R Markdown for Reproducible Research

Cole Arendt

2019-01-25

Contents

1 ABSTRACT

R Markdown is one of many tools available to the R programmer when it comes to building and sharing the modeling and data pipeline work that has been written in R. The ability to reliably reproduce results is essential to the success and impact of a researcher's work. We will show that R Markdown is a valuable tool for this task by introducing R Markdown, showing a handful of examples, and discussing best practices. When all prose, reporting, and output is generated by a single file based on data inputs, the researcher is freed from semantics and can enjoy truly data-driven results.

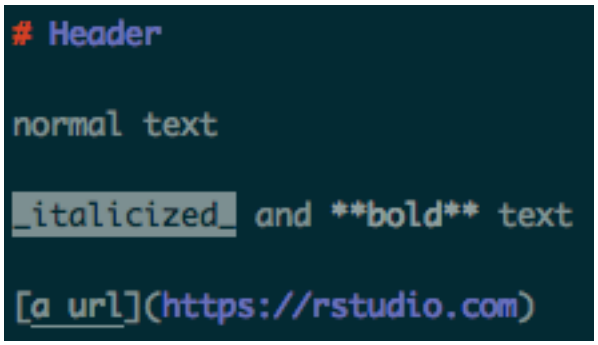

Figure 1: Markdown with syntax highlighting in a computer terminal

2 INTRODUCTION

The researcher's task is to uncover new things, challenge conventional wisdom, explore uncharted territory, and confirm or dismiss bias with facts. In order to accomplish such feats, it is important that a researcher's results be *reproducible,* that is:

> Results are reproducible if they can be reliably replicated or recomputed in another time or place.

For a researcher that is trying to have an impact that extends beyond the "here and now," reproducible results ensure that conclusions are valid in another time and place. Further, for industries like the Pharmaceutical Industry, reproducibility can be a mandate and legal requirement for valid results. As a result, software like R Markdown that helps ensure reproducible results is a valuable tool in a researcher's toolkit.

3 GETTING STARTED

3.1 WHAT IS MARKDOWN?

The story of R Markdown begins with Markdown, a markup language and formatting syntax designed to be:

- human readable
- flexible in its output

For example, where HTML would allow writing bold and italicized text like:

```
This is <b>bold</b> and <i>italicized</i> text
```

Markdown allows bold and italicized text with:

```
This is **bold** and _italicizied_ text
```

A subtle difference, but Markdown is more readable because there are not any letters injected, and it is *much faster* to type. It also has more simplicity in its objectives and flexibility in its output, so where HTML is rendered almost exclusively in browsers, Markdown can be relevant for the web, print, and even computer terminals (see Figure 1).

3.2 WHAT IS R MARKDOWN?

If Markdown is a lightweight markup language and text formatting syntax, R Markdown must be Markdown plus R code, right? Yes, with a catch. R Markdown actually supports *many* programming languages and output types. It got its start with R, and R is the primary language engine. However, R is an interface language. This means that R has been (from its outset) commonly used to "wrap" or call other programming

languages. This began as Fortran and C, but has expanded to include C++, Python, JavaScript, SQL, Stan, and more. As a result, R Markdown supports the inclusion of many of these programming languages.

3.3 THE YAML HEADER

R Markdown has a handful of noteworthy differences from general Markdown. First, every R Markdown document begins with a YAML header:

```
---
title: "R Markdown for Reproducible Research"
output: pdf_document
---
```

The YAML header uses YAML syntax to declare all sorts of information about the document. The full documentation on what options are available is here or here, but the primary elements are the document's Title and Output type. The output type has many options, and can actually support multiple options at the same time.

Some choices for Output Type:

- Static HTML page
- HTML website
- PDF document
- Microsoft Word document
- HTML Presentation
- PDF Presentation
- Microsoft PowerPoint Presentation
- Shiny (interactive) web application
- Blog, Book, etc.

3.4 THE REST OF THE DOCUMENT

Once the YAML header is defined, the body of the document contains:

- Markdown text
- inline code
- code chunks
- embedded items (images, etc.)

Inline code is written like `r my_function` and gets placed into the document like text. It can be very useful for displaying data-determined values programmatically so that they are dynamically changed when the document is regenerated (for instance, if new data is acquired).

Code chunks are much like inline code, but they stand alone and are more useful for lengthier code blocks that do complex logic, setup, value preparation, plotting, or other output generation. It is worth noting that code chunks are very flexible in their options and defaults, and the number of options can sometimes be overwhelming.

Embedded images and other resources are included into R Markdown much like they would be in an HTML page. The syntax looks like `![Text alias](path/to/resource)`.

Once the document is written, it is passed through the `knitr` package to "knit" the R Markdown document into its final output(s). This creates a new / fresh R context to render the document and thereby helps ensure reproducibility.

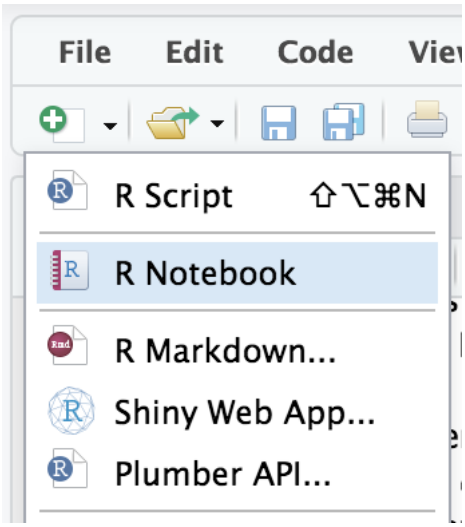

Figure 2: RStudio IDE New File Dropdown showing R Markdown and R Notebook options

3.5 WHAT IS AN R NOTEBOOK?

When creating a new document in the RStudio IDE (see Figure 2), you might notice an option to create a "R Notebook" or "R Markdown Notebook." This is offered in addition to the long-known "R Markdown," and the differences are subtle.

There can be many reasons that a user might prefer R Notebooks, but the most noteworthy difference between R Notebooks and plain R Markdown is that R Notebooks are somewhat *stateful*. This means that they cache and display output as it is generated, making it easier to interactively and iteratively build a narrative without having to re-render the document each time you have made a change.

It is worth noting that this output can be **problematic for reproducibility.** Because output is cached, there is *state* introduced that might unknowingly persist when it is no longer relevant or reproducible.

As a result, R Notebooks can be thought about as good for interactive exploration or discussion among developers, but moving to a standard R Markdown format is generally preferred once the report has stabilized or is being shared with an end user.

R Notebooks are identified by `output_type: html_notebook`. Remember that you can have multiple output types, so a YAML header like the following allows you to easily build multiple output types from the same document.

```
---
title: "R Notebook"
output:
  html_document:
    df_print: paged
  pdf_document: default
  html_notebook: default
---
```

3.6 PARAMETERIZED R MARKDOWN

A little known feature of R Markdown is the ability to add *parameters*. Parameterized R Markdown is defined by the addition of parameters to the YAML header, which will then be utilized within the rendering of the document.

For instance, a YAML header like the following might be used to define the report dates:

```
---
title: "R Markdown - Parameterized"
output: html_document
params:
  start_date: !r Sys.Date() - 10
  end_date: !r Sys.Date()
---
```

This allows these values to be accessible in code using `params$start_date` and `params$end_date`. It essentially makes the R Markdown document a *parameterized executable* or function, which can then be called using `rmarkdown::render()`. For example, the following executes this R Markdown report with non-default parameter selections:

```
rmarkdown::render(
  "./ex/ex-parameterized.Rmd",
  params = list(
    start_date = Sys.Date()-5,
    end_date = Sys.Date()
  )
)
```

There is much more available to explore in Parameterized R Markdown, like an easy-to-use user interface for selecting values. You can also host Parameterized R Markdown in a publishing platform like RStudio Connect that exposes a user interface to end users of a report without needing them to install R.

4 A RESEARCHER'S WORKFLOW

Now that we have thoroughly explored the what of R Markdown, let us now delve into *why* this tool is useful for researchers.

4.1 A COMMON PROBLEM

Let us imagine a researcher who has done a fair amount of data analysis, exploration, and deduction to come to a set of results, and that programming was involved in this researcher's endeavor. When the researcher has finished, she will likely take the results and use them to build a paper, report, presentation, or other output which she will use to share the output with colleagues or other researchers.

This process is tedious and error prone:

- moving the results to another medium (e.g. from script output to a PowerPoint)
- copying values, data, plots, images, etc.
- injecting values into prose (e.g. There are 45 rows in this dataset)

Moreover, let us imagine that someone wants to reproduce her work, or that another set of surveys, experiments, or clinical trials are added to her dataset. Not only must she re-execute her code, she must also duplicate all of the transmission of information into her manual "publishing" process. If any data or information is accidentally "left in place" from her first pass, her published results will be un-reproducible and incorrect.

4.2 AN ELEGANT SOLUTION

Let us now imagine the same researcher begins her programming process by writing her prose and code in a single R Markdown document. In-line values in the document, plots, images, and other output is all

generated dynamically based only on the input data and runtime ecosystem (packages, etc.).

As a result, when another researcher wants to reproduce her results or when she wants to re-execute the paper on new data, the workload becomes "Click Knit." Further, our researcher can trust that the report accurately reflects the input data, since no human "copy/paste" or other editorial process has taken place.

4.3 WORKFLOW FOR DATA DRIVEN RESEARCH

This problem and solution can be reduced to a paradigm of *best practices* in which reproducible, data-driven research is greatly simplified by R Markdown. There are certainly more thorough articles on the topic, but we will focus on the highlights.

- Define input datasets, resources, or other dependencies
- Manage R packages in a reproducible fashion
- Write narrative and code together into an R Markdown document
- Ensure that the narrative is programmatically dependent on the data
- Configure settings so that the R Markdown document produces final output

4.4 AN EXAMPLE

Let us imagine that we are beginning a project where we explore the classic `iris` dataset. We might approach the problem by first specifying the `iris` dataset as our only input. To ensure auditability and reproducibility, I might save and open source the dataset as well as specify it as a parameter for my research in the YAML header.

```
---
title: "Iris Research"
author: "Cole Arendt"
date: "1/6/2019"
output: pdf_document
params:
  input_data: "iris.rds"
---
```

Now, I need R packages to make my work more efficient. I have chosen `dplyr`, `tidyr`, and `ggplot2`. Though I reference these packages in my code, I should also keep track of which versions of the package I am using. For this task, I might use `packrat::.snapshotImpl(".", snapshot.sources = FALSE)`.

Finally, we come to my actual research. As I explore this dataset, I must ensure that my results are programmatic. This means plots, values, and prose should be dependent only on my input dataset.

When complete, I will be able to generate my research paper with one line of code (or one click):

```
rmarkdown::render("./ex/ex-research.Rmd")
```

If I received updated, corrected, or otherwise different data (i.e. "iris_altered.rds"), changing my single input parameter and re-rendering the report will reliably rebuild my analysis off of different data. Further, any other researcher can take my document and R packages to reproduce or validate my results. Feel free to give it a shot yourself!

5 TIPS AND TRICKS

5.1 CODE CHUNK OPTIONS

Every code chunk has options that control its behavior. These are specified in the header above the code chunk:

```
{r my_chunk_name, my_option="one", my_other_option="two"}
```

While there are *many* such options, some of the most ubiquitous are:

- `echo` - whether to "echo" the executed code in the output document
- `include` - whether to include output of the chunk in the output document
- `message` - whether to include messages from the chunk in the output document

Further, it is advisable to remember that the defaults can be set with code like:

```
knitr::opts_chunk$set(echo = FALSE)
```

5.2 KNIT OPTIONS

"Knitting" is the process of taking an R Markdown document and generating its final output. As a result, options passed to the knit process are "global" (i.e. affect the entire document). Most knit options are defined in the YAML header, and common options include:

- templates, formatting, or other output styles
- parameters
- output document type

5.3 TABLE OF CONTENTS

If your document is very long, a table of contents is very helpful. Unfortunately, a table of contents is not included in the PhUSE style guide, so we excluded it from this paper. However, you can easily include a table of contents by adding a section like the following to your YAML header.

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

```
output:
  pdf_document:
    toc: true
```

5.4 URLS AND INTERNAL LINKS

Often within a research article, report, presentation, or other work, we are standing on the shoulders of giants. As a result, it is very important that we appropriately reference their work. External URLs are standard markdown format `[link text](link url)`, and these can be used to reference internal headers as well. However, for pdf and html document compatibility, the following is recommended for internal links:

- Label a header with something like `# MY HEADER {#the-header}`
- Reference the header using `\@ref(the-header)`

Referencing images can be done in a similar fashion. It unfortunately requires using a different output format for PDFs (`bookdown::pdf_document2`), but can be done using:

- Label the image with `![Image alt text](./image/path){#the-label}`
- Reference the image using `Figure \@ref(the-label)`
- Images built in a code chunk can also be referenced with `\@ref(fig:code-chunk-name)`

5.5 RSTUDIO DOCUMENTATION

RStudio has adopted R Markdown as the standard for some of its most important communication. If you are interested in examples, we

- host our company blog and community blog using blogdown
- write (most) product documentation using bookdown
- build many popular websites using pkgdown
- write presentations and papers like this one using R Markdown

We like to think that the sky is the limit when it comes to the reproducible results that can be achieved using R Markdown, and are always excited to see the impressive works that the open source community builds using this tool.

6 OTHER TOOLS

6.1 INCLUDING PYTHON

Python is a general purpose programming language that is increasingly common in data science. It has great utility in many regards and many useful modules (akin to packages) that can be useful to a researcher.

R Markdown, despite its name, is actually remarkably language agnostic. You can include code chunks in other languages (like Python, bash, Stan, and SQL) in your R Markdown documents.

Python, in particular, has recently improved support in R Markdown and the RStudio IDE. Specifically, the `reticulate` package makes it possible to specify the python version and conda / virtual environment, as well as persist objects across code chunks.

There is also native support for including python plots and other output in your rendered R Markdown document. There is much written already about this work flow. If interested, the following articles are helpful:

- RStudio IDE features
- Calling Python from R
- An example Shiny application

6.2 CUSTOMIZED EMAILS

While emails may not be very relevant to the *researcher's* task, per se, they are a common means of communication for many supportive functions. R Markdown reports have what is called *output metadata* that can be consumed by external services. This, combined with packages like `blastula` that enable customized email bodies, can facilitate programmatically generating reports, documents, or presentations that are programmatically delivered in a customizable and personalizable fashion to the consumers of such content.

RStudio Connect is an example of a platform designed to leverage this tooling for this end, and there are several examples to explore if interested.

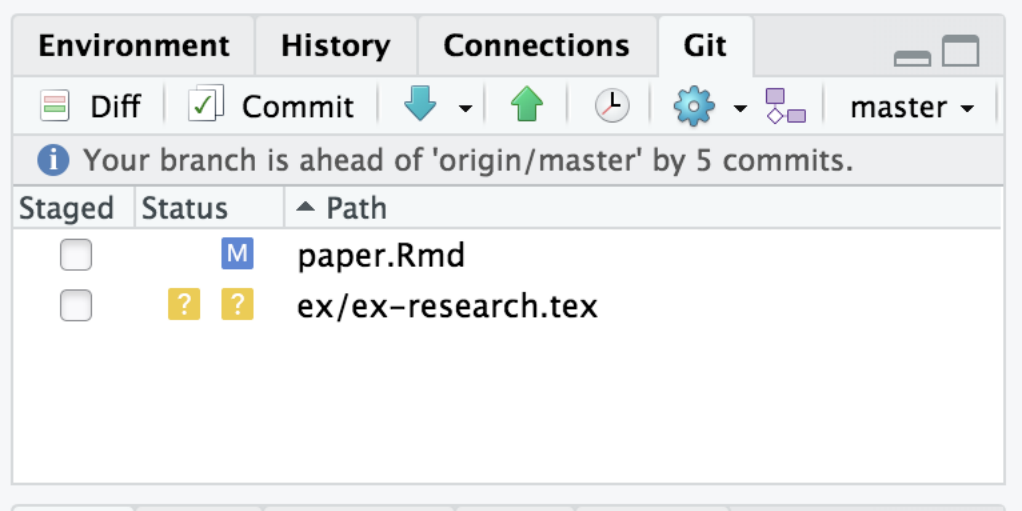

Figure 3: The RStudio IDE `git` pane with new commits and a modified paper

6.3 VERSION CONTROL AND GIT

Being able to reproduce a single version of one's research is fantastic... but what if you could reproduce *every* version of one's research? That is the sort of magic that "Version Control" provides.

`git` is the most popular version control tool, followed by `subversion`. These tools are oriented around software engineering and tracking / "diffing" changes in source documents (mostly code) over time.

Since our research in this workflow is entirely driven by data along the highway of code, it makes sense to spend some time getting used to these tools. If we make version control a part of our process, we get reproducible research up to a moment in time. Further, since R Markdown consists of plaintext documents, we acquire easy auditability of code that can be compared over time.

There are several helpful tutorials and books on the topic, principally Happy Git with R, and even a built-in `git` pane within the RStudio IDE to simplify some operations (see Figure 3).

7 CONCLUSION

R Markdown is a remarkably powerful tool in the researcher's tool belt. Given a community of powerful minds contributing to its future, it will only improve in its ability to reliably reproduce researcher's results with minimal effort. I imagine a world where reporting standards (like the PhUSE paper formatting) are defined in an R Markdown template that any researcher or contributor can easily use or extend as needed. It is the power of an open source community building quality and reproducible tools that make such a future possible.

8 RESOURCES

The source and examples from this paper are available on GitHub:

https://github.com/colearendt/rmarkdown-for-reproducible-research

There are also many other interesting resources on this topic which we did not have the opportunity to include:

- http://ropensci.github.io/reproducibility-guide/sections/tools/
- http://shop.oreilly.com/product/0636920051435.do
- https://rviews.rstudio.com/2018/11/01/r-markdown-a-better-approach/
- https://rviews.rstudio.com/2017/01/25/r-markdown-for-the-enterprise/