Producing Reports With knitr

Data analysis reports

- Data analysts write a lot of reports, describing their analyses and results.
- Beginners often write R scripts & email those scripts & graphs around.
- This can be problematic and cumbersome
- Writing reports in Word or LaTeX can incoporate analysis and ouput graphs together in one doc.
- But tweaking the report and parameters can be difficult:
  - have to fix or change code, run it, get output and copy over to document
  - will have to do this multiple times as you tweak, gets confusing
- Having the code, analysis and output in one document that is runable helps
- It makes changing formatting or analysis much easier and is easier to re-run with changes (no copying and pasting)

Literate programming

- Ideally analysis reports are *reproducible* documents
- If an error is discovered, or if some additional subjects are added to the data, you can just re-compile the report and get the new or corrected results.
- The key R package is `knitr`.
  - It lets you to create a document that is a mixture of text and chunks of code.
- When the document is processed by `knitr`, chunks of code will be executed, and graphs or other results inserted into the final document.
- This sort of idea has been called **literate programming**.
- `knitr` lets you to mix basically any sort of text with code from different programming languages, but we recommend that you use `R Markdown`, which mixes Markdown with R.
- Markdown is a light-weight mark-up language for creating web pages (the name is a play on the word markup where you annotate a document with info to display text, e.g. html tags)

Creating an R Markdown file

- Within RStudio, click File New File R Markdown
- You can stick with the default (HTML output), but add a title and yourself as the author.

Basic components of R Markdown

- The initial chunk of text (header) contains instructions for R to specify what kind of document will be created, and the options chosen.
- You can use the header to give your document a title, author, date, and tell it that you're going to want to produce html output (in other words, a web page).

```
---
title: "Initial R Markdown document"
author: "Tim DEnnis"
date: "May 1, 2020"
output: html_document
---
```

- Fields are optional - you can delete the ones you don't want

- Double quotes optional needed if you include colon
- RStudio creates the document with some example text to get you started.
- Note below that there are chunks like

```
summary(cars)
```

```
##      speed           dist
##  Min.   : 4.0   Min.   :  2.00
##  1st Qu.:12.0   1st Qu.: 26.00
##  Median :15.0   Median : 36.00
##  Mean   :15.4   Mean   : 42.98
##  3rd Qu.:19.0   3rd Qu.: 56.00
##  Max.   :25.0   Max.   :120.00
```

- This is a code chunk that will be run by `knitr` and output the results inline.

Markdown

- A system for writing web pages by marking up the text with symbols like asterick, pound, underscore
- The marked-up text gets *converted* to html, replacing the marks with the proper html code.
- Let's see how this works:

1. delete all of the dummy text that was generated when we created the rmarkdown document and write some markdown.
2. make things **bold** using two asterisks around what you want to bold, like this: `**bold**`, and you make things *italics* by using underscores, like this: `_italics_`.
3. You can make a bulleted list by writing a list with hyphens or asterisks, like this:

```
* bold with double-asterisks
* italics with underscores
* code-type font with backticks
```

or like this:

```
- bold with double-asterisks
- italics with underscores
- code-type font with backticks
```

- Each will appear as:
- bold with double-asterisks
- italics with underscores
- code-type font with backticks
- You can use whatever method you prefer, but *be consistent*. This maintains the readability of your code.
- You can make a **numbered list** by just using numbers.
- You can even use the same number over and over if you want:

```
1. bold with double-asterisks
1. italics with underscores
1. code-type font with backticks
```

This will appear as:

1. bold with double-asterisks
2. italics with underscores

3. code-type font with backticks

- Make section headers of different sizes by putting `#` symbols in front of the text, like so:

```
# Title
## Main section
### Sub-section
#### Sub-sub section
```

- *compile* the R Markdown document to html by clicking Knit button in the upper-left.
- You might have to install some packages, hit ok and let RStudio install packages.
- Ok now let's try challenge 1

Challenge 1

Create a new R Markdown document. Delete all of the R code chunks and write a bit of Markdown (some sections, some italicized text, and an itemized list).

Convert the document to a webpage. > ## Solution to Challenge 1 > > In RStudio, select File > New file > R Markdown... > > Delete the placeholder text and add the following: > > `> # Introduction > > ## Background on Data > > This report uses the *gapminder* dataset, which has columns that include: > > * country > * continent > * year > * lifeExp > * pop > * gdpPercap > > ## Background on Methods > > >` Then click the 'Knit' button on the toolbar to generate an html document (webpage). {: .solution} {: .challenge}

A bit more Markdown

- You can make a hyperlink like this: `[text to show](http://the-web-page.com)`.
- You can include an image file like this: `![caption](http://url/for/file)`
- You can do subscripts (e.g., F_2) with `F~2~` and superscripts (e.g., F^2) with `F^2^`.
- If you know how to write equations in LaTeX, you can use `$ $` and `$$ $$` to insert math equations, like `$E = mc^2$` and

```
$$y = \mu + \sum_{i=1}^p \beta_i x_i + \epsilon$$
```

- You can review Markdown syntax by navigating to the "Markdown Quick Reference" under the "Help" field in the toolbar at the top of RStudio.

R code chunks

The real power of Markdown comes from mixing markdown with chunks of code. This is R Markdown. When processed, the R code will be executed; if they produce figures, the figures will be inserted in the final document.

- The main code chunks look like this:

```
gapminder <- read_csv("gapminder.csv")
```

- That is, you place a chunk of R code between "'{r chunk_name} and "'.
- You should give each chunk a unique name, as they will help you to fix errors and, if any graphs are produced, the file names are based on the name of the code chunk that produced them.

Challenge 2

Add code chunks to:

- Load the ggplot2 package
- Read the gapminder data
- Create a plot

Solution to Challenge 2

{: .solution} {: .challenge}

How things get compiled

- When you press the "Knit" button, the R Markdown document is processed by `knitr` and a plain Markdown document is produced (as well as, potentially, a set of figure files): the R code is executed and replaced by both the input and the output; if figures are produced, links to those figures are included.
- The Markdown and figure documents are then processed by the tool `pandoc`, which converts the Markdown file into an html file, with the figures embedded.

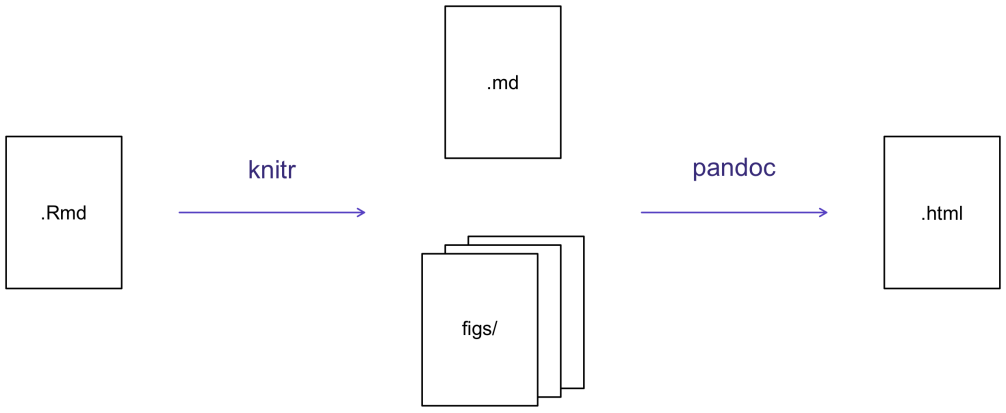

Figure 1: How R markdown gets compiled

Chunk options

*Options that affect how our code chunks are treated. Examples:

- Use `echo=FALSE` to avoid having the code itself shown.
- Use `results="hide"` to avoid having any results printed.
- Use `eval=FALSE` to have the code shown but not evaluated.
- Use `warning=FALSE` and `message=FALSE` to hide any warnings or messages produced.
- Use `fig.height` and `fig.width` to control the size of the figures produced (in inches).

Write:

- Often there will be particular options that you'll want to use repeatedly; for this, you can set *global* chunk options, like so:
- `fig.path` option defines where the figures will be saved.
- `/` here is really important; without it, the figures would be saved in the standard place but just with names that begin with `Figs`.

- If you have multiple R Markdown files in a common directory, you might want to use `fig.path` to define separate prefixes for the figure file names, like `fig.path="Figs/cleaning-"` and `fig.path="Figs/analysis-"`.

Challenge 3

Use chunk options to control the size of a figure and to hide the code.

Solution to Challenge 3

{: .solution} {: .challenge}

- You can review all of the `R` chunk options by navigating to the "R Markdown Cheat Sheet" under the "Cheatsheets" section of the "Help" field in the toolbar at the top of RStudio.

Inline R code

- You can make *every* number in your report reproducible.
- Use ‘r and ‘ for an in-line code chunk, like so: ‘r round(some_value, 2)‘.
- The code will be executed and replaced with the *value* of the result.
- Don't let these in-line chunks get split across lines.
- Perhaps precede the paragraph with a larger code chunk that does calculations and defines variables, with `include=FALSE` for that larger chunk (which is the same as `echo=FALSE` and `results="hide"`).
- Rounding can produce differences in output in such situations. You may want `2.0`, but `round(2.03, 1)` will give just `2`.
- The `myround` function in the R/broman package handles this.

Challenge 4

Try out a bit of in-line R code.

Solution to Challenge 4

Here's some inline code to determine that 2 + 2 = `` `r 2+2` ``.

{: .solution} {: .challenge}

Other output options

- You can also convert R Markdown to a PDF or a Word document.
- Click the little triangle next to the "Knit" button to get a drop-down menu.
- Or you could put `pdf_document` or `word_document` in the initial header of the file.

Tip: Creating PDF documents

Creating .pdf documents may require installation of some extra software. If required this is detailed in an error message.

- TeX installers for Windows.
- TeX installers for macOS. {: .callout}

Resources

- Knitr in a knutshell tutorial
- Dynamic Documents with R and knitr (book)
- R Markdown documentation
- R Markdown cheat sheet
- Getting started with R Markdown
- R Markdown: The Definitive Guide (book by Rstudio team)
- Reproducible Reporting
- The Ecosystem of R Markdown
- Introducing Bookdown
- R Markdown Tutorial for RLadies LA