R Markdown Basics

Paul Johnson, Director, CRMDA <pauljohn@ku.edu>

Feb. 20, 2019

Abstract

This discusses the basics of 'R markdown' that are important for authors who want to prepare documents that go beyond the bare rudiments of markdown. In our Center, we expect all authors who prepare documents with 'R markdown' should understand the potential and limitations of markdown when they use it.

Introduction: Terminology

Markdown (John Gruber, `https://daringfireball.net/projects/markdown`) is a different way to write documents that include computer code. It started as a replacement for HTML code used in Websites. Because HTML and LaTeX `markup` was tedious and unreadable, he proposed instead a `markdown` language. The intention was to have documents that are understandable for non-technical readers, but capable of presentation as Web pages or PDF document. The aspirations of the markdown community are now considerably broader. There are many competing dialects of the markup language. There are efforts to generalize the back end format to Open Document, Microsoft Word, and other formats.

`R markdown` is a particular kind of markdown document. Authors should be cautious about following formatting advice for other types of markdown when working on `R markdown`. The distinguishing feature of `R markdown` is that it cooperates with R. Like LaTeX with Sweave, code chunks can be included. When the document is compiled, the code is executed in R and results are tabulated into the output ("knitting" the chunks). The chunk conversion process creates an intermediate file format, usually a `pandoc` markdown file, often with a suffix like `.utf8.md`. That markdown document is then converted to the eventual back end format through one or more additional transitions.

It is important to remember that `R markdown` is not the same as "markdown," for which there are many different dialects. The markdown standard to which `R markdown` is closest is `pandoc` markdown because the program `pandoc` is widely used to convert the `.utf8.md` file into HTML or PDF. However, that is not necessarily the case. On many Web servers, a

1425 Jayhawk Blvd
Watson Library, Suite 470
Lawrence, KS 66045-7594

Phone: 785-864-3353
Web: `http://crmda.ku.edu`
Email: `crmda@ku.edu`

program called Jekyl (`https://jekyllrb.com/`) is used to convert markdown files to static HTML pages for quick rendering.

The `R` packages `knitr` and `rmarkdown` provide functions required to convert code chunks and create the intermediate markdown document. The free-standing program `pandoc` plays the pivotal role in most personal computer applications. It converts markdown files to HTML or `LaTeX`.

Refer Elsewhere for Comprehensive Instructions

We thought we might make YAMG, *Yet Another Markdown Guide*, but now elect instead to explain the big picture issues and focus on practical problems that we expect authors to have when they use the `stationery` document classes.

Reference Guides from the RStudio Corporation (and affiliates)

1. *R Markdown Reference Guide*. Keep a copy of this document available.
2. An RStudio oriented guide, *R Markdown Cheatsheet*
3. *Markdown Basics*. See also the `R markdown` page at Rstudio corporation, `http://rmarkdown.rstudio.com`, which offers a "Get Started" Tutorial.
4. Yihui Xie. 2015. *Dynamic Documents with R and Knitr. 2ed* Boca Raton, Florida: Chapman Hall/CRC. `http://yihui.name/knitr`.

5. On July 7, 2017, Xie made available a new book manuscript, *bookdown: Authoring Books and Technical Documents with R Markdown*.

User-friendly overviews and tutorials on `R markdown` usage.

1. Cosma Shalizi, "Using R Markdown for Class Reports", `http://www.stat.cmu.edu/~cshalizi/rmarkdown`.
2. Wisconsin University Social Science Computing Collaborative. "R for Researchers: R Markdown".
   `http://www.ssc.wisc.edu/sscc/pubs/RFR/RFR_RMarkdown.html`
3. Karl Broman, "Knitr with R Markdown".
   `http://kbroman.org/knitr_knutshell/pages/Rmarkdown.html`

In my experience, there are several document features that can be produced with `R markdown`, but just about as many `LaTeX` features are no available via pure markdown. That puts us in a "gray area," where we are either forced to change our writing style or code the document differently. This is discussed in the section .

Document-wide options

`R markdown` is prepared in a text document that has two parts. The first part, which can be referred to as the header or preamble, includes "metadata". The metadata is information about the document, the authors, and how it must be compiled.

The header is written in the YAML format (YAML originally meant *Yet Another Markup Language*, now stands for *YAML Ain't Markup Language*; see The Official YAML Web Site).

The header for the document you are reading now in PDF is:

```
---
title: "R Markdown Basics"
author:
- affiliation: CRMDA
  description: Director
  email: pauljohn@ku.edu
  name: Paul Johnson
output:
  pdf_document:
    fig_caption: true
    keep_tex: true
    latex_engine: pdflatex
    highlight: haddock
    citation_package: natbib
    pandoc_args: [ --listings ]
    template: "theme/report-template.tex"
fontsize: 12pt
vignette: >
    %\VignetteIndexEntry{stationery}
    %\VignetteEngine{knitr::rmarkdown}
    %\VignetteEncoding{UTF-8}
header-includes:
- |
  ```{=latex}
  \usepackage{xcolor}
  \usepackage{fancybox}
  \usepackage{calc}
  \usepackage{subfig}
  ```
addr:
    l1: 1425 Jayhawk Blvd
    l2: Watson Library, Suite 470
    l3: Lawrence, KS 66045-7594
```

```
    r1: "Phone: 785-864-3353"
    r2: "Web: http://crmda.ku.edu"
    r3: "Email: crmda@ku.edu"
logo: theme/logo.pdf
keywords: guides, report, pandoc, R markdown, sweave, knitr
---
```

Observations about YAML

Parameter and values (`key:value` pairs) are separated by `:`.

One must use the colon, not (**NOT!**) the equal sign).

Logical values are specified by lowercase, unquoted `true`, `false`, or `null`. If a value (say, a document title) includes a colon, quotation marks should be used.

Finding the correct keys

Authors will need to acquire experience about valid keys for their intended output format. The `key: value` pairs in the header are meaningful if the document processing software takes them into account. The correct keys are specified in the document template, so they vary from one document template to another. Otherwise, they are just decorations that on which authors waste their time. To test that idea, authors might insert this in the header of the current document

```
christmaslist: "pandora account", "gold watch"
```

and recompile. There are no errors or warnings about unrecognized key values.

Flush-left `key` values are either top-level parameters (`title`, `date`, `fontsize`) or initiators of metadata blocks. The `header-includes` section is relevant only to the LaTeX stage of document preparation.

The structure of the `author` and `addr` blocks is unique to the template that governs this document (the template is specified in the output block). With different templates, the format of the `key:value` author information will differ.

The format is of the YAML header is VERY sensitive.

A. The header begins with three dashes, in column 1:

```
---
```

and ends with three dashes, in column 1

B. Subsections are represented by indentation. The leading users of `R markdown` prefer indentation using 2 spaces, not 4 as in base R code. We follow that in this document.

C. Errors in formatting are fatal. Using tabs to indent YAML code leads to a ***fatal error***. Thankfully, the error messages point directly at the offending line in the header. Inserting a tab before `highlight: haddock` results in this error:

```
Error in yaml::yaml.load(string, ...) :
  Scanner error: while scanning a plain scalar at line 10,
     column 19
found a tab character that violate intendation(sic) at line
   11, column 1
Calls: rmd2pdf ... parse_yaml_front_matter -> yaml_load_utf8
   -> <Anonymous>
Execution halted
```

The output stanza specifies the back end format

In the stationery package, we are focused primarily on HTML and PDF because, at the current time, these formats are the most dependable.

The output specification in this document means that the function `pdf_document` in the `markdown` package will be used to create an output object. The `.Rmd` file is converted to pandoc markdown, `.utf8.md`, then into LaTeX, then PDF.

It is tricky for authors to figure out which keys will work, and how their values will be specified. Lamentably, it may be necessary to read the source code of an R package like `rmarkdown` to understand all of the details. However, usually, there is some relief in the fact that there is one example working document to exemplify the settings for a particular back end and authors can adjust and modify those values. The `key:value` pairs may may a difference at several stages in the compilation process.

Users should be aware that a moderately complex `R markdown` document will usually be a success only if it is built with a particular output format in mind. Many document components are created with PDF **or** HTML output in mind and they are not generally compatible with both. The number of "cross-compiling" gotchas will probably decline with time, but currently there are many difficulties awaiting someone who writes a document intended for HTML and later wants to generate PDF from the same markdown document.

It is also worth mentioning that the `RStudio` pulldown button Knit can be misleading. It creates the impression that one can choose freely between "Knit to PDF" or "Knit to HTML,"

but that will usually fail. Be warned also that Rstudio will revise the content of the document header, possibly corrupting its formatting.

render equals knitr + pandoc

The process for rendering documents is discussed in the `stationery` vignette. The key point is that the code chunks must be digested by `R` and replaced with valid input and output text. After that, the document is converted by a sequence of transitions, `.Rmd` to `.utf8.md` to `.tex` to `.pdf`.

The conversion of the `.Rmd` file into `pandoc` markdown is known as "knitting". It is the same functionality that is performed by "sweaving" an R noweb `.Rnw` LaTeX document. The `knitr` package for R plays a vital role because it has the procedures that make this an `R markdown` document, rather than just a markdown document with code examples pasted in.

It is a little more difficult to understand the role of the `rmarkdown` package in the document transition process. Perhaps the reader will find some ancient history to be helpful.

In the olden days, say 2014, we could write R markdown documents that did not require the `rmarkdown` package. The workflow was simple.

1. Create a bare minimum YAML header

I once wrote a slide show called "Rmd on Rmd" and called the file `rmdonrmd.Rmd`. There was almost no YAML at all.

```
---
title: "Rmd on Rmd"
author: "Paul E. Johnson"
---
```

That document included R code chunks. It had everything I needed to do at the time. Except I did not understand the problem that the final format might be customized by an intricate array of settings. The importance of those settings will become apparent if we continue the story.

2. Render the document in 2 steps

To convert my cave-person markdown document to HTML, I would use 2 steps in the command line. First, handle the chunks with `knitr`.

```
R CMD knit rmdonrmd.Rmd
```

That created a new file, `rmdonrmd.md`.

I wanted this to turn into a presentation document in a Web format, and somehow I found out that if you told pandoc that your "to" format was `S5`, then a slidy presentation would be created. So I ran:

```
$ pandoc rmdonrmd.md -t S5 -o rmdonrmd.html
```

The `-o` parameter indicates we want an output file `rmdonrmd.html`.

At that time, I did not understand very well the problem that a document that uses the special features of one backend will generally be incompatible with other formats. As long as the document does not use any special features, however, it can be converted to a slideshow in PDF format. This (in 2014, at least) would generate that PDF:

```
$ pandoc rmdonrmd.md -t beamer -V theme:Warsaw -o rmdonrmd.pdf
```

The major danger, then and now, is that an incomplete the output file might be created. Some document elements that we are counting on fail to appear. Incorrect LaTeX markup results in "blank spots" in Web output, for example.

What's wrong with the old way?

The people that are extremely deep in the details of document preparation know that there are ***many*** details that authors might adjust. Do you want a table of contents? If yes, how many headings do you want to include? Do you want code listings to have colored backgrounds? How do you want keywords to be highlighted? Do you want figures to float freely in the document? If so, what algorithm is to be used.

All of these features are in play and most of us–authors–don't want to know all of the details about how the `R markdown` document is turned into the eventual result. The authors of the `rmarkdown` document are the ones who know the ins-and-outs and they created an elaborate system that reads the `keys` in the markdown document and then reformats it and, most importantly, calls `pandoc` with an immense list of command line options.

If I compile an `R markdown` document today, and I allow verbose output of the steps used to build the document, I see, at the very end, the `pandoc` function call.

```
/usr/bin/pandoc +RTS -K512m -RTS rmdonrmd.utf8.md --to slidy
--from
   markdown+autolink_bare_uris+ascii_identifiers+tex_math_single_backslash
--output rmdonrmd.html --smart --email-obfuscation none
   --self-contained
--template
   /usr/local/lib/R/site-library/rmarkdown/rmd/slidy/default.html
--include-in-header
   /tmp/RtmpAa6kai/rmarkdown-str2de413c62f71.html
```

```
--mathjax --variable
   'mathjax-url:https://mathjax.rstudio.com/latest/MathJax.js?config=TeX-A
--highlight-style pygments

Output created: rmdonrmd.html
```

This `pandoc` call converts a pandoc markdown file, `rmdonrmd.uft8.md` (which was created from rmdonrmd.Rmd) into HTML.

In a nutshell, we need the `rmarkdown` package because it can deal with the immense complexity of `pandoc`. The `rmarkdown` package is the glue. It creates output format objects that specify a large family of sensible settings that are sent to `pandoc`. Because there were some complications in the creation of documents with the custom templates in the `stationery` package, I had to study the code in `rmarkdown` for quite a while. Every time I came away with just one thought. It would have taken me several years to piece together all of those details.

Preparing an R Markdown Document

Use a text editor

An `R markdown` file is suffixed ".Rmd". It is a "raw text" file. Don't edit it with MS Word. Instead, use a programmers file editor, such as `Emacs`, `Notepad++`, or an integrated development environment, such as `RStudio`, to edit the file.

Be considerate of the .Rmd document's reader

Remember that the markdown philosophy is that a document should look reasonably nice, even before it is compiled. That means authors should take care that their document has reasonably "shaped" paragraphs. Lines should generally be 80 characters or less.

Document body has markdown, plus ...

The body of the `R markdown` document can include both "markdown" syntax, which we hope is compatible with either PDF or HTML backends, as well as a selection of backend-specific code input. In a document with an HTML backend, authors can write "raw" HTML when they can't find R markdown syntax that works.

In our experience, it is also *mostly* workable to include LaTeX
code in `R markdown` documents. Recall that documents in `R markdown` are converted from `Rmd` to `md` by the knitting process, and then into LaTeX by `pandoc`, and then into PDF by

a LaTeX compiler (`pdflatex`, `xetex` or similar). As a result, one can insert quite a bit of LaTeX code and it will often be successfully post-processed in the last stage of document compilation.

We have found that some LaTeX code does not work well, however, so it is not possible at the current time to say "everything works". However, many things do work.

As in LaTeX itself, when an author wants to add features, it is generally necessary to add `\usepackage{}` statements in the LaTeX
document preamble. We use a LaTeX template for this document; it includes many packages. Packages that are not included in the template can be tucked into the YAML preamble:

```
header-includes:
-  \usepackage{amsmath}
```

We did not realize that `amsmath` was necessary until an author typed in a valid formula that ended up with missing characters in the output file.

Basic R Markdown

These are the most frequently used markdown idioms that work well in both HTML and PDF backends. These things work well in either backend because they are understood to `pandoc`, which has the job to convert the `markdown` document into either LaTeX or HTML.

1. Character styling.
    - *Italics* result from asterisks (`*italics*`)
    - **bold face** results from two asterisks (`**boldface**`).
    - Inline `computer code` should be a typewriter fixed width font produced by using two "backticks" (`` `computer code` ``)
2. Section headers.
    - `#` is level 1 header
    - `##` is level 2 subheading
    - levels below are used differently in HTML and PDF. The ability of html documents to include specialized subsections is one of the advantages of that backend. In R markdown,6 levels of headings are allowed.

    There is a problem to be aware of. A section header may not have a linebreak in it.

    An error so simple as writing a section title that has a line wrap will cause poor quality output.
3. Paragraph styling
    - Paragraphs are created by blank lines

- Single-spaced line: Two blank spaces at end of line signify a new line without creating new paragraph (blank line) below.
- Block quotes: Indented material like this:

  Shall I compare thee to a summer's day? Thou art more lovely and more temperate.

begins with a > character in column 1, followed by a space:

```
   Shall I compare thee to a summer's day?
   Thou art more lovely and more temperate.
```

The second and following lines should be entered without blank lines.

- code listings that are not R chunks, such as

```
int x = 3
```

are requested by three ticks before and after

```
   ```
   int x = 3
   ```
```

4. Lists.
   - Numbered lists can be created by putting a number in column 1 of a new line, followed by a period.
   - Unnumbered (Bullet) lists can be created by putting an asterix in column 1 of a new line, followed by a space.
   - Lists can be nested by typing four blank spaces for each level of nesting.
     - A continuation of the existing list's style can be obtained by inserting a plus sign.
   - It is possible to insert paragraphs within lists. These should be indented 3 spaces.
5. Hyperlinks, such as CRMDA, for which the markdown is `[CRMDA](https://crmda.ku.edu)`.
6. Citations. Need to know how to use natbib citations in a markdown document ?? ???

This *basic markdown* is compatible with either HTML or PDF output. While I am writing in a document intended for PDF, I can use LaTeX code `\emph{italicize}` to *italicize*, as you can see, but I can also use markdown syntax `*italicize*` to *italicize*. If I were intending this document for HTML, I could insert `<i>italicize</i>` to reach the same result.

Where feasible, we should use the generic markdown syntax rather than backend-specific markup, because this keeps open the door that we might want to change the backend from PDF to HTML at some point. Of course, *the key assumption is that the backend-neutral methods actually work* to get the job done, and that is simply untrue in my experience. Nevertheless, this is a valuable aspiration and it is a focal point of current software development.

Math in Markdown documents

There's a fairly broad range of mathematical markup that will work in R markdown documents intended for PDF and HTML. So far as we can tell, documents intended for PDF (which can add LaTeX packages) can exploit all of the mathematical features, while HTML documents do not have that same advantage. There are other weaknesses in the math tool chain for HTML documents, to which we shall return.

Here are the highlights.

1. In-line math markup is dollar-sign bracketed, just as in LaTeXitself. We write `$f_{ij} x_{i} \times y_{j}^2$` for $f_{ij}x_i \times y_j^2$. Using the usual LaTeX format, we can write the Greek letters, binary relations, and other important symbols. Behold:

$$\alpha, \beta, \gamma, \Gamma, \chi, \approx, \neq, \geq, \leq, \pm, \sim, \rightarrow, \pi, \text{and} \infty.$$

2. Displayed equations–equations that are centered or otherwise set apart from the text–can be bracketed by `\[` and `\]`. In LaTeX, it is not longer recommended to use the double-dollar sign notation for displayed equations.

Compare the inline mode, $\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$ with the display equation:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i.$$

that results from

```
\[
\bar{x} = \frac{1}{n} \sum_{i=1}^{n}x_{i}.
\]
```

After some testing, it appears that in order to have equations with numbers (via labels, allowing cross references), the best avenue is to change the display markup to use `amsmath` equations. In the following, we use the LaTeX `\begin{equation}`, which comes from the LaTeX package *amsmath*.

```
\begin{equation}
\bar{x} = \frac{1}{n} \sum_{i=1}^{n}x_{i}.
\label{eq:mean}
\end{equation}
```

This equation is numbered, because the label macro is included within the equation:

$$\int \beta\alpha\Psi\Xi\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i. \tag{1}$$

Because we labeled the equation, we can now use cross references. For example, as the reader can see in equation (1), the arithmetic mean is a very lovely formula. If we insert equations before or after that one, they will be renumbered automatically and cross references will be adjusted.

Another benefit of introducing the amsmath support is that arrays can be demonstrated.

$$A_{m,n} = \begin{pmatrix} a & b & c \\ d & e & f \\ g & h & i \end{pmatrix} \tag{2}$$

Given the fact that the `equation` environment provides all of the needed features, it seems obvious that authors should prefer to use it, rather than the simpler `\[` markup.

This is not the right place for a comprehensive guide to LaTeX. Users will need to install a LaTeX distribution and then a user-friendly editor that will help create equation markup. Many users in the CRMDA lab use LyX, which offers a "View source" feature to inspect LaTeX coding for equations.

Syntax and code chunks

Code chunks are discussed in a separate vignette that is available with the `stationery` package, "Code Chunks". Code chunks are allowed in `R markdown` documents intended for either HTML or PDF.

In `R markdown` documents intended for HTML output, `knitr` is the only chunk-processing technology that is available. In `R markdown` documents, the outer boundaries of a chunk are demarcated by three back ticks, along with squiggly braces and the letter "r", which designates the language being processed (`knitr` can handle several languages).

```
```{r}
lm(y ~ x, data = dat)
```
```

In contrast, a chunk in a `noweb`/LaTeX document would look like this:

```
<<>>=
lm(y ~ x, data = dat)
@
```

Options which control chunk management are quite elaborate in `knitr`, but most authors will find that there are 5 (or so) options that they use frequently and the others can be discovered in the documentation when emergencies arise:

1. eval

2. echo
3. include
4. results

These control if the chunk is calculated, displayed, and included in the output.

`knitr` has fine grained settings for sizing, saving, and placing figures. These are discussed in our vignette, "Code Chunks". We expect that all authors will not have too much difficulty creating chunks that

1. are not evaluated, but are displayed "beautifully" to the reader.
2. are not displayed, but are evaluated, and the results may (or may not) be displayed for the reader.
3. create graphics, which are automatically included in the document.
4. create graphics (or other files) that are not automatically included in the document. Graphics are saved in image files that can be inserted at a different part in the document
5. import previous chunks for re-analysis.

Presentable tables

In a report document, it would be unusual to display a large chunk of R code or output. Instead, reports typically include stylized tables, graphs, and equations.

Unfortunately, this is a point at which the backend-neutral document processing strategies break down. Markdown allows some elementary tables, but these are unsuitable for statistical tables. Instead, one needs to create backend-specific tables in order to have something presentable.

The very nice table-making routines for R were originally developed for PDF/LaTeX, and alternative backends like HTML were added later. At the current time, regression tables can be generated in either LaTeX or HTML by the R package `texreg`, `xtable`, `rockchalk`, and many others.

Warnings aside, here are 2 table-making recipes

I have used two approaches to include tables in documents.

1. The "automatic" method. Write a code chunk that creates an output table that is good enough to go into the report without revision and allow the document to use it. The chunk option `results="asis"` is used to signal that the result of the chunk is valid markup that is to go directly into the backend document.
2. The "slow lane" method. Write a chunk that creates a table in a file on disk. Then one can review and edit the file, and include it in the document manually.

In "guides" and less formal documents, I often use the automatic method (beauty of presentation is not the primary emphasis). In reports to clients, I often need to use the second method because tables almost never come out in a perfectly presentable format.

Because I wrote the `outreg` function in the rockchalk a long time ago, it still holds a special place in my heart. I will use it for a demonstration. `outreg` works with standard R regressions (`lm`, `glm`). Suppose we have data about cancer prevalence (y) and predictors including exposure to plutonium (x1), vegetables (x2), and tobacco (x3).

Regressions are estimated. The second regression includes interactions.

```
m1 <- lm(y ~ x1 + x2 + x3, data = dat)
m2 <- lm(y ~ x1 + x2*x3 + x1*x3, data = dat)
```

The variable labels are created as `varLab` and they are supplied in the call to `outreg`.

```
```{r reg20, results="asis"}
varLab <- c(y = "Cancer", x1 = "Plutonium", x2 = "Veggies",
            x3 = "Tobacco", `x2:x3` = "Veg $\\times$ Tob",
            `x1:x3` = "Plut $\\times$ Tob")
or10 <- outreg(list("Cancer OLS" = m1, "Cancer Interaction" =
   m2),
                tight=FALSE, varLab = varLab)
```
```

The table is inserted flush left in the document at the point of the code chunk because I did not create a floating table object in which to situate it.

	Cancer OLS		Cancer Interaction	
	Estimate	(S.E.)	Estimate	(S.E.)
(Intercept)	-722.408***	(5.303)	-254.862***	(16.531)
Plutonium	15.974***	(0.078)	11.703***	(0.289)
Veggies	13.608***	(0.105)	8.488***	(0.224)
Tobacco	0.303**	(0.110)	-9.333***	(0.338)
Veg × Tob	_		0.105***	(0.004)
Plut × Tob	_		0.085***	(0.006)
N	1000		1000	
RMSE	23.882		17.000	
R^2	0.988		0.994	
adj R^2	0.988		0.994	

$*p \leq 0.05 ** p \leq 0.01 *** p \leq 0.001$

The other way to do this is to save the table in a file, and then include it in a document. I do this in some reports because the LaTeX (or HTML) tables created by software are never *exactly* right and I want to fine tune them.

```
cat(or10, file = "tmpout/or10file.tex")
```

Table 1: Regression Analysis

	Cancer OLS		Cancer Interaction	
	Estimate	(S.E.)	Estimate	(S.E.)
(Intercept)	-722.408***	(5.303)	-254.862***	(16.531)
Plutonium	15.974***	(0.078)	11.703***	(0.289)
Veggies	13.608***	(0.105)	8.488***	(0.224)
Tobacco	0.303**	(0.110)	-9.333***	(0.338)
Veg × Tob	_		0.105***	(0.004)
Plut × Tob	_		0.085***	(0.006)
N	1000		1000	
RMSE	23.882		17.000	
R^2	0.988		0.994	
adj R^2	0.988		0.994	

$*p \leq 0.05** p \leq 0.01***p \leq 0.001$

The file "or10file.tex" can be edited by hand, and then inserted into the LaTeX document (for the PDF backend). If I am using this method, I can insert a floating table with a proper caption and label. See Table 1, which results from the following.

```
\begin{table} \caption{Regression Analysis\label{tab:reg30}}
\input{tmpout/or10file.tex} \end{table}
```

Admittedly, this two step process to create the floating LaTeX label is not truly necessary. The `outreg` function could do that for us. The same result using the results="asis" chunk:

````
```{r reg40, results="asis"}
or10 <- outreg(list("Cancer OLS" = m1, "Cancer Interaction" =
   m2), tight=FALSE,
                varLab = varLab, title = "Regression Analysis
                   asdfasdf",
                label="tab:reg30")
```
````

Because this document is intended for the PDF backend, any LaTeX
generating table function should work. For linear regressions, multilevel models, and glm fits, I find the `rockchalk::outreg` output is fine, but other kinds of models often arrive with custom table-making routines. I have not found a single approach that works beautifully for all kinds of regressions, and I expect other authors will say the same thing because new packages to create LaTeX tables seem to crop up in the R repositories on a monthly basis. It is worth mentioning that some of the LaTeX table generators, like `texreg`, allow the use of `bookman` markup to beautify the placement of horizontal lines, so the document preamble must be modified to include that package.

I often need to include simpler tabular material in documents and I always try to use the

package `xtable`. `xtable` is not the simplest package, but it has been around longer than the others and it has more detailed features than any other package. In particular, the `longtable` class, which allows tables that gracefully split over "page breaks", is available. If I cannot get the result I want with `xtable`, the second approach I try is the `latex` function in the `Hmisc` package. There are plenty of other table-making R packages, but my opinion is that if one can master `xtable`, then one can generate just about any kind of table.

Warnings, cautions, etc.

Dialects

R markdown is similar to, but the exactly the same, as the general markdown specification. It is different from other specialized dialects, such as the GitHub Flavored Markdown and other variants. Be cautious about using style guides that are not intended explicitly for `R markdown` documents. There is a good chance the advice about preparing a document from those other sites is irrelevant to an `R markdown` document. Google searches will sometimes lead to the wrong answer for a particular form of Markdown.

Inaccurate rendering

Another major concern is that illegal LaTeX code in documents intended for HTML will be ignored in the document production process. The document compiles without error, but HTML result is incomplete. Where we expect to see math, there is *empty nothing*. By itself, this problem seems serious enough to make me think the HTML backend should not be used. The `markdown` → HTML conversion is the weak spot. The `markdown` → PDF transition is not similarly flawed. It halts with error messages. The conclusion from all of this is the following:

> *Authors must exert themselves to proofread the HTML output to make sure it appears as intended.*

Another problem we have discovered is that even valid HTML markup can sometimes be corrupted by `pandoc`. Valid HTML (and PDF, for that matter) can include empty spaces that are inserted by authors to make their markup easier to read. The spaces are confusing to `pandoc`, which may interpret them as markdown formatting statements (markdown uses spaces to indicate nesting of lists or code sections. At the current time, it appears to be necessary to process all HTML markup that goes into the markdown document to remove spaces in the beginning of lines.

In our experimentation with R markdown, we have noticed another quirk. I document intended for PDF, one writes `\LaTeX` to produce LaTeX. On the other, in a document intended for HTML, writing `\LaTeX` produces one of those empty spaces previously mentioned. Instead, in the document intended for HTML one must insert `$\LaTeX$`, as if this were a valid

mathematical expression, which it is not. Inserting `$\LaTeX$` in `R markdown` intended for PDF results in an error message:

```
! You can't use `\spacefactor' in math mode.
\@->\spacefactor
                  \@m {}
l.934 ...to compile and the error message \(\LaTeX

pandoc: Error producing PDF
```

Mathjax needed for HTML documents

Finally, there is a fundamental weakness of HTML documents. Math is not "in" the HTML file. Instead, there is javascript code that is shown in the Web browser using a framework called MathJax. MathJax allows inclusion of math markup in the page which–when the conditions are right–can be converted by the Web browser to look like equations. The MathJax display system works better and better over time; it *almost always works*. The Web browser must be able to retrieve files from the MathJax server in order to display the document. If the reader is offline, or the MathJax server fails. Users will see markup, rather than equations, when the network fails.

Illegal encoding of pasted fonts

While working on this document, I found a web page with a lovely Shakespeare sonnet. I carelessly copy/pasted that into a code listing. Look what happened at compile time:

```
pauljohn@delllap-16:vignettes$ ./rmd2html.sh Rmarkdown.Rmd
Error in tools::file_path_as_absolute(output_file) :
  file 'Rmarkdown.pdf' does not exist
Calls: rmd2html -> do.call -> <Anonymous> -> <Anonymous>
In addition: Warning messages:
1: In grep(paste(r, collapse = "|"), text, value = TRUE) :
  input string 1312 is invalid in this locale
2: In grep("^\\s*$", x) : input string 1312 is invalid in this
   locale
3: In grep("^! ", x) : input string 1312 is invalid in this
   locale
4: In grep("^\\s*$", x) : input string 1312 is invalid in this
   locale
5: In grep("^! ", x) : input string 1312 is invalid in this
   locale
6: In file.rename(file_with_ext(texfile, "pdf"), output_file) :
```

```
Execution halted
```

The error message means that there were some characters in the web page that were encoded in an unfamiliar format.

Bibliographical citation test is appropriate.

A citation with name and date is obtained as (Diggle, Heagerty, Liang, & Zeger, 2013). A citation that with just the date, i.e., Diggle and colleagues (2013, 37) is obtained differently. To get the full parenthesized statement with names and dates, we insert hard brackets `[`, an `@` sign, the bibtex tag, and acloser `]`. If we don't want their names, we insert a `-` sign. It is also possible to refer to a group of projects (Fitzmaurice, Laird, & Ware, 2011; Hsiao, 2014; McCullagh & Nelder, 1983).

In HTML output, the references are always generated as the very last piece of the document. We insert a section name in the very last line to prepare for than.

```
# References
```

In HTML output, the style of the bibliography will not match what we generally want. What we get is close, the Chicago author-year format. We need a "CSL" file to alter the style of the output. See authoring_bibliographies_and_citations.

If the back end is PDF, as in this document, there is some more encouraging news. A combination of YAML header magic and some "raw" LaTeX succeeds in creating a bibliography for this document.

Conclusion

For the sake of clarity, then, I state the following:

> If the backend in PDF, one should should use functions that provide tables using LaTeX code.

On the other hand,

> if the backend is HTML, then functions that create HTML markup code should be used.

Session Info

```
warnings()
sessionInfo()
```

```
R version 3.5.2 (2018-12-20)
Platform: x86_64-pc-linux-gnu (64-bit)
Running under: Ubuntu 18.10

Matrix products: default
BLAS: /usr/lib/x86_64-linux-gnu/blas/libblas.so.3.8.0
LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.8.0

locale:
 [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
 [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
 [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
 [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
 [9] LC_ADDRESS=C               LC_TELEPHONE=C
[11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C

attached base packages:
[1] stats     graphics  grDevices utils     datasets  methods
[7] base

other attached packages:
[1] rockchalk_1.8.140 stationery_0.98.7

loaded via a namespace (and not attached):
 [1] Rcpp_1.0.0       knitr_1.21       magrittr_1.5
    splines_3.5.2
 [5] kutils_1.64      MASS_7.3-51.1    mnormt_1.5-5
    lattice_0.20-38
 [9] pbivnorm_0.6.0   xtable_1.8-3     minqa_1.2.4
    carData_3.0-2
[13] stringr_1.3.1    plyr_1.8.4       tools_3.5.2      grid_3.5.2
[17] nlme_3.1-137     xfun_0.4         htmltools_0.3.6
   lme4_1.1-19
[21] yaml_2.2.0       digest_0.6.18    lavaan_0.6-3
   Matrix_1.2-15
[25] zip_1.0.0        nloptr_1.2.1     evaluate_0.12
   rmarkdown_1.11
[29] openxlsx_4.1.0   stringi_1.2.4    compiler_3.5.2
   stats4_3.5.2
[33] foreign_0.8-71
```

References

Diggle, P., Heagerty, P., Liang, K.-Y., & Zeger, S. (2013). *Analysis of Longitudinal Data* (2nd ed.). Oxford: Oxford University Press.

Fitzmaurice, G. M., Laird, N. M., & Ware, J. H. (2011). *Applied longitudinal analysis* (2nd ed.). Hoboken, N.J: Wiley.

Hsiao, C. (2014). *Analysis of panel data* (3rd ed.) (No. 54). New York, NY: Cambridge University Press. (OCLC: ocn872561839)

McCullagh, P., & Nelder, J. A. (1983). *Generalized Linear Models* (No. 37). London: Chapman and Hall.