Lab 01 - Introduction to R and R Markdown

2026-01-23

Contents

Getting Started with R Markdown

This is an R Markdown document.

At present, you are reading it formatted as an HTML document, but as we have seen, it can also be rendered as a PDF – when working with an .Rmd file, you can choose how it is presented by selecting the arrow next to the Knit ball of yarn an choose between "Knit to HTML" or "Knit to PDF". For most of the work in this class, we will be knitting to a PDF to facilitate submissions to canvas.

When you Knit an R Markdown file, it will ask you to first save and name your document. I recommend sticking to a consistent format throughout the semester: something like `lastname_[lab/hwk]_#.Rmd`. For example, I would name this `paris_lab_1.Rmd`. After you knit, this will also be the name of your pdf.

For this lab, we will be following along with this document which we can begin by downloading the .Rmd file associated with it here; this is allow us to see side-by-side how the .Rmd file looks compared to the finished output.

For example, reading this online we see that **this sentence is bold**, while in the .Rmd file we see that the sentence is a plain-text sentence surrounded by two asterisks (**) on each side. This is an example of using markdown to format our documents. For this lab, we will be following along with the .Rmd file, filling bits in as prompted. Let's begin by introducing some of the more common markdown syntax that we will use in this class.

Text Editing

We can use .Rmd files to write text just as we would with a standard word processor. As we saw previously, using **two asterisks** creates bold text, whereas using *single asterisks* creates text that is italicized.

Headers can be created using the # symbol. Using one creates a large header, two a smaller header, and so on. You can see examples of this in the text above (this section on "Text Editing" uses two to create a header). Take note that you need to leave a space between the # and the header text – forgetting to do so will render the text exactly as you have written it.

#For example, this will not show up as a header

We can also create sequential lists. To do so, start new lines with sequential numbers

1. Like this
2. And this
3. And finally this

We can create un-ordered lists with using stars or dashes

- Here is an item
- Here is another item
  - And if I indent, I can make nested lists

We can also create horizontal dividers with a line of stars (at least 3, but any number greater is ok). This is a good way to separate your answers from a given problem

---

Things are more clear if I put my answers between bars, but I need to be sure to leave a blank line between my text and the bottom star

---

Question 1:

Create a new R Markdown file and copy the entirety of this question over to the new file (we will do this for all questions in this lab). Then, proceed with the instructions below.

Between the stars below, do the following:

1. Use two # to create a header that says About Me
2. Type your first name in bold and your last name in italics
3. Create a bullet point list of the people sitting on either side of you
4. Create a numbered list of your 3 least favorite animals

Once you have done this, Knit to PDF by clicking the bar of yarn above and verify that everything looks like it should.

Getting Started with R Studio

Whereas R is the programming language responsible for doing our relevant computation, our interactions with R will be primarily through R Studio. When you first open R Studio, your screen will partitioned into 3 panes: *Console*, *Environment*, and *Files* (and often, *Editor*)

Console

The console makes up the lower left-hand side of the screen, and it is here that we can interact with R directly. Some information will appear on start-up, followed by a prompt (indicated with `>`). Try copying and pasting the following lines into your console one at a time and pressing enter:

```
5 + 2
sqrt(25)
pi
(3^2 + 1)*5
x <- 5
```

Environment

The environment panel is located in the top right and includes all of the named variables that we have created in R. If you copied the lines above correctly, you should see a single line here, indicating that the variable `x` has the value `5`. When starting a new R session, this pane will be empty.

Output/Files

In the bottom right is the Output pane which, in addition to giving us information about our file system also includes tabs for Plots and Help. Copy each of the two lines below to see how this pane changes

```
?sqrt
plot(1:10)
```

You can click any of the tabs in this pane to change what is currently being shown.

Editor

Finally, we have the editor. If you start a new R Studio session, there will be nothing in the editor, but this will change if we open either a .R or .Rmd file. An R Script is the simplest type of file which can be created from the menus at the top:

> File -> New File -> R Script

The editor pane will appear with a blank R file in which you can write and store R code. Try copying the R commands from the *Console* section into an R Script, click on the line with code and click "Ctrl+Enter" on your keyboard. R Studio should execute this code and display the results in the Console. You do not have to save this file, and you can close it when you are finished.

Getting Started with R

While it is true that R scripts (files that end in .R) are used for the vast majority of written R code, we will primarily be using R Markdown files (files that end in .Rmd).

Perhaps the most obvious utility in using R Markdown documents for this course is their ability to interweave regular text with R. We can include and run R code by including it in a *code chunk*, such as the one shown below:

```
# This is a code comment, beginning with #
x <- 4
x^2
```

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

This code will run and display its results when we push the Knit button, but we can also run the code without knitting by place by again clicking the line with our cursor and hitting "Ctrl+Enter"

For this portion of the lab, we will be introducing some of the basic mechanics in the R language. This will include `vectors`, similar to the variables we discussed in class, as well as `data.frames`, the primary data type we will be using for collections of observations.

You are welcome to follow along on the course website or read the R Markdown directly. In either event, I highly encourage you to try running the code that we see, either by coping it from the website and pasting into the console, or by running with the R Markdown code chunks.

Vectors

Creating and subsetting

Vectors in R are the simplest type of object. Every vector has two attributes that will dictate how we use it: length and type. We have already seen vectors of length one, which are created by assigning a number to a name using `<-`

```
# This is a numeric vector of length 1
x <- 4
x
```

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

We can create vectors of greater than length one using `c()`, which is short for "combine"

```
y <- c(2, 4, 6, 8, 10)
y
```

```
## [1]  2  4  6  8 10
```

Sequences of vectors can also be created with `:`

```
z <- 1:5
z
```

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

Every element of a vector has a position that we can access directly using square brackets `[]` in a process called *subsetting*. Here, we grab the 3rd element of the vector `w`

```
w <- c(3, 6, 9, 12)
w[3]
```

```
## [1] 9
```

Question 2:

Again, copy the entirety of this question into the R Markdown file you created for Question 1.

Let's practice creating vectors and subsetting with a short exercise.

1. First, create an R code chunk between the rows of stars below (Ctrl+Alt+I is quick way to do this)
2. Next, create a vector called `x` that has all of the numbers from 11 to 20
3. Use square brackets and subsetting to select the first five numbers of this vector.

Note in R, like most programming languages, there are many ways to accomplish any task. ****

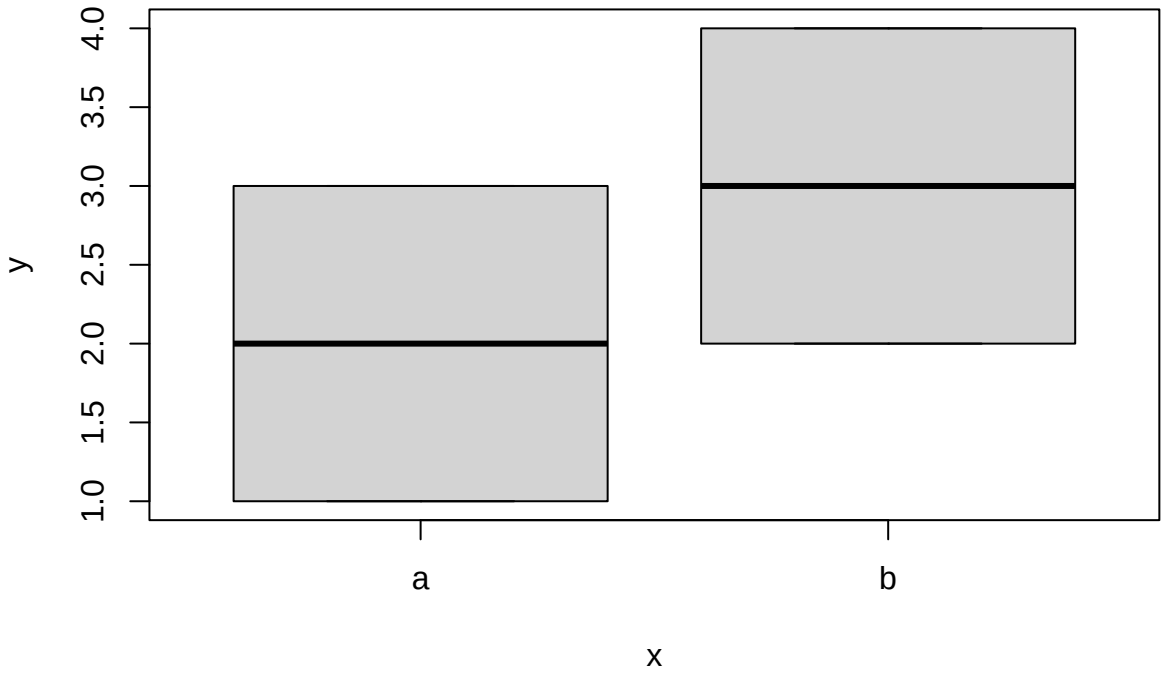
Vector types

Just as we saw in class that variables can be quantitative or categorical, so it is with vectors in R. There are primarily 4 types of vectors we need to be aware of:

1. *numeric* - for example: `x <- c(1, 2, 3)`
2. *character* - for example: `x <- c("a", "b", "c")`
3. *logical* - for example `x <- c(TRUE, FALSE, TRUE)`
4. *factor* - for example `x <- as.factor(c('a', 'b', 'a'))`

Factors and characters in R look extremely similar but behave in slightly different ways on some important functions. To oversimplify R considers characters to be random strings of text that don't have any meaningful statistical properties. Factors is how R denotes nominal variables and recognizes observations are part of some group. Below is a brief demostration of this. As it stands the plot will render (be made) for the factor below but will not run if we use characters which have been hashtag'd out (which is how R knows to ignore/not run code). You don't need to do anything but feel free to test it by un-hashtagging the lines of code which will produce an error when ran

```
my_num <- 1:4
my_factor <- as.factor(c('a', 'b', 'a', 'b'))
plot(x = my_factor,
     y = my_num)
```




```
 my_character <- c('c', 'd', 'c', 'd')
 #plot(x = my_character,
#      y = my_num)
```

Don't worry about memorizing everything – we will continue to review them together as they appear in the course. For now, we will be satisfied with simply having made our introductions.

As we mentioned in class, the type of a variable will often dictate what we are able to do with it. For example, it should make sense to take the mean of a numeric variable, but less so with a character vector. We do this using the R function `mean()`:

```
# This will return a warning and NA (NA = Not Available/Missing Value)
z <- c("a", "b", "c")
mean(z)
```

```
## Warning in mean.default(z): argument is not numeric or logical: returning NA
```

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

```
x <- c(1, 2, 3)
mean(x)
```

```
## [1] 2
```

The function off the top of my head that changes the most is probably summary(). See below for examples

```
z <- c("a", "b", "c", "c")
y <- as.factor(z)
x <- c(1, 2, 3, 20)

summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    1.75    2.50    6.50    7.25   20.00
```

```
summary(z)
```

```
##    Length     Class      Mode
##         4 character character
```

```
summary(y)
```

```
## a b c
## 1 1 2
```

Some situations can be confusing based on appearances. The `class()` function will tell us what *type* a vector is if we are not sure

```
# This looks numeric
z <- c("1", "2", "3")
z
```

```
## [1] "1" "2" "3"
```

```
mean(z)
```

```
## Warning in mean.default(z): argument is not numeric or logical: returning NA
```

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

```
# but actually, it is a character
class(z)
```

```
## [1] "character"
```

Usually in situations like this, we can *coerce* a vector to being a different type, though I will try to make you aware in advance when a situation like this will arise:

```
# coerce the z variable to be a numeric vector
z_num <- as.numeric(z)
z_num
```

```
## [1] 1 2 3
```

```r
class(z_num)
```

```
## [1] "numeric"
```

```r
mean(z_num)
```

```
## [1] 2
```

Deciding what class a vector should have is often context specific. For example, having numeric vectors is ideal if we plan on using them for mathematical operations (i.e., finding the mean, adding them together, taking a square root), which may make sense for things like age, population size, or temperature. Other times, a variable with numbers may make more sense as a character vector to represent categories: this could be things like room numbers, the year an automobile was manufactured, or group assignments.

One important characteristic of a vector to note: all elements of a vector *must* be the same type. They are either all numeric, all logical, or all character. R will force this by default:

```r
z_mix <- c("a", 1, TRUE)

## They are all forced to be character
class(z_mix)
```

```
## [1] "character"
```

```r
#we can try to force the class to be different
#by assigning numeric to the class of z_mix
class(z_mix) <- "numeric"
```

```
## Warning in class(z_mix) <- "numeric": NAs introduced by coercion
```

```r
z_mix
```

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

```r
#No way to force "a" to be a number, not surprising
```

Characters are the most broad (ie least useful) of the data types and is usually avoided when possible. Nominal variables in R are better coverted to *factors* which is an alterantive class.

Data Frames

Most of the data we will be working with in this class will be stored in objects known as data frames. Put simply, a data frame is a collection of vectors that are all the same length.

```r
# Create two vectors of the same length
x <- c("red", "green", "blue")
y <- c(1, 4, 10)

# We can create a data frame with columns named `name` and `value`
df <- data.frame(name = x, value = y)

df
```

```
##    name value
## 1   red     1
## 2 green     4
## 3  blue    10
```

As we can see, this has the format of having rows as observations and columns being variables. We can select one of the variables from our `data.frame` using `$` which will extract the vector

```
# Extract the "name" column
df$name
```

```
## [1] "red"   "green" "blue"
```

```
# Extract the "value" column and use it to find median
median(df$value)
```

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

Data frames are organized (and subsetted) similar to vectors except now we have two dimensions instead of one. We still use the [ ] system but now we need to insert a comma, the left hand side of the comma is the row and the right hand side is the column (rise over run is how I remember)

```
#refresh what the data frame looks like to ourselves
df
```

```
##    name value
## 1   red     1
## 2 green     4
## 3  blue    10
```

```
#Collect just the first column
df[,1]
```

```
## [1] "red"   "green" "blue"
```

```
#Collect the second row
df[2,]
```

```
##    name value
## 2 green     4
```

```
# Collect the third row's second column entry
df[3,2]
```

```
## [1] 10
```

```
# We can also say what we do NOT want with an negative sign
# I want the data frame but I don't want the second row
df[-2,]
```

```
##   name value
## 1  red     1
## 3 blue    10
```

While we can create data frames directly, it is far more likely that we will be reading into R data that has already been created. The most common way to do so is with the `read.csv()` function (CSV stands for comma separated values). This can be read directly from your computer OR you can pass in a URL

```
# Use read.csv to pull Happy Planet data
planet <- read.csv("https://collinn.github.io/data/HappyPlanet.csv")
```

Once a data frame is read in, you will see it show up in your environment. Right away, we get two critical pieces of information: this data frame has 143 observations and 11 variables (recall that rows are observations and columns are variables).

If we click on the blue arrow to the left, we will get a drop down showing us a more detailed description of the variables we do have. Note also the additional information given about the vectors (num = numeric, int = integer, chr = character).

This information can also be produced by using the function str (for structure).

```
str(planet)
```

```
## 'data.frame':    143 obs. of  11 variables:
##  $ Country       : chr  "Albania" "Algeria" "Angola" "Argentina" ...
##  $ Region        : int  7 3 4 1 7 2 2 7 5 7 ...
##  $ Happiness     : num  5.5 5.6 4.3 7.1 5 7.9 7.8 5.3 5.3 5.8 ...
##  $ LifeExpectancy: num  76.2 71.7 41.7 74.8 71.7 80.9 79.4 67.1 63.1 68.7 ...
##  $ Footprint     : num  2.2 1.7 0.9 2.5 1.4 7.8 5 2.2 0.6 3.9 ...
##  $ HLY           : num  41.7 40.1 17.8 53.4 36.1 63.7 61.9 35.4 33.1 40.1 ...
##  $ HPI           : num  47.9 51.2 26.8 59 48.3 ...
##  $ HPIRank       : int  54 40 130 15 48 102 57 85 31 104 ...
##  $ GDPperCapita  : int  5316 7062 2335 14280 4945 31794 33700 5016 2053 7918 ...
##  $ HDI           : num  0.801 0.733 0.446 0.869 0.775 0.962 0.948 0.746 0.547 0.804 ...
##  $ Population    : num  3.15 32.85 16.1 38.75 3.02 ...
```

```
#an alternative is the head() fucntion which produces the first six rows of the data frame but won't re

head(planet)
```

```
##     Country Region Happiness LifeExpectancy Footprint  HLY   HPI HPIRank
## 1   Albania      7       5.5           76.2       2.2 41.7 47.91      54
## 2   Algeria      3       5.6           71.7       1.7 40.1 51.23      40
## 3    Angola      4       4.3           41.7       0.9 17.8 26.78     130
## 4 Argentina      1       7.1           74.8       2.5 53.4 58.95      15
## 5   Armenia      7       5.0           71.7       1.4 36.1 48.28      48
## 6 Australia      2       7.9           80.9       7.8 63.7 36.64     102
##   GDPperCapita   HDI Population
## 1         5316 0.801       3.15
## 2         7062 0.733      32.85
## 3         2335 0.446      16.10
## 4        14280 0.869      38.75
## 5         4945 0.775       3.02
## 6        31794 0.962      20.40
```

```
#this is one of my most used functions. Alternatively tail() returns the
#bottom 6 rows
```

Clicking on the white box pane to the right of `planet` brings us to the tabular view in R, giving us the opportunity to navigate and explore the data more directly

Question 3:

For this question, we will be using the `HappyPlanet` data that we have just looked at:

- **Part A** Copy the code above to read the Happy Planet data into your own R Markdown file, saving the dataset to a variable called `planet`
- **Part B** Looking at the Happy Planet data, explain in one or two sentences what constitutes an observation in this dataset (what is the data being recorded from)
- **Part C** Using `$` to extract columns from the dataset, find the mean life expectancy of all countries in the dataset? (Hint: what functions have we seen already in this lab?)
- **Part D** Using [,] to extract columns, what is the median GDP per Capita? Hint: Use the help pages if NA appears (specifically look for something like an na.rm parameter)
- **Part E** Are there any variables in this dataset that are stored as a numeric that would be better suited as a categorical variable? Explain your answer

---

Introductory Notes Review and Practice

We will work through a series of questions in this lab, which will cover some of the topics we went over in the Introduction slides. Copy the questions to the markdown file you have been making. As always you may work together with others, but each student will need to submit their own version of this file to Canvas as a .pdf with all answers filled out.

Part 1 - Conceptual Questions

Question 4: I said statistics is the study of variability. Please explain that in your own words.

Question 5: What is a census? Why are censuses generally considered good? Give two reasons why we do not always want to conduct a census.

Part 2 - Describing Data & Including Context

We have seen the terms *population, parameter, sample, statistic*, and *observation* in the Introduction. These terms are important for helping us describe data and understand what the purpose of a study is. Being able to read a summary of a study and label these individual parts is going to be an important skill we will use all semester. Our goal is to be able to communicate with each other.

When we are describing the *population*, the *sample*, and the *observations* in a study, we want to provide adequate context to explain the study and data.

To begin with, the topic that is going to be discussed and then the question(s) of interest should be stated or made clear at the beginning. Giving the audience an understanding of why the data exists will improve their understanding of the later description of the data. I want to stress this point. Context is critical in statistics and everything that is reported needs to be understood within that context.

For the techincal side, a generally accepted way is to use the 5 W's and HOW! This may be familiar to those of you who have taken media/journalism classes.

5 W's and H of Data

- **Who** – Who collected the data, who is the data collected on? This requires us to report how many "who's" we have (sample size).
- **What** – What variables was the data recorded on? That is, what pieces of information did we collect from our sample? Eg if we are interested in the voting population of the state of Iowa's views of gay marriage we would probably record if they support it, party affiliation, gender, age bracket, etc...
- **Where** – Where was the data collected? A poll done in Massachusetts offers limited information on the state of Iowa's views on gay marraige.
- **When** – When was the data collected? The voting population of the state of Iowa's views of gay marriage in 1995 offers limited information for today's views.
- **Why** – Why was the data collected? What research question(s) were the investigators trying to answer? This should be answered in the note on context above.
- **How** – How was the experiment ran? How was the data physically collected? How was a random sample randomly sampled?

We may not always use all of these in our own descriptions, but they are useful to add context to our data, and potentially see if there are any issues with the study. It is incorrect to believe that one needs to just respond to each question I presented above. Those are examples of common things that researchers convey to their audience that you need to cognizant about.

Describing Studies

Question 6: (Healthcare Opinions) In 2009, the PEW research group wanted to learn more about public opinion on the idea of the public option for health coverage. One thing that they wanted to know was the percentage of adult U.S. residents who favored a public option for health coverage in October 2009. In a poll of 1500 randomly selected Adult residents in the United States, they found that 55% of adult residents favored a government health insurance plan to compete with private plans. Source

- What is the point of the study? Why did they do this poll?
- Describe the population in this study:

- Describe the sample in this study:
- Describe an observation in this study:
- What is the variable of interest in this study? Is it categorical or quantitative?
- Do you think this data is useful for learning about healthcare opinions in 2024?

Question 7: (National household size) The American Community Survey (ACS) conducts yearly surveys. One thing that is of interest is the average household size. In April 2022, the ACS had surveyed 1,980,550 U.S. households and found the average household size to be 2.50. Source

- Describe briefly why the American Community Survey is ran. Hint: you may google the survey to learn more about it and it's goals.
- Describe the population in this study:
- Describe the sample in this study:
- Describe an observation in this study:
- What is the variable of interest in this study? Is it categorical or quantitative?

Question 8: (Consulting Problem) Making batteries last longer is an economically profitable area and many experiments occur with them. At Iowa State 64 laptop-style batteries were produced using the same standardized process. The cells were then either "slowly discharged" where the battery is slowly drained of power or "quickly discharged" where the battery is rapidly drained of power. All batteries would then be recharged and the process repeated. This continued for each battery until it fell below an acceptable operating condition (ie wouldn't hold a strong enough charge). The total number of charges/discharges for each battery was recorded.

- Describe an observation in this study:
- Describe the sample in this study:
- Describe the population in this study:
- What question do you think the researchers were trying to answer?
- What are the two variables that are being collected?
