An Introduction to the R Environment

Dr. Austin Brown

Kennesaw State University

8/16/2021

Why Learn R?

- You may be asking yourself, out of all of the possible analysis softwares which exist, why should I spend time learning R?
- Great question!
- R is a useful tool and worthwhile to learn for several reasons:
  1. It's free!
  2. Because it's open source, thousands of people have contributed packages and functions at a pace that proprietary softwares can't compete with
  3. It is very flexible and robust meaning there's a lot you can do with it (including creating these very slides!)
  4. It is becoming widely used across many industries

So What is R?

- R is a command-line, object-oriented programming language commonly used for data analysis and statistics.
- **Command-line** means that we have to give R commands in order for us to get it to do something

```
## I want to add 2 and 2

2 + 2
```

```
## [1] 4
```

So What is R?

- **Object-oriented** means that we can save individual pieces of output as some name that we can use later. This is a super handy feature, especially when you have complicated scripts!

```
## I can save 2 + 2 as "a"

a <- 2 + 2
a
```

```
## [1] 4
```

What can R do?

- What can R do? Well, for the purpose of data analytics, I am yet to find a limit of what it can do!

- In this class, we will be using R as a tool for analyzing categorical and quantitative data both visually and quantitatively.

- Note, this class is a first step in learning R. While we'll learn a lot of programming skills, and specifically for manipulating and analyzing data, there is so much more that can be learned.

Importing Data into R

- In order to analyze data in R, we need to be able to import data into R.

- There are a variety of ways of importing data into R, but they largely depend on the type of datafile that you are importing (e.g., Excel file, CSV file, text file, SAS dataset, SPSS dataset, etc.).

- While there are lots of different files which can be imported into R (Google is an excellent resource for searching for code for how to do something), we're going to focus on two main types: Excel and CSV

Importing Data into R

- Let's try importing a CSV file into R. This file is part of the famous Framingham Heart Study.
- First, download the "HEART" file from D2L. Save the file to somewhere on your desktop.
- Now to read in this CSV file, we will use the `read.csv` function.
- Every function in R requires arguments specified inside of parentheses.
- If you aren't familiar with the arguments for a particular function, R has a cool feature where we can type a `?` before the function in the console and press enter.

Importing Data into R

```
## What are the arguments for read.csv? ##
?read.csv
```

Importing Data into R

- As we can see, there are a lot of arguments we can specify. However, we don't need to specify most of them.

- All of the arguments which have an "“=" after them, like "`header = TRUE`," will retain that argument unless you explicitly change it. `header = TRUE` means that the columns of the CSV file have names. If they don't, then we would change it to, `header = FALSE` and the column names will have generic names, (`V1, V2, ... , VN`).

- So for us, because we know that the CSV has names for the columns, our code for importing will be:

```
heart <- read.csv("HEART.csv")
```

Importing Data into R

- Now that we've learned how to import a CSV file into R, let's learn how to import an XLSX file into R.

- First, though, we have to install a package into R which has an easy-to-use function for importing XLSX files.

- There are hundreds and hundreds of R packages which contain special functions for special purposes, and before we can use them, we have to install them into our local repository.

- The function we need is called `read_xlsx` which is within the package called `readxl`. Installation of a package is easy! The code we use is:

```
install.packages("readxl")
```

Importing Data into R

- Once the package has successfully installed, we don't quite have access to its functions.

- To gain access, we can either use the `library` function to give us access to the package's functions in the current session, or,

- we can type the name of the package followed by two colons and then the name of the function and all its associated arguments. I prefer this second method because some functions in different packages have the same name (a bit inconvenient, I know).

- So to be sure I'm using the function from the package I want, using this colon method is better.

Importing Data into R

- Download the XLSX file called "esoph" from D2L and save it to your desktop. This dataset contains information about esophageal cancer patients from a study in France.

- Now we can import this file into R using:

```
esoph <- readxl::read_xlsx("esoph.xlsx")
```

Exploring Dataframes in R

- So we've imported some datasets into R. . . how do we know that they imported correctly?
- There are two general approaches I'd recommend. One is visual and one uses the `summary` and `head` functions.
- In the upper right hand corner of the RStudio window, we see our heart dataframe we uploaded a bit ago. If we click on it, a new window will open up which shows us the structure of the dataframe, the values of the variables, and the variable names.

Exploring Dataframes in R

- This visual method is effective for relatively small dataframes ($<$ 10,000 rows), but can bog down your machine if you have a large dataframe.

- To get around this, we can just look at a few rows of the dataframe using the `head` function, which prints the first few rows of a dataframe to your console.

```
head(heart)
```

Exploring Dataframes in R

- As we visually inspect the first few rows of the HEART dataframe, we can see that the "Sex" variable appears to be *categorical* whereas the "Weight" variable appears to be *quantitative*.

- A **quantitative** variable is something which can be measured with a number, like dollars, time, height, weight, blood pressure, etc. R refers to these as "numeric" variables.

- A **categorical** variable is just the opposite. It is something which cannot be quantified and is more of a quality. These are things like sex, country of origin, hair color, cause of death etc. R refers to these as "character" variables.

Exploring Dataframe using R

- You may be asking yourself, “why does this matter?" It’s important for two primary reasons:

- First, the type of variable we are working with dictates to us which graphical and statistical methods exist for us to analyze that variable. What works for a categorical variable almost certainly won’t work for a quantitative variable.

- Second, in terms of R programming, we can look at the variable `Sex" and`Height" in the heart dataframe and conclude that these are categorical and quantitative variables, respectively. But when we read the heart dataframe into R using `read.csv`, we didn’t have to tell R what types of variables each column was; it by default scans each column and makes a best guess as to what type of variable the column contains.
  - So how can we know that R properly recognized the variables in the heart dataframe?

Exploring Dataframes in R

- One straightforward way to do this is by using the `summary` function.

- This function basically does what it sounds like: it gives us a brief summary of a dataframe. Let's check it out using:

```
summary(heart)
```

Examining Subsets of Dataframes in R

- Let's say I wanted to find the average or mean Age at Death from the Heart dataframe. How would I go about doing that?
- First, I need to know how to isolate that single variable by itself.
- To do this, we make use of the dollar-sign operator after the name of our dataframe.
- In your console, enter the following command, and see what happens:

```
heart$AgeAtDeath
```

Examining Subsets of Dataframes in R

- One of the cool aspects of RStudio is that when you press the dollar sign after a dataframe, whether that's in your script window or your console window, is that it automatically pops up a list of all the variables contained within that dataframe that you can navigate to with your arrow keys.

- Okay, so now that we know how to isolate `AgeAtDeath`, we find its sample mean by using the `mean` function

```
mean(heart$AgeAtDeath)
```

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

Examining Subsets of Dataframes in R

- When we ran that code, the result came up as NA which stands for "not-applicable." Why is this? Isn't `AgeAtDeath` a quantitative variable?

- Let's use the summary function on just this one variable to see what might be going on

```
summary(heart$AgeAtDeath)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   36.00   63.00   71.00   70.54   79.00   93.00    3218
```

Examining Subsets of Dataframes in R

- Since our output contains what's referred to as the "five-number summary," this means R recognizes the variable as a quantitative/numeric variable.

- But notice in the right-most column, we see that there are quite a few NA's or missing values for this variable. (Notice we also get the mean, but we want to get it using the the `mean` function!)

- So there's clearly something weird going on with the `mean` function which is creating the error when we ran `mean(heart$AgeAtDeath)`

- To figure out what the potential problem is, I recommend using `?mean`

Examining Subsets of Dataframes in R

- Notice the third argument in the `mean` function, `na.rm = FALSE`.

- If we scroll down a bit and read what this bit of code does, it basically says that it is a logical (i.e., true or false) argument asking if you want it to remove the NA values before calculating the mean or not. By default, it won't since it is set to FALSE already.

- So if we change this argument to "TRUE" we should get the same 70.54 mean that we saw using the summary function.

```
mean(heart$AgeAtDeath,na.rm=TRUE)
```

```
## [1] 70.53641
```

Examining Subsets of Dataframes in R

- Now, let's say I have a large dataframe with lots of columns of information, as you might see in the healthcare industry.
- But, for whatever analysis I'm wanting to do, I don't need all of the columns, just a few.
- In such a case, it might be useful to subset the dataframe and select only the columns we need (exactly the same as the SELECT clause in SQL).
- How do we go about doing this? Like many things in R, there are a few different ways to yield the same result, but I'm going to show you what I consider the most straightforward method.
- But first, you need to install a package called `dplyr`, which is part of a larger set of packages called, `tidyverse`.

Examining Subsets of Dataframes in R

- Because `tidyverse` contains `dplyr`, I recommend attaching the `tidyverse` package using the `library` command.

- Let's say using the Heart dataframe, I want to create a new dataframe which only contains the last four columns: `Chol_Status`, `BP_Status`, `Weight_Status`, and `Smoking_Status`.

- To do this, we will use the `select` function from within the `dplyr` package.

```
library(tidyverse)
heart_status <- heart %>%
  dplyr::select(Chol_Status,BP_Status,
                Weight_Status,Smoking_Status)
```

Examining Subsets of Dataframes in R

- To check and make sure the subsetting worked properly, we would use the same visualizing and summarizing approaches we used for importing data.

- In the last problem, we subset the Heart dataframe by columns. What if we wanted to subset by values in the rows (similar to the WHERE clause in SQL)?

- For example, let's say in the new heart_status dataframe we just created, we want to create a new dataframe where we only have those participants whose `Weight_Status` is "Overweight."

- Again, there are a few different approaches, but I would recommend using the `filter` function within the `dplyr` package.

Examining Subsets of Dataframes in R

```
heart_status_ow <- heart_status %>%
  dplyr::filter(Weight_Status == 'Overweight')
```

Examining Subsets of Dataframes in R

- To check and make sure this worked, I would recommend utilizing a new function called `table`. It will count up frequencies of unique responses for a particular variable.

- So in the heart_status dataframe, if we use the `table` function, we can see that there are 3550 participants who were categorized as overweight.

- If we look at the number of observations in the heart_status_ow dataframe, we can see we indeed have 3550 observations, meaning that `dplyr` did what it was supposed to do.

```
table(heart_status$Weight_Status)
```

```
##
##                 Normal  Overweight Underweight
##              6       1472       3550         181
```