Lecture 1 - Intro to R/RStudio

Will Calandra

Note: This is a summary of our first R education session for those who were absent (and those who want to review)! Please reach out to me (Will Calandra) at wcc44@georgetown.edu (mailto:wcc44@georgetown.edu) if you have any questions or concerns. Thanks!

Downloading R and RStudio

Before we begin, let's download the necessary software, which will be R and RStudio! R is the base language that we install to run on our computer, and RStudio is the IDE (Integrated Development Environment) where we will type and run our code!

There are two steps to this downloading process…

- Download R - https://www.r-project.org/ (https://www.r-project.org/)
    - This is the link to the R website. Click on "CRAN" under "Download" and pick the location closest to you!
    - Then at the top of the page, click "Download R for XX" - this depends on your operating system! Then click the "base" option, and then the "Download" link at the top of the page again!
    - If you did this right, a win.exe file should be downloaded. Open this and follow the instructions! This is a free and open license, so don't worry :)
- Download RStudio
    - Now that R is downloaded on your computer, we will download our IDE, RStudio. Navigate to the RStudio website: https://posit.co/download/rstudio-desktop/#download (https://posit.co/download/rstudio-desktop/#download)
    - Scroll and click on the "Download" button for RStudio Desktop (!!)
    - Doing this should take you to a table with the titles "OS" "Download" "Size" and "SHA-256." Click on the download link for your OS and follow the instructions with the .exe download!
    - Once completed, RStudio should be an application in your computer. Find it, put it in an accessible place, and fire it up!

If you need help with downloading the software, feel free to reach out or watch a YouTube video so you can see how we do it!

Now, to frame this lecture, watch this inspirational TED Talk about the power of data below:

TED Talk

Link: https://www.youtube.com/watch?v=8pHzROP1D-w (https://www.youtube.com/watch?v=8pHzROP1D-w)

- "Big Data is Better Data" by Kenneth Cukier, who is a journalist at the Economist and a famous speaker on technology and society
    - Please try to watch the entirety of the video! Cukier gives us an inspiring yet fair perspective on "big data," its potential, and its dangers - below are some of my thoughts on it as we think about the impact and implications of our work

Some thoughts…

- The world is full of patterns, seen and unseen, and it is our job to find trends in data that elucidate these patterns and tell a story. Information is the world's most valuable asset, and we can use it to learn and make discoveries.
- Data and processing information is not a new concept. However, the development of technologies that can harness large volumes of data has become a game-changer for every industry. R is a tool that we use to process information and turn it into actionable insight!
- We have become clever at representing real-word problems in a computer, through code and mathematics. We have turned traditional problems into "data problems" that can be simulated in a computer to learn FAST.
- In lecture, we discussed ethical concerns, potential for government regulations on algorithms, and how tech leaders are challenged with understanding how their solutions will scale and impact society. There is an entire field of data ethics in addition to studies in responsible AI which try to tackle concerns around privacy, discrimination, and various other issues that are prevalent in the infancy of the "big data" age.
- I really enjoyed our discussion, and those of you who attended lecture asked me some really thought-provoking questions. I was challenged by these and you did a good job driving the discussion forward!

RStudio IDE Tour:

- Thanks for watching… now head back to your RStudio! We now take a tour of the "Integrated Development Environment", or IDE, for R. REmember that an IDE is just a fancy way for saying an application that you use to write, test, and run your R code
- In the top right, you will see a glass box, which is for your "projects," serving as a folder in R for all of your files. It is important to put all your data and files in the same directory, so it is a good idea to stay organized by creating projects
    - Let's create a "Hoyalytics" project - click on it, click "new project," "new directory," "new project," give it the name "Hoyalytics," and save it on your computer where'd you like - all your work will be done in here!
- There are 4 panels in RStudio's IDE. Let's look at each:

Top left - source code

- This is where you will write the code that you share with us
- R reads the code that you type in the source panel from left to right, top to bottom, just like a book - this is important to know because order matters!
- You type in here, and click the "Run" button to run the line of code that your cursor is on - you can also highlight multiple lines of code and click "Run" to run multiple lines of code at once

Bottom left - the console

- This is actually the place where your source code is sent for R to read - you can test your code in the console to make calculations and such, but code that you type in here won't be saved or sent to anyone! The console is useful to look at your code, objects (more on this later), data, etc. - without impacting your source code!

Top right - the environment

- Here is where your "saved" values, data, objects, etc. are stored! When you run your code and save it in an object, that object and its characteristics will be displayed in here
- A handy tool to use is the "Import Dataset" capability where RStudio can help you import some data for you to work with!
- Clicking on the data and dataframes (more on this later) will allow you to view them, which can be helpful and easier than typing a "view()" function

Bottom right - files, plots, packages, and help

- Your directory, where you create and import files, will show up under the “files” tab - there is some functionality here to play around with, but think of this as just a useful tab to keep track of your files and know where things are in your computer
- As you run R code with visuals, they will display as they are produced in the “plots” tab - you can click the “zoom” button to blow them up full screen, and you can even “export” them as PDFs and such
- The “packages” tab includes packages (or libraries) in R, with the checked packages representing the packages installed on your computer - more on this in later lessons, but packages are libraries that are full of functions and capabilities that we can use (for free!) - clever people have developed packages that do neat stuff in R
- The “help” tab has some useful links with information that you can check out - we find the search bar in this tab to be the most useful for looking up certain packages or functions we may have questions about - this will give us what is called “documentation,” which details what functions do and tell us how to properly write code for these functions

Time to get started!

- So that covers the basics of RStudio… now we can create a file and get started!

Basic Die Roll Simulation Example

```
# Counter Example
```

- Starting with line 1, we have a comment, which is a way for the user to enter information about what they are doing without the line being interpreted by the compiler. In R, using the # symbol at the beginning of a line creates a comment for the entire line, where you can insert text referring to what the subsequent code is expected to do. In our example, we use #Counter Example to indicate to us that the following lines are meant to create a counter.

```
X <- 5
```

- In line 2, we define our first object, X, and assign it the value of 5. This is an integer object, meaning that the value 5 is stored in the name “X”, and we can reference it in the future by calling the object “X”. The “arrow” symbol <- is called the assignment operator and indicates that X is being assigned the value of 5. It can be created by using the less than “<” operator followed by the hyphen “-” symbol.

```
X <- X + 1
```

- In line 3, we are reassigning the object X with a new value. This line cannot be run unless the object has already been initialized (i.e. assigned a value) as we did in line 2. This will re-assign X to an integer value 1 greater than the value X is currently assigned. If you were to run lines 2 and 3, X would finish with a value of 6, but if you were to run line 3 again, X would now be assigned 7. This seems like weird behavior in R, and it has to do with your computer’s memory. for now, just know that it is good practice to define objects carefully and make sure that you only run each line of code once while you are doing your analysis.
- We are now going to set up the die roll simulation:

```
# Die roll
S <- 1:6
```

- The code in line 6 (S <- 1:6) aims to create a sequence of integer values from 1 to 6, inclusive, and assigns the list to the variable S. The 6 values that you see represent the 6 possible die values.

```
# Use function (argument1 = x, argument2 = y, ...)
S <- seq(from = 1, to = 6, by = 1)
```

- The code in line 9 also aims to create a sequence of integer values from 1 to 6, inclusive. This is our first example of a function, which can be formally defined as a set of instructions known by the compiler which are intended to achieve a specific purpose. In R, there are two main ways to look up what a function does. The first method is to go to the help tab (described in the previous section) and type in the name of the function. The other method is to type a question mark followed by the name of the function into the console (ex. ?seq()). In this case, the seq() function will be creating a sequence of integer values. “From,” “to,” and “by” are known as arguments (or parameters), and they will influence the process of the function. In the case of the seq() function, the “from” argument indicates the starting number of the sequence, the “to” argument indicates the concluding number of the sequence, and the “by” argument indicates the increment of the sequence. The single equals sign (“=”) is another assignment operator and must be used within the argument of the function to assign the correct value to the function’s parameter.

```
P <- c(1/6,1/6,1/6,1/6,1/6,1/6)
```

- Line 10 is creating a vector of 6 numbers and assigning it to the named variable “P.” Here we also see another function, called c(), which stands for concatenate. There are no specific names for concatenate, so we just put in the values of interest. We are using c() to create a vector object of 6 numbers which represent the probability of a six-sided fair die roll landing on each value. Why did we create this? Well, vectors are HUGE in R. Think about it: S is a sequence of 6 numbers (1, 2, 3, 4, 5, 6) with a specific order, so S is a vector. We create the corresponding vector P that corresponds with each entry of S as its probability of occurrence. We will combine these vectors in a clever way to get a randomized output of die values.

```
# Randomization set.seed() function
set.seed(27)
```

- Line 13 is a set.seed() function that is important to use whenever there is randomization. In the set.seed() function, R makes a random sequence of numbers, and keeps this randomization constant when you re-run code. This is important when we want to reproduce model results from randomized experiments. The number within the set.seed() function represents the “seed,” or sequence of random numbers. This number is arbitrary, but we’ll choose 27 here.

```
# Sample function (uses randomization above, run line 13 BEFORE 16 every time)
Samps <- sample(S, 23, replace = TRUE, prob = P)
```

- Line 16 is an object called “Samps,” which is where we will save the values of our sampling from the S sequence. Since S is a sequence from 1 to 6, we simulate the value of a die roll in our computer by randomly sampling from the S object. The “Samps” object is defined by a sample() function, which takes the corresponding parameters (sample, size, replacement, and probability). We can see that we cleverly use our objects to represent parameters, which is good practice, because we do not have to change our code each time a value in our objects is updated. Walking through this line of code, the sample() is the function, the “S” represents the object being sampled from (1 to 6), “23” represents the size of the sample (take 23 samples), “replace = TRUE” means that we are sampling with replacement, and the “prob = P” is the corresponding probability vector for each value of 1 to 6 (equal, ⅙ for each value). Run this and see what

happens! Notice how we run the set.seed() function each time we run the Samps line. This is important because the sample() function is random, so we want to reiterate over the 27th seed each time for consistency.

- Note how we didn't put the parameter names for S and 23. It is not required by R that you use the parameter names and the assignment operator "=" to assign their values, but it might help you out early on if you practice writing code in this manner (to better see what you are doing)!

```
print(Samps)
```

```
##  [1] 1 2 1 3 3 4 2 2 2 3 5 1 5 6 1 6 1 2 5 2 2 1 5
```

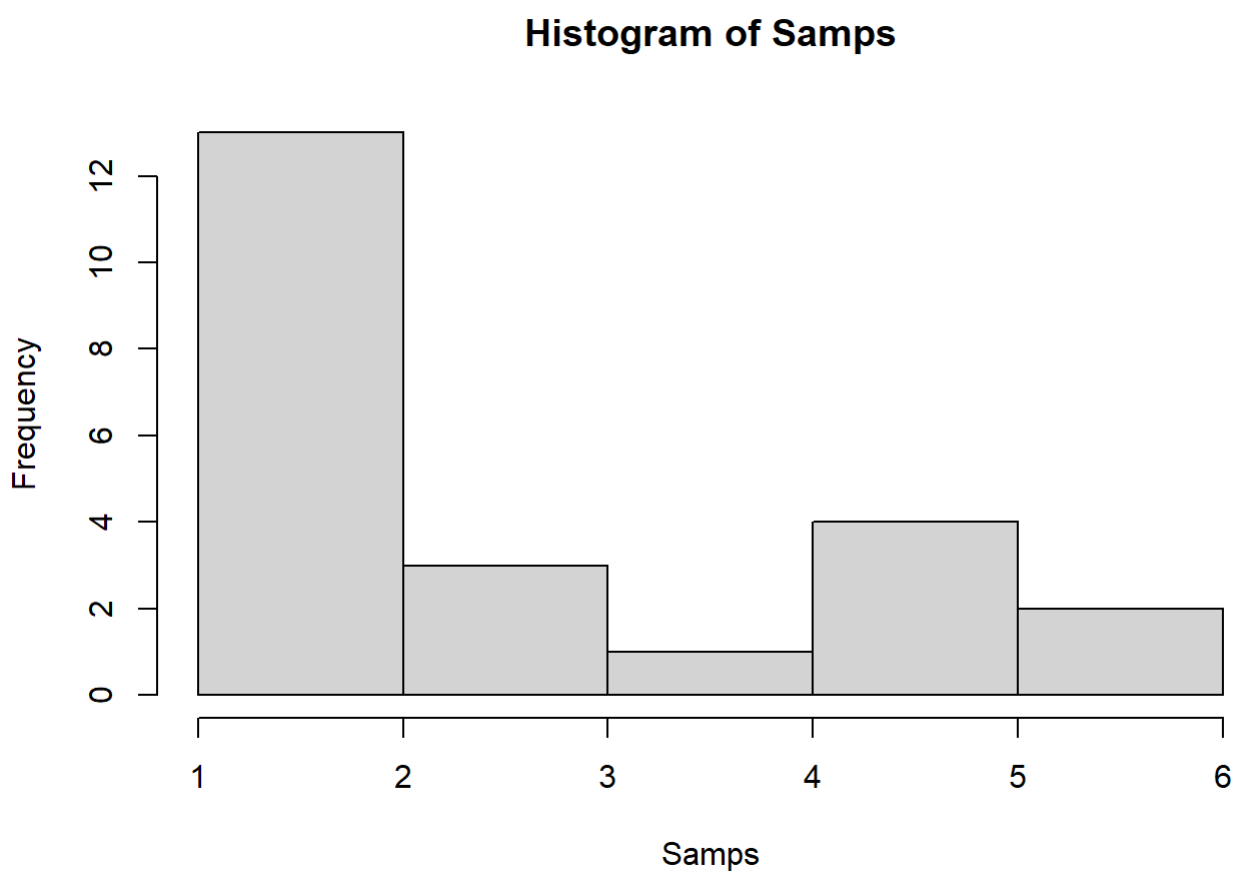
- A useful print() function call on line 17 lets us see some values of Samps, but not all of them. We'll see how we can do this later. Often, it's good practice to type print() in the console instead of your source code.

```
summary(Samps)
```

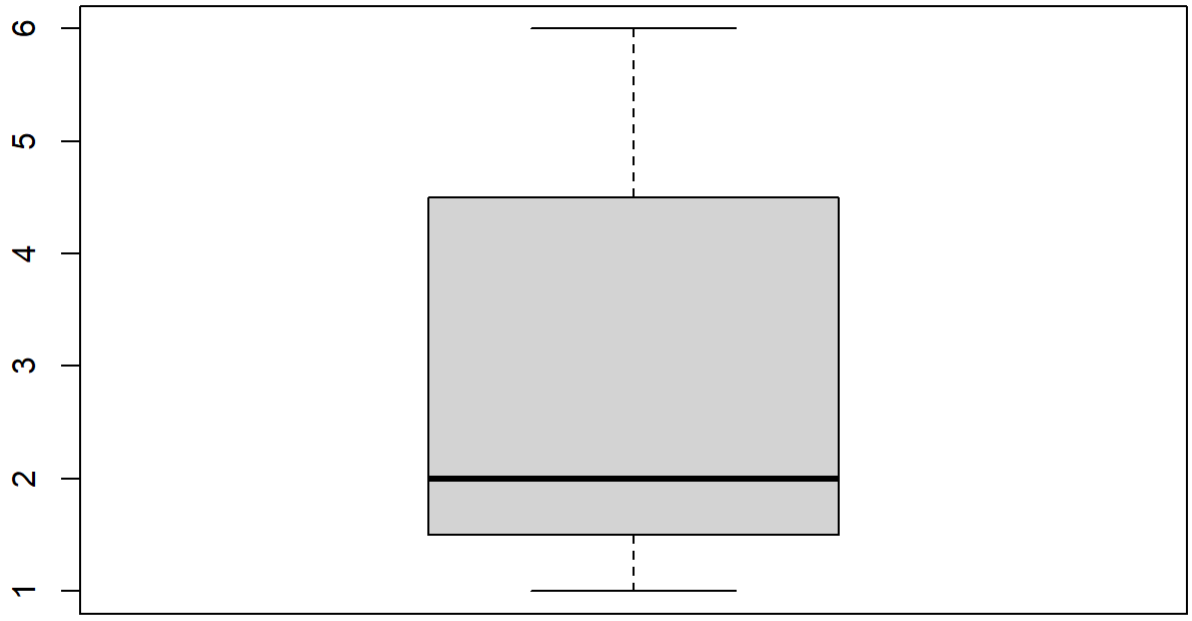
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.500   2.000   2.826   4.500   6.000
```

- Line 18 is a handy way to look at our results using the summary() function as it gives us the 5 number summary of "Samps," or our samples/die rolls - pretty neat!

```
# Visualizations!
hist(Samps)
```




```
boxplot(Samps)
```



- Our first visualizations in R! Notice the distribution of our die rolls: it's not really the theoretical uniform distribution we'd expect! That's because our sample size of 23 is small. Here's an exercise: try changing 23 to a much larger number. Then these visuals should be more appealing!

```
# Change data type and save
Samps <- as.data.frame(Samps)
```

- Notice how it is difficult for us to access the results of "Samps." This is because it is stored in R as a list. We want to be able to take a closer look and do more with our die rolls, so we will change the class of our object "Samps" from a "list" to what we call a "dataframe." Dataframes are practically Excel sheets; they are vectors/matrices representing columns and rows. The as.data.frame() function changes "Samps" into a dataframe… we will overwrite the Samps object to save this as "Samps" using the as.data.frame() function in line 25.

```
summary(Samps)
```

```
##      Samps
##  Min.   :1.000
##  1st Qu.:1.500
##  Median :2.000
##  Mean   :2.826
##  3rd Qu.:4.500
##  Max.   :6.000
```

- Line 27 offers a summary of Samps, proving that it is the same as our previous Samps object, just in a different class (form). Look at the top right in your global environment; you are able to click on Samps and access each die roll! Notice how it saved as a column name “Samps” with the values of our 23 die rolls. Congrats, you just simulated die rolls and got an output!

Notes

Thanks for reading! If you missed our lecture, trust me — watching, doing, and learning is more fun than reading, especially when it comes to learning how to code! This document should suffice for those of you who were absent, but it should not be your primary way of learning R… come to the lectures and use these for review! :)