stationery Package Overview: Guides, Slides, and Reports

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

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

Nov. 14, 2019

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 the same response. The `stationery` package for R is here to help. It offers working examples that new users may explore to find out what works.

The big idea of the *reproducible research documents* movement (Leisch, 2002; Stodden, Leisch, & Peng, 2014; Xie, 2015, 2016) is to replace traditional "cut and paste" research reports with a program-driven documentation system that blends calculations with report preparation. Researchers who use word processors have long been aware of the danger that computer code and output tables fall into misalignment. Hand edited tables are often incorrect, misplacing decimal points and parameter names. Proponents of reproducible documents ask us to change our workflow in a fundamental way.

In my experience, the benefits of this new approach are especially notable in class notes, slides, and small to medium sized reports. 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 is less danger that the computer code being discussed will not match the output.

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 (or almost 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, 2018) built its documentation framework on the literate programming philosophy, integrating Knuth's concepts of "weaving" and "tangling" into the R system for creating documentation (Leisch, 2002).

For most word-processor-using researchers, the transition will be jarring. First, the tools for statistical analysis (e.g., SAS, SPSS, Stata, or Mplus) that used to seem adequate are to be replaced with statistical software that is more readily integrated with document preparation. Second, the software for document authoring is being replaced. Microsoft Word is, simply put, insufficient. There is new priority on the integration of code and output within documents.

Once we understand that *big idea*, then comes another series of rude shocks. First, *there are several competing formats* for creating reproducible documents. The first examples prepared for this package were done with LaTeX in mind, but we have introduced a set of examples based on R markdown as well. The examples include code for "raw" LaTeX and LyX (a graphical interface for LaTeX document preparation, `http://www.lyx.org`).

Markdown document formats are, at least superficially, simpler. Their content may seem less obscured by formatting code. For example, in LaTeX to insert a bold font, one would write

`\textbf{this is bold}`, but in most dialects of markdown, a simpler `**this is bold**` would get the job done. Markdown is a reasonably legible format *as is*, and yet it can be compiled into more professional output formats. The leading voice for markdown has been John Gruber, whose Daring Fireball website (https://daringfireball.net/projects/markdown) offered the first working set of guidelines for markdown documents. Markdown is more like a movement than language, since there are many competing versions (called "flavors") of the language specification. The success of markdown depends, in a very practical way, on the success of John MacFarlane's ambitious software program `pandoc` (MacFarlane, 2018), a translator and compiler.

In my experience, it is relatively easy to build a simple document with either LaTeX or markdown. It is considerably more difficult to create a nice-looking document that incorporates the special features that we are used to in reports. And it is considerably more difficult to re-style the output to match user-driven specifications. The `stationery` package reflects the accumulated effort (and frustration) of several generations of graduate research assistants who have struggled to reconcile the reproducible document idea into demands of an organization that wants to issue visually consistent guides and reports.[1] Others who want to create their own distinctive styles can take the offerings here as evidence that there is light at the end of the tunnel.

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`. LyX is an editor that saves its files with the suffix `.lyx`, but can export to `.Rnw`.

back end: the delivered format.[2] The back ends considered here are Adobe portable document format (PDF) and hypertext markup language (HTML).

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

[1]Special thanks to Brent Kaplan, Ben Kite, and Charles Redmon.

[2]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.

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

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 "mix and match" front ends, back ends, and document 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. 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 provides R functions to initialize skeleton documents and convert them to a desired back end. It also provides a shell script with each document type that can be used to compile documents from the command line.

Finally, there are vignettes. In addition to the current document, there are vignettes named

1. Code Chunks, which compares Sweave and knitr style code chunks,
2. R markdown, which explores key elements in the newest front end, and
3. HTML Special Features, a tour of the promise and peril of using HTML as a back end for markdown documents.

2.1 `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 *frontend*2*backend.* 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).

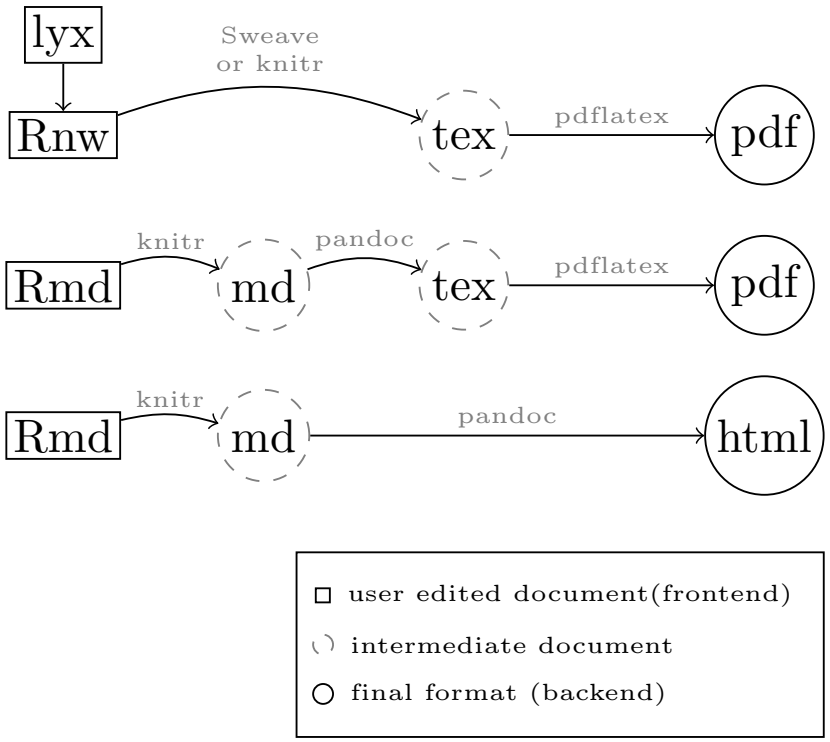
The process of converting a reproducible document from front end to back end is referred to as *compiling.* As illustrated in Figure 1, it is a multi-step process. Because the process can fail at any step, users are well advised to keep this fact in mind. There will be flaws in documents that can only be discerned by inspecting each stage in the process.

Between the front end and the back end intermediate files are created (and just as quickly erased). One of the current aims in software development is to make the transition process automatic and seamless, but errors at each stage are still fairly common. One challenge is to retain some flexibility for user input while weeding out compilation errors.

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

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 inserted in a LaTeX file, the document is referred to as a "noweb" file and the filename suffix is `.Rnw`. That file is 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 files will be compiled, usually by `pdflatex` or `texi2pdf`. 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.)

One meaningful difference between LaTeX-based documents and markdown-based documents is customization of document features in the compilation process. `pdflatex` and `texi2pdf` do not provide command-line options to change the look-and-feel of a document, while `pandoc` does. 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 a major advantage. This will be discussed further in section 3.2.

2.2 Incorporating computer code

Whether one is editing in a 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 about it). 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 markdown-based formats (sadly, not for the LaTeX based formats), the RStudio graphical interface will work 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)
```

Perhaps confusingly, the term template is used in different ways. The `rmarkdown::draft` function uses the name `template` to refer to larger collection of settings and document resources, while the same name is used by `pandoc` to refer only to particular file within the larger scheme. The `rmd2pdf-report` document skeleton, for example, specifies a template file in the document preamble:

```
template: "theme/report-template.tex"
```

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. The code chunks in the .Rmd file will be converted to output chunks in an intermediate file named `crmda.utf8.md`. 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-newtemplate.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.

```
$ ./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 character variables require quotation marks, and quotation marks need to be protected from interpretation by the shell. Here we protect double quotes in single quotes:

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

   We have made special effort to implement GNU style two-dash command line arguments (spaces are not allowed on either side of the equal sign).

4. Our LaTeX skeletons are provided with both `.lyx` and `.Rnw` files. The `.lyx` file can be opened with LyX. Compilation to PDF (via pdflatex) will be handled automatically by LyX. The LyX document has internal settings which indicate whether the code chunks will be handled by `Sweave` or `knitr`. LyX files can also be handled by the function `rnw2pdf` and the shell script of the same name.

5. If editing in `RStudio`, there is a button 

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. Our skeleton for `rmd2pdf` guide documents includes the following.

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

   The `RStudio` Knit 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` called `crmda_html_document` .

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

New versions of `RStudio` will notice that output format and offer a choice "Knit to crmda_html_document".

RStudio is also able to edit .Rnw files, but its File -> New setup is not customized for LaTeX files. Authors will need to create the new document in one of the ways mentioned above before editing it in RStudio.

4 Styling for Documents

4.1 Reports versus Guides

The `stationery` package document templates are intended to have consistent "look and feel" across document types. Reports created with markdown or LaTeX should look the same. Results created by documents using either Sweave or knitr for chunk processing should have a similar appearance. To the extent possible, the color schemes and arrangement of header and footer information should be similar (if not identical).

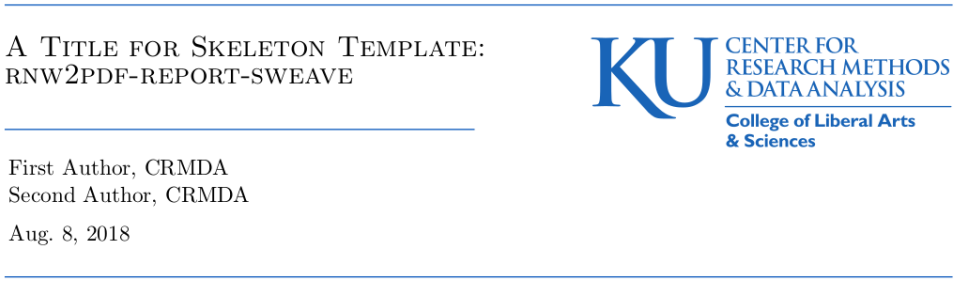
The stationery package includes two document styles, dubbed "guide" and "report". The difference between these two is not hard-and-fast. A **guide** is usually prepared as a teaching document. It may end up in a Webpage, where more color rather than less is expected. It is usually a less formal document. The final presentation is likely to include computer code and output excerpts. The 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 (Tantau, Wright, & Miletic, 2016) 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



(b) Footer

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.

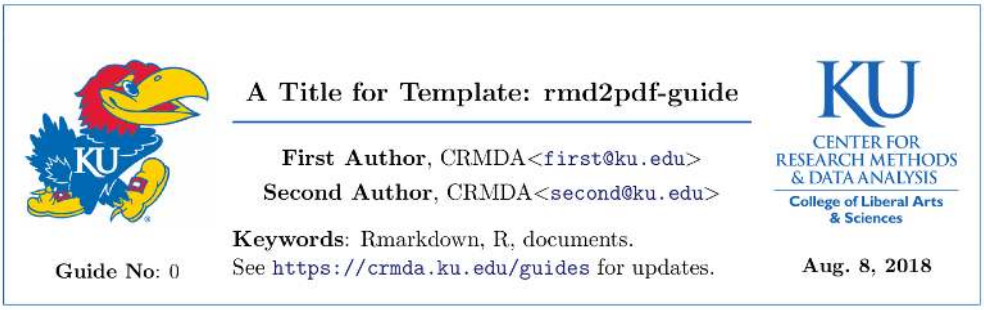
The markdown format introduced some interesting concepts to streamline document creation, so we begin with a markdown document's preamble. As exemplified in Listing 1, the markdown document begins with a section, written in YAML format, that species the title. A standard set of parameters is specified 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.dept.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.dept.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

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

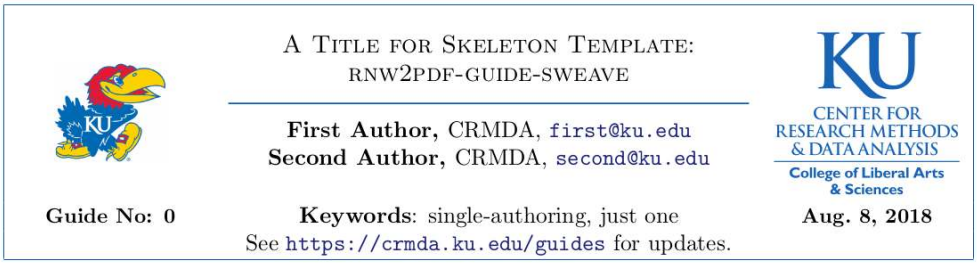
```
\guidesetup{
  author={
    lastname=Author,
    firstname=First,
    affiliation=CRMDA,
    email=first@ku.edu},
  author={
    lastname=Author,
    firstname=Second,
    affiliation=CRMDA,
    email=second@ku.edu},
  url={https://crmda.dept.ku.edu/guides},
  keywords={single-authoring, just one},
  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.dept.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



(b) Footer

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 (Thornton & Otto, 2018). 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-template.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. 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.[3] 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 end. 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 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 (`http://www.mathjax.org`). 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

[3] 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.

rendered. Second, 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 What do the front end formats have in common?

First, each one offers a method to include R code chunks. The formats and options differ, but the functionality is similar.

Second, all of these formats allow authors to include LaTeX expressions for math. Much, but not all, LaTeX code will work in either format. Inline and display equations can work well in either, while many of the attendant features (numbered equations, cross references) are not available when the back end is HTML.

Third, one can freely use "native" code that is suitable for back end. In a markdown document for which the back end is HTML, one can insert HTML code. It will be passed through without alteration to the back end in most cases. If the back end is PDF (via pdflatex), any legal LaTeX code can be inserted in a markdown document. To an considerable degree, this solves the problem that "there is no markdown for that." R functions that can write output in LaTeX or HTML format can be exploited in this way. There is no need to write a new function to create regression tables. One can shop among the alternatives, knowing only that the output format must be suited to the back end.

Nevertheless, the inclusion of native code in a markdown document is viewed as something of a defeat. Markdown enthusiasts would be more enthusiastic if we would either 1) live within the limitations of their framework, or 2) do the work to broaden the framework itself. For the first few years after markdown was introduced, it was possible to insert graphics in the document, but it was not possible to resize their display width or height in markdown code. To work around that, we inserted native HTML code to resize graphics. Several years later `pandoc` markdown introduced parameters to resize markdown images.

5.3 What considerations differentiate the front ends?

The original version of this document was prepared in R markdown that was compiled into HTML. There were several show-stopping flaws. 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. The document was changed to LaTeX and PDF. 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.)

I believe the following are reasonable conclusions:

1. If one intends to export as HTML, then markdown is, *by far*, the most reasonable choice for a front end. Markdown was developed, first and foremost, as a simpler way to generate Web pages. There are special features in the HTML back end that can only be exploited in markdown documents. There are always some proponents of LaTeX that would have us convert to HTML, but none of the suggested avenues preserve equations and tables (in my experience) as well as markdown to HTML conversion.

2. If one intends to export to PDF, then either 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. While it is true that markdown documents can benefit from insertion of LaTeX equations and some structures, they lack the ability to exploit many of the strengths in LaTeX document preparation.

Another concern is that R markdown implies an "extra step" in the compilation process. `.Rmd` is knitted into `.md`, which then must be converted to `.tex`, from which PDF is created. In contrast, a document prepared in `.Rnw` format will be converted straight to `.tex`. The "extra step" of translation (from `.md` to `.tex`) is avoided in LaTeX.

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 front end. 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.

5.4 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. One must either use a general purpose programmer's file editor and then compile the document using the R functions or command line scripts outlined above, or one may use an integrated development environment.

I recommend LyX (http://www.lyx.org) to many LaTeX learners. 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.

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 interactively. Rather, one must compile the whole document. I recommend running LyX from a terminal so that the R error messages will be easy to review, but that is only a modicum of relief. If one is editing in LyX (or in a LaTeX editor) and saves a file, the shell script rnw2pdf.sh will be able to compile the file. It will execute a sequence of steps roughly as follows:

```
$ 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
```

When this succeeds, the result is fine, but when it fails, finding errors can be difficult. Compiling the LaTeX or LyX document has an "all or nothing" feel. 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.

As an alternative workflow, 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 (`http://emacs.org`) offers Emacs Speaks Statistics (ESS; `https://ess.r-project.org`) 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. Neither Emacs nor RStudio make the creation of LaTeX markup especially easy, however.

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 with several competing sets of idioms. Features are added and changed on a weekly basis. That is to say, we are on the steeply sloped curve of technical innovation.

I extend an apology and salute to users of Emacs Org mode, which is an entirely different implementation of the same ideas as markdown (Org existed first). Org mode has a document header that is very similar to markdown, and it also has code chunks. We used Org mode to prepare a series of tutorials about Mplus programming[4] and the editing process is rather similar to markdown. The main reason that I do not emphasize it in this context is that the number of Emacs Org mode users is quite small compared to either LaTeX or markdown at the current time. The development effort on Org mode is unpredictable. It is much more difficult to find accurate information about how to customize output from Org mode files.

5.5 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 documents, we have the choice between the traditional `Sweave` chunk framework or the newer `knitr` framework. The syntax can be very similar, but `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 reproducible documents about other programming languages. I've explored markdown with knitr to weave documents about BASH shell programming, for example.

The major reason why I resist `knitr` is that the format of the output is very difficult to control. Over the years, I have used the Listings class in LaTeX with Sweave to customize the appearance of code input and output. So far, at least, I've not had success customizing the output from `knitr`.

6 Conclusion

The `stationery` package is a snapshot survey of what is practical, here and now, for the preparation of reproducible research reports. It seems likely that both LaTeX and R markdown documents will continue to exist (and compete for user loyalty) into the indefinite future. Neither of these authoring frameworks is without flaws, but each is workable.

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 single document. At the current time, there are practical problems that prevent us from achieving this goal with perfect fidelity. While the software to render regression tables has improved during the past 10 years, none of the available programs will offer a perfectly finalized presentation for all kinds of models. Rather, it appears we

[4] http://gitlab.crmda.ku.edu/crmda/semexample

do need to leave room for some "hand editing" of tables that are exported from R. In the production graphs for figures we are considerably closer to the perfectly presentable standard, of course, and tables may approach that standard at some point.

The transition to a reproducible document framework will present challenges for new users. Authors who have used LaTeX, or written with LyX, will find this more familiar. For those authors, it will be relatively easy to learn to insert code chunks and re-think the document compilation process. For authors that do not have LaTeX background, it seems certain that the LyX versions of the rnw2pdf templates, or perhaps the R markdown formats, will be more appealing than raw LaTeX. It is easier to produce a rudimentary document from scratch with R markdown, but using the skeletons we provide, it seems that all of these documents should be equally convenient. General opinion among the younger crowd on the Internet seems to more enthusiastically endorse R markdown, while the more experienced (dare I say "traditional"?) crowd prefers LaTeX.

It is perhaps not honest to predict that new users will find any of this to be easy or as convenient as a word processor. It is more likely to be correct and reproducible, but it is not easy. One can expect that authors, whether they start with `.Rnw` or `.Rmd` files, will hit bugs as they add features which seem natural and obvious. LaTeX equations, for example, can be included in either document format, but preparing them correctly is not easy. Features that work well in one format are either unavailable or more difficult in the other. While there are many Websites touting examples of "pretty" or "elegant" documents, it seems certain that new visitors to the reproducible documents will find bugs where the experts never dreamed them to venture. Experts know what won't work, but they don't usually emphasize the difficulties as much as the victories. Although the markdown community has many eager, outspoken proponents, I have to admit that every markdown project on which I've worked hit one or more deep, unsolvable bugs that required re-engineering of a presentation. One source of change/trouble is `pandoc` itself, which is evolving so rapidly that software distributors cannot keep up.

One issue that users should be aware of is that the compiler tool chains are generally fragile. Slight changes that "seem safe" can cause compilation failures with vague error messages. This is partly due to the fact that the LaTeX compiler has its own layer of difficult-to-understand errors. Also, the process of Sweaving/knitting will stumble on errors in R code and sometimes the weaved output file will cause errors in the LaTeX compiler. The LyX authoring environment offers a very convenient way to rapidly move up the LaTeX learning curve; some difficulties can be avoided. However, LyX does not have a convenient way to test code interactively, which makes the R work more frustrating. On the other hand, an authoring environment that allows one to interact with R more directly, such as Emacs or RStudio, places the LaTeX learning curve as an obstacle at the outset.

References

Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., ... Lister, A. (2018). rmarkdown: Dynamic documents for R [Computer software manual]. Retrieved from https://cran.r-project.org/web/packages/rmarkdown/rmarkdown.pdf (R package version 1.10)

Knuth, D. E. (1984a). Literate programming. *The Computer Journal*, *27*, 97–111. doi:10.1093/comjnl/27.2.97

Knuth, D. E. (1984b). *The TeXbook*. Reading, MA: Addison-Wesley.

Leisch, F. (2002). Sweave: Dynamic generation of statistical reports using literate data analysis. In W. Hardle & B. Ronz (Eds.), *Compstat 2002: Proceedings in the 15th Computational Statistics Symposium in Berlin* (pp. 575–580). New York: Physica-Verlag Heidelberg.

MacFarlane, J. (2018). Pandoc user's guide [Computer software manual]. Retrieved from https://pandoc.org/MANUAL.pdf

R Core Team. (2018). R: A language and environment for statistical computing [Computer software manual]. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://cran.r-project.org/doc/manuals/r-release/fullrefman.pdf

Stodden, V., Leisch, F., & Peng, R. D. (Eds.). (2014). *Implementing reproducible research*. Boca Raton, FL: CRC.

Tantau, T., Wright, J., & Miletic, V. (2016). The BEAMER class: User guide for Version 3.50 [Computer software manual]. Retrieved from http://www.ctan.org/tex-archive/macros/latex/contrib/beamer/doc/beameruserguide.pdf

Thornton, J., & Otto, M. (2018). Bootstrap [Computer software manual]. Retrieved from https://getbootstrap.com/

Xie, Y. (2015). *Dynamic documents with R and knitr* (2nd ed.). Boca Raton, FL: CRC.

Xie, Y. (2016). *bookdown: Authoring books and technical documents with R Markdown*. Boca Raton, FL: CRC.

Xie, Y. (2018). knitr: A general-purpose package for dynamic report generation in R [Computer software manual]. Retrieved from https://yihui.name/knitr (R package version 1.20)