# *RMarkdown for writing reproducible scientific papers*

*Mike Frank & Chris Hartgerink*

*2017-07-28*

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. 

## *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, **?** 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.

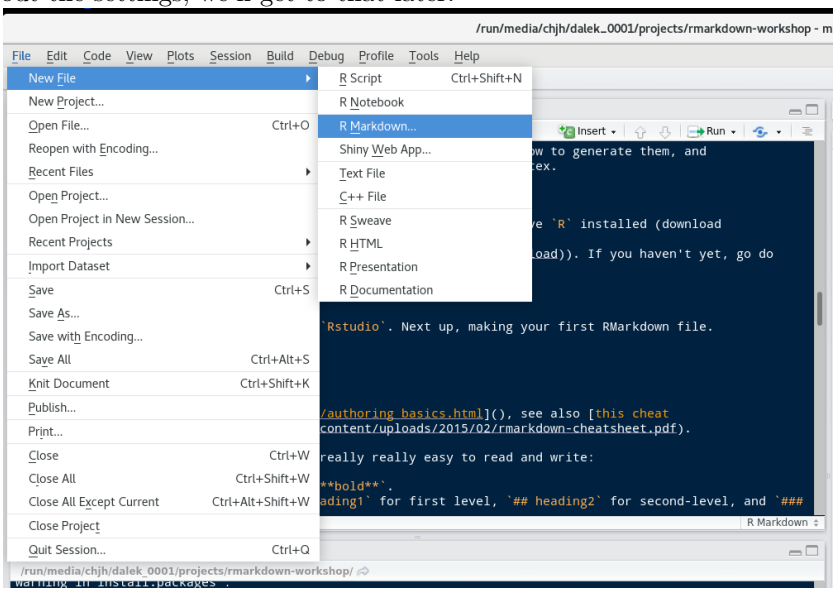
## *Installation*

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!

## *Getting started*

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.




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 get going!

## *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.

```
---
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]

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

[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.

### *Exercise*

1. Create a code chunk, and use `write.csv()` to write out some data. Subsequently, find that file on your computer.
2. Load some of your most used packages in a code chunk and regenerate the document.
3. Load some packages you’ve never heard of, like `adehabitatHS` or `QCA`. What happens?

4. Force RMarkdown to install packages that the user doesn't yet have when the document is generated.

### *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.

### *Tables and graphs*

### *Writing APA-format papers*

THIS # Code and prose, mixed together

First load packages, I always put these together at the top.

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

Now, here are some of the startup options I often use. Caching can be very helpful for large files, but can also cause problems when there are external dependencies that change.

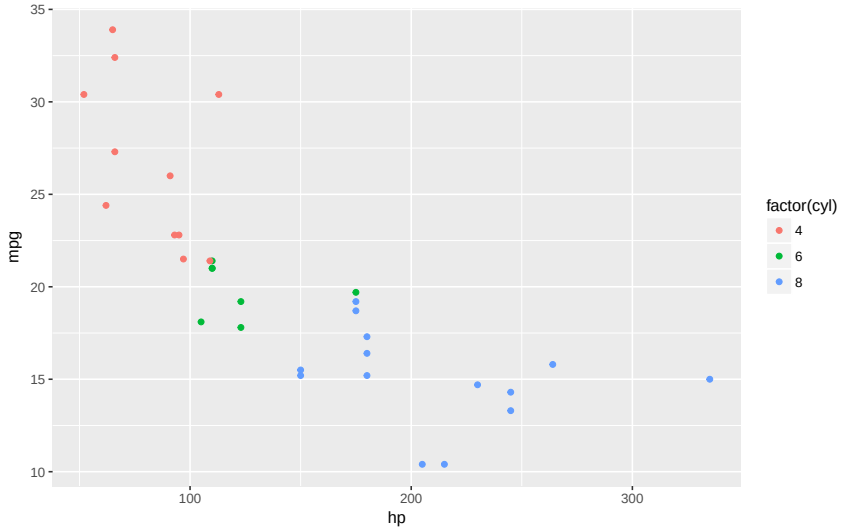
```
opts_chunk$set(fig.width = 8, fig.height = 5,
    echo = TRUE, warning = FALSE, message = FALSE,
    cache = TRUE)
```

And you can use various local and glbal chunk options like `echo=FALSE` to suppress showing the code (better for papers).

*Graphs*

It's really easy to include graphs, like this one.

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




*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`).

I also do a lot of APA-formatted statistics. We can compute them first, and then print them inline.

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

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 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.

## *Putting it all to use!*

## *Writing APA-format papers*

### *Using the `papaja` package*

### *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].

So in conclusion, and as described by Xie [2013], `knitr` is really amazing!

### *Computational reproducibility concerns*

`packrat` for package versioning

or include `session_info` from `devtools` at least.

```
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     2017-07-28
##
##  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
##  labeling     0.3     2014-08-23
##  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 3.4.0)
##  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)
```

could even use `docker`, but that's outside the scope of the course!

## *FAQ*

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.

Yihui Xie. *Dynamic Documents with R and knitr*, volume 29. CRC Press, 2013.