Writing with RMarkdown

Francis L. Yuen

2019-12-18

Learning goals:

1. Why we should use RMarkdown
2. How to combine R with your manuscript
3. How to automatically generate APA formatted papers

Before we start, we need to install a few packages:

```
install.packages("devtools")
install.packages("knitr")
devtools::install_github("crsh/papaja")
```

When you install these, it may ask if you want to update some of your current packages. I recommend that you **do not** update anything as it may create problems that I cannot fix for you…

You will also need to install **TeX**. Here are the versions I recommend:

For Windows:

MikTeX

For Macs:

MacTeX

Why use RMarkdown?

What is RMarkdown anyways?

RMarkdown is a special file format that lets you generate cool documents. It is much less user friendly than something like Word, but it is also **much more powerful** once you get over the learning hump.

Pros	Cons
- Automatically does APA formatting for you	- Very spooky at first
- Automatically cites and generates a reference page	- Can run into problems you don't know how to solve
- The same file can output PDF and Word files	- You might be too cool and intimidate your peers
- Reproducible!	

The best way to learn RMarkdown, just like R, is to actually practice. Let's create a new markdown file by going to File -> New File

-> RMarkdown. Enter any title you want and put your name as the author.

When you press 'Knit' for the first time, it will ask you where you want to save your RMarkdown file. Remember the reproducible workflow from before? Make a folder for where you want to store this markdown file AND all your data.

The Basics

There are many parts to an RMarkdown file. The first one is the header, which is the part at the top sandwiched between the `---`. There are a ton of fun things you can put here. For now, press the 'Knit' button and see what happens.

Now, change where it says `html_document` to `word_document` and knit again. Pretty cool, right?

Change it back to `html_document` for now. Notice how there are some #'s infront of certain text? What did those do when you knitted your document?

One single # changes the text that follows it into a first-level heading. Two ## changes it to a second-level heading. Utilizing these will allow you create organized and dynamic documents and reports. For example, make some random headings, then adjust your header section so it looks like this:

The indentation has to be correct here. Each level is separated by one `tab` indent, or two spaces

```
output:
  html_document:
    toc: true
```

Pretty neat, right? Now change it to this:

```
output:
  html_document:
    toc: true
    toc_float: true
```

Note: Different file formats will have different header options.

There are tons of other fancy things you can do. Check out **this link** for a comprehensive guide!

You can also do other things like:

- *Italicize* by doing `*this*` and **bold** by doing `**this**`
- Make lists by starting a new line with `-` followed by a space
- Hyperlink using this format: `[whatever text you want](actual link)`

Exercise 1

Try to reproduce this list:

- Everyone should use RMarkdown
- RMarkdown is *very* **cool**
- You can **Google** everything you need to know

How to combine R with your manuscript

So far we saw how to do things that word can do but in RMarkdown instead. That seems a bit silly: why wouldn't we just do all of that in Word? One of the biggest benefits of RMarkdown is that you can imbed Rcodes within your manuscript. That means instead of making a graph in excel, saving the graph, then pasting that graph in Word, you can just have it ready to go in the document, which comes with several advantages:

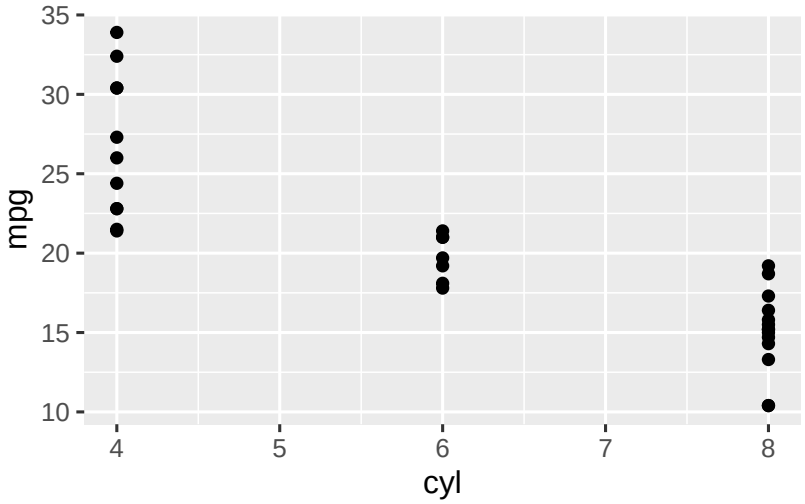
1. Someone looking at your RMarkdown file can know exactly how you generated the graph and where the data came from
2. You can make minor adjustments to your graph easily by changing the code
3. If the raw data ever changes (e.g. you collected more data and/or there were mistakes in the raw data), you **don't** have to remake the graph because the code is there!

There are two ways to imbed code: inline code and code chunks. Inline codes are generally used for small recalls that you want within your text (e.g. recalling statistical test results and *p*-values.) This is extremely useful for preventing mistakes: instead of copy and pasting results from SPSS and maybe accidentally copying the wrong thing, you are literally asking R to write the results for you. We will get into inline codes a bit later when we discuss how to write APA formatted papers. For now, we will start with **code chunks**, which look like this:

```
```{r}
```
```

That symbol is NOT an apostrophe; the button is usually under the Esc key. You can also press `Ctrl + Alt + I` to insert a code chunk.

The space between is where you write your R script, and RMarkdown treats it as such. For example, let's plot something simple from `mtcars`:

```
```{r}
ggplot(mtcars, aes(x = cyl, y = mpg)) +
geom_point()
```
```

If you press knit, what happens?

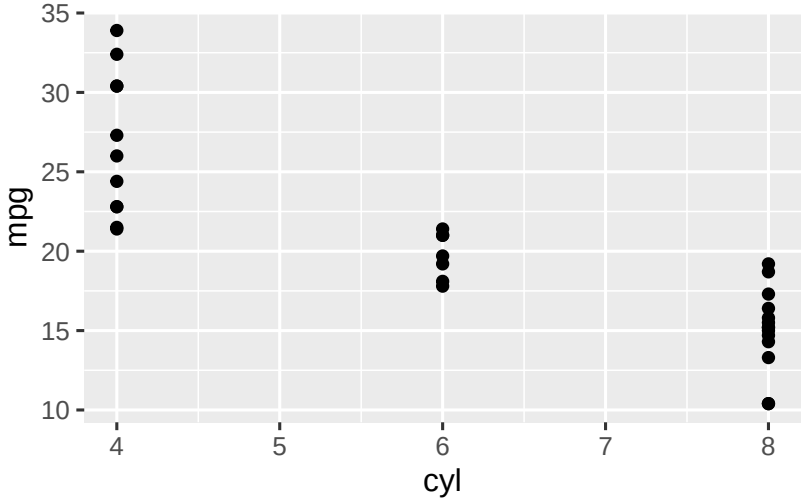
```
ggplot(mtcars, aes(x = cyl, y = mpg)) + geom_point()
```



This is not too bad for a report since everyone can see your code, but certainly not viable for a manuscript. We can add `chunk options` to each code chunk to customize how we want each section to be displayed. Here are some common options:

- `echo` = FALSE prevents code from displaying
- `include` = FALSE prevents code AND results from displaying (useful for a code chunk that just loads your packages)
- `fig.width` and `fig.height` lets you adjust your plot's dimensions
- `warning`, `error`, and `message` lets you hide error messages that R might spit out

Now add this chunk option to your code chunk like this:
"'{r echo = FALSE}
If you knit now, you should only see the plot:



It's annoying to have to set chunk options everytime. It's very useful for manuscripts to have a default determined right from the

start. A useful function for that is:

```
knitr::opts_chunk$set(fig.width = 8, fig.height = 5,
    echo = FALSE, warning = FALSE, message = FALSE,
    cache = FALSE)
```

This sets a default code chunk setting for all of your code chunks in the document, and you can override it in each individual chunk. Add this to your `setup` chunk at the start of the document.

Generating APA-formatted papers

We're in the end-game now. Learning all of these things is ultimately so we can knit a fully reproducible manuscript that is APA-formatted. If you have `papaja` installed, you're already half way there. Start up a new markdown file again, but this time choose `From Template`, there should be an option for APA 6th edition.

You probably noticed that we now have **a lot** more stuff in our header. Don't worry about these for now, they are there for customization. We will only cover three core aspects of writing a manuscript in RMarkdown: Bibliography, imbedding graphs, and recalling statistics.

Bibliography

This might be already familiar if you already use a reference manager. If you're still manually typing in all your references (no shame, I did that just over a year ago), this will be life changing. First, you need to install your favorite reference manager. Bibdesk is very popular among Mac users, I personally use Jabref because I have a PC. Choose whatever you want as long as you save the resulting .bib file within your Rproject.

Using your reference manager, create a new .bib file and add a few citations in there. If you would prefer to figure out how to use a reference manager later, you can skip this step for now and use my library for now, which can be downloaded here. Make sure to save your library in the same folder as your markdown file.

Go back to your APA styled markdown file. In the header section, change the bibiography to whatever you named your .bib file (if you used the sample one, it will be `References.bib`). Now, delete the sample abstract and write a sentence like this:

@Hamlin2007's study is very cool, but some other studies are just as cool [@Hamlin2011].

Knit the document and see what happens. Not only does it do the in-text citation correctly for you, it also adds it to your reference list

Note: Every reference manager assigns different `code names` to each paper. I am unsure how each of them format it, but usually it's LastnameYEAR.

at the end. Now you no longer have to carefully check that everything you cite in the paper is in your bibliography!

Imbedding figures

As we practiced earlier, using a code chunk with `echo` set to `FALSE` will allow you to output a graph. What does that look like using the APA template?

If you choose to use a previous dataset, make sure to follow best practices of workflow and save the dataset in the correct place to read in!

Also don't forget to add library(tidyverse) to the setup chunk near the beginning of the document!

Exercise 2

Create a graph of your choice from either one of the sample datasets (`mtcars` or `iris`), or from a sample dataset we used in earlier workshops. Put the code chunk in the **results** section, and write a few sentences explaining the results.

Sample answer

```
data <- mtcars
data <- data %>% group_by(gear) %>% summarize(mean_hp = mean(hp))
data$gear <- as.factor(data$gear)
ggplot(data, aes(x = gear, y = mean_hp)) + geom_col(position = position_dodge(0.9)) +
    theme_apa()
```

Adding captions is an important part of making good graphs, and you can make captions easily by modifying the code chunk with a `fig.cap=` option like this:

"'{r fig1, fig.cap = "\label{fig:fig1}This is my practice plot for Exercise 2."}

The fig1 right after the r is the 'name' of the chunk. These must be unique and helps you keep track of what each chunk does. the fig:fig1 part of fig.cap identifies which chunk it should reference when it makes the caption.

Exercise 2.5

Add a caption to your graph from Exercise 2.

We can do the same thing with tables. The `kable()` function is a nice start, although if we want an APA formatted table, we should use `apa_table()`. Let's reuse the data you made earlier for Exercise 2, except turn it into a table this time:

"'{r results = 'asis'}
apa_table(data)
"'

`apa_table()` is weird and you have to have the results = 'asis' as a code chunk option in order for it to work. Not a good idea to set this as a default because it messes up many other common functions, so it's better to manually put it in every time you have a code chunk that uses `apa_table()`

Referencing statistics

Note: These are not necessarily the most appropriate tests, and are only chosen for their simplicity to demonstrate functions in RMarkdown.

One of the most common mistakes in manuscripts is writing the strong number/statistic. It sounds silly, but it's something that can slip under the radar easily especially if no one else knows what exact details of your analyses (or if your raw data changes and you accidentally forget to change one number in your manuscript). Thankfully, we can

avoid these problems with RMarkdown. Let's illustrate this with some simple tests.

We will examine the `iris` dataset and hypothesize that different species have different sepal width. Let's first do some descriptives:

```
d <- iris
d_summary <- d %>% group_by(Species) %>% summarize(mean_sepal_width = mean(Sepal.Width))
```

Looks like there is a difference between the means. A simple 1x3 ANOVA will help us figure out if there is an effect:

```
d <- iris
d_summary <- d %>% group_by(Species) %>% summarize(mean_sepal_width = mean(Sepal.Width))
anova <- summary(aov(Sepal.Width ~ Species, data = d))
```

Call `anova` in your console. Looks good! How do we format this so we can reference specific numbers? The `apa_print()` function is handy here. It will take statistical results and reformatted to be easily incorporated into the manuscript:

Note: Not ALL analyses are supported right now. Google the documentation for `apa_print()` to see a full list of supported analyses.

```
anova_apa <- apa_print(anova)
```

Now we can use **inline code chunks** to call specific parts of the anova. To add an inline code chunk, you start with ` r and close it with a `, similar to code chunks except you only use one `.

Click on `anova_apa` or use `View(anova_apa)`. You will notice several options for you to reference. The ones that are most commonly referenced are the `statistics` and `full_result`. Let's say I want to report all of the results, this is what it would look like:

> A 1x3 ANOVA with sepal width as the dependent variable and iris species as the independent variable revealed a significant main effect of species (` r anova_apa$full_result`)

Don't worry about text size here. The format that this document uses is not APA. When you use the APA template everything will look fine.

What does that look like when you knit it?

> A 1x3 ANOVA with sepal width as the dependent variable and iris species as the independent variable revealed a significant main effect of species ($F(2, 147) = 49.16$, $MSE = 0.12$, $p < .001$, $\hat{\eta}^2_G = .401$)

We can also spit out a nice ANOVA table using what we learned earlier:

The labeling is messed up here because this is not an APA formatted paper.

```
apa_table(anova_apa$table, caption = "ANOVA table for demonstration purposes",
    note = "What a nice looking table")
```