biol827-graduate_statistics

Dr. Jacob C. Cooper & Dr. Melissa Wuellner

2024-08-26

Table of contents

Preface

Welcome to Biology 105 at the University of Nebraska at Kearney! Material in this class was designed by Dr. Melissa Wuellner and adapted by Dr. Jacob C. Cooper for use in *R*.

In this class, you will learn:

1. The basics of study design, the importance of understanding your research situation before embarking on a full study, and practice creating research frameworks based on different scenarios.
2. The basics of data analysis, including understanding what kind of variables are being collected, why understanding variable types are important, and basic tests to understand univariate distributions.
3. Basic multivariate statistics, including ANOVA, correlation, and regression, for comparing multiple different groups.
4. The basics of coding and working in *R* for performing statistical analyses.

This site will help you navigate different homework assignments to perform the necessary *R* tests. Furthermore, this GitHub repository contains all of the homework dataframes, so you will *not* have to manually enter assignments if you use *R* to complete your assignments.

Welcome to class!

Dr. Jacob C. Cooper, BHS 321

1 Introduction

This is a book created from markdown and executable code.

See Knuth (1984) for additional discussion of literate programming.

```
1 + 1
```

```
[1] 2
```

2 Intro to *R*

In this class, we will be using *R* to perform statistical analyses. *R* is a free software program designed for use in a myriad of statistical and computational scenarios. It can handle extremely large datasets, can handle spatial data, and has wrappers for compatibility with *Python*, *Bash*, and other programs (even *Java*!).

3 Setup

First, we need to download *R* onto your machine. We are also going to download *RStudio* to assist with creating *R* scripts and documents.

3.1 Installing *R*

First, navigate to the *R* download and install page. Download the appropriate version for your operating system (Windows, Mac, or Linux). **Note** that coding will be formatted slightly different for Windows than for other operating systems.

Follow the installation steps for *R*, and verify that the installation was successful by searching for *R* on your machine. You should be presented with a coding window that looks like the following:

```
R version 4.4.1 (2024-06-14) -- "Race for Your Life"
Copyright (C) 2024 The R Foundation for Statistical Computing
Platform: aarch64-apple-darwin20

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

If that screen appears, congratulations! *R* is properly installed. If the install was not successful, please talk to Dr. Cooper and check with your classmates as well.

3.2 Installing *RStudio*

RStudio is a GUI (graphics user interface) that helps make *R* easier to use. Furthermore, it allows you to create documents in *R*, including websites (such as this one), PDFs, and even presentations. This can greatly streamline the research pipeline and help you publish your results and associated code in a quick and efficient fashion.

Head over the the *RStudio* download website and download "*RStudio* Desktop", which is free. Be sure to pick the correct version for your machine.

Open *RStudio* on your machine. You should be presented with something like the following:

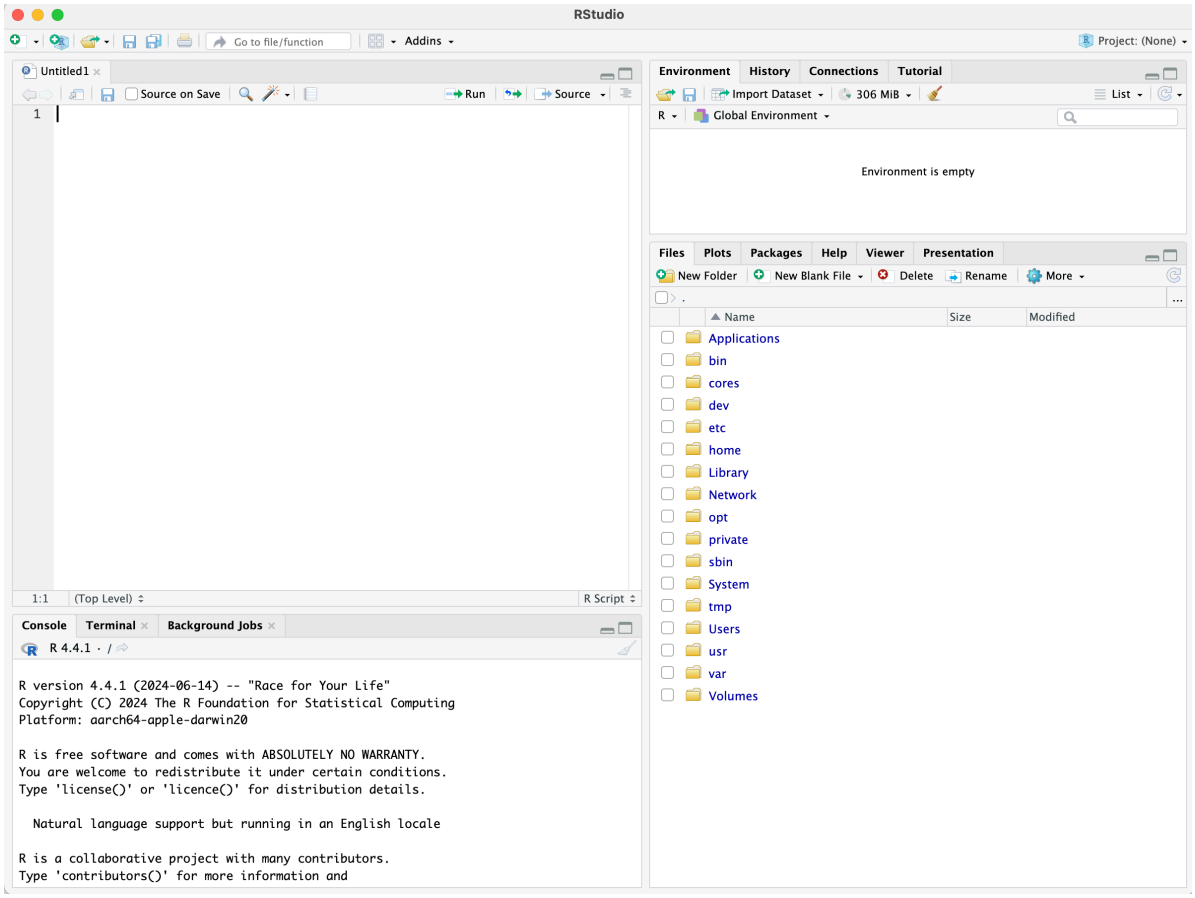


Figure 3.1: *RStudio* start window. Note that the screen is split into four different quadrants. Top left: *R* documents; bottom left: *R* program; top right: environment window; bottom right: plots, help, and directories.

In *RStudio*, the top left window is always going to be our coding window. This is where we will type all of our code and create our documents. In the bottom left we will see *R* executing the code. This will show what the computer is "thinking" and will help us spot any potential issues. The top right window is the "environment", which shows what variables and datasets are stored within the computers' memory. (It can also show some other things, but we aren't concerned with that at this point). The bottom right window is the "display" window. This is where plots and help windows will appear if they don't appear in the document (top left) window itself.

Now, we will create our first *R* document!

4 Creating an *RMarkdown* document

4.1 Setup

In this class, we will be creating assignments in what is called *RMarkdown*. This is a rich-text version of *R* that allows us to create documents with the code embedded. In *RStudio*, click the "+" button in the far top left to open the `New Document` menu. Scroll down this list and click on `R Markdown`.

A screen such as this will appear:

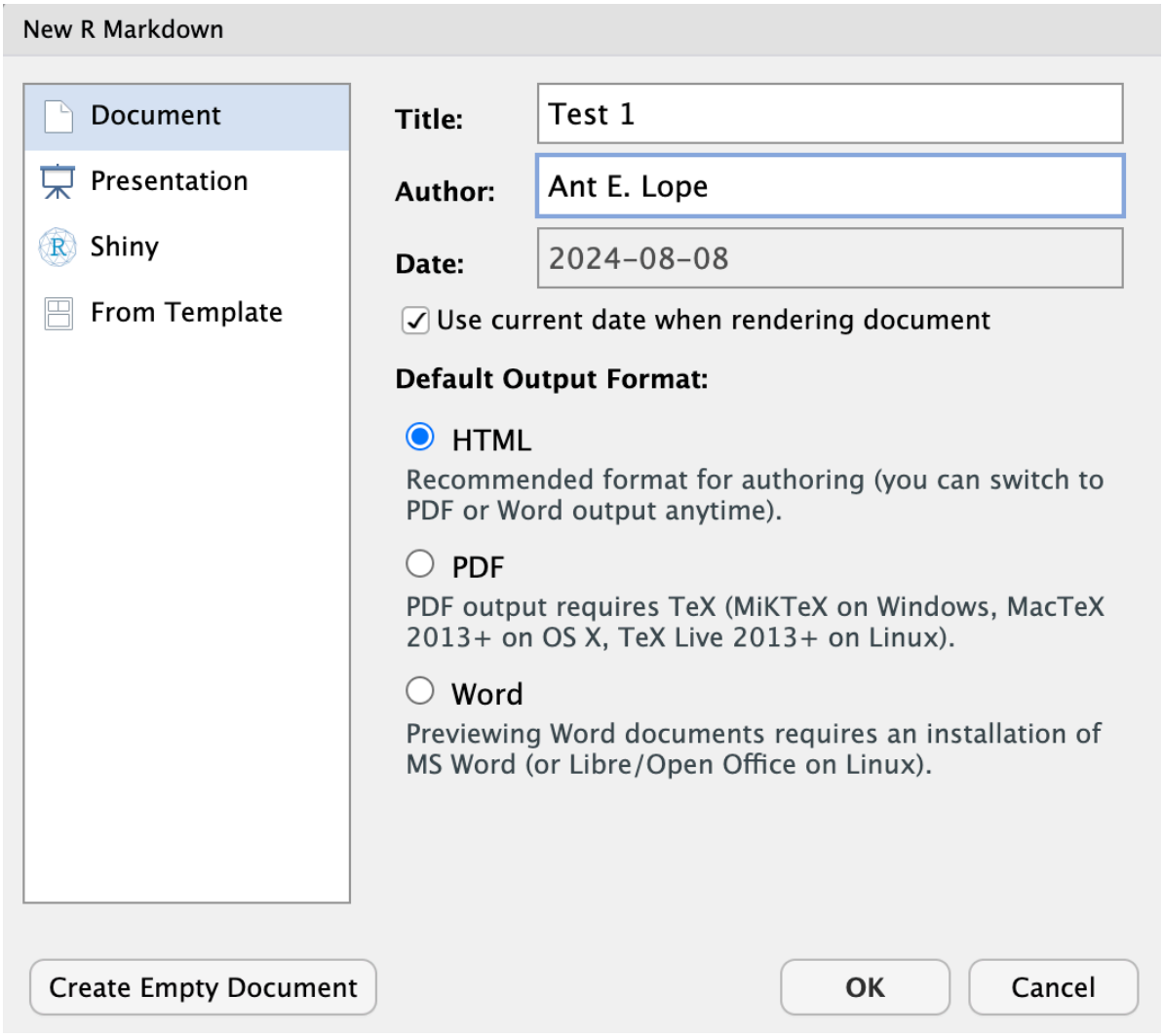


Figure 4.1: A new file window for an *RMarkdown* file.

After entering a title and your name and selecting `document` in the left hand menu, click `OK`.

Figure 4.2: An example of a markdown script.

In the image above, we can see what a "default" *RMarkdown* script looks like after creating the file. At the top of the document, between all of the dashes, we have the `yaml` header that tells *R* what kind of document will be created, who the author is, and tells it to use today's date. In this class, we will be saving documents as `html` as they are the easiest documents to create and save. These documents will include all of your code, text, and even any plots you may create!

Plain text in the document will be rendered as plain text in the document. (I.e., whatever you type normally will become "normal text" in the finished document). Lines preceded with `#` will become headers, with `##` being a second level header and `###` being a third level header, etc. Words can also be made italic by putting an asterisk on each side of the word (`*italic*`) and bold by putting two asterisks on each side (`**bold**`). URLs are also supported, with `<>` on each side of a URL making it clickable, and words being hyperlinked by typing `[words to show](target URL)`.

We also have code "chunks" that are shown above. A code chunk can be manually typed out or inserted by pressing `CTRL` + `ALT` + `I` (Windows, Linux) or `COMMAND` + `OPTION` + `I` (Mac). Everything inside a "code chunk" will be read as *R* code and executed as such. Note that you can have additional commands in the *R* chunks, but we won't cover that for now.

4.2 Using code chunks

In your computer, erase all information except for the `yaml` header between the dashes on your computer. `Save` your file in a folder where you want your assignment to be located. It is important you do this step up front as the computer will sometimes save in random places if you don't specify a file location at the beginning. *Don't forget to save your work frequently*!

````
This is a test of the *RMarkdown* code.

```{r}
print("Hello world!")
```
````

Figure 4.3: Text to type in your *Rmarkdown* document.

After typing this into the document, hit `knit` near the top of the upper left window. *R* will now create an HTML document that should look like this:

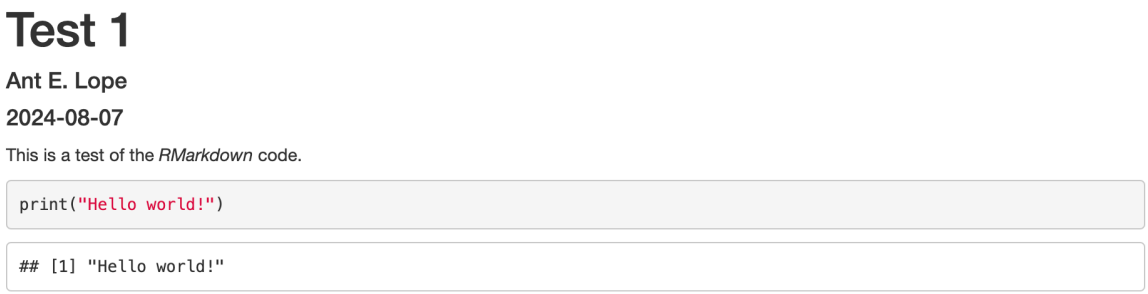


Figure 4.4: The output from the above code knitted into a document.

We can see now that the `HTML` document has the title of the document, the author's name, the date on which the code was run, and a greyed-out box with color coded *R* code followed by the output. Let's try something a little more complex. Create a new code chunk and type the following:

```
x <- 1:10
```

This will create a variable in *R*, `x`, that is sequentially each whole number between 1 and 10. We can see this by highlighting or typing only the letter `x` and running that line of code by clicking `CTRL` + `ENTER` (Windows / Linux) or `COMMAND` + `ENTER` (Mac).

```
x
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

If you look at the top right window, you will also see the value `x` in the environment defined as `int [1:10] 1 2 3 4 5 6 7 8 9 10`. This indicates that `x` is integer data spanning ten positions numbered 1 to 10. Since the vector is small, it displays every number in the sequence.

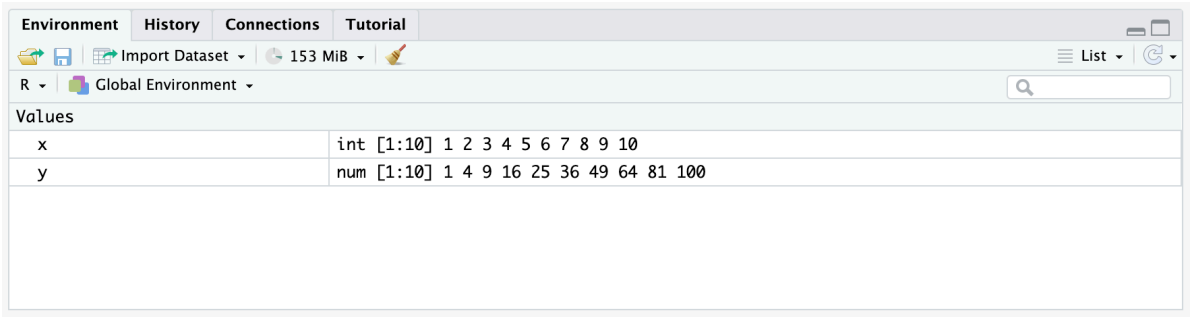


Figure 4.5: *RStudio* environment window showing saved objects. These are in the computer's memory.

Let's create another vector `y` that is the squared values of `x`, such that $y = x^2$. We can raise values to an exponent by using `^`.

```
y <- x^2
y
```

```
 [1]   1   4   9  16  25  36  49  64  81 100
```

Now we have the value `y` in the environment that is the square of the values of `x`. This is a `numeric` vector of 10 values numbered 1 to 10 where each value corresponds to a square of the `x` value. We can raise things to any value however, including x^x!

```
x^x
```

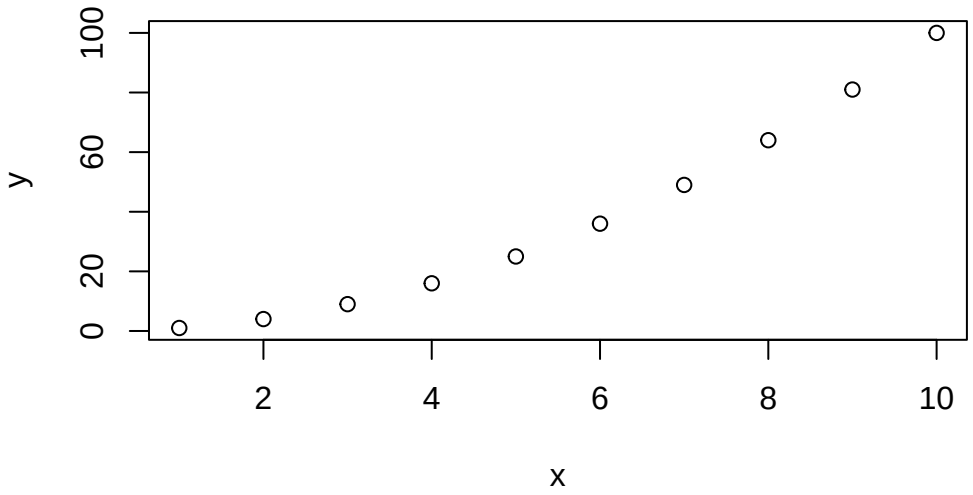
```
 [1]           1           4          27         256        3125       46656
 [7]      823543    16777216   387420489 10000000000
```

As we can see, since I didn't "store" this value as a variable in *R* using `<-`, the value is not in the environment.

4.3 Plotting

Now, let's try creating a plot. This is easy in *R*, as we just use the command `plot`.

```
plot(x = x, y = y)
```



By specifying the `y` and `x` components in `plot`, we can quickly generate a point plot. We can alter the visual parameters of this plot using a few different commands. I will outline these below with inline notes. Inline notes in the code can be made by using a `#` symbol before them, which basically tells *R* to ignore everything after the `#`. For example:

```
print("Test")
```

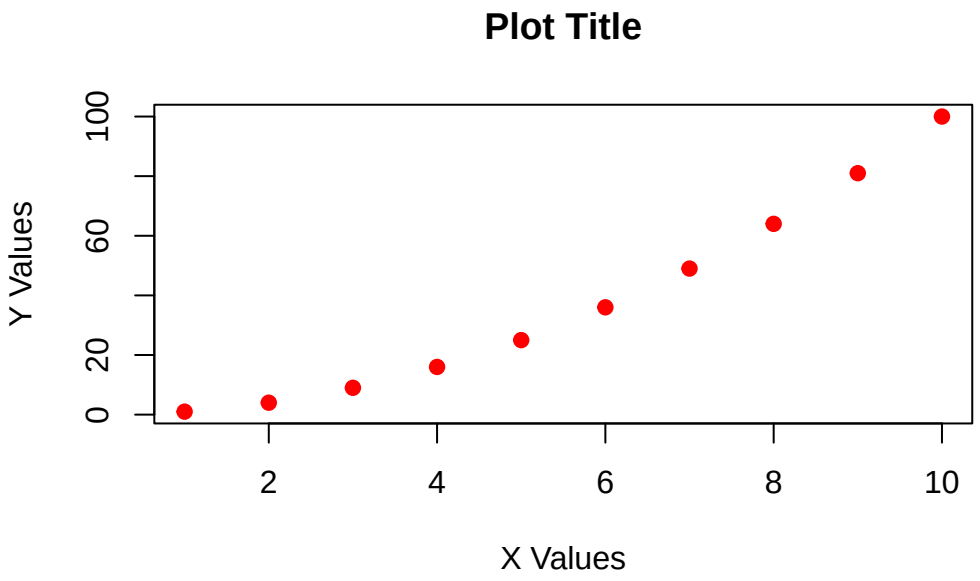
```
[1] "Test"
```

```
# print("Test 2")
```

This prints the word `Test`, but doesn't print `Test 2`.

Now let's make the plot with some new visual parameters.

```r
plot(x = x, # specify x values
     y = y, # specify y values
     ylab = "Y Values", # specify Y label
     xlab = "X Values", # specify X label
     main = "Plot Title", # specify main title
     pch = 19, # adjust point style
     col = "red") # make points red
```



5 Working with data

Throughout this course, we are going to have to work with datasets that are from our book or other sources. Here, we are going to work through an example dataset. First, we need to install *libraries*. A *library* is a collated, pre-existing batch of code that is designed to assist with data analysis or to perform specific functions. These *libraries* make life a lot easier, and create short commands for completing relatively complex tasks.

In this class, there are two libraries that you will need *almost every week*! First, we need to install the libraries. The main libraries we need for this class are:

1. `cURL`: this package allows us to download things from URLs. We will be using this to download data files. *Otherwise, you will have to enter all data by hand!*
2. `tidyverse`: this package is actually a group of packages designed to help with data analysis, management, and visualization.

```
# run this code the first time ONLY
# does not need to be run every time you use R!

# curl allows for internet downloads
install.packages("curl")

# tidyverse has a bunch of packages in it!
# great for data manipulation
install.packages("tidyverse")

# if you ever need to update:
# leaving brackets open means "update everything"
update.packages()
```

After packages are installed, we will need to load them into our *R* environment. While we only need to `install.packages` once on our machine, we need to load libraries *every time we restart the program*!

```
library(curl)
```

```
Using libcurl 8.7.1 with LibreSSL/3.3.6
```

```
library(tidyverse)
```

```
-- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
v dplyr     1.1.4     v readr     2.1.5
v forcats   1.0.0     v stringr   1.5.1
v ggplot2   3.5.1     v tibble    3.2.1
v lubridate 1.9.3     v tidyr     1.3.1
v purrr     1.0.2
-- Conflicts ------------------------------------------ tidyverse_conflicts() --
x dplyr::filter()     masks stats::filter()
x dplyr::lag()        masks stats::lag()
x readr::parse_date() masks curl::parse_date()
i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to beco
```

You should an output like the above. What this means is:

1. The core packages that comprise the tidyverse loaded successfully, and version numbers for each are shown.
2. The conflicts basically means that certain commands will not work as they used to because *R* has "re-learned" a particular word.

To clarify the conflicts, pretend that you can only know one definition of a word at a time. You may know the word "cola" as a type of soda pop or as a drink in general. However, in Spanish, "cola" refers to a line or a tail. While we can learn both of these definitions and know which one is which because of context, a computer can't do that. In *R*, we would then have to specify which "cola" we are refferring to. We do this by listing the package before the command; in this case, `english::cola` would mean a soda pop and `spanish::cola` would refer to a line or tail. If we just type `cola`, the computer will assume one of these definitions but not even consider the other.

We won't have to deal with conflicts much in this class, and I'll warn you (or help you) if there is a conflict.

5.1 Downloading data

Now, we need to download our first data set. These datasets are stored on GitHub.

6 Summary

In summary, this book has no content whatsoever.

```
1 + 1
```

```
[1] 2
```

References

Knuth, Donald E. 1984. “Literate Programming.” *Comput. J.* 27 (2): 97–111. https://doi.org/10.1093/comjnl/27.2.97.