stationery Package Overview: Guides, Slides, and Reports

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

Paul E. Johnson, Director, CRMDA

Aug. 8, 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? A university administration recently exclaimed, "I have no idea what that means!" It appears many members of the faculty have the same response. The stationery package for R, and the documentation provided with it, may help.

Researchers who have been using word processors are used to copying and pasting graphs and tables from statistical output. 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. Make the software for statistical analysis work more seamlessly with the document authoring process.

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 is no 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 not sufficient. 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 authoring.

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

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

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 statistical code is to be embedded in a research report, then comes a series of rude shocks. The first shock is that *there are several competing formats for doing this kind of work.* The stationery package includes working examples for several different file types. Our original work was done with LaTeX in mind. Authors can work in LaTeX or with LyX (http://www.lyx.org), a graphical user interface for LaTeX that works like a word processor. We also offer the markdown document format. Proponents claim that markdown is simpler and more human-readable, although we notice many limitations along the way. 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 gained ground in large part due to John MacFarlane's ambitious software program `pandoc` (https://johnmacfarlane.net/pandoc), a format translator that is used in all desktop computer systems for converting markdown documents to PDF and HTML.

The other shock coming for new users is that the quality of the output is generally unfamiliar, if not unacceptable. The `stationery` package is intended to ease the frustration. It offers working examples (and instructions) for generating output that is more compatible with our usual standards.

A glossary of the most important terms is provided next.

front end: the format in which a document is prepared. We consider markdown and LaTeX-based formats. Markdown files are saved with a suffix .Rmd, while the LaTeX versions are often 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 styles. We hope that authors who are accustomed to writing on stationery, with a pleasant header

[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 referred 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

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. Rmarkdown: 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.

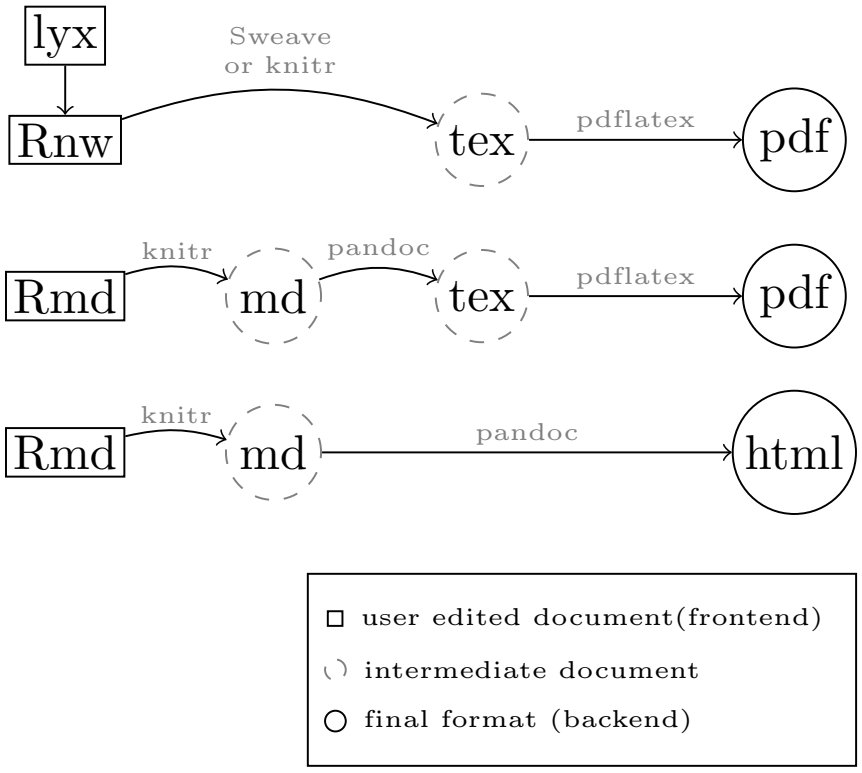
`stationery` package document templates

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 *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. 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. 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 refer to chunk converion as "Sweaving" (because S was the precursor to R). The newer `knitr` 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 via a variety of formats (referred to as environments in LaTeX). The traditional method was the LaTeX class for `Verbatim` output, but a newer, more desirable method uses the `Listings` class. 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 Styling for Documents

Reports versus Guides

The stationery package includes two document styles, dubbed "guide" and "report". A **guide** is 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 document. 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 Xb and Xc). 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.

The slide format we offer is based on LaTeX Beamer () slide format using a customized theme that we prefer. 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.

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 using the Sweave and knitr chunk processing engines. These guides will be visually consistent. The same is true for report documents produced by Sweave and knitr.

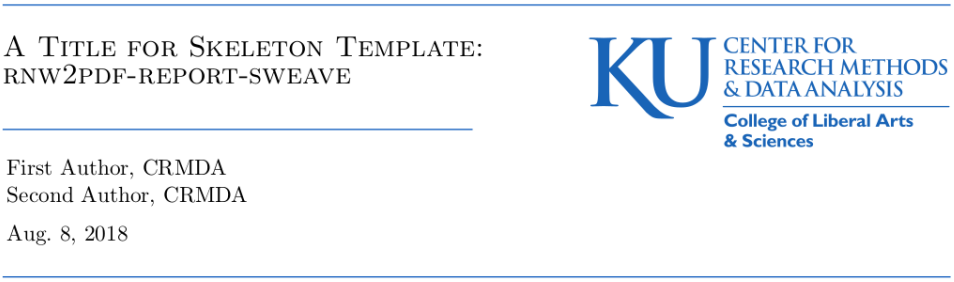
Distinctive Headers and Footers

In the stationery package, we provide enhanced headers (and footers where appropriate) that incorporate organizational graphics and address information. This is done by incorporating information in the beginning of the document which is later formatted by a template. We have created functions for LaTeX documents that mimic some features of markdown documents.

In order to introduce our LaTeX modifications, we start with an example of a markdown prefix. As exemplified in Listing 1, at the top of the markdown document, there is a section, written in YAML format. The names of the parameters can be drawn from a standard set specified by the pandoc program, but also there can be customized values specified by a document template. 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 "Rmarkdown Basics." The header and footer created from the document are presented in Figure 3.

Figure 2: Latex Output (Report Document)

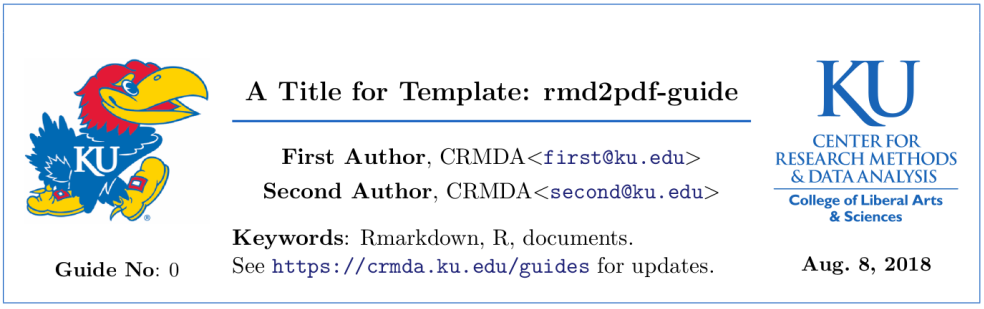
(a) Header



(b) Footer

Figure 3: Markdown Output (Guide Document)

(a) Header



(b) Footer

The clarity of separation between the document header information and the document content is widely regarded as a benefit of the markdown document. Another benefit of the markdown system is that the header is flexible; one can insert names of additional co-authors in the obvious way.

In LaTeX, it is somewhat difficult to achieve the same degree of flexibility, but the problem is solved by some LaTeX functions in our templates. These use recent innovations in the LaTeX programming interface (with the help of several LaTeX experts who answer questions in stackexchange). The top portion of the LaTeX document is illustrated in Listing 2. The input format is designed to be as similar to the markdown format as possible, but there are slight differences due to inherent

Listing 1: Beginning of a markdown preamble

```
---
title: "A Title for Template"
subtitle: "rmd2pdf-guide"
guidenum: 00
guideurl: https://crmda.ku.edu/guides
keywords: Rmarkdown, 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
```

differences in technology. One way in which the LaTeX approach is preferable is that the address information in the footer can be stored in a separate file and re-used where necessary (which markdown does not allow). The output in the header and footer is displayed in Figure 4.

How can users customize this? The most obvious user-customization is the replacement of the logo icons for guides and reports. In the beginning of all our basic documents, there is a function that retrieves logo files (graphics and the footer address information for LaTeX documents). As provided, the getFiles function will retrieve empty white graphics, but there are two ways to customize. The simple, but tedious method, is to replace the files "theme/logoleft.png" and "theme/logo-vert.png". The slightly more tedious, but automatic way, is to create an R package to hold the logo information (we can supply a working example of a package named "crmda"). Then to customize the document, the author will then alter the following code chun, replacing `pkg = stationery` to `pkg = "your_pkg_name"`.

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

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 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 Usage overview

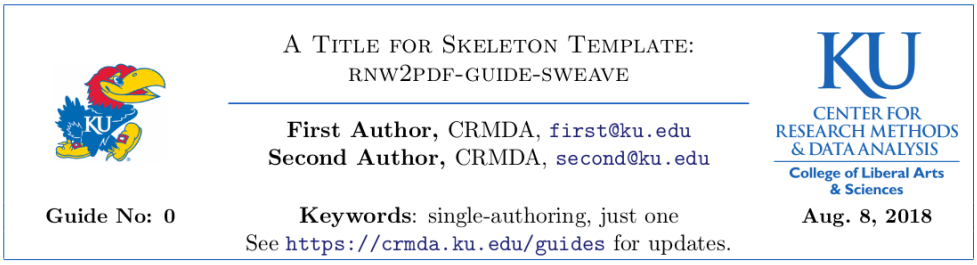
4.1 Create a starter "skeleton" document

The stationery package includes a function called `initWriteup`. This can create simple ready-to-compile examples for all document types. Here we will illustrate the process of initiating a guide document with Rmarkdown and will have the HTML backend. Start R in a folder where you would like to create a write-up and run

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

Figure 4: LaTeX Output (Guide Document)

(a) Header



(b) Footer

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

1. a skeleton template, skeleton.Rmd (which the author should rename and edit),
2. an instructional guide, instructions.Rmd,
3. a compiler script, `rmd2pdf.sh`,
4. a directory `theme`, in which a template and some other configuration files are copied

The output will tell you where the directory ended up, say:

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

Most users will rename the file “skeleton.Rmd” and the subfolder ”rmd2pdf-report” to something more relevant to a project. Here we might use the name `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 (as specified by the package `rmarkdown`).

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 as our function `initWriteup`. One could run

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

4.2 How to Compile a Document

Users should try to compile our document before changing it. After making revisions, users are always urged to test the document.

The file can be compiled in several ways.

1. Open an R session and use the function `rmd2pdf()` in the `stationery` package. For example,

```
library(stationery)
rmd2pdf("crmda.Rmd")
```

The function `rmd2pdf` allows a great many possible parameters. This function orchestrates a two part process that goes on within the `rmarkdown` package. First, a document format object must be created. That function is handled by the `pdf_document` in the `rmarkdown` package (or, for HTML output, `html_document`). After that, the document is rendered by a process that harvests the document format values and sends the work to the pandoc function. The tricky part of this process is that the document itself may specify a parameter that is overridden by `pdf_document` (or `html_document`). For example, if the document header requests a table of contents by stating `toc: true`, running `pdf_document` without specifying the parameter `toc = TRUE` will cause the default `toc = FALSE` to be used. These subtle issues are solved by the usage of the `rmd2pdf` and `rnw2pdf` function as the primary compiling tools.

The `rmd2pdf` function allows parameters to be selectively replaced without changing the document in question. As a use case, suppose we write 3 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)
```

This approach can be used to replace any of the parameters for which the `rmarkdown::pdf_document` function is scanning. For R markdown, our template is `theme/report-boilerplate.tex`. To employ an alternative template, the argument `template = ■theme/report-newboilerplate.tex`" can be used.

There are differences in format between the values in the markdown document preamble and the R function call. R uses `TRUE` and `FALSE` where markdown use `true` and `false`, for example.

2. 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
```

3. If editing in `R studio`, there is a button "Knit." This should be used with *extreme caution*!

There is a small triangle beside the Knit button, and it offers a selection of document back ends. Be careful to choose the correct one. In this case, we choose "Knit to PDF".

If the user makes a mistake and selects the wrong back end, then `R studio` does something that is quite unexpected. It alters the document preamble to insert its best guess about the desired format. The document header will be adjusted, sometimes mangled so that even "Knit to PDF" will fail.

4.3 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 boilerplate of javascript and cascading style sheets from bootstrap is inserted into the HTML header.

Rather than editing the document over-and-over to see the effects, we suggest instead either using the function or the compiler 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. To achieve the same effect by editing the markdown document the correct syntax will be `theme:  null` (neither capitalized nor quoted).

Of course, the disadvantage of removing the theme is that the benefits of the theme are lost. In the accompanying vignette "Rmarkdown HTML Special Features," we illustrate the use of some special features in the bootstrap theme.

The allowed bootstrap themes 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
```

4.4 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, it is recommended to 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, we can usually find out what's wrong.

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
    /tmp/pj3/Untitled/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/rmarkdown-str33e24223b3a.html --mathjax --variable
    'mathjax-url:https://mathjax.rstudio.com/latest/MathJax.js?config=TeX-AMS-MML_HTMLorMML'
    --filter /usr/bin/pandoc-citeproc
```

5 Working with LaTeX

Like so many other concepts and tools in this area, TEXwas a creation of Donald Knuth at Stanford. TEXwas the precursor to LATEX.

The CRMDA maintains a Web page about LATEX: https://crmda.ku.edu/latex-help

That page has basic guides and information about the KU dissertation template. I prefer to work with LATEXdocuments, for a number of reasons. The quality of the PDF output is nicer, in my opinion, and more predictable. However, the major reason I prefer LATEXis that the Sweave option `split=TRUE` is allowed. That option creates separate `*.tex` output files, for each code chunk. The developers of the

Rmarkdown documents framework disapprove of 'split' and elected, consciously, not to implement it. A couple of the questions not considered in our Web page are the following.

6 Choosing among formats

We provide 8 document formats because there are several acceptable methods to achieve the same goal. At the current time, there is a considerable amount of enthusiasm about markdown document that generate HTML output. Our experience is that PDF output is considerably more dependable and one ?? ??

6.1 Which backend?

Should I end up with HTML or PDF? The answer depends on the intended audience/client. If a "paper" must be submitted, obviously 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 HTML is the obvious choice (unless you simply want to convey a PDF document via the Web). Our HTML template includes a cascading style sheet feature that allows both color-highlighted sections and tabbed sub-sections.

There is concern about mathematics in HTML documents that deserves mention. For many years, the inability of Web pages to display equations was a major problem. Many tedious, ugly methods were developed. A more-or-less workable solution was developed, a framework called MathJax. MathJax allows inclusion of math markup in the page which--when the conditions are right--can be converted by the Web

browser to look like equations.

If the user is offline, or if the MathJax server is not available, then the HTML document's math will not display. If one wants to put math into a document, using HTML is inherently risky. If one needs to be 100% sure that math will display as intended, choose to create PDF documents.

6.2 Which Frontend? Write in LaTeX or Rmarkdown?

This will be the answer:

> When choosing the frontend, consider the backend. Where you want to end up determines where you start.

While working on this document, I prepared an original version in Rmarkdown that was compiled into HTML. Because some features failed to compile, I changed the backend to PDF in the report style. As a result, several features that are unique to the HTML backend had to be removed. HTML offers access to some special document formatting features that are simply not available in PDF, and the converse isalso true.

I believe the following are good conclusions:

1. 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.
2. 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. Conversion from LaTeX to HTML is less decidedly less unsatisfactory.

Since one can put much LaTeX markup into a markdown document, perhaps the difference is not so great as it seems. The Rmarkdown compilation process (see Appendix \ref{appendix1}) generates a LaTeX file at an intermediate stage, so in some sense the same PDF result ought to be possible with Markdown or LaTeX document preparation. However, in practice, we find differences in conveniences for authors.

I would summarize the situation with a poem:

> Markdown documents intended for HTML allow some LaTeX code.
>
> Markdown documents intended for PDF allow more LaTeX code.
>
> Almost all HTML code is tolerated well in a Markdown document intended for HTML.
>
> No HTML code is tolerated in a document intended for PDF.

The main point is that if one writes a markdown document, using special features intended for the backend, then it is generally not possible to, at the last minute, change the output format from HTML to PDF, or vice versa. HTML output has advantages in Web style features, while PDF documents have advantages in ”on paper” presentations.

For novices, LaTeX seems more difficult than markdown. Perhaps this is not such a big hurdle as it used to be because [LyX](http://www.lyx.org) is available. In my opinion, ‘LyX‘ makes preparing a LaTeX document much easier than it is to prepare

a similarly complicated markdown document.

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.

6.2.1 Important caution about Math in the HTML backend

Math is not incorporated in HTML in the same way as PDF. Compiling a document into PDF uses a program like ‘pdflatex‘ to put the equations ”in” the document. They are displayed in (more or less) the same way on various browsers and operating systems. The same is not true for math in HTML documents. Simply put, **math is not

allowed in HTML**. We think it is allowed--our eyes tells us it is

allowed--because we browse Web pages that show equations. However,

this is an illusion achieved by extraordiary measures involving

Javascript and third party servers. The beautifully formatted \LaTeX

equation is not ”embedded” in the HTML, it is instead delivered as

code ”available for rendering” in the Web browser. The HTML code is

converted, via javascript and functions supplied interactively from

the MathJax Web server.

Markdown to HTML allows most valid HTML markup, but Markdown to PDF does not allow all LaTeX.

This is a somewhat surprising difference. All HTML markup I’ve tried

works well in an Rmarkdown document that aims to go into HTML.

However, not all LaTeX markdown is allowed in an Rmarkdown document

going to PDF. And even less LaTeX code works well if the intended

backend is HTML. One cannot insert italics with LaTeX in a document

intended for HTML. For example, writing ‘\emph{italics}‘ or bold

‘\textbf{bold}‘ in the style of LaTeX code will have no effect in an

HTML document. However, if PDF output is used, then both of those

LaTeX codes work. As evidence, note I get \emph{italics} and bold

\textbf{bold} in this PDF document.

6.3 Should one prefer ‘Sweave‘ or ‘knitr‘?

This question is meaningful only in ‘noweb‘/ LaTeX documents. In Rmarkdown, ‘knitr‘ is the only available method to process code chunks. In ‘noweb‘, one can choose between ‘Sweave‘ and ‘knitr‘. Perhaps that suggests that, if one must learn one set of chunk options, then ‘knitr‘ options are the right place to start (since they can be used in documents intended for HTML or PDF). The chunk options allowed the original ‘Sweave‘ were ‘echo‘ (include code with output?), ‘eval‘ (run the chunk calculations?), ‘include‘ (display the chunk in the document?), ‘fig‘ (code generates a figure?) and ‘results‘ (output in LaTeX is handled differently than raw \TeX). There are a few others, but that is most of the story. ‘knitr‘ honored most of the Sweave options and then added many more (see [knitr code chunk options](https://yihui.name/knitr/options)).

One benefit of Rmarkdown 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.

7 Troubleshooting

The first step is understanding the trouble. The trouble stems from the fact that each document must be transformed through several stages to reach the final result. Understanding that, and learning about the problems that appear at each stage, can help with the troubleshooting strategies that we recommend.

7.1 Compilation Stages\label{appendix1}

Steps to compile documents break down into 2 phases.

1. Handle code chunks

2. render the resulting document.

A noweb file is converted from Rnw into PDF by a sequence of transitions.

1. ‘Rnw -> tex‘. This is called ”weaving” or ”knitting”, depending on whether Sweave or knitr is the code processing engine. R finds the code in the Rnw file and inserts results into a new LaTeX file. The difference between weaving and knitting will be explained below.

2. ‘tex -> pdf‘. The default is ‘pdflatex‘ for this step, but the alternative ‘xelatex‘ is growing in popularity because it more gracefully handles Unicode characters (utf8). If the document is edited with [LyX](http://www.lyx.org), there is an

implicit step 0,

0. ‘LyX -> Rnw‘.

In the LyX pull down menu system, this is represented by Export -> Sweave.

A markdown file is converted from Rmd to HTML by this sequence of transitions

1. ‘Rmd -> md‘. The ”knitting” process replaces code chunks by R input and output, converting the R markdown file into an ordinary markdown file. In my system, an ”md” file is generated, and then a second ”utf8.md” is generated to clean up the file encoding.

2. ‘md -> HTML‘. Currently, most people use the program ‘pandoc‘ for this. A version of ‘pandoc‘ is distributed with Rstudio for the convenience of users. Linux users probably have ‘pandoc‘ available as standard system packages and the Rstudio version be removed.

The production of PDF from markdown, involves an additional transition.

1. ‘Rmd -> md‘. Knitting converts code chunks into R input and output that is inserted into an ‘md‘ file created in the ‘pandoc‘ markdown style.

2. ‘md -> tex‘. A LaTeX file is created by ‘pandoc‘. In the header of the ‘md‘ document, one can set a number of parameters to alter the LaTeX generation process. For troubleshooting, ”keep_tex: yes” to keep the ‘tex‘ file.

3. ‘tex -> pdf‘. The default program for this has been ‘pdflatex‘. It may be important to know that a LaTeX document may need to be run through ‘pdflatex‘ several times because cross-references among pages and equations need to be made consistent.

The R packages ‘rmarkdown‘ and ‘knitr‘ orchestrate the process that builds the instructions to ‘pandoc‘. In ‘rmarkdown‘, the function ‘render‘ orchestrates all of the work. It calls chunk calculator and assigns work among the various conversion programs. The functions in the stationery package named ‘rmd2pdf‘ or ‘rmd2html‘ are ”wrapper” functions that adjust settings sent to ‘render‘.

7.2 Avoiding compilation trouble

The document production phase can fail at many steps (see Appendix \ref{appendix1}). While editing a document, authors are well advised to heed the advice:

\begin{quote}

\textbf{Compile early, compile often!}

\end{quote}

When a mistake is inserted, it is best to find it as soon as possible.

7.3 When debugging, check intermediate files

The compiler scripts may erase intermediate files. While debugging a document, we want to disable that clean-up step so that we can see what goes wrong. In an Rmd document that ends up in PDF, for example, we should be able to inspect an ‘md‘ file and a ‘tex‘ file. We can not only inspect those files, but we can also attempt to compile them in isolation so that we can see what is going wrong.

This document includes the header argument ‘keep_tex: true‘, which means we save a copy of ”reports_and_guides.tex”.

While developing this document, some of the problems with backend-specific code have come to the forefront. The HTML backend allows pleasant color-coded section headings which included in

the PDF output. Tables that work well in PDF documents don't work in HTML, and vice versa. Some HTML tables that are legal HTMLdon't cooperate with 'pandoc'.

It is also worth mentioning that error messages are not always informative. In fact, we sometimes don't get error messages when we should. Rather, we simply receive bad output. While developing this document I noticed that when a user includes erroneous LaTeX code in a markdown document, a flawed HTML output is generated without error or warning. On the other hand, changing the intended backend to PDF causes the compiler to fail and issue an error message. If one is exporting to HTML, then, a very careful proofreading of the output to check conversion of LaTeX code into HTML is necessary.

8 What do we Really, Really Need?

Things we wish we could have in HTML output (that we can get in PDF output)

Numbered equations, easy cross references, numbered tables and figures

Things we wish we could have in PDF output (that we can get in HTML)

A splash of color, mainly. This is possible in PDF, but more

difficult, at least on the surface.

8.1 Math

We are a Center focused on methodology. It is necessary to be able to write about math, preferably with a standard, uniform mathematical markup language, such as \LaTeX.

Many social scientists are not familiar with LaTeX document preparation. That was a hurdle that kept many authors with Microsoft Word, even when they were frustrated with it. The difficulty of using LaTeX was solved, to a significant extent, by LyX, an open source graphical interface. With LyX, or other editors that could generate LaTeX output (such as Scientific Word, TexMacs, or Abiword), authors who were not computer programmers could learn enough LaTeX to finish their projects.

8.2 Literate documents: include code and output

We need to be able to write about computer code. The "old fashioned" way is to copy/paste code and output into documents. That's somewhat error prone and difficult to keep up-to-date.

Donald Knuth, a famous Stanford professor of computer science, proposed strategies to integrate the production of documents with the development of computer code. Rather than creating code in one file, and documentation in another, the idea was that the two parts of our work should be blended in a "literate programming" exercise.

The literate programming idea is more a general way of life than it is a particular document production strategy. It is, partly, aimed at programmers who don't like to write instruction manuals. In the end, however, it may have more impact on non-programmers who need to prepare technical reports that include computer code examples.

In computer programming, one of the biggest impacts of literate programming is the proliferation of systems for preparation of documentation within code files. In the 1990s and early 2000s, when I was working on the Swarm Simulation System, we used a framework called Autodoc that allowed us to write instructions into Objective-C code

that were later harvested and turned into instructional manuals. See, for example, the documentation for the [Opinion Formation model]

(http://pj.freefaculty.org/Swarm/MySwarmCode/OpinionFormation/Opinion-Docs). The

Autodoc program was poorly documented and not easy to get, but soon after that, a new coding system called Doxygen became widely available. Doxygen, developed for creation of instruction manuals for C++ programs, was a major success in computer programming. Like Autodoc, Doxygen gave programmers a relatively convenient method to explain what they were doing without wasting too much time.

In the modern experience of most CRMDA staffers, the "documentation inside code" approach is visible in the Roxygen markup method used for functions in R documents. Any function worth using should have Roxygen markup.

8.3 Where we have been

In order to embrace the importance of using either markdown or \LaTeX, one must first abandon the idea that Microsoft Word can ever be useful for serious authorship. That's a big step for many graduate students and professors.

If we look past GUI "what you see is what you get" (*wyswig*) word processors, where do we go? For a long time, the only answer was \LaTeX. However, there was a fatal weakness in \LaTeX. It is intended for PDF output, not Web pages. Exports into HTML were problematic. LaTeX was not only difficult for some to use, but it also did not benefit from fancy features that were becomming available in the Internet, especially cascading style sheets and Javascript.

This gap in the document production process created a need for a new methods. In the 1990s, there was quite a bit of effort to make user friendly Web page editors, so that authors could have a Word-like experience that would generate HTML. The end result, generally, was difficult-to-maintain HTML documents. It was generally not feasible to edit and revise documents, the HTML generated was both extremly complicated and generally unsatisfactory.

Another strategy was the development of alternative markup languages. In a way similar to \LaTeX, these markup languages (e.g., 'docbook') drop the idea that the user should have a *wyswig* experience. Instead, the author would again become a programmer who would insert symbols to create sections.

Markdown was developed as a rejection of both ugly markup documents (either LaTeX or HTML) and *wyswig* editors. The idea is that documents should be text files that are readable *as is* but also convertible into other backends. The ”markdown” movement seeks to deliver an easier-to-edit, less difficult-to-read, and easier-to-convert format. The leader is John Gruber, whose Website is boldly named [”daringfireball”](https://daringfireball.net/projects/markdown).

Markdown is intended to be easy-to-read, so that even if it is not compiled into a backend, it might be presentable to an audience of non-programmers. That general idea seems unrealistic to me, there is almost never going to be an audience (for CRMDA, at least), which is eager to look at a markdown file. The dropback argument is that a markdown file is more easily produced by a novice who does not want to learn to use \LaTeX. This seems more persuasive.

The strength of markdown is that it makes it fairly easy to produce HTML documents that utilize some (not all) of the strengths of the World Wide Web's most commonly used method of communication.

Where markdown is not capable, one can still write ”raw HTML” in the middle of a markdown document.

9 Required footer

The guide documents we require authors include a final chunk that includes the session information, to be used in bug-tracking.

Reports do not include raw output, so it is not recommended to insert that raw output in the report. Instead, we ask report writers toinclude a final R chunk that saves the session information in a file in the same directory as the pdf output.

“‘{r session, include=F}

zz <- ”stationery-sessionInfo.Rout”

capture.output(warnings(), file = zz)

capture.output(sessionInfo(), file = zz, append = TRUE)

“‘

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

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). *Compstat 2002 - Proceedings in Computational Statistics*, chapter Dynamic generation of statistical reports using literate data analysis, (pp. 575–580). Physika Verlag, Heidelberg, Germany.

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.

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