Using R Markdown

Put your name here

Put the date here

Introduction

This module will teach you how to use R Markdown to create quality documents that incorporate text and R code seamlessly.

Are you in your project?

If you are looking at this document, that means you successfully completed the "Intro to R" module. At the end of that module, you created a new project called `Using_R_Markdown` and you should have placed this file in it. Let's make sure you're in your project.

Look at the upper right corner of the screen. Under your username, does it say `Using_R_Markdown`? If so, congratulations! You are in your project and you can safely skip down to the next section "What is R Markdown?"

If you're not in the `Using_R_Markdown` project, click on whatever it does say in the upper right corner (probably `Project: (None)`).

When you do the next step, this file will likely close and you'll have to re-open it again!

You can click "Open Project" but it's likely that the `Using_R_Markdown` project appears in the drop-down menu in your list of recently accessed projects. So click on the project `Using_R_Markdown` and then you'll have to re-open this file. . .

. . . and we're back, I hope.

What is R Markdown?

The first question should really be, "What is Markdown?"

Markdown is a way of using plain text with simple characters to indicate formatting choices in a document. For example, in a Markdown file, one can make headers by using number signs (or hashtags as the kids are calling them these days[1]). The Markdown file itself is just a plain text file. To see the formatting, one has to send the file through a process called knitting, which is described below.

R Markdown is a special version of Markdown that also allows you to include R code alongside the text.

Knitting a document

Going from the raw text of an R Markdown document to formatted output is called "knitting". There is a button in the toolbar right above the text that says "Knit". Go ahead and push it. See what happens.

Once the pretty output is generated, take a few moments to look back and forth between it and the original R Markdown text file (the plain text in RStudio). You can see some tricks that we won't need much (embedding web links, making lists, etc.) and some tricks that we will use everyday (like R code chunks).

[1] Also called "pound signs" or "octothorpes". This is also an example of formatting a footnote!

The PDF file will be the "pretty" output file that you submit when you turn in your assignment. At first, you'll want to work back and forth between the R Markdown file and the PDF file to get used to how the formatting codes in the plain text file get translated to output in the PDF file. After a while, you will knit to PDF less often and work mostly in the R Markdown file, only knitting to PDF when you are finished and ready to submit your final draft.

Literate programming

R Markdown is one way to implement a "literate programming" paradigm. The concept of literate programming was famously described by Donald Knuth, an eminent computer scientist. The idea is that computer programs should not appear in a sterile file that's full of hard-to-read, abstruse lines of computer code. Instead, functional computer code should appear interspersed with writing that explains the code.

Reproducible research

One huge benefit of organizing your work into R Markdown documents is that it makes your work *reproducible*. This means that anyone with access to your data and your R Markdown file should be able to re-create the exact same analysis you did.

This is a far cry from what generally happens in research. For example, if I do all my work in Microsoft Excel, I make a series of choices in how I format and analyze my data and all those choices take the form of menu commands that I point and click with my mouse. There is no record of the exact sequence of clicks that took me from point A to B all the way to Z. All I have to show for my work is the "clean" spreadsheet and anything I've written down or communicated about my results. If there were any errors along the way, they would be very hard to track down.[2]

Reproducibility should be a minimum prerequisite for all statistical analysis. Sadly, that is not the case in most of the research world. We are training you to be better.

Structure of an R Markdown document

Let's start from the top. Look at the very beginning of the plain R Markdown file. (If you're in RStudio, you are looking at the R Markdown file. If you are looking at the pretty PDF file, you'll need to go back to RStudio.) The section that starts and ends with three hyphens is called the YAML header. (Google it if you really care why.) The title of the document appears already, but you'll need to substitute your name and today's date in the obvious places. Scroll up and do that now.

You've made changes to the document, so you'll need to push the "Knit" button again. Once the knitting is done, look at the PDF document. The YAML header has been converted into a nicely formatted document header with the new information you've provided.

Next, there is some weird looking code with instructions not to touch it. I recommend heeding that advice. This code will allow you to answer questions and have your responses appear between pretty blue lines. You need to make sure there are blank lines before and after the `\answerbegin` and `\answerend` commands. Let's try it:

ANSWER

Replace this text here with something else. Then knit the document and see how it appears in the PDF file.

[2] If you think these errors are trivial, Google "Reinhart and Rogoff and Excel error" to read about the catastrophic consequences of seemingly trivial Excel mistakes.

To be clear, the colorful "answer" bars are not part of the standard R Markdown tool set. That's why we had to define them manually near the top of the file. Note that the weird code itself does not show up in the PDF file. It works in the background to define the blue bars that show up in the PDF file.

Next we have our first section header, which in the R Markdown file looks like `## Introduction`. The number signs are Markdown code for formatting headers. Observe:

Section header

Not quite as big

We could actually use a single number sign, but `#` makes a header as big as the title, which is too big. Therefore, I prefer to use `##` for section headers and `###` for subsections.

You do need to make sure that there is a blank line before and after each section header. To see why, look at the PDF document at this spot: ### Is this a new section? Do you see the problem?

Put a blank line before and after the line above that says "Is this a new section?" Knit one more time and make sure that the line now shows up as a proper section header.

Other formatting tricks

You can make text *italic* or **bold** by using asterisks. (Don't forget to look at the knit document to see the result.)

You can make bullet-point lists. These are also made with asterisks, but you'll need to start after a blank line, then put the asterisks at the beginning of each new line, followed by a space. If you want sub-items, indent four spaces and use a minus sign followed by a space. (Search online for syntax that will allow even deeper levels of nesting.)

- Item
    - Sub-item
    - Sub-item
- Item
- Item

Or you can make ordered lists. Just use numbers and R Markdown will do all the work for you. Sub-items work the same way as above. (Again, make sure you're starting after a blank line and that there is a space after the periods and hyphens.)

1. First Item
    - Sub-item
    - Sub-item
2. Second Item
3. Third Item

We can make horizontal rules. There are lots of ways of doing this, but I prefer a bunch of asterisks in a row.

There are many more formatting tricks available. For a good resource on all R Markdown stuff, click on this link for a "cheat sheet". And note in the previous sentence the syntax for including hyperlinks in your document.

R code chunks

The most powerful feature of R Markdown is the ability to do data analysis right inside the document. This is accomplished by including R code chunks. An R code chunk doesn't just show you the R code in your output file; it also runs that code and generates output that appears right below the code chunk.

An R code chunk starts with three "backticks" followed by the letter `r` enclosed in braces, and it ends with three more backticks. (The backtick is usually in the upper-left corner of your keyboard, next to the number 1 and sharing a key with the tilde ~.) In RStudio, click the little dark green, right-facing arrow in the upper-right corner of the code chunk below. (The icon I'm referring to is next to a faint gear icon and a lighter green icon with a downward-facing arrow.)

```r
# Here's some sample R code
test <- c(1, 2, 3, 4)
sum(test)
```

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

After pushing the dark green arrow, you should notice that the output of the R code appeared like magic. If you knit and look at the PDF output, you should see the same output appear. If you hover your mouse over the dark green arrow, you should see the words "Run Current Chunk". We'll call this the Run button for short.

We need to address something here that always confuses people new to R and R Markdown. A number sign (aka "hashtag") in an R Markdown document gives us headers. In R, however, a number sign indicates a "comment" line. In the R code above, the line `# Here's some sample R code` is not executed as R code. But you can clearly see that the two lines following were executed as R code. (Perhaps even more confusingly, the R output in the PDF file is preceded by two number signs.) So be careful! Number signs inside and outside R code chunks behave very differently.

Typically, the first code chunk that appears in our document will load any packages we need. We will be using a package called `mosaic` throughout the course. This package was designed by stats educators to provide some useful tools for teaching statistics to undergraduates. We load it now. Click on the Run button (the dark green, right-facing arrow) in the code chunk below.

```r
library(mosaic)
```

```
## Loading required package: dplyr

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: lattice

## Loading required package: ggformula

## Loading required package: ggplot2

##
## New to ggformula?  Try the tutorials:
##  learnr::run_tutorial("introduction", package = "ggformula")
##  learnr::run_tutorial("refining", package = "ggformula")
## Loading required package: mosaicData
```

```
## Loading required package: Matrix
```

```
##
## The 'mosaic' package masks several functions from core packages in order to add
## additional features.  The original behavior of these functions should not be affected by this.
##
## Note: If you use the Matrix package, be sure to load it BEFORE loading mosaic.
```

```
##
## Attaching package: 'mosaic'
```

```
## The following object is masked from 'package:Matrix':
##
##     mean
```

```
## The following objects are masked from 'package:dplyr':
##
##     count, do, tally
```

```
## The following objects are masked from 'package:stats':
##
##     binom.test, cor, cor.test, cov, fivenum, IQR, median,
##     prop.test, quantile, sd, t.test, var
```

```
## The following objects are masked from 'package:base':
##
##     max, mean, min, prod, range, sample, sum
```

The output here is very messy because it just consists of a bunch of messages that are generated when trying to load the package. (These are not errors, but you have to read the message carefully to know that; errors will also appear as red messages below the code chunk. Usually, errors appear with the word "Error", so it's typically clear.) Let's change the code chunk in RStudio to look like this instead. (As before, press the Run button.)

```r
library(mosaic)
```

The `warning` and `message` flags in the code chunk header control whether we see the annoying messages that are generated in the output. Keep in mind that this is one of the few times we will actively try to suppress the output. In general, the whole point is to see both the R code and its output simultaneously.

Okay, let's do something interesting now. We'll take a look at the `mtcars` data set. First, let's explore what happens when we push the Run button in a code chunk containing several commands that generate output.

```r
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```r
tail(mtcars)
```

```
##                 mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

```r
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```r
summary(mtcars)
```

```
##       mpg             cyl             disp             hp       
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0  
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
##       drat             wt             qsec             vs        
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
##        am              gear            carb      
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
##  Median :0.0000   Median :4.000   Median :2.000  
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812  
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

If you're looking at this in RStudio, it's a big mess. RStudio did its best to give you what you asked for, but there are four separate commands here. The first two (`head` and `tail`) print some of the data, so the first two boxes of output are tables showing you the head and the tail of the data. The next two (`str` and `summary`) normally just print some information to the Console. So RStudio gave you an R Console box with the output of *both* of those commands.

If you look at the PDF file, you can see the situation isn't as bad. Each command and its corresponding output appear nicely separated there.

Nevertheless, it will be good practice and a good habit to get into to put multiple output-generating commands in their own R code chunks. Run the following code chunks and compare the output to the mess you saw above:

```r
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```r
tail(mtcars)
```

```
##                 mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

```r
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```r
summary(mtcars)
```

```
##       mpg             cyl             disp             hp       
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0  
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
##       drat             wt             qsec             vs        
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
##        am              gear            carb      
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
##  Median :0.0000   Median :4.000   Median :2.000  
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812  
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

This won't look any different in the PDF file, but it sure looks a lot cleaner in RStudio.

What about the two lines of the first code chunk we ran above?

```
test <- c(1, 2, 3, 4)
sum(test)
```

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

Should these two lines be separated into two code chunks? If you run it, you'll see only one piece of output. That's because the line `test <- c(1, 2, 3, 4)` works invisibly in the background. The vector `test` gets assigned, but no output it produced. Try it and see (push the Run button):

```
test <- c(1, 2, 3, 4)
```

So while there's no harm in separating these lines and putting them in their own chunks, it's not strictly necessary. You really only need to separate lines when they produce output. (And even then, if you forget, RStudio will kindly give you multiple boxes of output.)

Working at the Console

One thing that gives new R users headaches is understanding the difference between working in an R Markdown file and working at the Console.

It's important to know that an R Markdown document creates variables in a completely separate space called an *environment*. This environment is different from the Global Environment that you see in the Environment pane in the upper right.

For example, suppose we define a new variable called `test2` in a code chunk. FOR PURPOSES OF THIS EXERCISE, DO NOT HIT THE RUN BUTTON YET! But do go look at the PDF file.

```
test2 <- c("a", "b", "c")
test2
```

```
## [1] "a" "b" "c"
```

The first line defines `test2` invisibly. The second line just asks R to print the value of `test2`, so in the PDF file, we see `"a" "b" "c"` just as we expect.

Okay, now go to the Console in RStudio (in the lower left corner of the screen). Try typing test2. You should get an "Error: object 'test2' not found."

Why does this happen? Remember that R Markdown has stored the `test2` variable in its environment, but the Global Environment doesn't know about it yet. Look in the upper right corner of the screen, under the "Environment" tab. You should see `test`, but not `test2`.

Okay, NOW GO BACK AND CLICK THE RUN BUTTON IN THE LAST CHUNK ABOVE. The output appears in RStudio below the code chunk and the Global Environment has been updated.

The take home message is this:

Be sure to run all your code chunks in RStudio!

If you forget, everything will still work fine in the R Markdown file, but working interactively in the Console might cause things to break.

One more bit of weirdness to explore. Run the following code chunk:

```
test3 <- "Hello!"
test3
```

```
## [1] "Hello!"
```

You can see the output here and you can see the `test3` variable defined in the Global Environment.

Now go to the Console and type the following:

test3 <- “WTF?”

What do you think is going to happen when you run the following code chunk? Make a guess in your head, and then try it to find out.

```
test3
```

```
## [1] "Hello!"
```

Okay, so that sort of makes sense. We redefined `test3` to be “WTF?” and the above output in RStudio reflected that. But now go find this spot in the PDF file.

It still says “Hello!” What’s going on?

Again, remember that the R Markdown file creates its own environment, and that environment is separate from the Global Environment. When you knit the PDF, that process doesn’t look at the Global Environment at all. It only looks at code chunks here in the R Markdown file. The last place it can find a definition of `test3`, it said

```
test3 <- "Hello!"
```

So how do we avoid problems like this? In RStudio, look in the toolbar above this document, toward the right. You should see the word “Run” with a little drop-down menu next to it. Click on that drop-down menu and select “Run All”. Do you see what happened? All the code chunks ran again, and that means that anything in the Global Environment will now be updated to reflect the definitions made in the R Markdown document.

It’s a good idea to “Run All” when you first open a new R Markdown document. This will ensure that all your code chunks have their output below them (meaning you don’t have to go through and click the Run button manually for each chunk, one at a time) and the Global Environment will accurately reflect the variables you are using.

You can “Run All” from time to time, but it’s easier just to “Run All” once at the beginning, and then Run individual R code chunks manually as you create them. It’s also a good idea to “Run All” at the end of your session as well to make sure you haven’t introduced any inadvertent errors.

If you get really screwed up and if it seems like the R Markdown code isn’t in line with what you see in the Global Environment, you can always clear the Global Environment. Look in the Environment tab and find the icon that looks like a broom in the toolbar. Click it now and confirm that “Yes” you want to remove all objects from the environment.

Now try to run the following code chunk:

```
test
```

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

You should get an error. But if you knit to PDF right now, it will work just fine. (Again, think about it and make sure you understand why.)

Go to the Run menu and “Run All”. Not only does your Global Environment repopulate correctly, but the code chunk above no longer shows an error.

Inline R commands

You don't need a standalone R code chunk to do computations. One neat feature is the ability to use R to calculate things right in the middle of your text.

Here's an example. Suppose we wanted to compute the mean mpg for the cars in the `mtcars` dataset. We could do this, of course:

```r
mean(mtcars$mpg)
```

```
## [1] 20.09062
```

But we can also do this inline by using backticks and putting the letter `r` inside the first backtick. Go to the PDF document to see how the following sentence appears:

The mean mpg for cars in the `mtcars` dataset is 20.090625.

You can (and should) check to make sure your inline R code is working, but you don't necessarily need to go to the PDF file to find out. In RStudio, click so that the cursor is somewhere in the middle of the inline code chunk in the paragraph above. Now type Ctrl-Enter or Cmd-Enter (PC or Mac respectively). A little box should pop up that shows you the answer!

Notice that in addition to the inline R command that calculated the mean, I also enclosed `mtcars` in backticks to make it stand out in the output. I'll continue to do that for all computer commands and R functions. But to be clear, putting a word in backticks is just a formatting trick. If you want inline R code, you also need the letter r followed by a space inside the backticks.

Exporting the PDF file

When you are all done with these R Markdown assignments, you will need to submit the PDF back to your professor (probably in Canvas). Remember, though, you are working in the cloud. The PDF file that was generated is not living on your personal computer. It appears in a window spawned by some distant server. (And by "distant", I mean the basement of the Giovale Library.)

So how do we get files off the server and onto our personal computers?

It's especially tempting to use what appears to be a functional "Download" button right there in the PDF viewer. This is the first place students try to go, but this doesn't seem to work consistently!

Instead, close the PDF viewer and go back to the Files pane of RStudio Server. Check the box next to the name of the file you want to export. In this case, it will be `Using_R_Markdown.pdf`. (Make sure you select the PDF file, not the Rmd file!)

The toolbar in the Files pane has buttons for "New Folder", "Upload", "Delete", "Rename" and "More". Click on "More". In the drop-down menu, select "Export". In the dialog box that pops up, leave the file name alone and click "Download". This will place the file back in the "Downloads" folder on your personal computer. From here, you'll follow the instructions of your professor for how to submit the PDF file for grading. (It will likely be an assignment in Canvas.)

I cannot emphasize this enough:

If you do this wrong, you run the risk of producing a blank or otherwise corrupted file. If you submit that file, there is no way for your assignment to be graded, so you will receive no credit!

Do not use the "Download" utility in the PDF viewer! Close the PDF viewer and Export the file through RStudio as explained above!

Copying and pasting

In future assignments, you will be shown how to run statistical analyses using R. The first part of each assignment will give extensive explanations of the statistical concepts and demonstrations of the necessary R code. The latter part of the assignment will be one or more exercises that ask you to apply your new-found knowledge to run similar analyses on your own with different data.

The idea is that you should be able to copy and paste the R code from the previously worked examples. But you must be thoughtful about how you do this. The code cannot just be copied and pasted blindly. It must be modified so that it applies to the exercises with new data. This requires that you understand what the code is doing. You cannot effectively modify the code if you don't know which parts to modify.

There will also be exercises in which you are asked to provide your own explanations and interpretations of your analyses. These should **not** be copied and pasted from any previous work. These exercises are designed to help you understand the statistical concepts, so they must be in your own words, using your own understanding.

In order to be successful in these modules, you must do the following:

1. **Read every part of the module carefully!**
    - It will be tempting to skim over the paragraphs quickly and just jump from code chunk to code chunk. This will be highly detrimental to your ability to gain the necessary understanding—not just to complete the assignment, but to succeed in statistics overall.
2. **Copy and paste thoughtfully!**
    - Not every piece of code from the early part of the assignment will necessarily apply to the later exercises. And the code that does apply will need to be modified (sometimes quite heavily) to be able to run new analyses. Your job is to understand how the code works so that you can make changes to it without breaking things. If you don't understand a piece of code, don't copy and paste it until you've read and re-read the earlier exposition that explains how the code works. And if you're still confused, ask for help.

One final note about copying and pasting. Sometimes, people will try to copy and paste code from the PDF output file. This is a bad idea. The PDF document uses special characters to make the output look pretty, but these characters don't actually work as plain text in an R markdown document. The same applies to things copied and pasted from a Word document or a website. If you need to copy and paste code, be sure to find the plain text R Markdown file (the one with the .Rmd extension here in RStudio) and copy and paste from that.

Conclusion

That's it! There wasn't too much you were asked to do for this assignment that will actually show up in the PDF output. (Make sure you did do the three things that were asked of you however: one was adding your name and the date to the YAML header, one was typing something between the blue answer bars, and the last was to make a section header appear properly.) As you gain confidence and as we move into more serious stats material, you will be asked to do a lot more.

Now you know how to use an R Markdown file to generate pretty output files. If you look in your project folder, you should see three files. One is the `.Rproj` file that you were instructed never to touch. The one with extension `.Rmd` is your R Markdown file. It is really nothing more than a text file; you could open it up in the most basic notepad program on your computer with no trouble. Then you also have a `.pdf` file that is the pretty output file generated when you knit to PDF. (There may also be other internal files, perhaps starting with a period, in the folder, but you can ignore those too.)

Don't forget to Export the PDF file using RStudio (**not the Download button in the PDF viewer**) so that you can turn it in.