In-Depth R Markdown

PS 811: Introduction to Statistical Computing

March 26, 2020

Abstract

This document serves as lecture notes for R Markdown (March 27, 2020). It demonstrates key capabilities of R Markdown for preparing HTML documents and PDF documents: incorporating R code along with Markdown writing, controlling the output of R code, writing math, creating tables and figures, bibliographies, rendering to multiple output formats, and more.

Contents

1 Overview

1.1 The Main Ideas

Why is it valuable to write technical documents using code? What's the matter with Microsoft Word or other word processing programs? It's long been understood in the academic community that there is more that goes into a piece of academic writing than *word processing* alone. Word processing preoccupies the writer with a constant focus on the appearance of the document. Technical writing, meanwhile, raises problems that require specialized tools for controlling the *content* and *reproducibility* of the document.

What does the emphasis on content control mean? Technical writing incorporates content from several inputs: data, statistical output, external citations, and more. The researcher needs to incorporate numbers, tables, and graphics from their analysis. The researcher needs to create a bibliography from their in-text citations. The researcher may rearrange the document, changing the ordering of numbered sections, tables, figures, references, and so on... systems like R Markdown allow the user to control these elements of the *content* without bothering the user with aesthetic considerations (renumbering sections, deleting bibliography entries, manually creating tables...).

These tools also ensure the reproducibility of a project/document. Because your document runs on code, it has all the benefits of a computer program: it does what its told, no matter how many times you ask it. In an age of "replication crises" and outright research fraud, tools that ensure robust replication of your analysis are important to ensure that your work reflects your most recent data, that your results are accurately transcribed from your statistical results to your paper, and that your work is useful to other researchers who want to implement similar or updated analyses.

1.2 Finding Help

This document will get you started with many important R Markdown capabilities and best practices for writing academic papers, but it isn't a one-stop-shop for all of your R Markdown questions. You may consider consulting other R Markdown resources that have appeared earlier in this course:

- The example project from our first full lecture features an R Markdown document.
- "A Basic R Markdown Document" that I circulated a few weeks ago, which contains some (but not all) of the lessons contained in this current document.
- The syllabus contains even more links.

The official R Markdown documentation contains a ton of helpful information:

- R Markdown: The Definitive Guide
- bookdown: Authoring Books and Technical Documents with R Markdown
- Sometimes it is helpful to read others' experiences with error messages and other bugs by looking at a software package's "issues" on Github. Here are issues pages for `knitr`,

rmarkdown, and bookdown.

1.3 Lay of the Land

We've seen a lot of names for different softwares: R Markdown, bookdown, knitr, pandoc…what's the deal? How do they relate to one another?

First, there are technologies that exist independently of R Markdown.

- LaTeX is an old document preparation system for turning plain text files (`.tex`) into PDF documents. Famous for its ability to render math and control the placement of tables and figures. LaTeX refers to the "engine" that builds a PDF document as well as the markup syntax (the "programming language") that you write in. Many people still use this today, but R Markdown essentially contains all the power of LaTeX and more.
- `markdown` is a simple text markup syntax for dictating how plain text should be styled (e.g. on a web page as `HTML` code).
- `pandoc` is a document conversion system for converting between source code file formats (`.md`, `.tex`, `.html`, and so on).

"R Markdown" is an ecosystem of R packages for converting `.Rmd` files into your desired output, using the technologies above.

- R Markdown can be thought of as a language, combining markdown syntax and "R code chunks" into one file, allowing your final document to reflect both analysis and writing.
- The `knitr` package controls the actual conversion of `.Rmd` files to `.md`, `.tex`, and so on. This process is also called "knitting."
- `rmarkdown` is also a package. It has functions for controlling the way `knitr` renders documents into its final output, which we cover in Section 5.
- `bookdown`: A package that lets you use R Markdown to write more complex documents such as books. Also contains updated output templates that we will use for academic writing.

If you were able to complete Skills Task 1, you should have everything installed. The only missing element might be `bookdown`. If this current document fails to build for you, you may need to install `bookdown`.

```
install.packages("bookdown")
```

2 Setup

Every R Markdown document contains a *header* and a *body*. The header is written with a syntax called `YAML`. You don't need to know much about how YAML *really* works, you just need to know how to use it for R Markdown.

Use the YAML header to supply metadata about a document—title, author, date, and so on—as

well as parameters that control the way a document is built. These build options may be global (that is, irrespective of the output) or they may be output-specific. Consult the `.Rmd` source code for this document for examples.

The *body* of a document contains the content: a combination of writing (primarily markdown syntax) and statistical code (mainly R, but also other languages are possible).

3 R Code

There are two ways to include R code in an R Markdown document: using "chunks" or "inline" R commands. Here are examples and guidance concerns for each.

3.1 Code Chunks

This is the standard method for including a big hunk of R code. Here is an example where we load packages.

```
library("here")
library("tidyverse")
library("broom")
library("scales")  # useful for writing markdown
```

Chunks can be named. Naming chunks is helpful for keeping yourself organized. More importantly, chunk names help you debug code after you hit a compilation error, since R will tell you that it encountered errors within a certain chunk. The chunk above is named `packages`.

RStudio contains helpful buttons to run entire chunks, run all chunks up to this point, etc.

3.1.1 Chunk Options

Aside from the code that you enter into a chunk, you can control the behavior of the chunk as it renders. These chunk "options" are variables that you declare when you create a chunk. Learn about the possible options here and here. A selection of options are:

- `eval`: is the chunk evaluated (interpreted) by R upon compilation? Defaults to `TRUE`.
- `include`: is anything from the chunk included in the output? Defaults to `TRUE`. Overwrites many other options such as `echo`, `results`, etc.
- `echo`: is the code printed in the output? Default is `TRUE`
- `results`: is the output printed in the document? Default is `"markup"` (styled output).
- `cache`: is the output "cached." If output from a code chunk is cached, R will *skip the evaluation of that chunk* if the code is unchanged. This can be helpful to avoid rerunning costly code chunks, but can create problems if you are editing code chunks in a nonlinear fashion.[1]

[1] You may need to delete the cache folder created by R Markdown every once in a while.

Here is a code chunk where I use chunk options to hide the results and cache the chunk output.

```
prims <-
  haven::read_dta(here("data", "hall_house-primaries.dta")) %>%
  print()
```

3.1.2 Default Chunk Options

In all likelihood, you want to create a document that follows a coherent style, and so some chunk options will be more appropriate than others *across the entire document*. When I'm writing a paper, for example, I never want code to print in my final document. You can control/override default chunk options as follows:

```
knitr::opts_chunk$set(
  include = FALSE, echo = FALSE, collapse = TRUE,
  warning = FALSE, message = FALSE,
  cache = TRUE,
  fig.align = "center",
  fig.width = 4, fig.height = 3,
  out.width = "80%"
)
```

For me, setting default chunk options is a necessary part of every document I create with R Markdown.

3.1.3 Side-note: the `here` package

We have used the `here` package so far in this semester. One area where it comes in handy is with R Markdown documents. R Markdown documents don't typically build themselves in your project root. They build themselves in their location within the project. For instance, when I build this document, what does R Markdown think the working directory is?

```
getwd()
## [1] "/Users/michaeldecrescenzo/Box Sync/teaching/811-computing-s20/code/Rmd"
```

This is different from working in R interactively, where the working directory should be the project root. We robustify our code against this weirdness using the `here()` function. This way, all of our file paths *begin at the project root*, even if the "working directory" is arbitrarily set somewhere else within the project. Observe:

```
here()
## [1] "/Users/michaeldecrescenzo/Box Sync/teaching/811-computing-s20"
```

3.2 Inline R code

The other way to include R code is using *inline* commands. An inline command includes R output directly in the text. For example, I can use code to say that the dataset we imported contains 504 observations and 70 variables.

Why is this helpful? This cuts down on the number of numerical errors that I would otherwise make by trying to transcribe results from R into my paper. I don't have to copy and paste any numbers to tell you that this dataset covers elections between 1980 and 2010, or that the mean of the dependent variable in the analysis below (to two decimal places) is precisely 0.59.[2]

3.2.1 Controlling inline output

When you're writing text, you often want to round numbers or convert to percentages on the fly. For example, the number 0.77777. Functions in the `{scales}` package let me process numbers by rounding them (0.8) or converting to percentages (77.777%) while controlling their numerical precision.

4 Writing

Markdown is the easy part.

You create section headers using the pound key (#). More pound keys create lower-level sections (see the code!)

You can write with *italics*, or in **bold face**, or in `monospace, which looks like code`. You can make

- unordered
- lists
- pretty easily,

as well as

1. ordered lists.
2. Notice how the numbering
3. in the source file
4. doesn't have to match the output!

You can directly embed URLs, for instance, to https://www.google.com, or you can include a hyperlink to a webpage.

For technical documents, we can include footnotes.[3] Citations are also possible but we will cover that below in Section 4.4.

[2] This last example shows how I can even include R code inside of LaTeX math.

[3] In case you want to include some details.

4.1 Math

You can write math using LaTeX syntax. You include is just as if it were LaTeX: inline math is included between dollar signs (`$\alpha + \beta x$` becomes $\alpha + \beta x$), and equations are included in LaTeX "environments." I recommend using the `align` environment for equations, since it can be used to align multiple equations along a symbol (e.g. an equals sign).

$$y_i = \mathbf{x}_i^{'}\beta + \epsilon_i \tag{1}$$

$$\epsilon_i \sim \text{Normal}(0, \sigma) \text{ for } i \in \{1, 2, \dots, n\} \tag{2}$$

When you render to PDF, R Markdown uses LaTeX directly, so math (and other LaTeX) just...works. When you render to HTML, math is rendered using an online program that imitates LaTeX called "MathJax." MathJax doesn't have as full of capabilities as LaTeX does, so it's good practice to limit your math'ing to the stuff that is enabled by the `amsmath` LaTeX package (which MathJax handles). Luckily, that is most of what you'll ever want to do with LaTeX math.

You can learn more about common LaTeX math symbols here.

4.2 Tables and Figures

R Markdown is great because you can create tables and figures directly in your report—no need to save graphics and then paste them into a document or anything.

Let's make a graphic and a table reflecting a regression discontinuity analysis from Hall (2015).

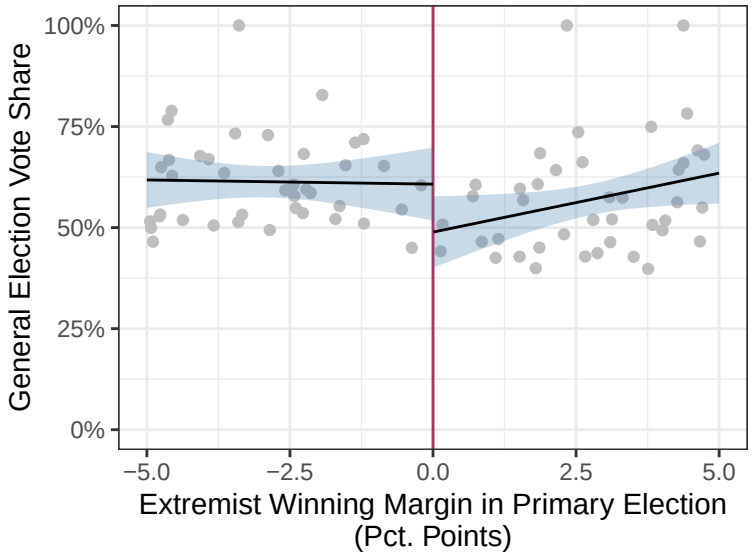

Figure 1: Predicted Effect of Extremist Nomination at Discontinuity

	RD Bandwidth = 0.05
Constant	0.61*
	(0.05)
Extremist Margin	−0.21
	(1.39)
Extremist Win	−0.12
	(0.06)
Extrmist Margin x Extremist Win	3.12
	(2.01)
R^2	0.08
Adj. R^2	0.04
Num. obs.	83
RMSE	0.13

*$p < 0.05$

Table 1: Regression discontinuity model results. Treatment effect is the "Extremist Win" coefficient

4.3 Cross-references

More fun content control with R Markdown. I can refer to figure and table numbers without actually needing to know their numbers—I only need to refer to their "labels."

Figures are labelled according to the chunk name. For example, I can refer to Figure 1, and the number is automatically generated during the compilation process—I don't control the numbering. This is helpful in case you ever reorder figures or tables. You never have to redo your numbering. Never. R Markdown takes care of it.

Tables can sometimes be a little different, depending on how you create your table. Since I used `texreg` to create the table in this document, I can supply an argument to `texreg()`'s `label =` argument and then refer to Table 1 that way. If you create tables using `knitr::kable()`, you can refer to the chunk name in which you create the table like you do with figures.

This output formatting is actually not enabled with default R Markdown output types. It is a special feature of the `bookdown` output formats specified in this document's YAML header. **For this reason, you should always** use `bookdown::pdf_document2` instead of `pdf_document` as your PDF output format. Same goes for `bookdown::html_document2`.

More things can be cross-referenced, including chapters, sections, equations, and more. Learn more here.

4.4 Citations and Bibliographies

R Markdown has the power to create citations and bibliographies automatically. A few things need to come together.

- You need a bibliography file. In this example, it is the `bibliography-example.bib` file. The `.bib` file type is a LaTeX thing (BibTeX) that R Markdown inherited.
- The `bib` file contains bibliographic information for sources that you want to cite. Google Scholar and other journal repository websites will give you BibTeX entries automatically (see also the Google Scholar Button browser extension), but they are often trash and need to be edited slightly.
- You tell R Markdown where to look for bibliographic information by specifying the path to your `bib` file in the YAML header. In this case, we are storing the `bib` file in the same folder as our `Rmd` file, so we put `bibliography: bibliography-example.bib` in the header.
- Finally, cite specific pieces in the text of your paper using the *cite key* for a specific source. For instance, I may want to say that the ideology scores used in Hall (2015) are a campaign donation-weighted average of member NOMINATE scores (see Hall and Snyder 2015).

Learn more about how to create different citation formats here. Some additional notes:

- I am of the opinion that most cite keys generated by Google, JStor, etc., are so stupid. Cite keys should have a standardized format of `author-lastname:year:title-slug`. Go look up your favorite paper on Google Scholar and see at the awful cite key that it suggests. Now vow to always do better.
- For BibTeX output, you can specify using either `biblatex` or `natbib` citation packages. This doesn't change the way you interface to bibliographies in your R Markdown document, but it does effect which citation/bibliography styles you can use. I chose `authoryear` for this document, which is a cite style for `biblatex` and is probably appropriate for most of your paper needs. Some journals have citation style (`csl`) packages that work only with `natbib`, for example. I don't think you really have to worry about this, since copy editors at any journal will fix your bibliography to conform to their style. If you really want to be an overachiever you can download your favorite citation style files here.

5 Output Formats

R Markdown can render the same document into several different output formats. For instance, that YAML header for this document defines possible outputs as PDF (rendering using the function `bookdown::pdf_document2()`), HTML (`bookdown::html_document2`), and even Microsoft Word (`bookdown::word_document2`).

If you want to control which output format you render to, use the "Knit to…" button in RStudio, or you could use the `rmarkdown::render()` function to specify that you want to render an `Rmd` document to a specific output format.

5.1 Customizing output appearance

You can learn about how to customize your output appearance in the R Markdown and Bookdown documentation sites. One additional source is the Pandoc documentation website here. I have already given you workable PDF output options in this document though; messing with all the tiny stuff isn't always worth it.

Outside of the Pandoc variables, you can customize the appearance of LaTeX output by loading additional packages. For this document, I added packages by linking to a LaTeX *preamble* document in the YAML. This preamble doesn't do much, but it changed the fonts, so that's something. You can basically include whatever preamble code you want to control the way LaTeX builds your document, but the R Markdown default PDF template includes a lot of helpful packages that get you through most scenarios.

HTML output is similar; you can use CSS files to control the way your HTML document is styled. If you aren't a web designed, CSS is kinda challenging.

Rather than change features little-by-little, you might consider rendering your entire document using a different stylistic template. For example, check out the `rticles` package templates or the `tufte`-style output templates in the `bookdown` package (which look like this).

5.2 Other R Markdown Technologies

You can also use R Markdown to make slides (using Xaringan for instance, or LaTeX Beamer), websites using blogdown, dissertations, and so much more.

References

Hall, Andrew B (2015). "What happens when extremists win primaries?" In: *American Political Science Review* 109.01, pp. 18–42.

Hall, Andrew B and James M Snyder (2015). "Candidate Ideology and Electoral Success. Working Paper: https://dl.dropboxusercontent.com/u/11481940/Hall Snyder Ideology.pdf". In: