RMarkdown for writing reproducible scientific papers

Mike Frank & Chris Hartgerink

2017-07-31

There is also a slidedeck that goes along with this here, which is worth looking at if you don't know what you're doing on this page and what to look at. Going through this document takes at most two hours, assuming you already know some basic `R` programming. If you find any errors and have a Github account, please suggest changes here. All content is CC 0 licensed, so feel free to remix and reuse!

Introduction

This document is a short tutorial on using RMarkdown to mix prose and code together for creating reproducible scientific documents. It's based on documents that both Chris and Mike wrote about independently (see here and here if you're interested). In short: RMarkdown allows you to create documents that are compiled with code, producing your next scientific paper.[1]

[1] Note: this is also possible for Python and other open-source data analysis languages, but we focus on R.

Now we're together trying to help spread the word, because it can make writing manuscripts so much easier! We wrote this handout in RMarkdown as well (looks really sweet, no?).

Who is this aimed at?

We aim this document at anyone writing manuscripts and using R, including those who...

1. ...collaborate with people who use Word
2. ...want to write complex equations
3. ...want to be able to change bibliography styles with less hassle
4. ...want to spend more time actually doing research!

Why write reproducible papers?

Cool, thanks for sticking with us and reading up through here!

There are three reasons to write reproducible papers. To be right, to be reproducible, and to be efficient. There are more, but these are convincing to us. In more depth:

1. To avoid errors. Using an automated method for scraping APA-formatted stats out of PDFs, Nuijten et al. [2016] found that over

10% of p-values in published papers were inconsistent with the reported details of the statistical test, and 1.6% were what they called "grossly" inconsistent, e.g. difference between the p-value and the test statistic meant that one implied statistical significance and the other did not. Nearly half of all papers had errors in them.

2. To promote computational reproducibility. Computational reproducibility means that other people can take your data and get the same numbers that are in your paper. Even if you don't have errors, it can still be very hard to recover the numbers from published papers because of ambiguities in analysis. Creating a document that literally specifies where all the numbers come from in terms of code that operates over the data removes all this ambiguity.

3. To create spiffy documents that can be revised easily. This is actually a really big neglected one for us. At least one of us used to tweak tables and figures by hand constantly, leading to a major incentive *never to rerun analyses* because it would mean re-pasting and re-illustratoring all the numbers and figures in a paper. That's a bad thing! It means you have an incentive to be lazy and to avoid redoing your stuff. And you waste tons of time when you do. In contrast, with a reproducible document, you can just rerun with a tweak to the code. You can even specify what you want the figures and tables to look like before you're done with all the data collection (e.g., for purposes of preregistraion or a registered report).

Learning goals

By the end of this class you should:

- Know what Markdown is and how the syntax works,
- See how to integrate code and data in RMarkdown,
- Understand the different output formats from RMarkdown and how to generate them, and
- Know about generating APA format files with `papaja` and bibtex.

Getting Started

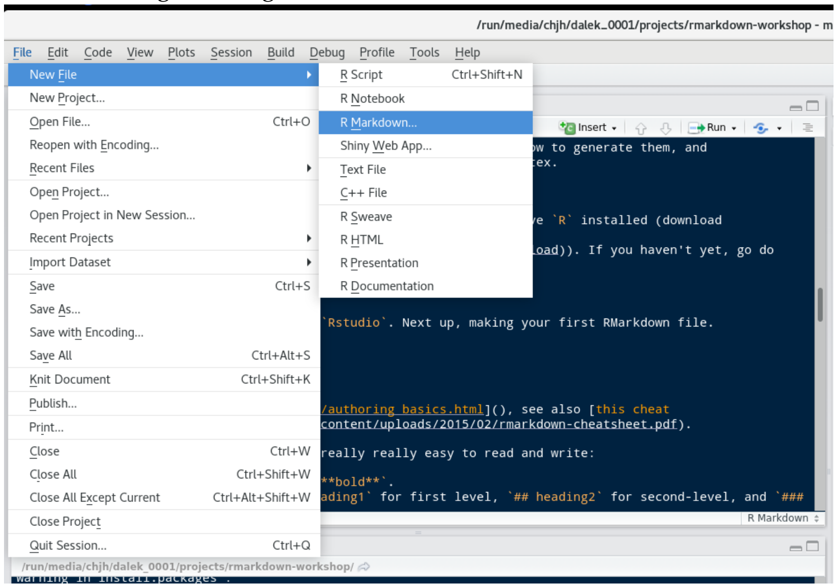
Installation

Before we get started with running everything, make sure to have `R` installed (download here) and `Rstudio` (download here). If you haven't yet, go do that first and we'll be here when you get back!

Exercise

Great, you installed both `R` and `Rstudio`! Next up, making your first RMarkdown file.

Fire up Rstudio and create a new RMarkdown file. Don't worry about the settings, we'll get to that later.



```
w to generate them, and
ex.

e `R` installed (download
oad)). If you haven't yet, go do

`Rstudio`. Next up, making your first RMarkdown file.

/authoring_basics.html](), see also [this cheat
content/uploads/2015/02/rmarkdown-cheatsheet.pdf).
really really easy to read and write:
**bold**`.
ading1` for first level, `## heading2` for second-level, and `###
```

If you click on "Knit" (or hit `CTRL+SHIFT+K`) the RMarkdown file will run and generate all results and present you with a PDF file, HTML file, or a Word file. If RStudio requests you to install packages, click yes and see whether everything works to begin with.

We need that before we teach you more about RMarkdown. But you should feel good if you get here already, because honestly, you're about 80% of the way to being able to write basic RMarkdown files. It's that easy.

Structure of an RMarkdown file

An RMarkdown file contains several parts. Most essential are the header, the body text, and code chunks.

Header

Headers in RMarkdown files, contain some metadata about your document, which you can customize to your liking. Below is a simple example that purely states the title, author name(s), date[2], and output format.

[2] Pro-tip: you can use the `Sys.Date()` function to have that use the current date when creating the document.

```
---
title: "Untitled"
```

```
author: "NAME"
date: "July 28, 2017"
output: html_document
---
```

For now, go ahead and set `html_document` to `word_document`, except if you have strong preferences for `HTML` or `PDF`.[3]

[3] Note: to create PDF documents you also need a TeX installation. Don't know what that is? You probably don't have it then.

Body text

The body of the document is where you actually write your reports. This is primarily written in the Markdown format, which is explained in the Markdown syntax section.

The beauty of RMarkdown is, however, that you can evaluate `R` code right in the text. To do this, you start inline code with 'r, type the code you want to run, and close it again with a '. Usually, this key is below the escape (`ESC`) key or next to the left SHIFT button.

For example, if you want to have the result of 48 times 35 in your text, you type ' r 48-35', which returns 13. Please note that if you return a value with many decimals, it will also print these depending on your settings (for example, 3.1415927).

Code chunks

In the section above we introduced you to running code inside text, but often you need to take several steps in order to get to the result you need. And you don't want to do data cleaning in the text! This is why there are code chunks. A simple example is a code chunk loading packages.

First, insert a code chunk by going to `Code->Insert code chunk` or by pressing `CTRL+ALT+I`. Inside this code chunk you can then type for example, `library(ggplot2)` and create an object `x`.

```
library(ggplot2)

x <- 1 + 1
```

If you do not want to have the contents of the code chunk to be put into your document, you include `echo=FALSE` at the start of the code chunk. We can now use the contents from the above code chunk to print results (e.g., $x = 2$).

These code chunks can contain whatever you need, including tables, and figures (which we will go into more later). Note that all code chunks regard the location of the RMarkdown as the working directory, so when you try to read in data use the relative path in.

Exercises

Switch over to your new RMarkdown file.

1. Create a code chunk. In it, use `write.csv(USArrests, "USArrests.csv")` to write out some data to your hard drive. Subsequently, find that file on your computer. Where is it?[4]
2. Load the `ggplot2` package (or any other that you use frequently) in a code chunk and regenerate the document. Now try load some packages you've never heard of, like `adehabitatHS` or `QCA`. What happens?[5]

[4] RMarkdown files operate out of the directory in which they are located. If you read in data, it's important to write paths starting from that "project root" directory – otherwise when you share the file the paths will all be wrong. Similarly, if you write out data they go in the root directory.

[5] We like to load all our packages at the top of the document so it's easy to know what you need to install to make it run. But it's not good form to put `install.packages` statements in your markdown – let users choose whether or not they want to install things, don't do it for them!

Markdown syntax

Markdown is one of the simplest document languages around, that is an open standard and can be converted into `.tex`, `.docx`, `.html`, `.pdf`, etc. This is the main workhorse of RMarkdown and is very powerful. You can learn Markdown in five (!) minutes Other resources include http://rmarkdown.rstudio.com/authoring_basics.html, and this cheat sheet.

You can do some pretty cool tricks with Markdown, but these are the basics:

- It's easy to get `*italic*` or `**bold**`.
- You can get headings using `# heading1` for first level, `## heading2` for second-level, and `### heading3` for third level.
- Lists are delimited with `*` for each entry.
- You can write links by writing `[here's my link](http://foo.com)`.

If you want a more extensive description of all the potential of Markdown, this introduction to Markdown is highly detailed.

Exercises

Swap over to your new sample markdown.

1. Outlining using headings is a really great way to keep things organized! Try making a bunch of headings, and then recompiling your document.
2. Add a table of contents. This will involve going to the header of the document (the `YAML`), and adding some options to the `html document` bit. You want it to look like this (indentation has to be correct):

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

Now recompile. Looks spiffy, right?

3. Try adding another option: `toc_float: true`. Recompile – super spiffy. There are plenty more great output options that you can modify. Here is a link to the documentation.

Adding More Code (Tables and Graphs)

We're going to want more libraries loaded (for now we're loading them inline).

```
library(knitr)
library(ggplot2)
library(broom)
library(devtools)
```

We often also add `chunk options` to each code chunk so that, for example:

- code does or doesn't display inline (`echo` setting)
- figures are shown at various sizes (`fig.width` and `fig.height` settings)
- warnings and messages are suppressed (`warning` and `message` settings)
- computations are cached (`cache` setting)

There are many others available as well. Caching can be very helpful for large files, but can also cause problems when there are external dependencies that change.

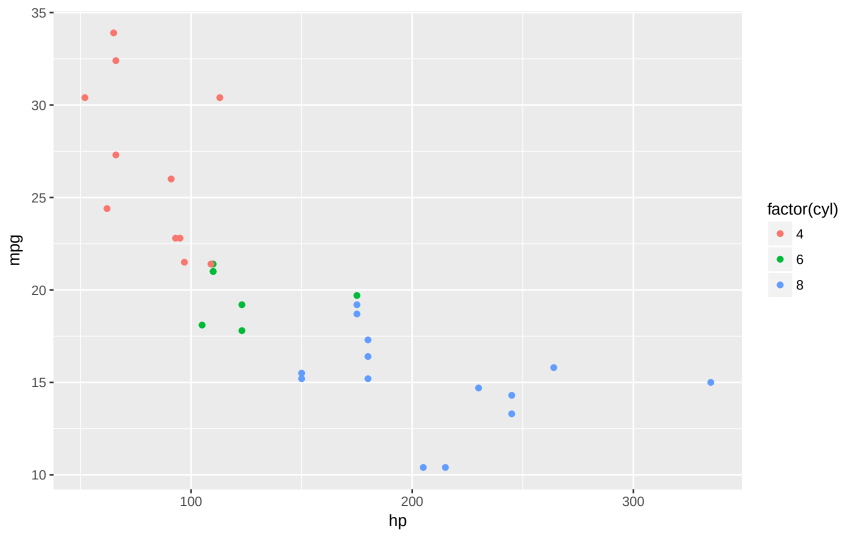
I sometimes set these as defaults, too:

```
opts_chunk$set(fig.width = 8, fig.height = 5,
    echo = TRUE, warning = FALSE, message = FALSE,
    cache = TRUE)
```

Graphs

It's really easy to include graphs, like this one. (Using the `mtcars` dataset that comes with `ggplot2`).

```
qplot(hp, mpg, col = factor(cyl), data = mtcars)
```



All you have to do is make the plot and it will render into the text.

Tables

There are many ways to make good-looking tables using RMarkdown, depending on your display purpose.

- The `knitr` package (which powers RMarkdown) comes with the `kable` function. It's versatile and makes perfectly reasonable tables. I also like that it has a `digits` argument for controlling significant figures.
- For HTML tables, I like the `DT` package, which provides `datatable` – these are pretty and interactive javascript-based tables that you can click on and search in. Not great for static documents though.
- For APA manuscripts, it can also be helpful to use the `xtable` package, which creates very flexible LaTeX tables. These can be tricky to get right but they are completely customizable provided you want to google around and learn a bit about tex.

We recommend starting with `kable`:

```
kable(head(mtcars), digits = 1)
```

	mpg	cyl	disp	hp	drat	wt	qsec	vs	am	gear	carb
Mazda RX4	21.0	6	160	110	3.9	2.6	16.5	0	1	4	4
Mazda RX4 Wag	21.0	6	160	110	3.9	2.9	17.0	0	1	4	4
Datsun 710	22.8	4	108	93	3.8	2.3	18.6	1	1	4	1
Hornet 4 Drive	21.4	6	258	110	3.1	3.2	19.4	1	0	3	1
Hornet Sportabout	18.7	8	360	175	3.1	3.4	17.0	0	0	3	2
Valiant	18.1	6	225	105	2.8	3.5	20.2	1	0	3	1

Statistics

It's also really easy to include statistical tests of various types.

For this I really like the `broom` package, which formats the outputs of various tests really nicely. Paired with knitr's `kable` you can make very simple tables.

```
mod <- lm(mpg ~ hp + cyl, data = mtcars)
kable(tidy(mod), digits = 3)
```

term	estimate	std.error	statistic	p.value
(Intercept)	36.908	2.191	16.847	0.000
hp	-0.019	0.015	-1.275	0.213
cyl	-2.265	0.576	-3.933	0.000

Of course, cleaning these up can take some work. For example, we'd need to rename a bunch of fields to make this table have the labels we wanted (e.g., to turn `hp` into `Horsepower`).

We often need APA-formatted statistics. We can compute them first, and then print them inline.

```
ts <- with(mtcars, t.test(hp[cyl == 4], hp[cyl ==
    6]))
```

This pre-computation makes it easy to write:

There's a statistically-significant difference in horsepower for 4- and 6-cylinder cars ($t(11.49) = -3.56$, $p = 0.004$).

To insert these stats inline I wrote e.g. `round(ts$parameter, 2)` inside an inline code block.

Note that rounding can occasionally get you in trouble here, because it's very easy to have an output of $p = 0$ when in fact p can never be exactly equal to 0.

Exercises

1. Using the `mtcars` dataset, insert a table and a graph of your choice into the document.[6]

[6] If you're feeling uninspired, try `hist(mpg)`.

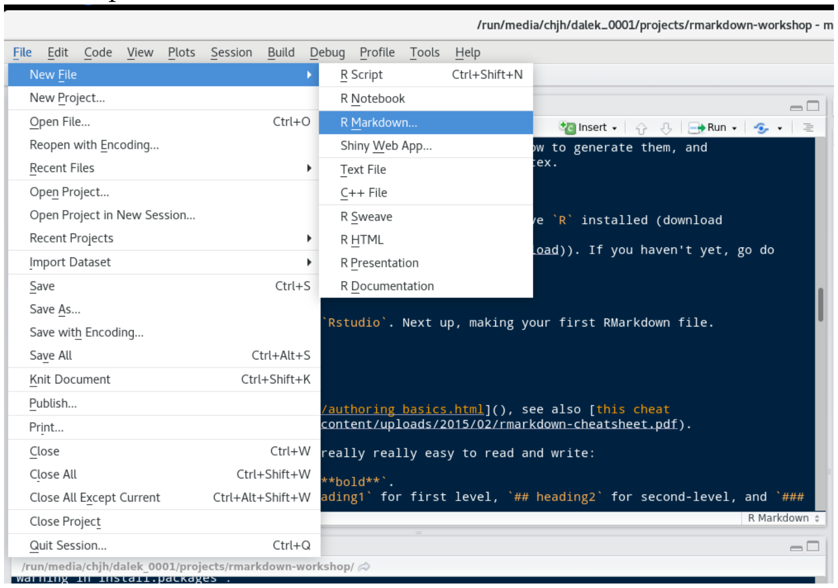
Writing APA-format papers

The end-game of reproducible research is to knit your entire paper. We'll focus on APA-style writeups. Managing APA format is a pain in the best of times. Isn't it nice to get it done for you?

Exercise

We're going to use the `papaja` package. To install this, you need the `devtools` package (required above). One that's installed and you've loaded it (`library(devtools)`), then run `devtools::install_github("crsh/papaja")`.

Once you've got this working, start a new RMarkdown using the default template for `papaja`.



```
ow to generate them, and
tex.

ve `R` installed (download
load)). If you haven't yet, go do

`Rstudio`. Next up, making your first RMarkdown file.

/authoring_basics.html](), see also [this cheat
content/uploads/2015/02/rmarkdown-cheatsheet.pdf).
really really easy to read and write:
**bold**`.
ading1` for first level, `## heading2` for second-level, and `###
```

Compile this document, and look at how awesome it is. (To compile you need `texlive`, a library for compiling markdown to PDF, so you may need to wait and install this later if it's not working).

Try pasting in your figure and table from your other RMarkdown (don't forget any libraries you need to make it compile). Presto, ready to submit!

For a bit more on `papaja`, check out this guide.

Bibiographic management

It's also possible to include references using `bibtex`, by using `@ref` syntax. I like bibdesk as a simple reference manager that integrates with google scholar, but many other options are possible.

With a bibtex file included, you can refer to papers like Nuijten et al. [2016] in text, or cite them parenthetically [e.g., Nuijten et al., 2016]. Take a look at the `papaja` APA example to see how this works.

Further Frontiers

Other dissemination methods

We didn't spend as much time on it here, but one of the biggest strengths of RMarkdown is how you can easily switch between formats. For example, we made our slides in RMarkdown using `revealjs` (`ioslides` is also a good option).

We also like to share HTML RMarkdown reports with collaborators using the RPubs service. It's incredibly easy to share your work publicly this way – you just push the "publish" button in the upper right hand corner of the RStudio viewer window. Then you set up an account, and you are on your way. We find this is a great method for sending analysis writeups – no more pasting figures into email!

And of course because not everyone uses R, you can simply click on the knit options at the top of the code window to switch to a Word output format. Then your collaborators can edit the text in Word (or upload to Google docs) and you can re-merge the changes with your reproducible code. This isn't a perfect workflow yet, but it's not too bad.

Computational reproducibility concerns

Once you provide code and data for your RMarkdown, you are most of the way to full computational reproducibility. Other people should be able to get the same numbers as you for your paper!

But there's still one wrinkle. What if you use linear mixed effect models from `lme4` and then the good people developing that package make it better – but that changes your numbers? Package versioning (and versioning on `R` itself) is a real concern. To combat this issue, there are a number of tools out there. The `packrat` pacakge is a solution for package versioning for `R`. It downloads local copies of all your packages, and that can ensure that you have all the right stuff to distribute with your work.[7] Some other groups are using Docker, a system for making virtual machines that have everything for your project inside them and can be unpacked on other machines to reproduce your *exact* environment. We haven't worked with these but they seem like a good option for full reproducibility.

[7] We've found that this package is a bit tricky to use so we don't always recommend it, but it may improve in future.

If you don't want to pursue these heavier-weight options, one simple thing you *can* do is end your RMarkdown with a printout of all the packages you used. Do this by running `session_info` from the `devtools` package.[8]

[8] You can't do this in a manuscript, but you could document your environment in a separate RMarkdown file, for example, or embed the results in a comment.

```
devtools::session_info()
```

```
##  setting  value
##  version  R version 3.4.0 (2017-04-21)
##  system   x86_64, linux-gnu
##  ui       X11
##  language (EN)
##  collate  en_US.UTF-8
##  tz       Europe/Amsterdam
##  date     2017-07-31
##
##  package    * version date
##  assertthat   0.2.0   2017-04-11
##  backports    1.1.0   2017-05-22
##  base       * 3.4.0   2017-05-12
##  bindr        0.1     2016-11-13
##  bindrcpp     0.2     2017-06-17
##  broom      * 0.4.2   2017-02-13
##  codetools    0.2-15  2016-10-05
##  colorspace   1.3-2   2016-12-14
##  compiler     3.4.0   2017-05-12
##  datasets   * 3.4.0   2017-05-12
##  devtools   * 1.13.2  2017-06-02
##  digest       0.6.12  2017-01-27
##  dplyr        0.7.1   2017-06-22
##  evaluate     0.10.1  2017-06-24
##  foreign      0.8-67  2016-09-13
##  formatR      1.5     2017-04-25
##  ggplot2    * 2.2.1   2016-12-30
##  glue         1.1.1   2017-06-21
##  graphics   * 3.4.0   2017-05-12
##  grDevices  * 3.4.0   2017-05-12
##  grid         3.4.0   2017-05-12
##  gtable       0.2.0   2016-02-26
##  highr        0.6     2016-05-09
##  htmltools    0.3.6   2017-04-28
##  knitr      * 1.16    2017-05-18
##  lattice      0.20-35 2017-03-25
##  lazyeval     0.2.0   2016-06-12
##  magrittr     1.5     2014-11-22
##  memoise      1.1.0   2017-04-21
##  methods    * 3.4.0   2017-05-12
##  mnormt       1.5-5   2016-10-15
##  munsell      0.4.3   2016-02-13
##  nlme         3.1-131 2017-02-06
##  parallel     3.4.0   2017-05-12
##  pkgconfig    2.0.1   2017-03-21
##  plyr         1.8.4   2016-06-08
##  psych        1.7.5   2017-05-03
##  R6           2.2.1   2017-05-10
##  Rcpp         0.12.11 2017-05-22
##  reshape2     1.4.2   2016-10-22
##  rlang        0.1.1   2017-05-18
##  rmarkdown    1.6     2017-06-15
##  rprojroot    1.2     2017-01-16
##  scales       0.4.1   2016-11-09
##  stats      * 3.4.0   2017-05-12
##  stringi      1.1.5   2017-04-07
##  stringr      1.2.0   2017-02-18
##  tibble       1.3.3   2017-05-28
##  tidyr        0.6.3   2017-05-15
##  tools        3.4.0   2017-05-12
##  tufte        0.2     2016-02-07
##  utils      * 3.4.0   2017-05-12
##  withr        1.0.2   2016-06-20
##  yaml         2.1.14  2016-11-12
##  source
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  local
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  local
##  local
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  local
##  local
##  local
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.3.3)
##  CRAN (R 3.4.0)
##  local
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  local
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.3.3)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  local
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
##  local
##  CRAN (R 3.4.0)
##  local
##  CRAN (R 3.4.0)
##  CRAN (R 3.4.0)
```

References

Michèle B Nuijten, Chris HJ Hartgerink, Marcel ALM van Assen, Sacha Epskamp, and Jelte M Wicherts. The prevalence of statistical reporting errors in psychology (1985–2013). *Behavior research methods*, 48(4):1205–1226, 2016.