Reproducible Reports with RMarkdown

Due: November 19, 2019

Why R Markdown?

R Markdown provides a framework for authoring dynamic documents, useful across scientific disciplines. An R Markdown file is a record of your research! It contains:

- the code that a scientist needs to reproduce your work *and*
- the narration that a reader needs to understand your work

Link to Markdown Video

R Markdown Setup

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

When you create a new R Markdown document, a pop-up box will appear, allowing you to create a title to your document, input your name (or group names), and select what type of file output you would prefer (Word, pdf, HTML).

- The title, author(s), and output option you selected will appear at the top of the page in what is called the YAML header. This header is everything between the top `---` and the bottom `---`.
- By default it will also add a line for today's date (the day you created the document).
- If you wish to knit to a pdf, Markdown requires that you have a LaTeXcompiler (like MiKTeX), **unless** you knit on the RStudio Cloud.
  - See the latex.pdf in the Resources folder for more info on using LaTeX!

Compiling a Markdown Document

Typically an R Markdown file contains a combination of text (with markdown formatting) and R code chunks. To compile, the .Rmd file is fed to `knitr` the program that executes the R code chunks and creates a new markdown (.md) document that includes the R code and it's output.

Then, the markdown file (.md) generated by `knitr` is then processed by `pandoc`, the program that creates the finished product (web page, PDF, Word document, slide show, handout, book, dashboard, package vignette, etc.). The figure below shows the compiling pathway a Markdown document goes through to get you your report.

This may look complicated, but you do not have to do any of this process on your own. Instead R Markdown makes it extremely simple by encapsulating this entire process into a single render function (that is attached to a single "Knit" button).

Figure 1: Compiling in RMarkdown

Part 1

(a) Create your own `R` Markdown document, with the title "Lab 11" and include your group members' name. Select the HTML option for compiling.
(b) Click the "Knit HTML" button!
(c) Save the .Rmd to your Lab 11 Workspace with an intuitive name!

Components of an `R` Markdown file

As you can see, there are some interesting parts to the template that we got. Let's walk through some of the components!

Typeface within Markdown

It is perhaps the simple Markdown typeface formatting that has swayed a large portion of data analysts to convert. Working with a Markdown document is very simple!

- Your sections are declared using `#` signs, where `#` indicates a main header, `##` a sub-header, and so on.
- You can begin a new line by indenting (hitting the space bar) twice before returning.
- You can make a word italicized by using either single underscores (`_`) on either side (`_word_`) *or* single asterisks (`*`) on either side (`*word*`).
- You can make a word boldface by using either double underscores (`__`) on either side (`__word__`) *or* double asterisks (`**`) on either side (`**word**`).

Part 2

(a) Change the names of the section headers to the sections you **must** include in any statistical report your write in this course!
(b) If there are subsections within a section (e.g. Exploratory Data Analysis lies within Statistical Procedures Used)
(c) Knit the Rmd to an HTML to make sure your changes look correct!

`R` Code within Markdown

The Markdown file we started with contained three `R` code chunks. You can tell they are `R` code chunks by the `r` at the beginning of the chunk (between the `{` and `}`).

The Markdown file knows that these blocks of code, often referred to as code chunks, are `R` code, because the chunk begins with

```

and ends with

```

However, we notice that each of these chunks contains things other than the `{r}` that we often see in our `R` code. Let's break these components apart:

- `include`: TRUE or FALSE, where FALSE indicates that the chunk will not be included in the document after it is run
- `echo`: TRUE or FALSE, where TRUE indicates the code **will** be output

Part 3:

(a) Load in the packages that we typically use in the `setup` code chunk.
(b) Knit the Rmd to an HTML.
(c) **Do any messages or warnings output from loading in these packages?**
(d) **Why or why not?**

There are other options for the code chunks that are not included in our template .Rmd file. Some of these options are:

- `eval`: TRUE or FALSE, where TRUE indicates that the code **will not** be executed
- `message`: TRUE or FALSE, where FALSE indicates that **no** messages from the code will be output
- `warning`: TRUE or FALSE, where FALSE indicates that **no** warnings from the code will be output
- `results`: modify the placement of the code output,
  - 'hide' - do not display results
  - 'hold' - put all results below all code
- `fig.width` & `fig.height`: modify the dimensions of plots (in inches)
- `fig.align`: modifies the alignment of the plots/output ('left', 'right', or 'center')
- `fig.cap`: adds a caption to the plots as a character string (e.g. "Caption")
- `tidy`: TRUE or FALSE, where TRUE modifies the code output to **not** run off the page

Part 4:

(a) Change the third code chunk to be a scatterplot of speed versus distance, using `ggplot`.

(b) Make sure you include:

- nice axis labels on your plot (units are great!)
- a figure that is centered on the page
- a figure caption printed out in the HTML (hint: look at `fig.cap`)
- specifications for the output figure to be 5 inches by 5 inches

(c) Knit the Rmd to an HTML to make sure your plot looks correct!

Reminder: You should always run each line of code to verify that it works *before* pushing the "Knit" button.

Appendix Formatting

The final piece in each of these `R` code chunks is the name of the code chunk (e.g. `setup`, `cars`, `pressure`). This is optional, but is useful when you want to reference a code chunk later on.

Often in statistical reports, we wish for our code to not be output in the document (`include = FALSE` or `echo = FALSE`), but wish for all of the code used to be output in an Appendix.

We can accomplish this task by adding an `R` code chunk at the end of the document that references **all** of the document's code chunks.

This looks like:

```
{r, ref.label = knitr::all_labels(), echo = TRUE, eval = FALSE}
```

These options do the following:

- `ref.label = knitr::all_labels()` tells `R` to pull *all* of the code from *all* of the previous code chunks
- `echo = TRUE` outputs the code from these code chunks
- `eval = FALSE` does not evaluate the code from the chunks, instead only prints it – this is because the code has already been evaluated

Part 5:

(a) Set the option for every code chunk to `echo = FALSE`

(b) Add an Appendix section at the end of the report

(c) In the Appendix, add a new empty `R` code chunk that:
- References all of the code created in the report
- Does not evaluate the code but prints the code

(d) Knit the Rmd to an HTML to make sure your Appendix looks correct!

Global Options (for All Code Chunks)

If you would like to not have to specify the options *for every* code chunk, you can specify **global** options in a code chunk at the beginning of the document.

This is what you see included in the `setup` code chunk at the beginning of the document. Inside this code chunk are all of the global options (for every code chunk) you would like.

This looks like:

```
knitr::opts_chunk$set(echo = FALSE,
                      eval = TRUE,
                      fig.width = 5,
                      fig.height = 3)
```

As you want for this code chunk to to be only used to specify the code options, you should add an `include = FALSE` argument to the header (so that the code will not be output).

Part 6:

(a) Specify global options for:
- your code to **not** be output on the page
- your figures to be 5 inches by 5 inches
- your figures to be center aligned

(b) Knit the Rmd to an HTML to make sure your changes look correct!

Publishing Statistical Findings

For many statistical reports we wish to report the results of the model(s) we fit. However, the "typical" output of a `lm()`, `t.test()`, or `anova()` model looks a bit cluttered for a professional looking statistical report.

Insert the `broom` and `knitr` packages.

I have advertised my love of the `broom` package previously, specifically using the `tidy()` function. I also referenced the `kable()` function from the `knitr` package in Lab 8, to make a nice looking HTML table from a statistical summary.

Let's play with both here to see what they can do!

Summarizing Data

On many occasions, it is nice to have a table summarizing your data in the body of your report. This table makes it easier for your reader to inspect different attributes of your data (e.g. sample size, variables, summary measures).

Part 7: In a new `R` code chunk

(a) Load in the `knitr` package

(b) Using the `summarise()` function in `dplyr` create a table containing:
- the mean and standard deviation of the distances
- the mean and standard deviation of the speeds
- the size of the data

Hint: For help look back at the beginning of Lab 8!

(c) Pipe the resulting table into the `kable()` function

(d) Make sure the column names are nicely formatted **and** explanatory!

(e) Knit the Rmd to an HTML to make sure your table looks correct!

Summarizing Statistical Models

On other occasions, we want to output a summary of the statistical model we fit to our data. The "typical" output of a `lm()` for simple linear regression looks like:

```
##
## Call:
## lm(formula = dist ~ speed, data = cars)
##
## Coefficients:
## (Intercept)        speed
##     -17.579        3.932
```

This doesn't look very nice in a professional report, since many people would be confused as to what the `Call: lm(formula = dist ~ speed, data = cars)` part even means.

To clean this up, we can use the `tidy()` function from the `broom` package!

Part 8: In a new `R` code chunk

(a) Load in the `broom` package

(b) Fit a simple linear regression to stopping distance, as a function of the car's speed.

(c) Use the `tidy()` function to output a nice summary table of the model's output.

Hint: If you don't quite get a nice table with `tidy()`, try using `kable()`!

Part 9:

(a) Make sure that the *code* associated with these new code chunks **is not** output on the page.

(b) Make sure that the *output* (tables) of these new code chunks, however, **are** output on the page.

(c) Verify that the code associated with these new code chunks **is** output in the Appendix!

Part 10: Explore *at least* 1 of the 4 additional options in RMarkdown

Less spicy

(a) Include a picture with a caption at the beginning of the report

(b) Include a bulleted list of the sections included in the report at the beginning of the report

(c) Include a mathematical formula for the SLR fit to the `cars` dataset
Hint: The LaTeX reference should help!

(d) Include a table of contents at the beginning of your HTML file
Hint: Page 4 of the RMarkdown Reference Guide should help!

More spicy