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.

If not, click on whatever it does say there (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). 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 about the text that says "Knit PDF". 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. 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).

From here on out, work back and forth between the R Markdown file and the PDF file generated as output. Start getting used to how the formatting codes in the plain text file get translated to pretty output in the PDF file.

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.[1]

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. 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 PDF" button again. Once the knitting is done, look at the PDF document again. 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 in pretty blue boxes. Let's try it:

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

To be clear, this "answer" environment is not part of the standard R Markdown tool set. That's why we had to define it 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 `answer` environment that does 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:

[1] 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.

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.

Other formatting tricks

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

You can make bullet-point lists. These are also made with asterisks, but at the beginning of a 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.

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 the syntax for including hyperlinks in your document in the raw R Markdown file.

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 when you knit the file.

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 ~.) Observe:

```r
# Try some R code in here
test <- c(1, 2, 3, 4)
sum(test)
```

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

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 `# Try some R code in here` is not executed as R code. But you can clearly see in the PDF output that the following two lines were executed as R code. Perhaps even more confusingly, the R output 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:

```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: ggplot2

## 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.

##
## 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, cov, D, 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
```

Take a look at the PDF document. As advertised, you see not only the R code, but also the result of running the R code. In this case, the output is very boring because it just consists of messages that are generated when trying to load the package. Let's change the code chunk to look like this instead:

```
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. Let's look at the `mtcars` data set:

```
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
```

```
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
```

```
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 ...
```

```
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
```

The commands are shown along with the output they generate.

Environments

You may or may not have noticed something odd above. A few code chunks back, we defined a new vector using the code `test <- c(1, 2, 3, 4)`. From our experience in the Intro to R module, we would expect that `test` would appear in the Global Environment. Look over to the Environment pane. The variable `test` is not present. Go to the Console and type `test` to verify that, indeed, R seems to know nothing about `test`. So how did the next line of code `sum(test)` work if R knows nothing about `test`?

The answer is that the R Markdown document creates its own separate environment, completely apart from the Global Environment that you see in the Environment pane in the upper right.

This causes people who are new to R a lot of headaches. Here's a common scenario: suppose I am working in an R Markdown document and a while ago I defined a vector called `test`. I have now forgotten what `test` contains, so I want to see it. I could type a new R code chunk in my R Markdown document with test in it, like so:

```
test
```

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

Looking at the PDF output, that certainly does the trick. But that's kind of a hassle. I want to check `test` quickly, so I type `test` at the Console. But I get an error because `test` is not in the Global Environment. It's only created when I knit the R Markdown document, and then it's only available to code chunks later in the R Markdown document.

There is a way to work interactively at the Console with objects defined in the R Markdown document. Look at the toolbar above this document in RStudio, off to the right of the "Knit PDF" button. There is a "Run" button. Click the drop-down arrow to see the options. One can "run" one or more code chunks from this document and then that code will run in the Global Environment. Try clicking on "Run All".

You'll see two interesting things. First, `test` is now part of the Global Environment, so now you can work with it at the Console it you'd like. Of course, remember that anything you do at the Console will not be reflected in the R Markdown document. So if you do something interesting and you want to keep it and execute it as part of the code in the R Markdown document, you need to create a code chunk and type the command there. Second, you can also see in the Console that all the other code ran too and its output appears in the Console.

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 I wanted to compute the mean mpg for the cars in the `mtcars` dataset. I could do this, of course:

```
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.

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.

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.)

Conclusion

That's it! 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.)