Introduction to RMarkdown

Data Science for Biologists, Spring 2020

Stephanie J. Spielman

Introduction to RMarkdown

This is an **RMarkdown** document. R is a computer language, and markdown is a simple formatting syntax, known as a "markup language" (groan puns groan, buckle up), for authoring HTML, PDF, and MS Word documents within plain-text environments. **RMarkdown is a unified framework for integrating markdown formatted-text with R code. It is the best.**

RMarkdown documents begin with a section known as *YAML front matter* ("YAML" = "YAML ain't markup language". This is known as recursive acronym. Welcome to the wide world of "Programmers Attempt Naming Humor."). The front matter, whose contents are bounded by three dashes `---` before and after, specifies document settings including information for how the document should be *rendered* into its final product, as well as any other parameters that are used throughout the RMarkdown file.

As indicated in the YAML front matter, this document will produce an HTML file when you click the **Knit** button (a package called `knitr` will show up and created your beautiful HTML). The *syntax* (code coloring) will be highlighted according to the "tango" scheme (we'll come back to this later!).

Resources

Your resources for working with RMarkdown include:

1. RStudio Rmarkdown documentation
2. RMarkdown Cheatsheet
3. The definitive RMarkdown reference by the inventor of RMarkdown, Yihui Xie, a very badass RStudio Engineer who also gives hilarious talks. This book is *incredibly comprehensive*, beware.
    - Here is a talk he gave last year on a new R package `bookdown`, which uses RMarkdown to make E-books and more (as in, every single online book in this class was made using `bookdown`).

Briefly, markdown syntax

Below are LIMITED examples of writing markdown documents. For a comprehensive set of ways to customize your documents, see the official basic syntax

We can write headers of varying size with different numbers of hashtags (remember, this is NOT R code - these are not comments!).

Biggest header

Slightly smaller

And smaller

And smaller!!!

so smol this is the smolest

We can also make bulleted lists, as shown below. There always needs to be a new line before the start of a list, otherwise it will NOT render properly. (See this for yourself by deleting the next new line and knitting. No more bullets :( )

- **bold text** and **bold text**
- *italics text* and *italics text*
- `code formatted text`, but this is NOT actual code!!
  - A nested list item is tabbed in
- We can also use stars for bullets
- And dashes too

> This is known as a blockquote. It's sometimes useful.

Any HTML component will also work in a markdown document. If you know HTML, style it up as you wish. If you don't know HTML, you're probably a happier person.

One important HTML bit to know, however, are comments. *HTML comments* are which are enclosed with `<!--` and `-->`. When you click **Knit**, the engine will ignore these bits - they are comments! Regular R-style comments won't work in markdown - in markdown, R comments are headers.

Working with code chunks

Below is an *R code chunk*, which RStudio has kindly shaded in grey for easier visibility.

```
## This is a code chunk.
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

There are several ways you can modify the behavior of code chunks, by adding arguments inside the `{r}` part at the beginning.

0. The `{r}` itself means "This is R code." It is possible, in fact, to specify *other languages(!)* rather than R, for example you can run bash code within RMarkdown documents by writing `{bash}` instead. **This is beyond the scope of our class.** We will only be writing R code here!
1. You can name a chunk by adding a name like `{r, code-chunk-name}`.
2. There are several parameters which you can include that modify how the `knitr` engine deals with the code chunk, some of which are described below. Each parameter should be set to either `TRUE` or `FALSE`. These defaults are implicitly applied if you don't specify:
   - `include=TRUE`: Should this code chunk itself be **included** in the final output. `include=FALSE` means the code will still be run but the knitted document will NOT show the code or its output.
   - `echo=TRUE`: Should the **output** from the code be displayed in the knitted document?
   - `eval=TRUE`: Should we even run this code chunk? If `FALSE`, the code chunk will appear as code but it won't have been run, so it also won't have any output.
   - `warnings=TRUE`: If the code produces any warning messages, should they be displayed in the knitted document?

- `messages=TRUE`: If the code produces any general messages, should they be displayed in the knitted document?
- `error=FALSE`: If the code is broken and gives errors, the default behavior (`FALSE`) is to halt knitting entirely until the user fixes the bug. To allow knitting to proceed and show the error message, set `error=TRUE`
- `collapse=FALSE`: If `FALSE`, the code output will be displayed in a separate white box from the code itself. If `TRUE`, output will be embedded into the code itself.

EXERCISE SET ONE Practice writing code chunks in the space below. After you write each chunk, make sure it works as expected by pressing the **green arrow** at the top-right corner of the chunk. This will **Run** your code chunk and reveal its output in place, without going through the hassle of knitting the entire document. You can clear this output by clicking the "x" in its top right corner. You are also welcome to Knit each time if that works for you!

> Beware: Pressing the green arrow doesn't always mean the code was run in the Console. If/when you encouter errors that suggest the code you thought you ran didn't run, it's because the code was run..elsewhere.

1. Write a code chunk named "chunk-1" that reveals the *last six rows* (hint: use the function `tail()`) of the `iris` data frame.
2. Write a code chunk named "chunk-2" that calculates the maximum value of iris sepal lengths (hint: remember the function `max()` and the `$` for working with columns). In your chunk, set the option `collapse=TRUE` to see how output looks different.
3. Write a code chunk named "chunk-3" that makes a *histogram* of iris petal widths. Be sure to style your plot appropriately with clean labels. Execute this code chunk with and without the added parameter `messages=FALSE`. What difference do you see?
4. Write a code chunk named "chunk-4" that makes a *scatterplot* of iris petal widths across petal lengths, where points are colored according their sepal widths. Be sure to style your plot appropriately with clean labels.
    - For an added challenge, change the *color palette* to your liking with the function `scale_color_gradient()`
    - For an added-added challenge, change the color palette using one of the "Brewer" palettes. You will need to use the function `scale_color_distiller()` for this. To accomplish this, you'll need to load the library "RColorBrewer" (it has already been installed for your convenience). You should install directly in the Console, but load the library *in the code chunk*. At the bottom of this RMarkdown document, you can see the colorblind-friendly palettes from Brewer.
5. Modify the *size* of the output figure in "chunk-4" by adding parameters like `{r, fig.width=7, fig.height=5}` until you find a width/height that conveys your figure well.
6. For many questions you will be prompted to write answers. I have set up a way for your answers to appear *in blue* to help your grader grade. You will write your answers inside *answer tags*, as shown below (only visible in the Rmarkdown document, not yet visible in the knitted output!). I will provide an *HTML comment* stating answer goes here (you can either leave or delete this comment, it doesn't matter), but your answer must be INSIDE the space delimited by `<answer>` and `</answer>`. Practice writing some text inside the answer tags, knit, and view the output to get a sense of how this will work.

At this point, if you haven't already, Knit the whole document to see how your chunks appear!

Working with global options

EXERCISE SET TWO Modify the YAML frontmatter and the R `setup` chunk to customize this document as your own. The `setup` chunk is a standard part of RMarkdown documents used to set **global options** (i.e., applies to whole document) and values used throughout the entire document.

After each step, KNIT the document to confirm it worked.

1. Working with YAML frontmatter
   - Modify the YAML frontmatter "author" field to show YOUR name, not mine!
   - Add a *new* field to the YAML frontmatter called "date". This should be a top-level field similar to title and author (unlike, for example, the highlight field which is nested under output/html_document). In this field, type today's date in any format you like (words, numbers, whatever - as long as it is in quotes)!
   - Change the value in the new "date" field to this exact text, including the quotes: "2020-02-05". This will automatically calculate and reveal today's date.
   - Modify the "highlight" field to a different syntax highlighter, and re-Knit. All you need to do is delete the word "tango" and replace it with your choice, such as "espresso". *Ignore any directions this website provides - the point of the link is just to see the palettes.* Choose your own adventure!!

- Modify the front matter to change the text `html_document` to `pdf_document`, and knit the output. It's a PDF now! Yes! Magic!

2. Working with the `setup` R chunk.

   We'll start working with the line `knitr::opts_chunk$set(echo = TRUE)`, which sets the global options for how the `knitr` package should treat code chunks. Here, it says that all code chunks should have `echo=TRUE`, i.e. output should never be suppressed. We can continue to add arguments to this global setup:

Again, after each step, KNIT the document to confirm it worked.

- Modify this line to read `knitr::opts_chunk$set(echo = FALSE)` and re-knit. How has the document changed?
- Revert back to `echo=TRUE`, but add an additional argument `eval=FALSE` as:`knitr::opts_chunk$set(echo = FALSE, eval=FALSE).=`. **Before knitting**, make an educated guess for how the document will change. Then knit and see!
- Remove the `eval=FALSE` argument, and add different arguments to change the size of ALL FIGURES that the document makes. Set the global figure width to 3 and the global figure height to 4 (Hint: we did this already in Exercise Set One!)
- It is usually in the setup chunk where we load all libraries that our code will rely on. You'll see that `tidyverse` is currently the only library loaded. Add another line into the setup chunk that also loads the library `RColorBrewer` (this has already been installed for your convenience).

Brewer Palettes

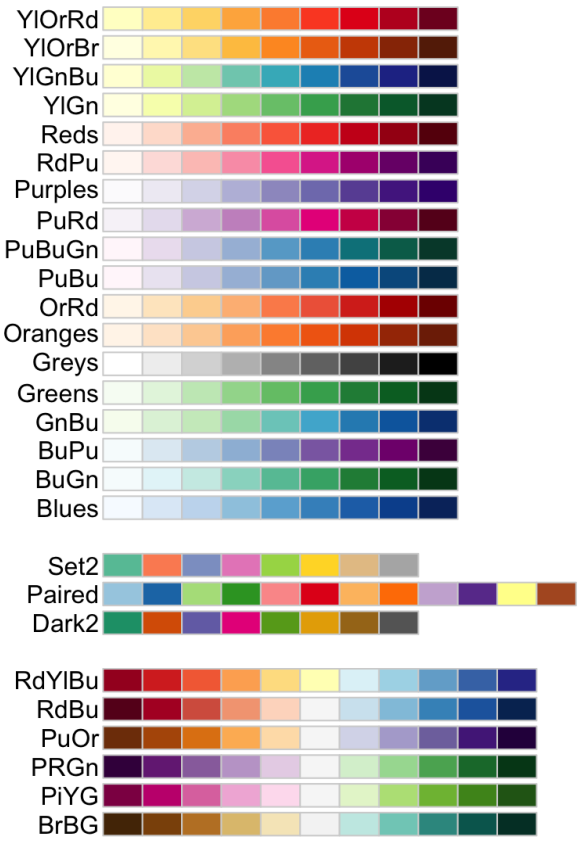

Figure 1: Colorblind-friendly Brewer Palettes