An Introduction to the R Environment

Dr. Austin Brown

Kennesaw State University

Why Use R?

▶ You may be asking yourself, out of all of the possible visualization softwares which exist, why should I spend time learning and using 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
  5. We can create visualizations and perform quantitative analyses in the same system.

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 3 and 5
3 + 5
```

```
[1] 8
```

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!

```
## Saving the result of 3 + 5 as a ##
a <- 3 + 5
a
```

```
[1] 8
```

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 visualizing categorical and quantitative data through various means.

Importing Data into R

▶ In order to visualize 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, etc.).

▶ While there are lots of different files which can be imported into R (Google and ChatGPT are excellent resources 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.

▶ Since we set up our first class session by connecting to GitHub, we should have this file in our project folder already.

▶ Now to read in this CSV file, we will use the `read_csv` function which is part of the `readr` package.

▶ Every function in R requires arguments specified inside of parentheses.

Importing Data into R

▶ We can think of packages like toolboxes within a mechanic's workshop. Each toolbox contains different tools.

▶ A tool is like a function; we use specific tools (functions) to solve specific problems!
  ▶ `read_csv` is a tool we use to solve the problem of reading CSV files into R.

▶ The `read_csv` tools is stored within the `readr` toolbox (package).

Importing Data into R

▶ Now, while many packages come installed in RStudio automatically, there are far, far more which we have to install from the web, including `readr`.

▶ To install a package, we use the `install.packages` function:

```r
## Installing the readr package ##
install.packages('readr')
```

Importing Data into R

▶ Now, we can also think of functions like mathematical functions; we have to supply the function with special inputs called *arguments* in order to get the desired output.

   ▶ For instance, in the `install.packages` function, we had to specify to the function which package we wanted to install.

▶ How do we know what arguments to specify for a given function?

▶ There are lots of different ways, but one way we can do so is by using the ? operator.

```
## What are the arguments for read_csv? ##
?readr::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 "col_names = TRUE", will retain that specific argument unless you explicitly change it. col_names = TRUE means that the columns of the CSV file have names. If they don't, then we would change it to, col_names = 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:

```
library(readr)
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 by using the esoph Excel file.

▶ The tool we use to do this is a function called read_xlsx which is part of the readxl package.

▶ After installing readxl, we can read in the file by:

```r
## First, install readxl ##
install.packages('readxl')
## Load the readxl package ##
library(readxl)
## Importing the 'esoph' dataset ##
esoph <- read_xlsx("esoph.xlsx")
```

Exploring Dataframes in R

▶ So we've imported some datasets into R...how do we know that they imported correctly?

▶ The best way that I have found which uses a combination of some traditional functions is a function called `glimpse` which is part of an incredibly useful data wrangling package called `dplyr`.

```
## First, Install dplyr ##
install.packages('dplyr')
## Load the dplyr package ##
library(dplyr)
## Get a glimpse of the data ##
heart |>
  glimpse()
```

Exploring Dataframes in R

```
Rows: 5,209
Columns: 17
$ Status         <chr> "Dead", "Dead", "Alive", "Alive", "Alive", "Alive", "Al~
$ DeathCause     <chr> "Other", "Cancer", NA, NA, NA, NA, NA, "Other", NA, "Ce~
$ AgeCHDdiag     <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 57, 55, 79,~
$ Sex            <chr> "Female", "Female", "Female", "Female", "Male", "Female~
$ AgeAtStart     <dbl> 29, 41, 57, 39, 42, 58, 36, 53, 35, 52, 39, 33, 33, 57,~
$ Height         <dbl> 62.50, 59.75, 62.25, 65.75, 66.00, 61.75, 64.75, 65.50,~
$ Weight         <dbl> 140, 194, 132, 158, 156, 131, 136, 130, 194, 129, 179, ~
$ Diastolic      <dbl> 78, 92, 90, 80, 76, 92, 80, 80, 68, 78, 76, 68, 90, 76,~
$ Systolic       <dbl> 124, 144, 170, 128, 110, 176, 112, 114, 132, 124, 128, ~
$ MRW            <dbl> 121, 183, 114, 123, 116, 117, 110, 99, 124, 106, 133, 1~
$ Smoking        <dbl> 0, 0, 10, 0, 20, 0, 15, 0, 0, 5, 30, 0, 0, 15, 30, 10, ~
$ AgeAtDeath     <dbl> 55, 57, NA, NA, NA, NA, NA, 77, NA, 82, NA, NA, NA, NA,~
$ Cholesterol    <dbl> NA, 181, 250, 242, 281, 196, 196, 276, 211, 284, 225, 2~
$ Chol_Status    <chr> NA, "Desirable", "High", "High", "High", "Desirable", "~
$ BP_Status      <chr> "Normal", "High", "High", "Normal", "Optimal", "High", ~
$ Weight_Status  <chr> "Overweight", "Overweight", "Overweight", "Overweight",~
$ Smoking_Status <chr> "Non-smoker", "Non-smoker", "Moderate (6-15)", "Non-smo~
```

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 visualization methods would be most appropriate.

▶ 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 `readr::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 `glimpse` function.

▶ This function basically does what it sounds like: it gives us a brief glimpse of a dataframe. Let's check it out!

Exploring Dataframes in R

```r
heart |>
  glimpse()
```

```
Rows: 5,209
Columns: 17
$ Status         <chr> "Dead", "Dead", "Alive", "Alive", "Alive", "Alive", "Al-
$ DeathCause     <chr> "Other", "Cancer", NA, NA, NA, NA, NA, "Other", NA, "Ce-
$ AgeCHDdiag     <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, 57, 55, 79,~
$ Sex            <chr> "Female", "Female", "Female", "Female", "Male", "Female-
$ AgeAtStart     <dbl> 29, 41, 57, 39, 42, 58, 36, 53, 35, 52, 39, 33, 33, 57,~
$ Height         <dbl> 62.50, 59.75, 62.25, 65.75, 66.00, 61.75, 64.75, 65.50,~
$ Weight         <dbl> 140, 194, 132, 158, 156, 131, 136, 130, 194, 129, 179, ~
$ Diastolic      <dbl> 78, 92, 90, 80, 76, 92, 80, 80, 68, 78, 76, 68, 90, 76,~
$ Systolic       <dbl> 124, 144, 170, 128, 110, 176, 112, 114, 132, 124, 128, ~
$ MRW            <dbl> 121, 183, 114, 123, 116, 117, 110, 99, 124, 106, 133, 1-
$ Smoking        <dbl> 0, 0, 10, 0, 20, 0, 15, 0, 0, 5, 30, 0, 0, 15, 30, 10, ~
$ AgeAtDeath     <dbl> 55, 57, NA, NA, NA, NA, NA, 77, NA, 82, NA, NA, NA, NA,~
$ Cholesterol    <dbl> NA, 181, 250, 242, 281, 196, 196, 276, 211, 284, 225, 2-
$ Chol_Status    <chr> NA, "Desirable", "High", "High", "High", "Desirable", "~
$ BP_Status      <chr> "Normal", "High", "High", "Normal", "Optimal", "High", ~
$ Weight_Status  <chr> "Overweight", "Overweight", "Overweight", "Overweight",~
$ Smoking_Status <chr> "Non-smoker", "Non-smoker", "Moderate (6-15)", "Non-smo-
```

Exploring Dataframes in R

▶ Sex, for example, we can imagine is a categorical variable as
   its values, male and female, are characteristics and not
   numbers.

  ▶ We can tell R read it in as a categorical variable because it is
     coded as character (notice the <chr> to the right of the Sex
     variable name).

▶ Height, on the other hand, we can imagine is a quantitative
   variable as its values are numbers!

  ▶ We can tell R read it in as a quantitative variable because it is
     coded as double (notice the <dbl> to the right of the Height
     variable name).

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.
   ▶ You can think of the dollar-sign operator like a door to your home. The name of the dataframe is the house itself, the dollar-sign is the door, and the variable name is the person we want to talk to inside of the house.
   ▶ So the structure is: `House$Person`

▶ 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 dataset 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?

▶ One trick I often use when I get unexpected output is to look at the documentation for the function I'm trying to use with the help of the ? operator.

▶ 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.

Examining Subsets of Dataframes in R

▶ So if we change this argument to "TRUE" we should get the right answer:

```
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 your own fields of study.

▶ 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.

▶ 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, which uses the `dplyr` package.

Examining Subsets of Dataframes in R

▶ 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.

```
heart_status <- heart |>
  select(Chol_Status,BP_Status,
         Weight_Status,Smoking_Status)

heart_status |>
  glimpse()
```

```
Rows: 5,209
Columns: 4
$ Chol_Status    <chr> NA, "Desirable", "High", "High", "Hi
```

Examining Subsets of Dataframes in R

```r
heart_status <- heart |>
  select(Chol_Status,BP_Status,
         Weight_Status,Smoking_Status)

heart_status |>
  glimpse()
```

```
Rows: 5,209
Columns: 4
$ Chol_Status    <chr> NA, "Desirable", "High", "High", "Hi
$ BP_Status      <chr> "Normal", "High", "High", "Normal",
$ Weight_Status  <chr> "Overweight", "Overweight", "Overwei
$ Smoking_Status <chr> "Non-smoker", "Non-smoker", "Moderat
```

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?

▶ 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

```r
heart_status_ow <- heart_status |>
  filter(Weight_Status == 'Overweight')
```

Examining Subsets of Dataframes in R

▶ To check and make sure this worked, I would recommend utilizing two new functions called dplyr::group_by and dplyr::count. Basically what they is counts up frequency of unique responses for a particular variable.

▶ So in the heart_status dataframe, if we use these, we can see that there are 3550 participants who were categorized as overweight.

Examining Subsets of Dataframes in R

▶ 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.

```
heart_status |>
  select(Weight_Status) |>
  group_by(Weight_Status) |>
  count()
```

```
# A tibble: 4 x 2
# Groups:   Weight_Status [4]
  Weight_Status     n
  <chr>         <int>
1 Normal         1472
2 Overweight     3550
3 Underweight     181
4 <NA>              6
```