Guide for authoring reports with Sweave code chunks

Paul E. Johnson, Director, CRMDA <pauljohn@ku.edu>
February 24, 2018

This shows how we use R (R Core Team, 2017) to make report documents using the CRMDA style. This is a LaTeX "noweb" guide document generated with the knitr code chunk engine.

1 Introduction

This is a LaTeX "noweb" report generated with the Sweave engine.

Create a skeleton document by opening R and running

```
library(stationery)
initWriteup("rnw2pdf-report-sweave")
```

That will create a folder "`writeup/rnw2pdf-report-sweave`" (unless you request otherwise by setting the `dir` argument). We can compare it to the very similar document produced with "knitr" (in the companion folder rnw2pdf-report-knitr).

We suggest you

1. Compile this document *as is* to test your setup
2. Make revisions incrementally, and re-compile often to make sure nothing has been broken.

Don't make changes that you don't understand in the code chunks above or the last chunks below.

2 LyX: Cautions

The document preamble has manual settings for margins (geometry) as well as hyperlinks (PDF hyperref). Don't use the LyX pull down menu to revise them. Edit preamble or config files for that. Repeat **CAUTION**: Don't change the page margins or settings for hyperlinks with pull down menus.

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

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

3 What to edit

Title and author information

The first block in the document has the title and author information.

Footer information

The footer in this document uses data that is provided in a file named "addressFooter.tex". After the document is compiled for the first time, that document should be available in the theme folder.

About the theme folder

The theme folder should be empty when the initProject() function is run.

There is an R code chunk above called "texcopy". It will copy configuration files from the package into the theme folder. After running this for the first time, those files will not be automatically replaced by the scripts.

That means authors are free to edit them to fit their needs.

If the author makes an error in editing a theme file, it is safe to delete the erroneous file and run the compile script again. That will copy a fresh version of the theme file into the directory.

4 Check our documentation

There are several vignettes distributed with this package. Please review them.

1. "crmda": the package framework overview
2. "code_chunks": discusses display of code in LaTeX documents
3. "instructions-rnw2pdf-report-sweave"

5 Compile as usual, or with rnw2pdf

The skeleton file is provided in 2 formats, LyX and Rnw.

In either case, please understand that compiling is a two step process.

1. knitting: Run R to do the calculations in the R code chunks and write out a LaTeX file
2. compiling: Run a LaTeX program, such as pdflatex or xelatex to convert the LaTeX file to pdf. It is usually necessary to run the compiler two or more times, along with a separate bibliography program. If it is available, we suggest an aggregator named texi2pdf, which will handle this effort.

Edit the LyX file.

There are 4 methods, we hope one will suit your workflow.

1. Use the LyX editor. The file can be compiled to PDF in LyX, just like any other LyX file. LyX handles conversion from LyX to Rnw to tex to PDF. This has the same effect as using LyX from the command line. The following will create the PDF file using pdflatex as the final comiler:

   ```
   $ lyx -e pdf2 skeleton.lyx
   ```

   Because lyx uses a separate working directory for the compilation work, the project directory stays clean. None of the intermediate LaTeX files (*.log, *.log, *.bbl) will appear.

2. Open an R session and make sure the working directory is the same as the project writeup.

   ```
   rnw2pdf("skeleton.lyx", engine = "Sweave")
   ```

3. The shell script `rnw2pdf.sh` is provided in the same folder. It can be run in the shell as

   ```
   $ ./rnw2pdf.sh --engine='"Sweave"' skeleton.lyx
   ```

   In the discussion in the next sub section, we outline usage of additional arguments with rnw2pdf for the compilation of Rnw files. All of those arguments are equally applicable in this context.

4. In case you want to track the steps of compiling one by one, open the file in LyX. Use the pull down menu `File` → `Export` → `Sweave`. That will create a file named "skeleton.Rnw". This is the equivalent of the command line statement

   ```
   $ lyx -e sweave skeleton.lyx
   ```

   After that Rnw file is created, proceed as described in the next subsection.

   This two-step process is valuable for debugging. It makes it easier to spot trouble by focusing on the separate transitions.

Edit the Rnw file

The Rnw file we provide is produced by LyX, it is an intermediate step in the document production sequence. A two step compilation procedure is necessary. First, one must convert the "Rnw" file to "pdf" (with Sweave), and then the weaved tex file is compiled into pdf by pdflatex (or one of the other LaTeX compilers).

The work flow here will vary, depending on your experience and the editor you choose to use. Here are some possibilities:

1. You may have a "noweb" aware editor. Emacs, Rstudio, and others have menus that can initiate the work of weaving and rendering the document.

2. Open an R session and make sure the working directory is the same as the project writeup.

   ```
   rnw2pdf("skeleton.Rnw", engine = "Sweave")
   ```

Additional arguments can be used, mainly to control the verbosity of the output and the creation of subsidiary files. Our function, by default, will create and R file summary of the command chunks. This file is referred to as a "tangled" (if using knitr, it is referred to as a purled file).

3. The shell script `rnw2pdf.sh` is provided in the same folder. It can be run in the shell as

```
$ ./rnw2pdf.sh --engine='"Sweave"' skeleton.Rnw
```

The command script answers to all of the arguments followed by the R function `rnw2pdf`. The usage is nearly identical. Where the R function call would be

```
rnw2pdf("skeleton.Rnw", purl = FALSE, clean = FALSE, verbose = TRUE, keep_tex =
```

the shell command would be

```
$ ./rnw2pdf --purl=FALSE --clean=FALSE --keep_tex=TRUE --verbose=TRUE skeleton
```

The only difference in usage arises when a quoted string must be passed through. Suppose the files are in a subdirectory named "project". Inside the R code, the quoted string to specify the directory where the file resides (the working directory) would be like so:

```
rnw2pdf("skeleton.lyx", engine="Sweave" wd = "project")
```

the shell command would be

```
$ ./rnw2pdf --engine='"Sweave"' --wd='"project"' skeleton.Rnw
```

Note the single quotes that are protecting the double quotes.

4. Our shell script is not the only way to use command line tools to get this done. One can run shell commands such as:

```
$ R CMD Sweave skeleton.Rnw
```

That will create `skeleton.tex`, which we compile with

```
$ texi2pdf skeleton.tex
```

The major difference between running this and the script we provide is that our script will handle LyX files and it will, by default, will create a purled copy of the R code.

If you are editing these files in LyX, it is sufficient to simply compile as usual. That will handle the chore of converting a sequence of document types to arrive at PDF.

6 Code Chunk Check

What is the difference between a guide and a report? Simply put, a report document does not reveal source code and it should not distract the reader with code or "raw" output. A report document might just as well be typed by hand, if we could be sure all the numbers would be typed correctly and they could easily revised. In our report style, the author will not generally insert visible code chunks, so almost always the chunk will have the flag 'include=FALSE' or, if the chunk is included, the code will not be echoed, but perhaps a \LaTeX mark-up table or a figure may be placed into the document.

Our report documents ALMOST NEVER show "raw" R code to readers and very seldom will they display "raw" R output. Almost always, code chunks will have the flag "include=F" set and the document, when it reveals results, will, again, almost always, display a LaTeX formatted table that is placed inside a floating table or a figure that is placed inside a floating figure object.

It is a matter of style and author preference to decide how to include output within the report document. One approach is to use the chunk flags that directly display LaTeX output in the document. One must take special care to assure that the table is fully presentable. The alternative is to write the nearly presentable table on disk and then edit by hand to finalize the formats (usually we need to fix column and row names).

R Code Chunks

We use R R Core Team (2017) to do statistical analysis. We'd like to be as close as possible to the "reproducible document" idea. If R functions can produce perfectly presentable LaTeX output, then we use it.

On the other hand, one might write the output files and then manually insert them into the document. In our documents, we almost always have the global parameter 'split=TRUE', so that the code input and output chunks are saved in a directory we call 'tmpout'. Another LaTeX document can insert those chunks. We will demonstrate that here.

One document-weaving tip: save something for later.

In the usual "weave" documentation, a user is told to type in a chunk and then the output plops into the document "right there". I don't use that so often anymore, instead what I do very often is a trick I learned from Duncan Murdoch in the r-help email list.

Make sure that the document options are set with `split=T`. This works in LaTeX documents using Sweave or knitr to handle the code chunks. This causes each chunk's input and output to be saved to a separate file. This includes graphs and tables.

If I make a figure, the chunk will look like this

```
<<chunkfig, eval=F, include=F, echo=F, fig=T>>=
# R code for figure here
@
```

A file named "`tmpout/t-chunkfig.pdf`" will be created in the `tmpout` directory. The "`t-`" at the front of the file name is inserted because in the document setup, I chose the global prefix for output files as "t-". (Because documents can have different prefixes, it is possible then to have several R programs that output files into the same output folder. But I rarely do that because I don't want to get too confused about what file came from which program.) To insert that graphic in the document, I will write a LaTeX statement

```
\includegraphics[width=5in]{tmpout/t-chunkfig}
```

Note I don't put ".pdf" on the end of the file name, LaTeX finds the file named "t-chunkfig.pdf". I could use the LyX pull down Insert -> Graphics as well.

If the code makes a LaTeX table, I'll have this instead

```
<<chunktable, include=F, results=tex>>=
# R code here
@
```

That creates a file named "tmpout/t-chunktable.tex". Then put that into the document where you want with:

```
\input{tmpout/t-chunktable.tex}
```

Why do this? Why separate chunk output creation from inclusion in a document? The simple answer is that I might want to use that chunk in a different document. If I save a copy in the separate folder, then it is very convenient to come along later and make a separate slide show document displaying the same tables and/or figures. Or I might need to edit the chunk output before inserting it in the document.

The automatic "stick this output in where the chunk is placed" approach works great with lecture notes and guides because these things are easy to update and re-run.

Make Nice Looking Tables

The aim in R code is to generate "final" tables that are in LaTeX format and they are as close as possible to the final, presentable tables that a client can review in a report. We don't want the report reader to see ugly output:

```
       x                    y
Min.   :-2.3804   Min.   :-2.12355
1st Qu.:-0.5901   1st Qu.:-0.51290
Median : 0.4837   Median : 0.02596
Mean   : 0.2452   Mean   : 0.04523
3rd Qu.: 0.9004   3rd Qu.: 0.69839
Max.   : 2.4771   Max.   : 2.65579
```

In R, there are many (many!) packages and functions that can be used to generate acceptable LaTeX output. The bewildering diversity of these things is a problem. There are a host of packages that generate results that are nearly presentable, perhaps requiring only a minor adjustment of labels.

Summary statistics tables

xtable

Here is an example that uses rockchalk::summarize to gather summary statistics, which are then reformatted as a LaTeX table by xtable. Here's an xtable that displays most of the rows in the output from `rockchalk::summarizeNumerics`:

```
tab1 <- rockchalk::summarize(dat)
```

Table 1: In kable, I added the caption argument and got this unexpected float

	min	med	max	mean	sd	skewness	kurtosis	nobs	nmissing
x	-2.380358	0.4837183	2.477111	0.2451972	1.114731	-0.1423599	-0.6095874	100	0
y	-2.123550	0.0259646	2.655788	0.0452331	1.011241	0.1700460	-0.3880356	100	0

	x	y
min	-2.38	-2.12
med	0.48	0.03
max	2.48	2.66
mean	0.25	0.05
sd	1.11	1.01
skewness	-0.14	0.17
kurtosis	-0.61	-0.39
nobs	100	100
nmissing	0	0

To regulate the values in the rows, it is possible to choose explicitly, but the summarize function in rockchalk was revised to allow uses to more easily pin-point particular summary values.

It may be that people want the output rotated, so that the variable names are on the rows and the summary stats are in the columns. That's possible:

	min	med	max	mean	sd	skewness	kurtosis	nobs	nmissing
x	-2.38	0.48	2.48	0.25	1.11	-0.14	-0.61	100	0
y	-2.12	0.03	2.66	0.05	1.01	0.17	-0.39	100	0

The key issue is that the table is not perfectly ready for inclusion in a report. The row and column names might need beautification. That is why, realistically, it is generally easier to write those tables into tex files and revise them by hand, and then use LaTeX "\input{}" to include them in the document where appropriate.

knitr::kable

The following is a result from `kable` in the `knitr` package:

	min	med	max	mean	sd	skewness	kurtosis	nobs	nmissing
x	-2.380358	0.4837183	2.477111	0.2451972	1.114731	-0.1423599	-0.6095874	100	0
y	-2.123550	0.0259646	2.655788	0.0452331	1.011241	0.1700460	-0.3880356	100	0

The `kable` function is offered as a simple, usually robust table writer that will not deal with much "fancy" formatting. .

The `kable` function assumes that if the user specifies a title for the table, then it must mean that the user wants to have the table set as a floating table object. The same code that made the previous table is changed just slightly to produce a floating object. Look around in this document for a table named "In `kable`, I added the caption argument and got this unexpected float". (I found this frustrating because the `kable` function does not include documentation for insertion of a label that can be used for cross referencing.)

I'd rather not have kable insert the table float for me, I'd rather do it manually, as we can see in Table 2

Table 2: kable output in a float I created manually

	min	med	max	mean	sd	skewness	kurtosis	nobs	nmissing
x	-2.380358	0.4837183	2.477111	0.2451972	1.114731	-0.1423599	-0.6095874	100	0
y	-2.123550	0.0259646	2.655788	0.0452331	1.011241	0.1700460	-0.3880356	100	0

Table 3: A Regression from `outreg`

	First Model	
	Estimate	(S.E.)
(Intercept)	0.022	(0.104)
Excellent Predictor	0.095	(0.091)
N	100	
RMSE	1.011	
R^2	0.011	

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

Regression output

I'll illustrate output from the `outreg` function in the `rockchalk` package. This table may not be perfect by APA standards, but it is certainly good enough for our reports. We estimate a regression and save the LaTeX markup in a file named "tmpout/t-outreg1.tex". (This approach is necessary in this document because it is a vignette in the stationery package and the split=TRUE flag is not allowed in vignettes for packages on CRAN).

```
library(rockchalk)
m1 <- lm(y ~ x, data = dat)
vl <- c("x" = "Excellent Predictor")
or <- outreg(list("First Model" = m1), varLabels = vl, tight =
   FALSE)
cat(or, file="tmpout/t-outreg1.tex")
```

After the file is created, we then incorporate it in the usual way, embedding it in a floating object, Table 3.

There are many other regression-table-making functions available today. I made some lecture notes about it for the R summer workshops that we offer at KU (http://pj.freefaculty.org/guides/Rcourse/regression-tables-1).

Structural equation models

In the good looking table department, we also need to display structural equation models. This has been a long term objective in CRMDA and it is, for the most part, a solved problem.

In the `kutils` package, we made a function `semTable` that is intended to help. Please see Table 4.

```
library(kutils)
require(lavaan)
```

Table 4: A Confirmatory Factor Analysis Table

	Model			
	Estimate	Std. Err.	z	p
		Factor Loadings		
visual				
x1	0.90	0.08	11.13	.000
x2	0.50	0.08	6.43	.000
x3	0.66	0.07	8.82	.000
textual				
x4	0.99	0.06	17.47	.000
x5	1.10	0.06	17.58	.000
x6	0.92	0.05	17.08	.000
speed				
x7	0.62	0.07	8.90	.000
x8	0.73	0.07	11.09	.000
x9	0.67	0.07	10.30	.000
		Latent Variances		
visual	1.00[+]			
textual	1.00[+]			
speed	1.00[+]			
		Fit Indices		
RMSEA	0.09			

[+]Fixed parameter

7 Session Information

Leave the code chunks below. But the visible words and section name should be removed. Session Information is usually not written into a report, but an output file is created by the following pieces.

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

References

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