DEPARTMENT OF ECONOMICS
WORKING PAPER SERIES

2017-02

McMASTER UNIVERSITY

Department of Economics
Kenneth Taylor Hall 426
1280 Main Street West
Hamilton, Ontario, Canada
L8S 4M4

<http://www.mcmaster.ca/economics/>

Energy, Economics & Replication

Jeffrey S. Racine*

Department of Economics and Graduate Program in Statistics, McMaster University,
racinej@mcmaster.ca

February 13, 2017 (17:44)

Abstract

This article outlines recent developments in *Markdown* scripting languages that facilitate the production of replicable, publication quality, research. The approach is similar to that achieved by using, say, Sweave, R and LaTeX, but is written instead in simple Markdown syntax and not tied to any particular output format (e.g., MS Word) nor computational language (e.g., Python). The computational component can be written in C++, Python, SQL, Stan, Bash, or R by way of example. The Markdown script is seamlessly converted to any one of a number of output formats. The output format is essentially an afterthought, and could be rendered as a PDF (LaTeX or Beamer presentation), MS Word, HTML, EPUB, or gitbook document, by way of illustration. Conversion of the Markdown script to the desired output format is performed by pandoc (a universal document converter). These tools can dramatically reduce the amount of time required to complete a research project that can be trivially replicated. Recent enhancements to RStudio streamline the entire process of output format generation via a simple click of an icon or keystroke shortcut (the minimum requirement is R). Replicability is guaranteed by using the **checkpoint** package in R. This article was written using Markdown.

1 Markdown and Replicability

We begin by looking at some of the benefits of using Markdown, R Markdown, and R bookdown for conducting research, writing research reports, and rendering the Markdown script into a publication quality document.

1.1 Markup and Markdown

Some background is likely in order. *Markup* is a *heavyweight* programming language used to annotate text in digital documents. The phrase *Markup language* comes from the marking up of manuscripts, where handwritten markups were used to annotate printer instructions. Hypertext Markup Language (HTML) is currently the most widely used Markup language and underlies much of the content displayed on the internet. Markdown, on the other hand, is a *lightweight* Markup language that adopts a simple plain text formatting syntax.¹ Markdown was designed so that the Markdown script can be converted to any one of a number of output formats including HTML, PDF (LaTeX), MS Word, EPUB, gitbook and so forth. You create Markdown scripts with plain text editors. You convert Markdown scripts to the desired output format with pandoc, though this will be effortlessly performed on your behalf as will be seen.

The difference between the *heavyweight* HTML Markup code and the *lightweight* Markdown code can be seen in the following illustration.

HTML	Markdown
<!DOCTYPE html>	---
<html>	output: html_document
<body>	---
<h1>My First Heading</h1>	# My First Heading
<p>My first paragraph.</p>	My first paragraph.

*I would like to thank Yihui Xie for his guidance and feedback and for his incredible contributions to the R community.

¹Markdown is a subset of HTML.

HTML	Markdown
<code></body></code>	
<code></html></code>	

HTML code requires the use of idiosyncratic commands such as *anchors*² and other style elements, and clearly you have pre-committed to output rendered in HTML. But the Markdown code requires very little formatting yet can be rendered exactly as the HTML code on the left would render. Furthermore, by simply changing `html_document` to `word_document`, the output would be rendered as an MS Word document (a range of output formats are available).

1.2 Why adopt Markdown for your research?

There are many reasons to consider using Markdown for producing documents. One reason is that you focus on your narrative instead of fiddling around with stylistic elements such as page breaks, page layout, typesetting and fonts, to name but a few. Often you spend far too much time tweaking silly things that turn out to be inconsequential. For example, if you are an MS Word user you not only have committed to MS Word going forward, but you know that manual page breaks often change when you print your document on different printers, and you know that sharing your document with others may be problematic if they do not have the same version of MS Word installed on their computer and you are using the MS Word Equation Editor, and so forth. Now, what happens if you submit your research to a journal that requires LaTeX formatted documents? Well, you then expend a substantial amount of *energy* rewriting your document in LaTeX, which is not an *economical* use of your time.³

Another reason for adopting Markdown arises when you are required to provide the *same information* in *multiple output formats* that are not available in your current framework. For example, as an academic, you might need to provide a *Curriculum Vitae* (CV) as an MS Word document for annual reviews, as a PDF document for granting agencies, and if you post your CV online it is customary to post it both in HTML and PDF (LaTeX) format. Some readers might therefore need to maintain more than one version of their CV in different formats and then have to manually synchronize them. But if you author your CV in Markdown script, this process becomes trivially simple.⁴

Once you master a few simple rules for writing Markdown scripts, you can render the script in any one of a range of output formats with no change whatsoever to your Markdown script (and with only a very minor tweak to the YAML⁵ metadata for your script). So if one audience requests output in MS Word, you simply render the Markdown script in that format. Another audience might request LaTeX output, so you render in PDF (which produces the LaTeX code) and so forth. With Markdown you remain completely agnostic regarding the output format, which turns out to be a very good thing indeed.

But these benefits might strike some as insufficient grounds for migrating to a new environment. However, when you combine Markdown with a programming language such as C++, Python, SQL, or R (R Core Team 2016) by way of illustration, you embark on a replicable journey that can dramatically reduce the amount of energy spent on the design of your document, time better spent crafting your narrative and conducting your research. In this framework you only work with one document, the Markdown script itself, which facilitates replicability as you no longer have code in one directory, figures in another and so forth. Errors that arise from manual copying and pasting from one program to another are sidestepped. Minor changes to data or code that would previously require substantial effort to update tables, figures and so forth become moot.

²An *anchor* is a chunk of text which marks the beginning and/or the end of a *hypertext* link.

³Kindly pardon this somewhat tenuous link to *Energy* and *Economics* that appear in the title.

⁴If you currently maintain a LaTeX formatted CV named `cv.tex`, you can convert it to Markdown using pandoc via `pandoc -s cv.tex --to markdown --output cv.Rmd` and then, if any cleaning up is needed, you can clean up `cv.Rmd` and edit only this Markdown script going forward (if your existing CV is in MS Word `docx` format simply replace `-s cv.tex` with `-s cv.docx`).

⁵YAML (a recursive acronym, *YAML Ain't Markup Language*) is a data serialization language commonly used for configuration files. In Markdown scripts, YAML metadata appears at the very top of the file enclosed in three dashes at the top and bottom.

Extensive sensitivity analysis becomes trivial and, above all, at the end of the journey you have a document whose replicability is guaranteed, as outlined in more detail below.

1.3 Markdown, Sweave, and knitr

Readers familiar with Sweave or knitr (Xie 2016b) may be wondering how Markdown differs from these environments. The answer is that users of Sweave and knitr have pre-selected and committed themselves to a single output format (LaTeX) and expend a substantial amount of energy on the nuances of LaTeX itself, part of which can be conserved. R Markdown⁶ does, however, make heavy use of knitr; see Racine and Hyndman (2002) and Meredith and Racine (2009) for an historical perspective.

1.4 Markdown requires fewer keystrokes than its peers

LaTeX requires care to get things right, and of course going forward you are restricted to LaTeX. In Markdown you have YAML metadata similar to LaTeX metadata indicating the title, author, date etc. But after this initial *heavy* syntax, writing the Markdown script is much lighter and requires noticeably less effort, and you remain agnostic about the output format. The difference between the *heavyweight* LaTeX Markup code and the *lightweight* Markdown code can be seen below.

LaTeX	Markdown
<code>\documentclass[11pt,reqno]{amsart}</code>	---
<code>\title{This is a title}</code>	<code>title: This is a title</code>
<code>\date{July 1 2020}</code>	<code>date: July 1 2020</code>
<code>\author{Me R. D. Author}</code>	<code>author: Me R. D. Author</code>
<code>\begin{document}</code>	<code>bibliography: mybib.bib</code>
<code>\section{Introduction}</code>	<code>output: pdf_document</code>
<code>\bibliography{mybib}</code>	---
<code>\end{document}</code>	<code># Introduction</code>

Changing the line `output: pdf_document` in the Markdown code to `output: html_document` renders the Markdown script in HTML output format instead of LaTeX/PDF output format. Furthermore, adding the option `keep_tex: true` below `output: pdf_document` will output the LaTeX code should you require it. The proper syntax⁷ in the YAML metadata would be

```
output:
  pdf_document:
    keep_tex: true
```

1.5 Universal special characters in Markdown are simple

In certain languages, often *special* characters are *reserved*. For instance, typesetting characters such as hash signs, percentage signs, quotations and ampersands in LaTeX requires care (in LaTeX you must *escape* some characters i.e., write `\#`, `\%` and `\&`, and use two left/right single quotes to get quotation marks). In Markdown you can type the plain text symbols `#`, `%`, `&` and “your keyboard quotation mark” and they will be properly rendered in whichever output format you select (i.e., you use your keyboard quotation symbol at the beginning and end of a character string and it is properly rendered, e.g., “as”).

⁶See <http://rmarkdown.rstudio.com/articles.html> for assistance with using R Markdown.

⁷Note that spacing must adhere to YAML rules, so `pdf_document:` is indented two spaces, `keep_tex: true` four spaces, and a colon proceeds `pdf_document`.

1.6 Creating universal formatted lists in Markdown is simple

Lists in LaTeX, by way of illustration, require idiosyncratic syntax. In Markdown lists are far simpler (you simply announce a list item with a dash in the first column).

LaTeX	Markdown
<code>\begin{itemize}</code>	- Item 1
<code>\item Item 1</code>	- Item 2
<code>\item Item 2</code>	
<code>\end{itemize}</code>	

1.7 Creating LaTeX Beamer presentations in R Markdown is simple

Creating LaTeX Beamer⁸ slides requires idiosyncratic syntax. In R Markdown it is far simpler (four dashes begin a new slide, or use `# frametitle` to begin a new slide with the title `frametitle`).

LaTeX Beamer	Markdown
<code>\documentclass[xcolor=dvipsnames,mathserif,11pt]{Beamer}</code>	---
<code>\usepackage[]{graphicx}</code>	output: beamer_presentation
<code>\usepackage[]{color}</code>	---
<code>\begin{frame}{Title}</code>	# Title
<code>\begin{itemize}</code>	- Yadda
<code>\item Yadda</code>	- Yadda
<code>\item Yadda</code>	- Yadda
<code>\item Yadda</code>	- Yadda
<code>\item Yadda</code>	- Yadda
<code>\item Yadda</code>	----
<code>\end{itemize}</code>	
<code>\end{frame}</code>	
<code>\end{document}</code>	

Note that, in the table above, there is nothing special about the Markdown code on the right that mimics the LaTeX Beamer code on the left, yet when you render it in a presentation format such as Beamer, it renders as if you wrote the LaTeX Beamer code manually. And it will generate the LaTeX Beamer code for you if you simply include `keep_tex: true` in your script YAML metadata. You can change themes and default behaviour by modifying the YAML metadata. For instance, if you wanted the LaTeX code to be output, items on slides to display incrementally, and to use a particular theme, color theme, and font theme, you would include the following in your YAML metadata:

```
output:
  beamer_presentation:
    keep_tex: true
    incremental: true
    theme: AnnArbor
    colortheme: dolphin
    fonttheme: structurebold
```

For a list of Beamer themes see http://deic.uab.es/~iblanes/beamer_gallery.

⁸Beamer is a LaTeX document class for creating presentation slides.

1.8 Creating universal formatted tables in Markdown is simple

Formatting tables in LaTeX requires substantial effort. Markdown tables are much simpler. When you select the PDF output format, Markdown automatically renders a LaTeX table, but would also produce a correctly formatted table in MS Word, HTML etc.

LaTeX	Markdown
<code>\begin{longtable}[]{@{}lll@{}}</code>	Header A Header B Header C
<code>\toprule</code>	----- ----- -----
Header A & Header B & Header C <code>\tabularnewline</code>	A B C
<code>\midrule</code>	D E F
<code>\endhead</code>	
A & B & C <code>\tabularnewline</code>	
D & E & F <code>\tabularnewline</code>	
<code>\bottomrule</code>	
<code>\end{longtable}</code>	

For guidance on creating tables using Markdown see <http://pandoc.org/MANUAL.html#tables>.⁹ See also the function `kable()` in the `knitr` package.

1.9 Migrating to Markdown is easy

R Markdown (Allaire et al. 2015) is a simple formatting syntax for writing Markdown scripts in R and RStudio¹⁰ (RStudio Team 2015). R Markdown adopts LaTeX syntax for typesetting mathematics and the BibTeX system for incorporating references (`\cite{foo}`, a BibTeX/LaTeX construct, is rendered using `@foo` in R Markdown). With the assistance of the R `knitr` package, you combine (*knit*) your narrative and your programming code in the same script and the output format is rendered by pandoc. This framework automatically renders any graphics, tables, mathematics, footnotes and the like in whichever output format you choose. The output format could also be a web page, pandoc is that flexible; see the following link for more information (<http://rmarkdown.rstudio.com>). R Markdown scripts use the filename suffix `.Rmd`.

This is an R Markdown script that uses R bookdown extensions (<https://bookdown.org/yihui/bookdown>) (Xie 2016a). You will need R (<https://www.r-project.org>) and preferably the latest version of RStudio (<https://www.rstudio.com>) installed on your system. You will also need a working version of TeX (MS Windows users see <https://www.miktex.org>, macOS users see <https://www.tug.org/mactex>, Linux users see <http://www.tug.org/texlive>), and I recommend installing github (<https://github.com>) for version control if you are so inclined. Version control is central to good research practices allowing you to track changes, revert to previous versions of your Markdown script in case of mistakes, etc.

In addition, you will need to install the R `bookdown` package. Depending on your system, you may also need to install various packages or components (the packages `checkpoint`, `codetools`, `fpp`, and `tikzDevice` are required if you want to compile this R Markdown script, and the software component `pandoc` is needed if you wish to only install R). But a clean install of R, RStudio, TeX, and github plus the above ought to be sufficient for getting started.

You will note from the YAML metadata for this R Markdown script that you are able to render the output in five different formats (`pdf_document2`, `html_document2`, `gitbook`, `epub_book`, `word_document2`). You can change the default output format by pulling down the `knit` icon menu in RStudio.

⁹Note that the expression `http://pandoc.org/MANUAL.html#tables` is written in simple text in Markdown yet it gets rendered with the appropriate anchors in HTML, LaTeX/PDF, and gitbook output formats. That is, it is rendered as a *hyperlink* (click on the expression and it takes you directly to the web page in a browser), yet no additional instructions are required as this is automatically enabled by Markdown and pandoc.

¹⁰RStudio is a powerful integrated development environment, and RStudio version 1.0.0 or higher is required when using R Markdown and R bookdown.

One compelling reason for migrating to Markdown is that you need not modify the Markdown script in order to render output in different formats. If you insist on inserting non-supported pure LaTeX constructs then these might not get rendered into other formats. If your script is *pure R Markdown* it will render in HTML, PDF etc. It is recommended that you work with the HTML output format when writing your draft, and then switch to, e.g., PDF (LaTeX-based) output format for the final version. If you have been faithful to the R Markdown format you can switch effortlessly between these and other formats which is quite liberating as you are no longer shackled to a word processor, or committed to any particular output format.¹¹

1.10 Replicability is guaranteed

R Markdown combines knitr, pandoc and a range of packages that coexist in the background to produce publication quality replicable research. It is the knitr engine that handles the replicability of the R, Python, or C++ code, among others. But while knitr solves one aspect of the replicability problem, it does not solve them all. In particular, knitr cannot guarantee replicability should some routine (which resides in an R package) be modified at some point in the future. The R **checkpoint** package (Microsoft 2016) solves the problem by allowing you to *lock in* the package versions you use with a project. You can even specify that your checkpoint project depends on a specific version of R (use the option e.g., **R.version="3.3.2"** in your **checkpoint()** call). It's easy to use - all you need to do is to add a line like **checkpoint("2017-01-01")** in an R code chunk at the beginning of your R Markdown script. Enabling this results in the following:

- **checkpoint()** downloads all of the packages used by your project as they existed on, e.g., January 1 2017
- **checkpoint()** installs these packages in a folder specific to your project
- **checkpoint()** ensures that R uses these package versions when you, or anyone else, re-compiles your output document

The use of **checkpoint()** simplifies sharing your research with others, because you can be confident they will use the packages required to replicate your work. To incorporate **checkpoint()** in your R Markdown file, create an R code chunk in which you first load all R packages used in your Markdown script, followed by the **checkpoint** and **codetools** packages (you must install these first). The following R code chunk loads all required packages (for this script, **fpp** followed by **checkpoint** and **codetools**), then creates a hidden checkpoint directory named **.checkpoint** in the current working directory, then checkpoints for a specific date. This code should precede code chunks that make use of the required libraries (in this script, **fpp**). The first time you compile your R Markdown script this will install all libraries in the local checkpoint directory (this may take a minute or two). Make sure you set the working directory to the directory where your Markdown script resides (in RStudio, from the menu select **Session -> Set Working Directory -> To Source File Location**).

```
```${r,include=FALSE}
require(fpp)
require(checkpoint)
require(codetools)
dir.create(".checkpoint")
checkpoint("2017-01-01",R.version="3.3.2",checkpointLocation=getwd())
```
```

Now your R Markdown script is checkpointed, and furthermore if you execute the R command **knitr::purl("foo.Rmd")** in a separate R session where **foo.Rmd** is the name of your R Markdown script, this will strip off only the R code associated with your project, but with the checkpoint facility intact, and write just the R code to a file named **foo.R**. Now all you need to do is to circulate the file **foo.R** along with any supporting data files. For anyone running this R code on their computer, checkpoint will perform the same steps of installing the necessary packages, creating the checkpoint snapshot folder and producing the

¹¹If you wish the HTML output preview to appear in the viewer pane in RStudio while working (lower right corner of RStudio by default) instead of a separate viewer (the default in RStudio), from the RStudio menu select **Tools -> Global Options -> R Markdown -> Show Output Preview in Viewer Pane**.

Figure 1: Screen shot showing the R Markdown/bookdown code in RStudio (upper left), gitbook output (upper middle), MS Word output (upper right), HTML output (lower left), LaTeX/PDF output (lower middle), EPUB output (lower right).

same results, *guaranteed*. Note that there is some overhead associated with checkpointing, so while editing and knitting your script you might want to disable checkpointing by placing a `#` sign at the beginning of the last two lines of the above R code chunk (i.e., `#dir.create()` and `#checkpoint()` will comment out these R commands), then enable checkpointing for the production version only.

Note that the `checkpoint` package is not the only such approach. Another approach can be found in the R package `packrat` (Ushey et al. 2016). There is a philosophical difference between the two that is worth noting. In particular, `packrat` tries to lock in the package versions you currently have installed (which may differ from those that are current) and will try to restore these packages from locally saved copies of the packages instead of from CRAN (Comprehensive R Archive Network). `Packrat` places more of a burden on the user in that it requires a user to somehow get their `packrat` file and code to another user, which can be a rather cumbersome and tedious process. The `checkpoint` approach involves a checkpoint-server that knows the exact state of publicly available packages at a given point in time. Since everyone sees the same snapshot, everyone is on the same level playing field. So though `packrat` is more flexible, `checkpoint` presents a more streamlined solution that places minimal burden on both you and those with whom you circulate your R code.

1.11 With Markdown, a variety of output formats are available

The screen shot in Figure 1 demonstrates how you can select your output format once your R Markdown script has been written, and gain access to a variety of popular output formats with one mouse click. If you enlarge this figure, you can make out the `knit` icon on the upper left pane of RStudio. Simply clicking on this icon produces the output document (you pull down the `knit` menu to select from among a range of output formats). Producing the five documents in Figure 1 (HTML, MS Word, gitbook, EPUB and LaTeX/PDF) required one mouse click for each document.

1.12 Pros and cons of migrating to Markdown

Nothing comes for free. By adopting this framework for your research you may think you have lost control over some (minor) features you have grown accustomed to. However, the gains far outweigh such minor losses (and they are mostly illusory - you can exert the same fine tuned control over the output format you have become used to). Here is a partial list of some of the pros and cons that came to mind as I worked on this article.

- Pros:
 - You get the benefits of knitr, R, and LaTeX (e.g., work with only one script, get automatic generation of figures, tables, plots, references) so that you can create dynamic replicable documents
 - Floating tables and figures are automatically generated for PDF (LaTeX) output format, or HTML or gitbook etc.
 - Cross-referencing is portable between HTML, PDF (LaTeX) and other formats (so by simply changing `pdf_document` to `html_document` in the YAML metadata you can post a paper online in either format with zero modification)
 - The `tikzDevice` package is supported so that PDF (LaTeX) documents have the same fonts used in figures as those used to render the body of the paper (see the YAML metadata in this script for the entry `dev: tikz`; you need to install the `tikzDevice` package in order to use this feature)
 - You don't have to sweat the small stuff (e.g., notice you can use the ampersand in this script's title which could cause grief in LaTeX; you can use the keyboard quote which gets properly translated into LaTeX quotes when rendering in the LaTeX/PDF output format, etc.)
 - With the option `keep_tex: true` in the YAML metadata, the LaTeX source code is automatically generated when rendering to a PDF (LaTeX) output format, which can then be provided to a journal requiring manuscripts in LaTeX format (and figures are automatically generated and stored in the subdirectory `foo_files/figure-latex` where `foo.Rmd` is the name of your R Markdown script).
 - Caching of computationally intensive code chunks means they are only processed when the code itself changes (via knitr)
- Cons:
 - You lose access to tweaking some of the finer details in a LaTeX document (unless you really want to do some sleuthing)
 - Cross referencing of ordered (numbered) lists is not (currently) possible (HTML can't do this and Markdown is a subset of HTML)
 - MS Word output does not (currently) support section numbering

2 Authoring scripts in Markdown

We now look at some of the features and nuances that might be of interest when you begin authoring in Markdown.

2.1 How does the output format conversion take place?

Conversion takes place via pandoc (<http://pandoc.org>) which is a universal document converter. The conversion is hidden from the user and runs seamlessly in the background provided you adhere to some simple Markdown rules. If you install the latest version of RStudio, pandoc is automatically included and you can get to work immediately without worrying about the details. If you wish to use R exclusively (i.e., and not install RStudio), you will need to install pandoc separately (<http://pandoc.org/installing.html>).

2.2 How do I list multiple authors and their institutions?

Simply include a Markdown style list of names (i.e., start each line with a dash) in your YAML metadata and use footnotes for the institutions, email addresses etc. The footnote symbol will be a character in PDF (LaTeX), not a number. This is portable and is common practice for academic papers. Your YAML metadata should resemble the following:

```
author:
- Author One[University of Somewhere, a1@gmail.com]
- Author Two[University of Nowhere, a2@yahoo.com]
```

Note that the email tag is rendered as a clickable item in most output formats. Clicking on the email address will automatically pull up the reader's email client with your email address as the outgoing address (no input is required from you to render this useful feature). If you wanted the affiliation to instead appear below the author names, simply place them in the list in the order in which you want them to appear, e.g.,

```
author:
- Author One
- University of Somewhere, a1@gmail.com
- Author Two
- University of Nowhere, a2@yahoo.com
```

2.3 Spacing, Markdown, pandoc, numbered lists and special characters

Spacing is one of the most important ingredients for getting your Markdown script rendered universally by pandoc. pandoc gets its cues from the interaction of spacing (or lack thereof), line breaks and a few *special* characters that *only work* if Markdown syntax requirements are obeyed. For instance, a single asterisk followed by a space followed by a word, a space, and another single asterisk, i.e., `* word *`, renders as `* word` while the same sequence without spaces renders as italicized text, i.e., `*word*` renders as *word*. The same goes for the use of white space to separate certain elements such as new paragraphs, ditto for numbered lists. The following code demonstrates some basic rules for creating ordered lists in Markdown:

1. Numbered lists start with a number followed by a period and a space

(i) Sub items are indented four spaces and start with e.g., (i) (or (a) if you prefer)

If you want indented text to appear below a list item, create a blank line and then indent an *additional* four more spaces beyond the spacing of the previous item

4. Next item

This Markdown code gets rendered as

1. Numbered lists start with a number followed by a period and a space

(i) Sub items are indented four spaces and start with e.g., (i) (or (a) if you prefer)

If you want indented text to appear below a list item, create a blank line and then indent an *additional* four more spaces beyond the spacing of the previous item

2. Next item

Note the behaviour displayed by the above example. As long as the first item is an ordered list, all subsequent list items will be interpreted as *consecutive elements*. So your list could be 1. (i) (i) 1. (i) (i) (with appropriate line breaks and spacing) and it would display correctly as an ordered list.¹²

¹²As is the case with all Markdown syntax, these are just tags to interpret the start of a `ul` or `ol` HTML tag block.

If you adhere to these Markdown syntax requirements then your script will render properly in any output format, but if you get even *one space* wrong it may not. Fortunately, these simple spacing guidelines are easy to adapt to and quickly become second nature when authoring in Markdown.

Universal Markdown sections, subsections, subsubsections etc.

In Markdown you create a section header by beginning a line with one, two, three or more hash signs and following the hash signs with the name of the section. If you wish numbered sections, simply render your script with R bookdown which has extensions that allow for numbering and cross-referencing of sections. This will create section numbering with the format 1, 1.1, 1.1.1, etc. when rendered with bookdown. If you don't want to number a particular section (e.g., **References** is typically not a numbered section), simply add a minus sign in braces (i.e., {-}) at the end of the section header, per the above section header (note the lack of a number). If you want to turn off section numbering, simply add/modify `number_sections: true` to `number_sections: false` in the YAML metadata.

2.4 Universal R bookdown cross-referencing

In R bookdown there exist some idiosyncrasies when cross-referencing, but once you get used to them cross-referencing is straightforward (this is a useful feature not present in Markdown). You can cross-reference numbered sections via `\@ref(#id)` as in Section 2.4 once you append a hash sign and character string identifier to the end of the section header (note you don't use the hash sign when referencing). Or, you can use automatically generated tags for sections (not subsections however), which use the lowercase hyphenated words in the section header, e.g., `\@ref(authoring-scripts-in-markdown)` would render as Section 2.

Numbered equations are handled by adding, e.g., `(\#eq:eqname)` to a LaTeX equation.¹³ To reference numbered equations you use `\@ref(eq:eqname)`, without the hash sign, as in Equation (2).

$$\hat{\beta} = 1 \tag{1}$$

$$\hat{\alpha} = 2 \tag{2}$$

Cross-referencing figures is slightly different. You name the R chunk with a character string such as `figname` then refer to `fig:figname` via Figure `\@ref(fig:figname)`. The following R code chunk creates Figure 2.

```
```{r figname,fig.cap="Annual electricity sales for South Australia in GWh
from 1989 to 2008 (Source: Australian Energy Market Operator).",fig.align="center"}
require(fpp)
data(elecsales)
model <- auto.arima(elecsales,stepwise=FALSE,approximation=FALSE)
plot(forecast(model,h=6),ylab="Annual electricity sales (GWh)")
```
```

Cross-referencing tables is also slightly different. If you use the `kable()` function from the knitr package to format your tables, then you refer to the name of the code chunk, i.e., Table `\@ref(tab:tablename)`.

See the the `kable()` help file (<https://www.rdocumentation.org/packages/knitr/versions/1.15.1/topics/kable>) for `kable()` options.

¹³`eqname` is some arbitrary name you give to your equation for the purpose of cross-referencing; different equations will require different names.

Forecasts from ARIMA(1,1,0) with drift

Figure 2: Annual electricity sales for South Australia in GWh from 1989 to 2008 (Source: Australian Energy Market Operator).

Table 6: Testing

a	b	c
1	3	5
2	4	6

2.5 Universal R Markdown inline code chunks

As is the case for Sweave and knitr (there you use `\Sexpr{}`), you can use R markdown to refer to computational results in your narrative. For instance, consider the following code chunk:

```
set.seed(42)
X <- rnorm(100)
Y <- rnorm(100)
```

If you want to compute the correlation coefficient for the two variables in the above code chunk and report the value in a paragraph, don't cut-and-paste the numerical value. Instead, simply add an *inline* R Markdown code chunk, e.g., $\rho_{xy} = 0.0312798$ (you surround your R code with backticks and an initial `r` immediately after the opening backtick; R replaces the inline code with its results), and you can format it to control the number of digits using the R format command, as in $\rho_{xy} = 0.031$ (this was achieved using `$\rho_{xy} = \texttt{rformat(cor(X,Y),digits=2)}\$`). Should your data change, the value of the correlation coefficient will now always be correct (you have avoided a cut-and-paste operation).

2.6 Universal R bookdown Theorem, Proof, Lemma, Corollary, Proposition, Definition, and Example environments

You can create these LaTeX-like environments with three backticks followed by e.g., `theorem`, `thmfoo` in braces, then insert your theorem then end with another three backticks. The label `thmfoo` might be used to reference this theorem (you can use any character string you wish of course) as the following code chunk demonstrates.

```
```{theorem, thmfoo}
Here is my theorem.
```
```

Theorem 2.1. *Here is my theorem.*

You can reference this via `\@ref(thm:thmfoo)` as in Theorem 2.1. Hopefully with obvious modification you can create and reference other environments (e.g., `\@ref(lem:lemfoo)`, `\@ref(ex:exfoo)`, `\@ref(def:deffoo)` etc.).

Theorem 2.2. *Here is my other theorem.*

Lemma 2.1. *Here is my lemma.*

Example 2.1. Here is my example.

Definition 2.1. Here is my definition.

See <https://bookdown.org/yihui/bookdown/markdown-extensions-by-bookdown.html> for further details.

2.7 Universal Markdown verbatim text

Occasionally you want to
produce text verbatim.
Surround the text with three backticks
at the beginning and end of the material.

To do so inline, a single backtick surrounds the material.

2.8 Universal Markdown hidden comments

This can be accomplished via a simple Markup command where you enclose your comment in `<!--` and `-->`.

```
<!-- comment you want hidden -->
```

2.9 Universal Markdown superscripts and subscripts

To get Markdown subscripts and superscripts, enclose the text to be subscripted in tildes `~` (for superscripts use the carat `^`). For instance,

`H~2~O` is a liquid. `2^10^` is 1024.

produces H₂O is a liquid. 2¹⁰ is 1024.

2.10 Universal Markdown italics, bold face, and block quotes

You render text in *italics* by enclosing a word/phrase in asterisks `*`, **boldface** by enclosing in pairs of asterisks `**`, and block quotes by starting a line with the greater than symbol (`>`).

Block Quotes

2.11 Universal R Markdown math expressions

In Markdown you can enter text intended to be formatted as a math equation following standard TeX/LaTeX guidelines (e.g., `$$\hat{\beta}=(X'X)^{-1}X'y$$` gets rendered as $\hat{\beta} = (X'X)^{-1}X'y$). However, one common refrain voiced by users learning to type math in LaTeX is that you can't see the output from your math commands as they are being typed, which leads to mistakes and can be quite frustrating. In RStudio there is

an icon **Addins** at the top of the screen from which you can select **Input LaTeX Math**. This will bring up a tool where you can write your LaTeX math commands in the top line and it will display automatically as you type, so `\alpha` will immediately display α . When you click on the **Done** icon, the code will be inserted where your cursor lies in your Markdown script (wrapped in dollar signs naturally). Also, if you let your cursor hover over math typeset in LaTeX in your R Markdown script, a floating image of the typeset material appears, which is extremely helpful (this feature is restricted to authoring in RStudio).

2.12 Spell checking in RStudio

You can spell check a Markdown script in RStudio from the menu selection **Edit -> Check Spelling**.

2.13 Universal R Markdown BibTeX references

References are supported via BibTeX (<http://www.bibtex.org>). You create a standard BibTeX file (referenced in the YAML metadata via `bibliography: mybibfile.bib`). In R Markdown you refer to an entry with the `@` symbol followed immediately by the BibTeX entry name, e.g., `@Adom201743`. Note that enclosing the entry in brackets renders the citation in parentheses (e.g., `[@Adom201743]`).

Here are a few sample references: (Adom 2017), Gupta and Wohar (2017), Apergis and Apergis (2017) and Pircalabu et al. (2017). Note that you don't want numbering on the references section header, hence the addition of a minus sign inside braces at the end of the **Reference** header appearing at the end of this Markdown script. Note that the references will appear as the very last section when rendering R Markdown script.

2.14 Knitting (producing) your R Markdown output format document

To render an output document in RStudio you can click on the **knit** icon (top left pane of RStudio by default) and it will run your R code, generate tables and figures automatically, run **bibtex** and generate the desired output format all with a single mouse click. Or you can use the keystroke (`CMD-shift-k` for macOS, `<ctrl>-shift-k` for MS Windows) and do the same. This streamlines a pretty sophisticated process so that anyone can, with minimal effort, focus on the research at hand and generate publication-quality, professional, and replicable documents in an environment that is without peer and virtually OS independent.

To render an output document in R (not RStudio) using the R Markdown code `foo.Rmd`, simply type the following in the R console (you may need to first install pandoc (see <http://pandoc.org/installing.html>)).

```
rmarkdown::render('foo.Rmd')
```

Or, from the command line enter

```
Rscript -e "rmarkdown::render('foo.Rmd')"
```

I would encourage you to create a project for your Markdown script (in RStudio menu select **File -> New Project**), and I would encourage you to also add version control to your project (in RStudio menu select **Tools -> Version Control**).

Summary

This article showcases a set of free and open source tools¹⁴ that can be enlisted in the pursuit of replicable research. By putting this collection of tools through its paces, you can see how publication quality replicable research can be conducted in an efficient and remarkably simple manner. Extensions to Markdown present in

¹⁴We considered pandoc, Markdown, bookdown, checkpoint, TeX/LaTeX, BibTeX, knitr, R, and RStudio, among others.

R Markdown and bookdown, i.e., cross-referencing and a variety of LaTeX-like environments (e.g., Theorem, Example etc.), add missing features that are essential for authoring research reports. By exploiting recent extensions to the Markdown language, you can weave your narrative and your statistical analysis together in one Markdown script and render it in a variety of popular output formats. Furthermore, the entire process can be accomplished with one mouse click on the `knit` icon in RStudio or one simple R command (i.e., `render("foo.Rmd")`).

The underlying script for this article is authored using Markdown. All that is required of the user is that they install R and TeX along with a handful of R packages (e.g., `bookdown`, `codetools`, `checkpoint` etc.). By following a set of basic rules for authoring Markdown scripts, you can generate a *universal* Markdown script that can be rendered in a variety of output formats using pandoc, a universal document converter. Using the R `checkpoint` package *guarantees* future replication by locking in versions of software used for the research project. As the name indicates, R bookdown (a Markdown extension) is for writing books, and setting up a book project involves separate files for each chapter along with a few YAML metadata configuration files. But as this article demonstrates, you don't need to be writing a book to benefit from this welcome extension to the Markdown language. If you want to use bookdown for a more involved project (say, a Ph.D. dissertation) you might wish to delve deeper; see <https://bookdown.org/yihui/bookdown> for further information.

References

- Adom, Philip Kofi. 2017. "The Long-Run Price Sensitivity Dynamics of Industrial and Residential Electricity Demand: The Impact of Deregulating Electricity Prices." *Energy Economics* 62: 43–60.
- Allaire, JJ, Jeffrey Horner, Vicent Marti, and Natacha Porte. 2015. *Markdown: 'Markdown' Rendering for R*. <https://github.com/rstudio/markdown>.
- Apergis, Emmanuel, and Nicholas Apergis. 2017. "The Role of Rare Earth Prices in Renewable Energy Consumption: The Actual Driver for a Renewable Energy World." *Energy Economics* 62: 33–42.
- Gupta, Rangan, and Mark Wohar. 2017. "Forecasting Oil and Stock Returns with a Qual {Var} Using over 150 Years Off Data." *Energy Economics* 62: 181–86.
- Meredith, E., and J.S. Racine. 2009. "Towards Reproducible Econometric Research: The Sweave Framework." *Journal of Applied Econometrics* 24: 366–74.
- Microsoft. 2016. *Checkpoint: Install Packages from Snapshots on the Checkpoint Server for Reproducibility*. <https://CRAN.R-project.org/package=checkpoint>.
- Pircalabu, A., T. Hvolby, J. Jung, and E. Høg. 2017. "Joint Price and Volumetric Risk in Wind Power Trading: A Copula Approach." *Energy Economics* 62: 139–54.
- R Core Team. 2016. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing. <https://www.R-project.org/>.
- Racine, J. S., and R. Hyndman. 2002. "Using R to Teach Econometrics." *Journal of Applied Econometrics* 17 (2): 175–89.
- RStudio Team. 2015. *RStudio: Integrated Development Environment for R*. Boston, MA: RStudio, Inc. <http://www.rstudio.com/>.
- Ushey, Kevin, Jonathan McPherson, Joe Cheng, Aron Atkins, and JJ Allaire. 2016. *Packrat: A Dependency Management System for Projects and Their R Package Dependencies*. <https://CRAN.R-project.org/package=packrat>.
- Xie, Yihui. 2016a. *Bookdown: Authoring Books and Technical Documents with R Markdown*. <https://CRAN.R-project.org/package=bookdown>.
- . 2016b. *Knitr: A General-Purpose Package for Dynamic Report Generation in R*. <https://CRAN.R-project.org/package=knitr>.