knitr in a knutshell (http://kbroman.org/knitr_knutshell)

Knitr with R Markdown

The simplest way to write a quick report, mixing in a bit of R, is to use R Markdown (http://rmarkdown.rstudio.com), a variant of Markdown (http://daringfireball.net/projects/markdown/) developed by the folks at Rstudio (http://www.rstudio.com).

You should first read the page about Markdown (markdown.html).

R Markdown

R Markdown (http://rmarkdown.rstudio.com) is a variant of Markdown (http://daringfireball.net/projects/markdown/) that has embedded R (http://www.r-project.org) code chunks, to be used with knitr (http://yihui.name/knitr/) to make it easy to create reproducible web-based reports. The Markdown syntax has some enhancements (see the R Markdown page (http://rmarkdown.rstudio.com)); for example, you can include LaTeX equations (see Equations in R Markdown (http://www.rstudio.com/ide/docs/authoring/using_markdown_equations)).

Here's an example R Markdown document (../assets/knitr_example.Rmd), and the html document it produces (../assets/knitr_example.html).

Code chunks

The key thing for us to focus on are the code chunks, which look like this:

````
```{r simulate_data}
x <- rnorm(100)
y <- 2*x + rnorm(100)
```
````

In the midst of an otherwise plain Markdown document, you'll have a bit of R code that is initiated by a line like this:

````
```{r chunk_name}
````

After the code, there'll be a line with just three backticks.

````
```
````

It's usually best to give each code chunk a name, like `simulate_data` and `chunk_name` above. The name is optional; if included, each code chunk needs a distinct name. The advantage of giving each chunk a name is that it will be easier to understand where to look for errors, should they occur. Also, any figures that are created will be given names based on the name of the code chunk that produced them.

When you process the R Markdown document with knitr, each of the code chunks will be evaluated, and then the code and/or output will be inserted (unless you suppress one or both with *chunk options*, described below). If the code produces a figure, that figure will be inserted.

An R Markdown document will have often have *many* code chunks. They are evaluated in order, in a single R session, and the state of the various variables in one code chunk are preserved in future chunks. It's as if you'd pulled out all of the R code as a single file (and you can do that, using the `purl` command in knitr) and then source (http://stat.ethz.ch/R-manual/R-devel/library/base/html/source.html)d it into R.

Chunk options

The initial line in a code chunk may include various options. For example, `echo=FALSE` indicates that the code will not be shown in the final document (though any results/output would still be displayed).

````
```{r chunk_name, echo=FALSE}
x <- rnorm(100)
y <- 2*x + rnorm(100)
cor(x, y)
```
````

You use `results="hide"` to hide the results/output (but here the code would still be displayed).

````
```{r chunk_name, results="hide"}
x <- rnorm(100)
y <- 2*x + rnorm(100)
cor(x, y)
```
````

You use `include=FALSE` to have the chunk *evaluated*, but neither the code nor its output displayed.

````
```{r chunk_name, include=FALSE}
x <- rnorm(100)
y <- 2*x + rnorm(100)
cor(x, y)
```
````

If I'm writing a report for a collaborator, I'll often use `include=FALSE` to suppress all of the code and largely just include figures.

For figures, you'll want to use options like `fig.width` and `fig.height`. For example:

````
```{r scatterplot, fig.width=8, fig.height=6}
plot(x,y)
```
````

Note that if `include=FALSE`, all of the code, results, and figures will be suppressed. If `include=TRUE` and `results="hide"`, the results will be hidden but figures will still be shown. To hide the figures, use `fig.show="hide"`.

There are lots of different possible "chunk options" (http://yihui.name/knitr/options#chunk_options). Each must be real R code, as R will be used to evaluate them. So `results=hide` is wrong; you need `results="hide"`.

Global chunk options

You may be inclined to use largely the same set of chunk options throughout a document. But it would be a pain to retype those options in every chunk. Thus, you want to set some global chunk options at the top of your document.

For example, I might use `include=FALSE` or at least `echo=FALSE` globally for a report to a scientific collaborator who wouldn't want to see all of the code. And I might want something like `fig.width=12` and `fig.height=6` if I generally want those sizes for my figures.

I'd set such options by having an initial code chunk like this:

```
```{r global_options, include=FALSE}
knitr::opts_chunk$set(fig.width=12, fig.height=8, fig.path='Figs/',
                      echo=FALSE, warning=FALSE, message=FALSE)
```
```

I snuck a few additional options in there: `warning=FALSE` and `message=FALSE` suppress any R warnings or messages from being included in the final document, and `fig.path='Figs/'` makes it so the figure files get placed in the `Figs` subdirectory. (By default, they are not saved at all.)

Note: the ending slash in `Figs/` is important. If you used `fig.path='Figs'` then the figures would go in the main directory but with `Figs` as the initial part of their names.

The global chunk options become the defaults for the rest of the document. Then if you want a particular chunk to have a different behavior, for example, to have a different figure height, you'd specify a different option within that chunk. For example:

```
```{r a_taller_figure, fig.height=32}
par(mfrow=c(8,2))
for(i in 1:16)
  plot(x[,i], y[,i])
```
```

In a report to a collaborator, I might use `include=FALSE, echo=FALSE` as a global option, and then use `include=TRUE` for the chunks that produce figures. Then the code would be suppressed throughout, and any output would be suppressed except in the figure chunks (where I used `include=TRUE`), which would produce just the figures.

Technical aside: In setting the global chunk options with `opts_chunk$set()`, you'll need to use `knitr::` (or to have first loaded the knitr package with `library(knitr)`). As we'll discuss below, we'll use the rmarkdown package (https://github.com/rstudio/rmarkdown) to process the document, first with knitr and then with pandoc (http://pandoc.org), and `rmarkdown::render()` will use `knitr::knit()` but won't load the knitr package.

Package options

In addition to the chunk options, there are also package options (http://yihui.name/knitr/options#package_options), set with something like:

```
```{r package_options, include=FALSE}
knitr::opts_knit$set(progress = TRUE, verbose = TRUE)
```
```

I was confused about this at first: I'd use `opts_knit$set` when I really wanted `opts_chunk$set`. knitr includes *a lot* of options; if you're getting fancy you may need these package options, but initially you'll just be using the chunk options and, particularly, the global chunk options defined via `opts_chunk$set`. So mostly **ignore** `opts_knit$set()` in favor of `opts_chunk$set()`.

In-line code

A key motivation for knitr is reproducible research (http://en.wikipedia.org/wiki/Reproducibility#Reproducible_research): that our results are accompanied by the data and code needed to produce them.

Thus, your report should never explicitly include numbers that are derived from the data. Don't write "There are 168 individuals." Rather, insert a bit of code that, when evaluated, gives the number of individuals.

That's the point of the in-line code. You'd write something like this:

```
There are `r nrow(my_data)` individuals.
```

Another example:

```
The estimated correlation between x and y was `r cor(x,y)`.
```

In R Markdown, in-line code is indicated with `` `r `` and `` ` ``. The bit of R code between them is evaluated and the result inserted.

An important point: you need to be sure that these in-line bits of code aren't split across lines in your document. Othewise you'll just see the raw code and not the result that you want.

YAML header

Insert, at the top of your R Markdown document, a bit of text like the following:

```
---
title: "An example Knitr/R Markdown document"
author: "Karl Broman"
date: "3 Feb 2015"
output: html_document
---
```

The final document will then contain a nicely formated title, along with the author name and date. You can include hyperlinks in there:

```
author: "[Karl Broman](http://kbroman.org)"
```

and even R code:

```
date: "`r Sys.Date()`"
```

This is called the YAML (http://www.yaml.org) header. YAML is a simple text-based format for specifying data, sort of like JSON (http://www.json.org) but more human-readable.

You can leave off the author and date if you want; you can leave off the title, too. Actually, you don't need to include any of this. But `output: html_document` tells the rmarkdown package (https://github.com/rstudio/rmarkdown) to convert the document to html. That's the default, but you could also use `output: pdf_document` or even `output: word_document`, in which case your document will be converted to a PDF or Word `.docx` file, respectively.

Rounding

I'm very particular about the rounding of results, and you should be too. If I've estimated a correlation coefficient with 1000 data points, I don't want to see `0.9032738`. I want `0.90`.

You could use the R function `round`, like this: `` `r round(cor(x,y), 2)` `` But that would produce `0.9` instead of `0.90`.

One solution is to use the `sprintf` function, like so: `` `r sprintf("%.2f", cor(x,y))` ``. That's perfectly reasonable, right? Well, it is if you're a C programmer.

But a problem arises if the value is `-0.001`. `` `r sprintf("%.2f", -0.001)` `` will produce `-0.00`. I don't like that, nor does Hilary (https://twitter.com/hspter/status/314858331598626816).

My solution to this problem is the `myround` (https://github.com/kbroman/broman/blob/master/R/myround.R) function in my R/broman (https://github.com/kbroman/broman) package.

At the start of my R Markdown document, I'd include:

````
```{r load_packages, include=FALSE}
library(broman)
```
````

And then later I could write `` `r myround(cor(x,y), 2)` `` and it would give `0.90` or `0.00` in the way that I want.

Converting R Markdown to html

Via RStudio

If you use RStudio (http://www.rstudio.com), the simplest way to convert an R Markdown document to html is to open the document within RStudio. (And really, you probably want to *create* the document in RStudio: click File → New File → R Markdown.) When you open an R Markdown document in RStudio, you'll see a "Knit HTML" button just above the document. (It's a particularly cute little button, with a ball of yarn and a knitting needle.) Click that, and another window will open, and you'll see knitr in action, executing each code chunk and each bit of in-line code, to compile the R Markdown to a Markdown document. This will then be converted to html, with a preview of the result. (The resulting `.html` file will be placed in the same directory as your `.Rmd` file.) You can click "Open in browser" to open the document in your web browser, or "Publish" to publish the document to the web (where it will be viewable by *anyone*).

Another a nice feature in RStudio: when you open an R Markdown document, you'll see a little question mark button, with links to "Using R Markdown (http://rmarkdown.rstudio.com)" and to a Markdown Quick Reference. convenient "Markdown Quick Reference" document: a cheat-sheet on the Markdown syntax. Like @StrictlyStat (https://twitter.com/StrictlyStat/status/423178160968970240), I seem to visit the Markdown (http://daringfireball.net/projects/markdown) website almost every time I'm writing a Markdown document. If I used RStudio, I'd have easier access to this information.

RStudio is especially useful when you're first learning KnitR and R Markdown, as it's easy to create and view the corresponding html file, and you have access to that Markdown Quick Reference.

Via the command line (or GNU make)

To process an R Markdown document, you need the rmarkdown package (https://github.com/rstudio/rmarkdown) (which in turn will make use of the knitr package (http://cran.r-project.org/web/packages/knitr/) plus a bunch of other packages), as well as pandoc (http://pandoc.org).

To install the rmarkdown package, use `install.packages(rmarkdown)`.

The simplest way to install pandoc is to just install the RStudio Desktop software (http://www.rstudio.com/products/rstudio/#Desk), which includes pandoc, and then include pandoc without your `PATH`. On a Mac, you'd use:

```
export PATH=$PATH:/Applications/RStudio.app/Contents/MacOS/pandoc
```

In Windows, you'd include `"c:\Program Files\RStudio\bin\pandoc"` in your `Path` system environment variable. (For example, see this page (http://www.howtogeek.com/118594/how-to-edit-your-system-path-for-easy-command-line-access/), though it's a bit ad-heavy.)

To convert your Markdown document to HTML, you'd then use

```
R -e "rmarkdown::render('knitr_example.Rmd')"
```

(Note that in Windows, it's important to use double-quotes on the outside and single-quotes inside, rather than the other way around.)

Rather than actually type that line, I include it within a GNU make (http://www.gnu.org/software/make) file, like this one (https://github.com/kbroman/knitr_knutshell/blob/gh-pages/assets/Makefile). (Also see my minimal make (http://kbroman.org/minimal_make/) tutorial.)

Up next

At this point, I'd recommend going off and playing with R Markdown for a while. Write your next report with R Markdown, even if it takes you a bit longer. Write it using RStudio (http://www.rstudio.com), where the knitting process is easy and you have easy access to that "Markdown Quick Reference".

Then, read a bit about figures and tables (figs_tables.html), my comments on reproducibility (reproducible.html), and perhaps about Knitr with AsciiDoc (asciidoc.html) or Knitr with LaTeX (latex.html).
