ST 351
Lab 1 Notes
Introduction to R, Sampling, and Simulation

Bring your laptop to lab this week!

Objectives of this lab

- Successfully install R and RStudio onto your computer or laptop
- Identify and explain the uses of the different window panes in RStudio
- Create vectors and run basic commands using the Script Window in RStudio
- Import a data set into RStudio
- How and when to use the $ notation in R
- Obtain a simple random using R
- Generate a distribution of sample means using simulation in R

If you have a laptop, bring it with you to lab this week. If you don't have a laptop or forget to bring it, no worries – R and R Studio are installed on the computers in the lab room. However, you may want to access R and R Studio outside of lab. Your TA will help with installing R and R Studio on your laptop during lab. If you do not bring a laptop to lab, you can follow the instructions below to install it yourself. If you do not have a device on which to install R and R Studio, talk to your TA about alternatives outside of lab.

Part I: Getting Started With R

Introduction to R and RStudio

R is a programming language that is popular in both academia and industry for doing statistical analysis and creating graphics from data. Knowing how to program in R (even just a little) is a valuable skill.

You might have some experience working with data in a spreadsheet application like Microsoft Excel, where you point and click to enter data and use buttons, menus, or formulas to perform tasks. In R there is no point-and-click interface. You write code to do all your desired calculations and to create graphics.

Don't worry if you don't have any programming experience. It might take a little practice to get used to R, but we will be using only very basic code, and we'll guide you through all of the code you need to write.

RStudio is a desktop application that makes it easy to use R and create R programs (usually called scripts). It's not strictly required for using R, but it is highly recommended, and all of the documents in this course will assume you are using RStudio. Both R and RStudio are completely free.

Objective: Install R and RStudio onto your device (desktop or laptop computer) before continuing.

- See the document in the "Start Here" module in Canvas called *Installing R and RStudio* for help installing both on your computer.
- If you have any trouble installing both R and RStudio, talk to a TA during lab or contact a TA or the instructor.

R Studio

At this time, open RStudio on your computer.

You can see that the RStudio window is divided into four window panes:

- **Script Window** is in the upper left hand corner.
    - ***Note:*** *If you do not see two window panes on the left side of RStudio, do the following to obtain the Script window: File -> New File -> R Script to make a blank script window appear.*
    - ***You will use the Script window pane to write R code.***
- **Console Window** is in the lower left hand corner.
    - The console will display results from commands that you write in the script window.
    - You can also type commands directly into the Console Window Pane, but we'll generally type commands into the Script Window and then "run" the script.
- **Environment Window** is in the upper right hand corner. More on this window below.
- **Plot Window** is in the lower right hand corner.
    - This window pane will display plots that you make.
    - This is also the window pane where you can access help files. (More on this later)

Part II: Some Basics Commands and Functions

The Console

The pane labeled **Console** is where you can enter and execute R code. For example, one thing you can do is use the **Console** as a simple calculator. Try entering the following lines at the > prompt and see that you get the same output. (After entering the code next to the prompt, hit enter to execute the code.)

```
> 1 + 1

## [1] 2

> 3 * 12 + 6

## [1] 42

> 3^2

## [1] 9
```

```
> 10/2
```

```
## [1] 5
```

(Note: you won't see the ## next to your output. We'll use the ## just to indicate an "answer" in the labs.)

Statements, Variables, and the Environment

An R program consists of a series of **statements**.

Here are two examples of statements:

```
x <- 11
y <- x + 2
```

The first statement assigns the number 11 to the variable x. **Variables** store data. (Now we're talking about R variables, which are different from the statistical variables.) They are called variables because their contents can change - you can assign new data to a variable or alter the data already stored there. The <- operator stores the value on the right side in the variable named on the left side.

The second statement adds 2 to x and stores the result in y.

Enter each of the statements above into the console. Notice that there is no output, but the variables and their values show up in the pane labeled **Environment**. The environment pane shows all of the variables defined in your current R session.

You can see what is stored for each variable in the **Environment**, but you can also see what is stored for each variable name in the **Console** window. Return to the **Console** and type that variable name next to the > prompt:

```
> x
## [1] 10
> y
## [1] 12
```

Note: there is one other ways of assigning variables: an equal sign (=). That is, instead of typing x <- 11, we could type x = 11. Both are accepted by R. (Try this yourself to verify.) There are pros and cons for using either way of assigning variables, but we'll tend to use the <- assignment statement in this class.

The Script Window, Vectors, and Functions

The **Script Window** is in the top-left corner of RStudio and is a place where we can write all code to be executed in the **Console Window**.

There are advantages to writing all code in the Script Window:

- It is easier to type all the code we need in the **Script window** than individually in the **Console window**, especially if you have multiple lines of code!
- We'll be able to save what we type (the **script**) in the **Script window**! This will be important when you want to use R code from the past for something you are doing now, or you just want to take a break from R and come back to it later!
- You'll be able to identify errors in code more easily in the Script window. (Yes, we all make errors in coding! However, most are easy to fix!)

In RStudio, open a blank R script using **File → New File → R Script**. This will open the **Script Window** pane (if it wasn't already open), which is where you edit R scripts.

In the **Script window**, let's create a vector. ***A vector is an ordered list of values.***

- The values can be numbers, text, or logical (TRUE or FALSE), but they must all be the same type of value.
- Vectors are created using the c function. The c stands for "concatenate".

Enter the following statement in the first line in the Script window. This command defines z as a vector of three numbers:

```
z <- c(3, 17, 42)
```

In the second line of the Script window, type the following, which will tell R to display the vector z in the console window pane.

```
z
```

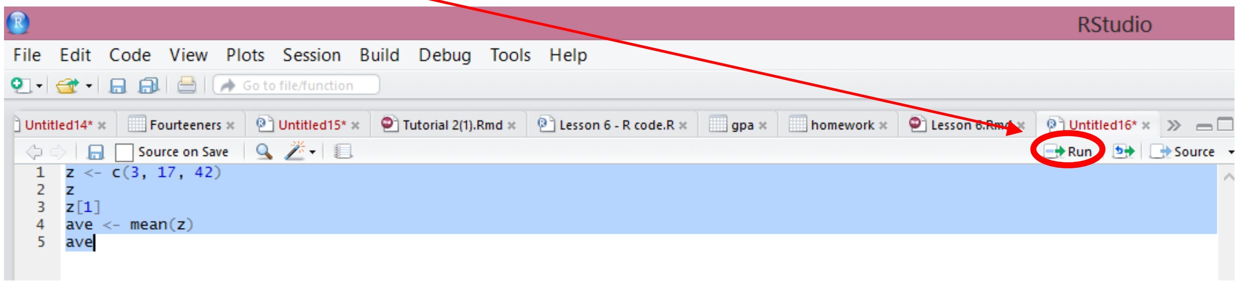
In the third line of the Script window, type the following, which tells R to display only the first value in the vector z.

```
z[1]
```

A **function** (or "command") takes some input and, after doing a calculation or performing some task, returns some output. For example, if we want to calculate the average (or "mean") of the three values in our vector z, we would use the `mean(z)` function. Note that the name of the vector goes inside the parentheses. Let's assign the mean of the vector z to a variable name, called "ave". We'll do that in line 4 in the Script window and then tell R to view the value stored in "ave" in the fifth line in the Script window. Lines 4 and 5 should look like this:

```
ave <- mean(z)
ave
```

To run the script, highlight any number of statements in your R script and click **Run** at the top of the Script Window pane. The highlighted statements will be executed in the Console. If you place your cursor somewhere in the code without anything highlighted and use the same command, it will execute just the line where your cursor is. Try executing your script's statement in the console by highlighting all five lines and clicking on "Run".


RStudio
File Edit Code View Plots Session Build Debug Tools Help
Go to file/function
Untitled14* Fourteeners Untitled15* Tutorial 2(1).Rmd Lesson 6 - R code.R gpa homework Lesson 6.Rmd Untitled16*
Source on Save
Run
Source

```
z <- c(3, 17, 42)
z
z[1]
ave <- mean(z)
ave
```

Use **File → Save As** to save the file with a name ending in .r or .R in whatever location you like.

➢ **You are strongly encouraged to save all R Scripts in a folder (called ST 351 R scripts) in a location where you will be able to access the folder later.**

Part III: Importing a Data Set into R Studio

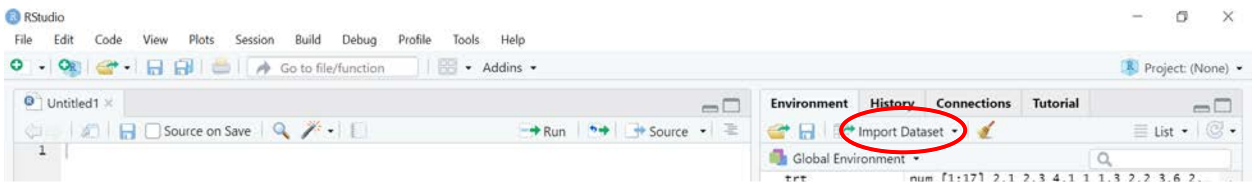
For many assignments, we will import data into RStudio from an existing data set that will be posted on Canvas. In Lab Assignment 1, you will use two data sets. Let's illustrate how to import one of those into RStudio: the **gettyself.txt** data set.

- *Note: all data sets will be posted in Canvas as a .txt file.*

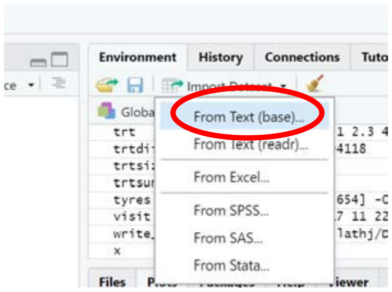
At this time, find the **gettyself.txt** data set in the *Lab Assignment 1* content page in the Week 1 module. (Note: the data sets are also posted in the Lab 1 Notes page on Canvas.) Follow these steps to import the data set into RStudio:

1. Save the `gettyself.txt` to a location that you'll be able to access when using RStudio. One recommendation is to create a folder for this class where you can store all your work for the class. Perhaps create a subfolder called "data sets" where you store all the data sets. *Make sure to save the data set as a .txt file.*

2. Click `Import Dataset` in the upper right hand corner of the R Studio window.

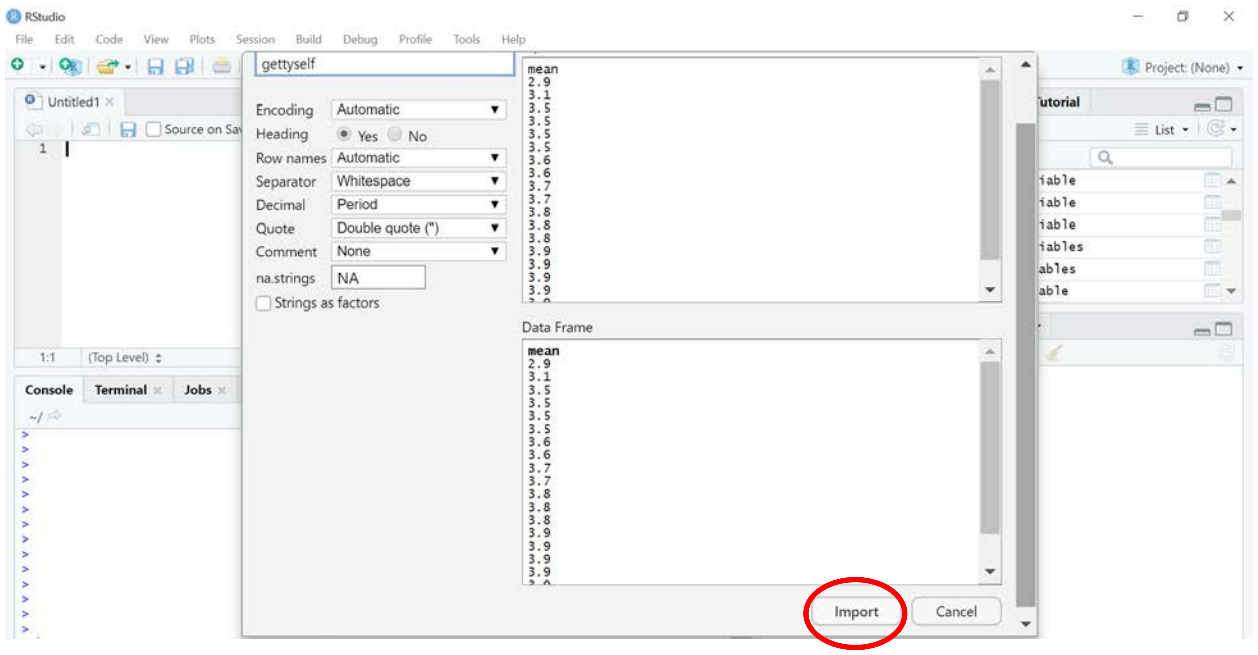


3. From the drop-down menu, click on the file type you want to use to import the data. Using the recommendation of saving the data sets as .txt files, select *From Text (base)*



4. Navigate to and select the file from the location where you stored it on your computer.

5. In the pop-up window, make sure "Heading" is set to *Yes*. (All data sets you will import will have the variable name at the top of the column. Therefore, all data sets will have a "heading".) Then click the *Import* button at the bottom.



gettyself

Encoding: Automatic
Heading: [✓ Yes] No
Row names: Automatic
Separator: Whitespace
Decimal: Period
Quote: Double quote (")
Comment: None
na.strings: NA
Strings as factors

```
mean
2.9
3.1
3.5
3.5
3.5
3.5
3.6
3.6
3.7
3.7
3.8
3.8
3.8
3.9
3.9
3.9
3.9
```

Data Frame

```
mean
2.9
3.1
3.5
3.5
3.5
3.5
3.6
3.6
3.7
3.7
3.8
3.8
3.8
3.9
3.9
3.9
3.9
```

The data set will be displayed in the *Script* window pane.

Note: The data set is automatically named in R with the same name: `gettyself`.

To see the first several rows of data in the data set, type the following in the Console window pane:

```
head(gettyself)
```

A few things to note about this data set:

- There is one variable in the data set (i.e. one column) called *mean*
- The values in the variable called *mean* represent the mean length of words from a self-selected sample of 10 words from the Gettysburg Address. (In Lab Assignment 1, you will have to self-select 10 words from the Gettysburg Address and calculate the mean length of the 10 words you selected. The data in this data set are from 629 recent ST 351 students who had to do the same thing. Therefore, the data in this data set contain 629 means from self-selected samples of 10 words from the Gettysburg Address.)

The $ notation

Whenever referencing a variable from a data set imported into RStudio, a $ must be used to separate the name of the data set from the name of the variable.

For example, if we want to view the 629 values in the *mean* variable in the **gettyself** data set, we **must** type the name of the data set followed by a $ followed by the variable name next to the prompt in the Console window pane (or in the Script window pane)

```
gettyself$mean
```

name of data set   variable name

So, if we wanted to calculate the mean of these 629 values in the *mean* variable, we would type the following command. (Open a new script in the Script Window (**File** → **New File** → **R Script**), as you will have to do this for the lab assignment!) After typing this command, use the instructions above to run the command to determine mean of all 629 sample means.

```
mean(gettyself$mean)
```

Part IV: Random Sampling and Simulation in R

Obtaining a Simple Random Sample

Toward the end of Lab Assignment 1, you are asked to generate 10,000 random samples of 10 words from the Gettysburg Address and calculate the mean length of words from each of the 10,000 random samples. To do so, we'll need to import a new data set that contains the words and length of words in the Gettysburg Address. The words and lengths of words are stored in the **gettyaddress.txt** data set in the Lab Assignment 1 content page in Canvas. Use the instructions above to import this data set. (If you are having trouble, make sure to ask your TA during your lab session!)

The **gettyaddress** has two variables (i.e. two columns):

• *word*	the words in the Gettysburg Address
• *length*	the length of each word in the Gettysburg Address (i.e. number of letters in the word)

To start, let's take one random sample of 10 words from the Gettysburg Address and look at the lengths of the 10 words. While we are randomly selecting 10 words, it's the length of those 10 words we're interested in for Lab Assignment 1. Therefore, we will sample from the *length* variable.

The function `sample()` allows us to get a random sample from a column of data.

The `sample()` function has two arguments (which go inside the parentheses):

- The first is what we want to sample from (in this case, the *length* of words in the Gettysburg Address). Because we're referencing a variable from an imported data set, we must use the $ notation:

```
gettyaddress$length
```

- The second argument is the size of the sample (we will sample 10 words randomly).
- **Separate the two arguments with a comma**

Type this command in a new line in the Script Window. Then run the command.

```
sample(gettyaddress$length, 10)
```

The ultimate goal is to calculate the mean length of words from your random sample. We can accomplish this goal in just one command – type this command in the next line of the Script window and run this command.

```
mean(sample(gettyaddress$length, 10))
```

A few notes about the above command:

- Recall, the argument inside the parentheses of the mean command is what we want to take the mean of. Because we want to take the mean of our random sample of 10 words from the Gettysburg Address, we put the entire command to take that random sample inside the parentheses.

- *Remember to make sure every open parenthesis has a closed parenthesis in the appropriate place in the command.*
- A second option is to store your random sample by giving the entire sample command a name. While you can give the entire sample command any name, I'll call it *mysample*. (Remember, only use one-word names (i.e. no spaces)!) Then we can take the mean of that variable name. Here is the entire code using this option:

```
mysample <- sample(gettyaddress$length,10)
mean(mysample)
```

Simulation

Important point! **More than one random sample is needed to make an informed decision if a random sample is a good representation of a population.**

Simulation is a method of quickly generating many, many results of the same command. We'll use R to generate 10,000 random samples of 10 words each from the Gettysburg Address and take the mean of each random sample. We will learn one set of code to perform a simulation here and learn a second way when we come back to simulation in the Lab 5 Notes.

Type these into new lines in the Script Window (or copy and paste into the Script Window). When typing these commands into the Script Window, use multiple lines! Make sure that you have the code in your Script Window exactly as it appears below:

```
randsample <- rep(0,10000)

for (i in 1:10000) {
 randsample[i] <- mean(sample(gettyaddress$length, 10))
}

mean(randsample)
```

You are now ready to complete Lab Assignment 1. It is highly recommended to work on lab assignments during your assigned lab session in case you have any technical problems with RStudio, have any errors in coding in R that you are unable to resolve, or have any other questions about R, RStudio or the assignment. You may not be able to complete the entire assignment during the lab session, but you should get a good start on it.

Error messages in R

One last comment. Everyone makes errors in coding in R. Don't let that fluster or frustrate you! In general, the best thing to do when you have an error message that you don't know how to fix is to simply copy and paste that error message into google. Chances are, someone else has had that error as well and will probably know how to fix it! Let's discuss a couple of common error messages that you might receive:

Forgetting to close your parenthesis:

Every open parentheses needs a closing parenthesis. Do you see the error in the code below?

```
mean(sample(gettyaddress$length, 10)
```

Type the above in the console window. Note that, now, instead of the `>` appearing in the lower left R console window, a `+` appears instead. This often indicates that you have forgotten to close a parenthesis. In fact, R does not even give you an `Error` message: it's just waiting for you to finish the command (i.e. close the parentheses in this example).

Remember R is case sensitive

```
mean(sample(gettyaddress$Length, 10))
```

Type the above code in the console window. Here, R gives the error message

```
Error in typeof(x) : object 'Length' not found
```

This is because R is case sensitive.

Forgetting a comma

When commands have multiple arguments, you must separate each argument with a comma. Forgetting a comma is a common error. Do you see the error in the code below?

```
mean(sample(gettyaddress$Length 10))
```

The point is that we all WILL make errors in coding. DO NOT PANIC!! Look for some of these common errors (forgetting a parenthesis, forgetting a comma, forgetting quotes, upper or lower case letters, spelling). There will be others. Most of the time you'll be able to figure it out on your own. If not, contact your TA and we'll try to help (but be specific about what you typed and what the error message is).