Joshua French

2021-01-26

Crash Course in R Markdown

What is R Markdown?

R Markdown is a tool for dynamic, reproducible creation of data-centric documents that include text, code, the results of executed code.

- R Markdown was created by R Studio (https://www.rstudio.com) in 2014 (or at least that's the oldest discussion I can find).
- It is a variant of Markdown, which was created by John Gruber (creator of Daring Fireball (https://daringfireball.net/)) and Aaron Swartz (co-creator of Reddit (https://www.reddit.com/) as a lightweight markup language for creating formatted text using a plain-text editor.
- In general, a markdown language is a language that is written in plain-text but that is rendered different based on "markups" in the text.
- Has some capabilities similar to Jupyter (https://jupyter.org/) notebooks.

R Markdown is a great way to:

- Create a data analysis notebook that documents how and why you did analysis a certain way.
- Share your work with others in a variety of formats.
- Make your work reproducible.

R Markdown is a very useful tool for the data scientist.

R Studio provides R Markdown (and other) cheatsheets at https://rstudio.com/resources/cheatsheets/

Your turn

The easiest way to create a new R Markdown file is File → New File → R Markdown. Do this on your computer to see what is produced.

What's in an R Markdown document?

An R Markdown document has three components:

1. An optional YAML header.
2. Formatted text.
3. **Chunks** of computer code.

Since the YAML header is optional, we'll talk about it last.

Formatting text in R Markdown

Headings and sections

Headings and sections of different levels are created with `#`.

- `#` indicates the first level
- `##` being the second level
- etc.

The words after the `#` will be larger than the regular text.

Note: In R Scripts and the R Console, the `#` symbol denotes a comment. In R Markdown files (.Rmd or .rmd), a comment is created by placing `<!--` and `-->` to the left and right, respectively, of the text you want to comment.

e.g.,

```
<!-- comment text -->
```

would comment the text "comment text".

The easiest way to comment text in R Markdown documents is to highlight the text and then press Ctrl/Cmd + Shift + c.

Your turn

Practice commenting some text in R Markdown.

Font styles and effects

- *Italic text*: `*text*` or `_text_`.
- **Bold text**: `**text**` or `__text__` (two underscores on each side)
- `Typewriter (code) text`: `` `text` ``
- Endash (–), emdash (—): `--` and `---`.
- $^{\text{superscript}}$ and $_{\text{subscript}}$: `^superscript^` and `~subscript~`
- ~~strikethrough~~: `~~strikethrough~~`
- Escaping special characters *, _, $, \, `: `*`, `_`, `\$`, `\\`, ``` `` ` `` ```
- Footnote[1]: `Footnote^[This is a footnote]`

Math is rendered using LaTeX (which is pronounced "Lah-tech" or "Lay-tech"). So you can use $ and $$ to render inline math and display math like you would in LaTeX.

E.g., Typing "The regression coefficient $\beta$" will produce "The regression coefficient $\beta$".

Your turn

1. Create a new R Script (Ctrl/Cmd + Shift + n).
2. Replace the file contents with the code below:

```
---
title: "Bad jokes"
output: html_notebook
---

# First level

## Second level

### Third level

This document is intended to help you see the effect of markup on a rendered document.

*This is italic text*

**This is bold text**

`This looks like computer code`
```

[1] This is a footnote

```
What's the difference between endash--endash and emdash---emdash?

How can I add a superscript to this? Like this: this^superscript^?

That's a joke if you didn't get the ~subtext~.

Wow, two bad jokes in a row. I'd like to strike ~~that~~ from my memory.

I wish I could \*escape\* these bad jokes.

But I guess this will just be an unpleasant footnote^[unpleasant footnote].
```

3. Save the file (Ctrl/Cmd + s) with the name "bad_jokes.Rmd".
4. Click the Preview button and view the results.

Numbered/ordered and unordered lists

Numbered/ordered lists

Numbered lists with sub-bullets:

```
A numbered list with subnumbering.

1. Item 1
    a. Sub-item 1 # two tabs, not one
    b. Sub-item 2
        i. Sub-sub-item 1 # four tabs, not two
2. Item 2
    a. Sub-item 1
```

Result: A numbered list with subnumbering.

1. Item 1
    a. Sub-item 1
    b. Sub-item 2
        i. Sub-sub-item 1
2. Item 2
    a. Sub-item 1

Note: There must be a line between any text and the number `1.`

Unordered lists

Unordered lists with sub-bullets:

```
Some text with bullets.

* Item 1
  * Sub-item 1
  * Sub-item 2
    * Sub-sub-item 1
* Item 2
  * Sub-item 1
```

Result: Some text with bullets.

- Item 1
    - Sub-item 1

    - Sub-item 2
        - Sub-sub-item 1
- Item 2
    - Sub-item 1

Notes:

- There must be a line between any text and the *.
- You can use `+` or `-` instead of `*` when creating unordered lists with the same result.

Your turn

Create an example of a numbered list with 4 levels and an unordered list with 3 levels.

Code chunks

Chunks of code can be added in an R Markdown by placing the relevant code between `` ```{r} `` and `` ``` ``. e.g.,

~~~
```{r, label = chunk_name}
1 + 1
```
~~~

You can generate a code chunk template using Ctrl/Cmd + Alt + i.

The `chunk_name` is optional but allows you give give chunks names labels that you can refer back to. This is useful for complicated documents and reports. (See Verbatim Code Chunks for how to get the above code to render.)

Note: each label should be unique within the document. I don't recommend adding chunk labels unless you actually need them, otherwise you are likely to reuse a chunk label and make a headache for yourself!

Chunk options

Code chunks have many options that can be used to customize the code. We will cover many important ones below. I will specify the default option in parentheses, (), after the option. So this is what will be used by default if you don't specify anything.

Yuhui Xie, creator of **knitr**, provides a complete list (https://yihui.org/knitr/options/) of chunk options on his website (https://yihui.org/).

Code evaluation

- `eval`: (`TRUE`) A logical value indicating whether the chunk should be evaluated in the Console.
    - You can also evaluate **selected lines** if you provide a numeric vector. e.g., `eval = 1:3` would evaluate lines 1 through 3.

Text output

- `echo`: (`TRUE`). A logical value indicating whether the source code should be displayed.
    - You can also echo **selected lines** if you provide a numeric vector. e.g., `echo = 1:3` would echolines 1 through 3
- `results`: (`'markup'`). Determines hwo the results are displayed.
    - Other options are `'asis'`, `'hold'`, `'hide'`.
- `collapse`: (`FALSE`). A logical value indicating whether the source and output should be collapsed into a single block.
- `include`: (`TRUE`) A logical value indicating whether to include the chunk output.
    - Set to `FALSE` when you want something to run but not be shown.

Other options: `warning`, `error`, `messages` (`TRUE`) determine whether warnings, errors, and messages will be displayed, respectively.

Code styling

- `highlight`: (`TRUE`) A logical value indicating whether the source code syntax should be highlighted with different colors.
- `comment`: (`##`) The text that will be shown prior to evaluated code in the output block.
  - Use `##` or `#>` to make the code blocks easier to copy and paste into the Console. `#>` is closer to what the Console output actually looks like.
- `tidy`: (`FALSE`) A logical value indicating whether the source code be automatically reformatted.
- `tidy.opts`: (`NULL`) A list of options that can be used to customize the `tidy` option above.
  - e.g., `tidy.opts = list(width.cutoff = 60)` controls the length of the code and tries to cut the code lines at 60 characters.

Plots

- `fig.show`: (`'asis'`). Instructions for displaying plots.
  - `'asis'`: Show plots in the order and places they would be if executed in the Console.
  - `'hold'`: Output all plots at the end of the chunk.
  - `'hide'`: Hide all plots.
  - `'animate'`: Combine and animate the plots
- `fig.align`: (`'default'`). A character vector specifying how to align the figures.
  - Allowable values are: `'default'`, `'left'`, `'right'`, and `'center'`.
- `fig.width`, `fig.height`: (7). A numeric value indicating the width and height of the result figures.
- `fig.cap`: (`NULL`). A caption for the figure.

Your turn

Copy and paste the code at https://github.com/jfrench/DataWrangleViz/blob/master/rmd-chunks-viz.Rmd into an R Markdown document and then Knit the file to see visual examples of the options described above.

Some general options not discussed

- Caching: This has to do with saving previous code that has been evaluated. This is important for documents with lots of code that is time consuming to run.
- Animation: You can add plot animations if you have the FFMpeg program installed https://ffmpeg.org/.

Inline code

You can run code in the middle of text using something like `` `r mean(1:3)` ``.

For example, `` `r 1 + 1` `` will produce 2.

Tables

By default, R will print tables (like data frames) exactly like you would see them in the Console output.

For example:

~~~
```{r}
head(cars)
```
~~~

```r
head(cars)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
```

```
## 5     8   16
## 6     9   10
```

The formatting can be improved with the `kable` function in the **knitr** package.

~~~
```{r}
knitr::kable(cars[1:6,], caption = "Kable table")
```
~~~

```
knitr::kable(cars[1:6,], caption = "Kable table")
```

Table 1: Kable table

speed	dist
4	2
4	10
7	4
7	22
8	16
9	10

YAML header

The YAML (Yet Another Markdown Language) header controls many aspects of the R Markdown document. Unless you're using R Markdown for advanced purposes (books, blogs, websites, running the same file with multiple input parameters) then you probably don't need to know a lot about the YAML header at this stage.

And if you do need to know about the YAML header in detail, then you probably need a more advanced introduction to R Markdown!

References

You can add a bibliography to your document using the YAML header.

You need to add something like the following to the header:

```
bibliography: filename.filetype
```

Many bibliography types are supported. A list of supported file types is currently available at https://rmarkdown.rstudio.com/authoring_bibliographies_and_citations.html

Citations must be placed in square brackets.

- Multiple citations should be separated by semicolons.

A citation will have a key associated with it in the bibliography file, and you must use `@key` to cite the relevant reference.

Here are some examples from the R Studio website:

```
Blah blah [see @doe99, pp. 33-35; also @smith04, ch. 1].

Blah blah [@doe99, pp. 33-35, 38-39 and *passim*].

Blah blah [@smith04; @doe99].
```

Many for details about bibliographies with R Markdown are available on the R Studio website, and are currently found at https://rmarkdown.rstudio.com/authoring_bibliographies_and_citations.html

Producing different document types

One of the incredible features of R Markdown is that you can easily produce multiple file types with a single click of the "Knit" button.

Adding the following code in the YAML header will produce a pdf document, an html notebook, an html document, and a Microsoft Word document, all with one click of the Knit button!

```
output:
  pdf_document: default
  html_notebook: default
  html_document:
    df_print: paged
  word_document: default
```

- `pdf_document`: Excellent for distributing a static (unchanging) document. Also great for submitting homework solutions to your instructor!
- `html_document`: Excellent for sharing interactive files. Can be viewed in a web browser.
- `html_notebook`: The excellent "R notebook".
  - It's like an `html_document` but the source code can be downloaded by toggling the "Code" button in the upper right hand corner of the file.
  - Great to share with collaborators because they can update the document, add analysis, etc.
  - Make sure to "Run All Code" using Ctrl/Cmd + Alt + r prior to "Previewing" the document or the code chunk output will not be included.
- `word_document`: If you have to have Microsoft Word editing capabilities, for some reason.

Your turn

- Execute the command `download.file("https://raw.githubusercontent.com/jfrench/DataWrangleViz/master/02` `destfile = "02-crash-course-in-rmd.nb.html")` in the Console.
- Close R Studio.
- Find the downloaded file.
- Toggle the "Code" button in the right hand corner of the file to download the original source code.

Other things you can create in R Markdown:

- Presentations: Use the `ioslides_presentation` or the `slidy_presentation` output type (there are others) to create a presentation.
  - Use `#` to start a new slide with a title.
  - Use `***` to start a new slide without a title.
  - Use `##` for a second level header.
- Shiny apps: This is essentially an interactive web app created using R Markdown.
- Websites
- Books (**bookdown**)
- Blogs (**blogdown**)
- Dashboards (**flexdashboard**)
- html_widgets (various packages)