stationery Package Overview: Guides, Slides, and Reports

Paul E. Johnson, Director, CRMDA

Aug. 9, 2018

Abstract

The stationery package offers working examples for preparation of "reproducible research" documents. It demonstrates how one can create and customize the style of both markdown and LaTeX documents. These formats utilize R's "code chunk" processing technology, so that code that creates figures and tables can be embedded into the document itself. The package includes 8 document templates. The vignettes that accompany the package provide elementary explanations of the formats and how they differ in practice.

1 Introduction

"Reproducible research documents?" A university administrator recently exclaimed, "I have no idea what that means!" Many members of the faculty seem to have have the same response. The `stationery` package for R may help would-be users to get started.

The *big idea* is to replace the traditional "cut and paste" approach to creating research reports with a program-driven documentation system that blends calculations with report preparation. Researchers who have been using word processors are aware of the danger that computer code, output tables, and graphs can fall out of alignment over time. Proponents of the *reproducible research documents* movement (Leisch, 2002; Xie, 2016, 2015; Stodden et al., 2014) ask us to change our workflow in a fundamental way, so that analysis, code, and output work together more seamlessly.

In my experience, the benefits of this change are most apparent in class notes and slides. A chronic problem for teachers is a misalignment of computer code and output in course materials. It is easy to forget to paste in an updated graph or modify a table. With a reproducible document, there less danger that the computer code being discussed will not match the output that is presented because they are all part of one piece.

For most word-processor-using researchers, the transition is jarring. First, the software for document authoring is being replaced. MS Word is, simply put, insufficient. Second, the format in which documents are stored and revised is replaced. Third, the methods of statistical analysis that used to seem separate from authoring (SAS, SPSS, etc) are now supposed to be integrated with it.

In the *new way* of doing things, we avoid typing tables or pasting in graphs. The analysis software will prepare article-ready tables and graphs that can be put to use with no revision. In the ideal case, an entire article, lecture, or book can be generated in one single execution that conducts

analysis, saves graphics, and assembles them together in the output document. This is in line with the "literate programming" movement started by computer scientist Donald Knuth (1984a; 1984b), who also created the TeX document preparation system. The statistical program R (R Core Team, 2017) built its documentation framework on the literate programming philosophy, integrating Knuth's concepts of "weaving" and "tangling" to produce documents and extract code files (Leisch, 2002).

Once we understand that *big idea*, then comes a series of rude shocks. First, *there are several competing formats for doing this kind of work*. Not only do they want to change the way they work, but we have to decipher several alternatives and select one. The `stationery` package tries to help with that, by offering working examples for several different file types. Our original work was done with LaTeX in mind. We offer documents in both "raw" LaTeX and LyX (`http://www.lyx.org`), a graphical user interface for LaTeX. We also explore opportunities in markdown format. Proponents of markdown are attempting to develop a simpler, more human-readable document format. The leading voice has been John Gruber, whose Daring Fireball website (https://daringfireball.net/projects/markdown) offered the first working set of guidelines for markdown documents. The markdown movement was advanced significantly by John MacFarlane's ambitious software program `pandoc` (2018), a format translator.

The other shock coming for new users is that the format of the output is much more difficult to customize than we expect in word processor software. The `stationery` package is intended to ease the frustration by providing templates that are visually appealing. These templates offer a starting point for customization of new styles.

Before proceeding to the discussion of the package, some key terms should be well understood.

front end: the format in which a document is prepared. Markdown files are saved with a suffix `.Rmd`, while the LaTeX files that include code chunks are saved as `.Rnw`.

back end: the delivered format.[1] The back ends considered here are Adobe portable document format (PDF) and hypertext markup language (HTML).

code chunk: A segment of R code that is embedded in a LaTeX or markdown document.

style: the description of the back end's format. Our examples compare pedagogical guides, formal reports, and slides as styles.

skeleton: a minimum working example that an author can revise into an essay.

weave: to replace code chunks with output, preparing for compilation of a report (synonym of "knit").

tangle: to extract code chunks into a free standing computer program (synonym of "purl").

2 What Do You Get with stationery?

The `stationery` package includes examples for eight types of documents (see Table 1). The document types result from "mixing and matching" of various front ends, back ends, and document

---

[1]In this context, it is a bit vague to say *front end* because each document is converted through several formats in the compilation process. Any intermediate format that precedes another might be edited directly and treated as a front end by an author.

Table 1: Document Types in Stationery

	Formats	Frontend	Backend	Style	Code Chunk Engine
1	rmd2html-guide	Markdown	HTML	Guide	knitr
2	rmd2pdf-guide	Markdown	PDF	Guide	knitr
3	rmd2pdf-report	Markdown	PDF	Report	knitr
4	rnw2pdf-guide-knit	Markdown	PDF	Guide	knitr
5	rnw2pdf-guide-sweave	LaTeX/LyX	PDF	Guide	Sweave
6	rnw2pdf-report-knit	LaTeX/LyX	PDF	Report	knitr
7	rnw2pdf-report-sweave	LaTeX/LyX	PDF	Report	Sweave
8	rnw2pdf-slides-sweave	LaTeX/LyX	PDF	Slides	Sweave

styles. We hope that authors who are accustomed to writing on stationery, with a pleasant header and footer, will find satisfactory results. Each document style includes theme files that can be revised to suit the author's needs. While it is not easy to create these style templates, we believe it will be easier for others succeed if they start from the examples we offer.

The package also provides functions to initialize skeleton documents and convert them to a desired back end.

Finally, there are vignettes, one of which is the present document, which serves as a package overview. In addition, we have vignettes named

1. Code Chunks: compares Sweave and knitr style code chunks.

2. R markdown: explores key elements in the newest front end document framework.

3. HTML Special Features: a tour of the promise and peril of using HTML as a back end for markdown documents.

`stationery` **package document templates**

In Table 1, readers will note that the document formats have three-part names like "`rnw2pdf-guide-sweave`". The first part of the name has the format *frontend2backend.* The "rnw" prefix is for LaTeX files (suffixes to be explained below) and the "pdf" is the output type. The middle part of the format label is either "guide", "report" or "slides". For document types that can be used with either `Sweave` or `knitr` code chunks, it is necessary to add a third part in the name (R markdown documents cannot use the `Sweave` engine).

The process of converting a reproducible document from front end to back end is referred to as compiling. It is always a multi-step process which we illustrate in Figure 1. Because the process can fail at any step, users are well advised to keep this fact in mind.

Between the front end and the back end we have intermediate files that are created (and just as quickly erased). One of the aims in software development is to make the transition process into an automatic, seamless process. The software design challenge is to retain some flexibility and respect for user input. It seems, almost invariably, that the software designer's idea of making this into a seamless process is to impose limitations on the author's ability to format and customize the document.

Figure 1: Compiling a Reproducible Research Document
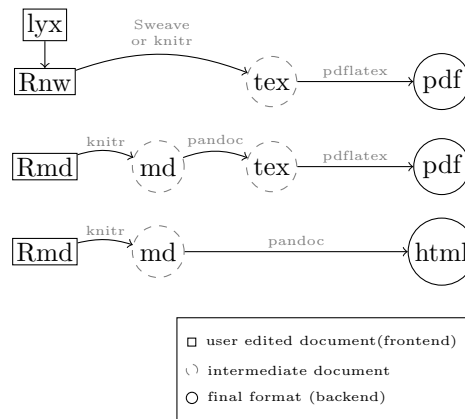
LaTeX authors are used to editing .tex files, but .tex is usually an intermediate format in this work. When code chunks are included, a document is usually given the suffix .Rnw, which is then converted into a .tex file by the chunk converter. (If one is editing a `LyX` file, the converter will export to .Rnw as a first step.) The intermediate LaTeX must be compiled (usually by `pdflatex`, `texi2pdf`, or similar programs). An R markdown document, which has the suffix .Rmd, can be prepared with various back ends in mind (we concentrate on PDF and HTML back ends, but `pandoc` can be used to convert markdown other formats, like MS Word.)

There is a meaningful difference between the programs that are used for compiling documents written in LaTeX and pandoc. The LaTeX programs do not provide command-line options to change the look-and-feel of results. In contrast, the `pandoc` program offers command line options that can over-ride many settings within the document. At the final stage of processing an R markdown document, for example, we usually arrive at a call to `pandoc` that has a somewhat elaborate format.

```
$ /usr/bin/pandoc +RTS -K512m -RTS crmda.utf8.md --to latex --from
markdown+autolink_bare_uris+ascii_identifiers+tex_math_single_backslash
--table-of-contents --toc-depth 2
--template theme/report-boilerplate.tex
--highlight-style haddock --latex-engine pdflatex --listings
```

If one has created tens or 100s of documents, the ability to re-style them in a script, rather than revising each of the individual documents, is an advantage.

Incorporating computer code

Whether one is editing in an LaTeX or a markdown file, there will be code chunks. In markdown documents, code chunks are bracketed by tick marks:

```
```{r chunkname , options}
code here
```
```

In LaTeX documents, code chunks are represented differently:

```
<<chunkname, options>>=
code here
@
```

One of the pivotal stages in document compilation is the conversion of code chunks into formats suitable for inclusion in the document. At this point, we arrive at a somewhat unhappy situation because there are competing programs and terminology. Knuth referred to the chunk conversion process as "weaving". The base R framework refers to chunk conversion as "Sweaving" (because S was the precursor to R; Leisch (2002)). The newer knitr(Xie, 2018) package for R refers to the chunk-conversion process as "knitting", although it is doing the same work as Sweaving. Another problem in terminology is that Knuth used the unlikely term "tangle" to refer to extraction of code chunks to create a "free standing" program (a program document separate from the commentary). Base R refers to that by the name "Stangle" while knitr package calls it "purling".

In LaTeX documents, computer code can be displayed in various ways. The traditional method was the LaTeX environment Verbatim, but a more flexible alternative is the Listings package. Our PDF documents use the Listings class. Our style documents include example settings that authors can revise if they want to adjust the shading, line numbering, or other characteristics.

3 Quick Start

3.1 Create a starter "skeleton" document

The stationery package provides initWriteup, a function to create simple ready-to-compile examples. Here we will illustrate creation of an R markdown guide that will have a HTML back end. Start R in a folder where you would like to create a write-up and run

```
library(stationery)
initWriteup(type = "rmd2pdf-report")
```

This creates a folder named rmd2pdf-report in which one should find

1. a skeleton template, skeleton.Rmd,

2. an instructional guide, instructions.Rmd,

3. a compiler script, rmd2pdf.sh,

4. a subdirectory theme, in which a template and some other configuration files are to be copied.

The return from initWriteup will indicate the full path to the new directory:

```
[1] "/home/pauljohn/wherever_you_say/rmd2pdf-report"
```

Most users will rename skeleton.Rmd to something more suitable. Here we discuss an example named crmda.Rmd.

There are other ways to create a document. We have formatted the folder structure of the package in a way that is consistent with the template format required by RStudio (http://www.rstudio.com), as specified by the package rmarkdown; Allaire et al., 2018).

For our markdown-based formats (sadly, not for the LaTeX based formats), the RStudio graphical interface will work as well. A user can open the File menu, choose New File -> R Markdown -> From Template. The formats "rmd2html guide", "rmd2pdf report" and "rmd2pdf guide" should be available. The `rmarkdown` function `draft()` performs exactly the same purpose. One could run

```
library(rmarkdown)
draft("crmda.Rmd", template = "rmd2pdf-report", package =
    "stationery", create_dir = FALSE)
```

3.2   How to Compile a Document

While editing a document, authors are well advised to heed the advice:

Compile early, compile often!

Users should try to compile our document before changing it. While revising the document, it is wise to compile often and notice errors as soon as they are committed.

The file can be compiled in several ways.

1. Use shell commands

   For the sake of completeness, we begin with the most basic method: run commands in a terminal. For a LaTeX file, such as `crmda.Rnw`, this is straightforward. First, convert the code chunks to R input:
   ```
   $ R CMD Sweave crmda.Rnw
   ```

   That creates `crmda.tex`, which is then converted to PDF output:
   ```
   $ texi2pdf crmda.tex
   ```

   The command line options to compile a file with knitr code chunks is a bit more elaborate. I generally use the R functions or shell scripts described next.

2. Open an R session and use the functions `rmd2pdf()`, `rmd2html()`, and `rnw2pdf()` in the `stationery` package.

   These functions will compile files that are saved in the `.lyx`, `.Rnw`, and `.Rmd` formats. For example,
   ```
   library(stationery)
   rmd2pdf("crmda.Rmd")
   ```

   This orchestrates a two part process that involves functions in the `rmarkdown` and `knitr` packages. The document header is scanned and then a suitable document format object is created. The heavy lifting is done by the `pdf_document` or `html_document` functions in the `rmarkdown` package. After that, intermediate documents must be rendered to the final result. The benefit of using the `stationery` functions is that they will review the in-document format settings, but can override them with function parameters. For example, one can control the creation of a table of contents by including `toc:   true` in the `.Rmd` document, or by running

`pdf_document` with the parameter `toc = TRUE`. Any of the document parameters can be selectively replaced.

As a use case, suppose we write three R markdown documents and we forget to specify the depth of the table of contents. Rather than editing each individual document to insert `toc_depth:1`, we might instead specify the depth as an R function argument:

```
rmd2pdf(c("crmda1.Rmd", "crmda2.Rmd", "crmda3.Rmd"), toc =
    TRUE, toc_depth = 1)
```

Similarly, a template can be substituted by employing the template argument, such as `template = "theme/report-newboilerplate.tex"`.

There are differences in format between the values in the markdown document preamble and the R function call. R uses `TRUE`, `FALSE` and `NULL` where markdown uses lower case `true`, `false`, and `null`.

3. Run the shell script provided with the template.

   The document skeleton is provided with a compiler script. It can do the same work as **rmd2pdf** inside R. The same output can be generated by running a command line script that does the same work.

```
$ ./rmd2pdf.sh crmda.Rmd
```

The compiler script is designed to accept the same arguments as the function rmd2pdf, but in the command line it is a little tricky to specify the options because we have to protect quotation marks from interpretation. For example, here we enclose double quotes in single quotes:

```
$ ./rmd2pdf.sh --toc=TRUE --toc_depth=1 --
    template='"theme/report-newboilerplate.tex"' crmda.Rmd
```

4. If editing in `RStudio`, there is a button [Knit ▾] that has a small triangular widget. This should be used with *extreme caution* because it can alter the document preamble.

   In the preamble of the markdown document, there will be an output stanza. This if from our skeleton for **rmd2pdf** guide documents.

```
output:
  pdf_document:
    citation_package: natbib
    fig_caption: yes
    latex_engine: pdflatex
    highlight: haddock
    pandoc_args: [
        --listings
    ]
    template:  theme/guide-boilerplate.tex
```

The `RStudio` triangular widget offers a selection of document back ends. Be careful to choose "Knit to PDF". If the user makes a mistake and selects, say, "Knit to HTML", then `RStudio` will replace the output stanza in the document. It will insert its best guesses about settings

for `html_document`. When `RStudio` inserts its best guess, it sometimes corrupts the format of the other output types and the document will fail to compile.

In our rmd2html documents, the output format is controlled by a function in `stationery`.

```
output:
  stationery::crmda_html_document:
    toc: true
    toc_depth: 2
    highlight: haddock
    theme: default
    citation_package: natbib
    css: theme/kutils.css
    template: theme/guide-boilerplate.html
```

New versions of `RStudio` will notice that the intended format is controlled by `crmda_html_document` and offer a choice "Knit to crmda_html_document".

4 Styling for Documents

4.1 Reports versus Guides

The `stationery` package document template are intended to have consistent "look and feel" across document fron ends. A guide document may be produced with an HTML back end, or in PDF, with either Sweave or knitr chunk processing engines. The same is true for report documents produced by Sweave and knitr. To the extent possible, the color schemes and arrangement of header and footer information should be similar (if not identical).

The stationery package includes two document styles, dubbed "guide" and "report". The difference between these two is not hard-and-fast. A **guide** is usually prepared as a teaching document. It may end up in a Webpage, where more color rather than less is expected. It is usually a less formal document. The final presentation is likely to include computer code and output excerpts. The guide format is are usually intended for education and training, but it is possible to use the guide style for other types of documents. The header is a three column structure with organizational and/or departmental logos on either side (see Figure 3 for an example).

A **report** is a more formal document. Its first page includes a header and a footer with organizational address information (see Figure 2). The header and footer appear only on the first page. It is more is suitable for preparation of a report to clients or a draft of a journal article. A report typically has less (maybe no) code and almost never will it include "raw output" from a computer program. A report includes tables and figures that are in a (nearly) publishable format. This document is prepared with the report template, but we do have some "raw" code examples.

The slide format is based on LaTeX Beamer (??) slide format using a customized theme that features the colors of our institution. Our slides use the Sweave engine. We have experimented with many slide producing strategies using markdown code and none of them have been dependable, so we set that aside for the moment.

Figure 2: Latex Output (Report Document)

(a) Header

A Title for Skeleton Template: rnw2pdf-report-sweave

CENTER FOR
RESEARCH METHODS
& DATA ANALYSIS
College of Liberal Arts
& Sciences

First Author, CRMDA
Second Author, CRMDA

Aug. 8, 2018

(b) Footer

Address line 1   Web: https://crmda.ku.edu
Address line 2   Email: you@where.edu
City State Zipcode   Phone: 123-345-5678

4.2 Distinctive Headers and Footers

In the `stationery` package, we provide enhanced headers (and footers where appropriate) that incorporate organizational graphics and address information. Document templates are provided for that purpose.

The markdown format introduced some interesting concepts to streamline document creation, so we begin with a markdown document's preamble. As exemplified in Listing 1, the markdown document begins with a section, written in YAML format, that species the title. A standard set of parameters is specified by `pandoc`, but additional parameters can be provided by our template and also by the R functions that compile the document. In this case, the parameters like `affiliation`, `email`, `l1` through `r3`, and the logo image files, are specified by our template. Many additional details about working with markdown are spelled out in the `stationery` vignette "R markdown Basics." The header and footer created from the document are presented in Figure 3.

Listing 1: Beginning of a markdown preamble

```
---
title: "A Title for Template"
subtitle: "rmd2pdf-guide"
guidenum: 00
guideurl: https://crmda.ku.edu/guides
keywords: R markdown, R, documents
author:
- name: First Author
  affiliation: CRMDA
  email: first@ku.edu
- name:  Second Author
  affiliation: CRMDA
  email: second@ku.edu
addr:
    l1: address row 1
    l2: address row 2
    l3: City State Zipcode
    r1: "Web: http://crmda.ku.edu"
    r2: "Email: author@ku.edu"
    r3: "Phone: 123-345-5678"
logoleft: theme/logoleft.pdf
logoright: theme/logo-vert.pdf
```

Figure 3: Markdown Output (Guide Document)

(a) Header

(b) Footer

address row 1   Web: http://crmda.ku.edu
address row 2   Email: author@ku.edu
City State Zipcode   Phone: 123-345-5678

One of the benefits of the markdown preamble is that one can specify a authors in a flexible way, providing one or more names that are gracefully handled during the rendering process.

In LaTeX, the tools to specify title and author information are not as flexible. The problem is solved by some LaTeX functions in our templates. These use recent innovations in the LaTeX programming interface. (Packages consistent with TexLive 2016 or newer will be required). The top portion of the LaTeX document, where we expect the author to include the metadata, is illustrated in Listing 2. The input format is similar to the markdown format, but there are slight differences due to inherent differences in technology. However, the output is the same (see Figure 4). The LaTeX format allows us to keep the footer information–the address–in a separate file, so it need not be typed into each individual document (markdown does not allow that).

Listing 2: LaTeX header information
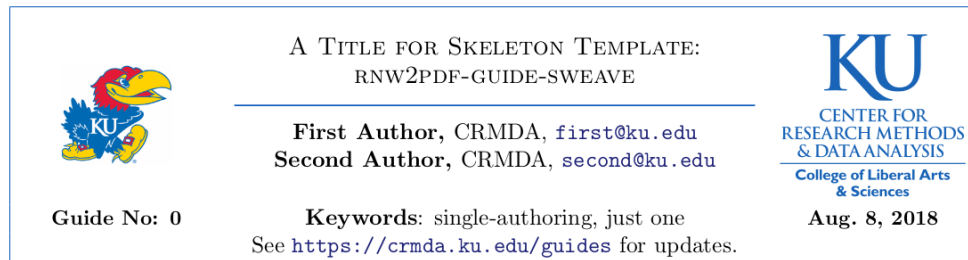
```
\guidesetup{
  author={
    lastname=Author,
    firstname=First,
    affiliation=CRMDA,
    email=first@ku.edu},
  author={
    lastname=Author,
    firstname=Second,
    affiliation=CRMDA,
    email=second@ku.edu},
  url={https://crmda.ku.edu/guides},
  keywords={single-authoring, just one},
  title={A Title for Skeleton Template: rnw2pdf-guide-sweave},
  leftlogo={theme/logoleft.pdf},
  rightlogo={theme/logo-vert.pdf},
  number=00,
}
\guidehdr
%footer information in theme/addressFooter.tex
\footersetup{
  leftside={
    lone={Address line 1},
    ltwo={Address line 2},
    lthree={City State Zipcode}},
  rightside={
    rone=Web: \url{https://crmda.ku.edu},
    rtwo=Email: \href{mailto:you@where.edu}{\url{you@where.edu}},
    rthree=Phone: 123-345-5678}
}
```

4.3 Customization

Hopefully, authors do not have too much trouble filling in their names and titles. There are not too many "gotchas", but the use of illegal symbols for the intended back end may cause trouble. For example, LaTeX documents assign special meaning to symbols like "%", "$", and "_" and these

Figure 4: LaTeX Output (Guide Document)

(a) Header

A TITLE FOR SKELETON TEMPLATE:
RNW2PDF-GUIDE-SWEAVE

First Author, CRMDA, first@ku.edu
Second Author, CRMDA, second@ku.edu

Guide No: 0   **Keywords**: single-authoring, just one   **Aug. 8, 2018**
See https://crmda.ku.edu/guides for updates.

(b) Footer

Address line 1	Web: https://crmda.ku.edu
Address line 2	Email: you@where.edu
City State Zipcode	Phone: 123-345-5678

will need to be protected by a backslash. In documents intended for HTML, titles or author names that include reserved symbols such as "<" ,">", or "&" are likely to cause trouble and should be avoided.

As the example code illustrates, we assume the logo information is saved in files with names like "`theme/logoleft.pdf`". Before first compiling our document, the author can copy image files into the theme directory to replace our defaults. If the author does not do so, the document compiler will retrieve the "plain white" image defaults and place them in the theme directory. Within the document, look for a code chunk like this which retrieves logo files.

```
```{r themecopy, include = FALSE}
library(stationery)
logos <- c(logoleft = "logoleft.pdf", logoright = "logo-vert.pdf")
getFiles(logos, pkg = "stationery")
```
```

If authors expect to create many documents, it is possible to automate this process. One can create an R package to hold the logo information (we can supply a working example of a package named "`crmda`"). To pull logo information from the user's package, one will replace our `pkg = "stationery"` with `pkg = "your_pkg_name"`.

4.4 HTML themes and the file size problem

The output size of HTML files may be quite large, even if the document itself has almost no content. This problem arises because `rmarkdown` uses a theme set based on the bootstrap library (). If the markdown preamble does not specify a theme, or it specifies any theme except `null`, then a large amount of javascript and cascading style sheet data from bootstrap is inserted into the HTML header. This can take up to 700KB of storage. All of the other elements in the document, including its graphs and features, will take additional space. An R package that includes three HTML vignettes will exceed the CRAN limit on the size of packages.

Rather than editing the document over-and-over to see the effects of themes, we suggest instead either using the The `stationery` package document template are intended to have consistent "look and feel" across formats. A guide document may be produced with an HTML back end, or in PDF, with either Sweave or knitr chunk processing engines. The same is true for report documents produced by Sweave and knitr. function or the x script. To prevent the use of a Bootstrap library theme, the `rmd2html` function can be run like so.

```
rmd2html("crmda.Rmd", theme=NULL)
```

On the command line, the parameter value NULL should not be quoted:

```
$ ./rmd2html.sh --theme=NULL crmda.Rmd
```

The nearly empty `skeleton.Rmd` provided with this package has a compiled size will be around 700KB. Preventing the insertion of the Bootstrap-based theme will reduce the HTML output file size to 62KB. (Note the author can insert "theme: null" in the markdown to prevent the use of a bootstrap theme (note that `null` is neither capitalized nor quoted). Of course, the disadvantage of removing the theme is that the benefits of the theme are lost. In the `stationery` package, there is a vignette "HTML Special Features," that explores these issues.

It is worth mentioning that there are many bootstrap themes (they are listed in the help page for `rmarkdown::html_document`). One can explore the impact of these themes on the final document by running, for example,

```
rmd2html("crmda.Rmd", theme = "spacelab")
```

or, from the command line,

```
$ ./rmd2html.sh --theme='"spacelab"' crmda.Rmd
```

4.5   Troubleshooting

Documents often fail to compile. There are many failure points and one might need to inspect the intermediate files and output at several stages. When there is trouble, recompile with parameter values `clean = FALSE`, `quiet = FALSE`, and either `keep_md = TRUE` (for HTML output) or `keep_tex = TRUE` (for PDF). By inspecting the intermediate files, editing them, and running the compiler commands again, one can usually find out what's wrong. During this process, it is beneficial to remember that the rendering process has separate stages, and each one can be run in isolation.

When `quiet = FALSE`, one of the especially important parts of the verbose output is the full command that is sent to `pandoc`. For example, compiling our minimal skeleton `crmda.Rmd` yields this intimidating list of command line options

```
/usr/bin/pandoc +RTS -K512m -RTS crmda.utf8.md --to html4 --from
    markdown+autolink_bare_uris+ascii_identifiers+tex_math_single_backslash --output
    crmda.html --smart --email-obfuscation none --self-contained --standalone --
    section-divs --table-of-contents --toc-depth 2 --template
    theme/guide-boilerplate.html --highlight-style haddock --css theme/kutils.css --
    variable 'theme:bootstrap' --include-in-header /tmp/RtmpU3qSFQ/R
    markdown-str33e24223b3a.html --mathjax --variable
    'mathjax-url:https://mathjax.RStudio.com/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML'
    --filter /usr/bin/pandoc-citeproc
```

This converts an intermediate markdown document `crmda.utf8.md` into `crmda.html`. The help page for pandoc lists the document parameters than can be specified in the command line (such as `--template` or `--table-of-contents`.

5 Choosing among formats

We provide 8 document formats because there are several acceptable methods. Each one has strengths and weaknesses. At the current time, there is a considerable amount of enthusiasm about markdown document preparation system. The markdown movement is the "bleeding edge," literally, as the style for document markup and the development of software to convert documents into the final format is currently underway. The core program that handles markdown documents, `pandoc`, is undergoing rapid development.[2] On the other hand, the LaTeX-based approaches are time-tested and the compilers for them have existed for years. There are fewer surprises (bugs), but there are more details for authors to learn.

A long run goal in the development of markdown is to create a single document that can be compiled into various formats. At the current time, that is an *aspiration*, not a reality. Authors who prepare documents intended for one format will generally use features that are fundamentally incompatible with the other formats. As a result, the choice of the back end must be made first, and after that one can choose among features in the front end that are compatible with the desired result. Where you want to end up determines where you start.

To sort through these considerations, the right place to start is the actually the desired end of the writing process. After we figure out the end, we can concentrate on the beginning.

5.1 Which back end?

Should I end up with HTML or PDF? The answer depends on the intended audience/client. If a "paper" must be submitted, choose PDF. If the document needs numbered equations, cross references, and "floating" tables and figures, choose PDF. If the document is intended for a Webpage, then choose HTML (although we prefer to PDF document via the Web in many cases).

HTML documents have hidden benefits and hidden costs. There is no free lunch. One can have features, but at the cost of large file sizes and (sometimes) unpredictable user experiences. Web browsers differ in their implementation of guidelines, javascript, and cascading style sheets. If one intends to convey a document in a replicable way to a wide audience using various kinds of computers, then PDF is the recommended format. The equations and figures are "in" the document and the author can be highly confident that they will have the same appearance across systems.

There are other limitations in HTML. There is no built-in method for display of mathematical equations. There have been efforts to ameliorate that problem over the years, the most recent of which is called MathJax (??). MathJax is a Web program that can be called when a user opens an HTML document. MathJax displays markup in a way that is similar to LaTeX output. However, in order for this to work, the conditions must be "just right". First, if the remote server that provides MathJax is unavailable, then the math in the document will not be rendered. Second,

---

[2]Frequent changes in `pandoc` caused the RStudio program distribution to include a snapshot of `pandoc` so that their example documents can be compiled in a predictable way. We are warned that our templates for `pandoc` are likely to be made unworkable by revisions in the future and revisions will be necessary.

some special care is required to allow MathJax equations in HTML documents that are based on markdown templates. The stationery package includes a special document format function, `crmda_html_document`, that circumvents the limitations.

5.2 Which Frontend? Write in LaTeX or R markdown?

The original version of this document was prepared in R markdown that was compiled into HTML. There were several show-stopping flaws. The document was changed to LaTeX and PDF. Some features failed to compile, and some errors in my own code were silently ignored by the version of `pandoc` that was available in 2017. When the back end format changed to PDF, several special features of the HTML back end had to be removed. (Again, there is no free lunch.)

Similarly, R packages that can generate output that works well in HTML do not generate similarly good LaTeX output.

I believe the following are good conclusions:

1. If one intends to export as HTML, then markdown is, *by far*, the most reasonable choice for a frontend. Markdown was developed, first and foremost, as a simpler way to generate Web pages. There are converters that try to go from LaTeX to HTML, but in my experience, none of them preserve all of the features that are likely to exist in an even moderately complicated LaTeX document.[3]

2. If one intends to export to PDF, then markdown or LaTeX can be useful. But LaTeX is probably better. LaTeX is primarily intended for the creation of publication-quality documents in PDF format.

Since one can put much (not all) LaTeX markup into a markdown document, perhaps the difference is not so great as it seems. Because a markdown documents must be knitted, and then converted to LaTeX, and then to PDF, there is an "extra step" in translation that may corrupt some features.

The fact of life, at least at the current time, is that it is not painless to change the output format without making wholesale changes in the document fron tend. If one writes a markdown document, using special features intended for the back end, then it is generally not possible to, at the last minute, change the output format. HTML output has advantages in Web style features, while PDF documents have advantages in "on paper" presentations.

5.3 Where does my editor fit into this?

We need an editor that allows the insertion of code chunks and has the ability to compile documents (recall Figure 1). Microsoft Word is not a workable alternative, which is a source of concern for may social scientists.

I recommend LyX (http://www.lyx.org) to many LaTeX novices. It is a graphical interface for preparation of LaTeX documents. LyX can export LaTeX files and it offers some good tools to learn about how to write LaTeX. LyX was used to create the skeletons provided with `stationery`. We prepare the documents as .lyx files, and then export to an equivalent .Rnw file.

---

[3]Tables and equations with cross references are almost always damaged in conversion.

LyX offers many benefits, but it has one limitation (which all LaTeX editors will share). LyX allows us to enter code chunks in the document, but it does not offer a convenient way to test chunks and interact with an R session while we are writing the document. If one is editing in LyX (or in a LaTeX editor), the following shell commands can be used to convert the current document, `stationery.lyx`, to pdf:

```
$ lyx -e sweave stationery.lyx # creates a Rnw file
$ R CMD Sweave stationery.Rnw # creates a tex file
$ texi2pdf stationery.tex # creates pdf , could use pdflatex
```

Compiling the LyX document has an "all or nothing" feel, where errors in R code will cause compilation to fail. One can edit an .Rnw file in any editor intended for LaTeX, of course, but those editors will have the same limitation that one cannot test R code chunks interactively.

One can instead edit the `.Rnw` or `.Rmd` files with editors have an "inferior mode" that can start an interactive R session. The author can create a code chunk and test it interactively before trying to compile the whole document. There are several editors that have that ability. The venerable editor Emacs (??) offers ESS (Emacs Speaks Statistics) which is popular with experts for a number of reasons. Perceiving Emacs to be difficult to use, many novices enjoy RStudio, which has the ability to work with markdown or LaTeX files.

While working on the markdown documents in this package, we are struck by the fact that markdown is a movement, a *frame of mind*, rather than a product. It is a rapidly moving target. Features are added and changed on a weekly basis, it is usually difficult to know what will work and what will not. That is to say, we are on the steeply sloped curve of technical innovation and there is no reason to expect it to stabilize in the near term.

5.4 Should one prefer Sweave or knitr?

In R markdown, `knitr` is the only available method to process code chunks. There is no Sweave chunk engine for markdown.

For LaTeX docuements, we have the choice between the older `Sweave` chunk framework or the newer `knitr` framework.

knitr introduced a more fine-grained set of options for the chunks (see "knitr code chunk options," https://yihui.name/knitr/options).

One benefit of R markdown with `knitr` is that it is possible to make documents about other programs (not just R). I've explored knitr to weave documents about BASH shell programming, for example.
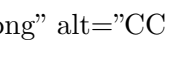
6 Conclusion

The `stationery` package is a snapshot survey of what is practical, here and now, for the preparation of reproducible research reports. There are some authors who take an extreme position that an entire research project, all phases in the analysis of data, should be wrapped together into a tightly wrapped bundle. At the current time, there are practical problems that prevent us from achieving this goal with perfect fidelity, but we can take some significant steps forward to integrate code and results into professionally acceptable guides and reports.

Authors who have used LaTeX, or written with LyX, will have some advantages as they enter this area. For those authors, it will be relatively easy to learn to insert code chunks and insert an extra step in the document compilation process. For authors that do not have LaTeX background, it seems certain that the R markdown format will be more appealing. It is easier to produce a rudimentary document with R markdown. That is to be expected, of course, since the markdown movement is propelled by a generalized rejection of ugly, tedious markup documents (either LaTeX or HTML).

Taking the details into account, however, we expect that authors that start with `.Rnw` or `.Rmd` files will likely encounter plenty of trouble as they try to integrate many appealing features. LaTeX equations, for example, can be included in either document format, but preparing them correctly is not easy. It is often frustrating to find that features that work well in one format are either unavailable or more difficult in the other framework. While there are many Websites touting examples of "pretty" or "elegant" documents, the these pages sometimes obscure the fact that their authors understand the compilation problems on a deeper level than the ordinary users. Those authors can more easily avoid pitfalls simply because they know, ahead of time, what will and won't work. This is especially true in the markdown community, because the `pandoc` program has a relatively long list of idiosyncrasies that the experts understand and avoid, while others are not so fortunate.

The experience in developing the examples provided with the package is that compiler tool chains are generally fragile and errors can be difficult to isolate and solve. The LyX authoring environment offers a very convenient way to rapidly move up the LaTeX learning curve, while it frustrates the author who wants to test code chunks while drafting a document.

Available under

[Created Commons license 3.0 <img src="http://crmda.dept.ku.edu/images/cc-3.0.png" alt="CC BY"

style="width: 75px;height: 20px;"/>](http://creativecommons.org/licenses/by/3.0/)

References

References

Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., & Chang, W. (2018). *rmarkdown: Dynamic Documents for R*. R package version 1.10.

Knuth, D. E. (1984a). Literate programming. *The Computer Journal*, 27(2), 97–111.

Knuth, D. E. (1984b). *The TeXbook*. Reading, Mass: Addison-Wesley Pub. Co. OCLC: 09322854.

Leisch, F. (2002). Dynamic generation of statistical reports using literate data analysis. In W. Härdle & B. Rönz (Eds.), *Compstat 2002 - Proceedings in Computational Statistics* (pp. 575–580). Physika Verlag, Heidelberg, Germany.

MacFarlane, J. (2018). *Pandoc: a Universal Document Converter*.

R Core Team (2017). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Stodden, V., Leisch, F., & Peng, R. D., Eds. (2014). *Implementing Reproducible Research.* Boca Raton: Chapman and Hall/CRC, 1 edition edition.

Xie, Y. (2015). *Dynamic Documents with R and knitr, Second Edition.* Boca Raton: Chapman and Hall/CRC, 2 edition edition.

Xie, Y. (2016). *bookdown: Authoring Books and Technical Documents with R Markdown.* Boca Raton, FL: Chapman and Hall/CRC, 1 edition edition.

Xie, Y. (2018). *knitr: A General-Purpose Package for Dynamic Report Generation in R.* R package version 1.20.

```
zz <- "stationery.Rout"
capture.output(sessionInfo(), file = zz, append = FALSE)
if (!is.null(warnings())){
    capture.output(warnings(), file = zz, append =  TRUE)
 }
```