frame is being selected here.

5a.

```
subset(d, d$Education == "College")
```

5b.

```
subset(d, d$Score > 20)
```

5c.

```
subset(d, d$Subject != "Alex")
```

The last example is very important to remember because occasionally we might want to remove an outlier. This is how we would do so.

For R, you can compare two values with a statement called a conditional. The conditionals in R are `==` (two equals signs, which tests whether two variables are equal or if each element of a variable is equal to a certain value), `>` and `<` (doing greater than or lesser than), and `!=` which tests not equal. When doing a conditional, R returns `TRUE` or `FALSE`.

Try typing the following. What happens?

```
d$Sex == "F"
```

Now use code chunks to answer the next two questions. Please include in your annotation which question you are answering.

6. Using the subset command, what command would you type to select only the high school education group?

7.What command would you select to select the English test?

8. Based on what you learned above, how would you remove Bob's scores?

Summarizing data

One easy way to look at the summary of a dataframe is to use the `summary()` command. This command gives the summary of an object, no matter what kind of object it is.

Go ahead and type the following:

```
summary(d)
```

This gives us a summary of each of the variables including the minimum, maximum, mean, and median. This can be helpful to view what is happening in a data frame and reporting basic statistics. Another way of doing this is by using the dataframe$variable notation and using a basic command like `mean()` or `sd()`

For example, to get the mean of Score, I would type:

```
mean(d$Score)
```

```
## [1] 26.94444
```

9. How would I get the standard deviation of score?

Using aggregate to get statistics separated by a grouping variable

Another really important way to analyze data is to compare groups. In the `d` dataframe, note that only one variable is actually containing numbers. All the other variables tell us about groups. For instance, the Person variable tells us what person's score it is. The Sex variable tells us the sex of the person.

We may want to look at summary statistics broken up by group. For instance, we may want to compare Math versus English scores. The aggregate command allows us to do so. The aggregate command is complicated and has several different parts. Here is the command that would give us the mean for Score, broken up by Test.

```
aggregate(Score~Test, d, FUN = mean)
```

```
##      Test    Score
## 1 English 28.00000
## 2    Math 25.88889
```

The aggregate command has three parts. The first part is a **formula** which is a way of breaking up the data. The second part is the dataframe we are looking at. The third part is the function we are applying.

The formula has two parts, divided by a tilde (~) which is the key to the left of the 1 key on your keyboard. The first part of the formula is the dependent variable. This is the numeric variable that we want to analyze. The second part is the grouping variable. This is the variable which we use to determine the groups. In this case, we use Score as the dependent variable and Test as the grouping variable.

If we wanted to compare females to males, we would change the grouping variable from Test to Sex. Type the following:

```
aggregate(Score~Sex, d, FUN = mean)
```

What results do you get?

If we want to change the type of function, we can change the `FUN = mean` part to some other command. If we really want to raise the fun, we might look at a median instead of a mean.

```
aggregate(Score~Sex, d, FUN = median)
```

Using this logic, create code chunks to do the following:

10. Get the mean score for each person. This will involve using Person as your grouping variable.

11. Get the standard deviation for each test (English versus Math).
12. Get the mean and median for Score broken up by Education level. Is this what you would predict?.

Knitting the R Notebook

Now you're done with Interactive Assignment 1. It's time to knit the notebook. Knitting the notebook produces a nice output of your data, including graphs and other analyses. In fact, this lab that you're reading is a knitted R notebook.

When you knit your R notebook, you can do so by clicking File -> Knit Document or choosing the tiny drop-down menu above this menu where it says "Preview Notebook" or "Knit Document". From this menu, you can choose what format you want to use to knit the notebook. I like this option better.

Go ahead and knit the notebook. You will see some text appear in your console. If you see an error, look at the line number where the error is. Then go to that line number and see what is causing the error. There is probably a typo in your code, or you accidentally deleted some of the punctuation around a code chunk. Sometimes you will have part of a code chunk left or put an annotation in a code chunk. Once you fix the error, try again.

Now you have finished, upload your assignment. If you can't get your document to knit, please let me know, or upload your .Rmd file, which is the text of your saved notebook. Then I can see why it's not knitting.