STAT-S 432 SP2019: Lecture 1

DJM, Revised: NAK

January 10, 2019

Mark Up, Markdown

- You are probably used to word processing programs, like Microsoft Word, which employ the "what you see is what you get" (WYSIWYG) principle:
- you want italics, and, lo, they're in italics
- you want some to be in a bigger, different font and you just select the font, and so on.
- This works well enough for basic purposes but is not a viable basis for a system of text formatting
- because it depends on a particular program (a) knowing what you mean and (b) implementing it well.
- The essential idea in a **mark-up language** is that it consists of ordinary text, *plus* signs which indicate how to change the formatting or meaning of the text.
- Some mark-up languages, like HTML (Hyper-Text Markup Language) use very obtrusive markup; others, like the language called **Markdown**, are more subtle.

Example

For instance, the last few sentences in Markdown look like this:

```
* The essential idea in a
__mark-up language__ is that it consists of ordinary text, _plus_ signs which
indicate how to change the formatting or meaning of the text.

* Some mark-up
languages, like HTML (Hyper-Text Markup Language) use very obtrusive markup;
others, like the language called __Markdown__, are more subtle.
```

Rendering

- Every mark-up language needs to be **rendered** somehow into a format which actually includes the fancy formatting, images, mathematics, computer code, etc., etc., specified by the mark-up.
- For HTML, the rendering program is called a "web browswer".
- Most computers which know how to work with Markdown at all know how to render it as
  - HTML (which you can then view in a browser),
  - PDF (which you can then view in Acrobat or the like),
  - Word (which you can then view in the abomination of Redmond).

Advantages

- portability across machines
- no proprietary software (I'm looking at you Bill)
- stability

R Markdown is, in particular, both "free as in beer" (you will never pay a dollar for software to use it) and "free as in speech" (the specification is completely open to all to inspect).

Even if you are completely OK with making obeisance to the Abomination of Redmond every time you want to read your own words, the sheer stability of mark-up languages makes them superior for scientific documents.

Rendering and Editing

To write R Markdown, you will need a text editor, a program which lets you read and write plain text files. You will also need R, and the package `rmarkdown` (and all the packages it depends on).

- Most computers come with a text editor (TextEdit on the Mac, Notepad on Windows machines, etc.).
- There are also lots of higher-powered text editors; I use Emacs, but I admit it has a harsh learning curve.
- You *could* use Word (or some other WYSIWYG word processor), taking care to always save your document in plain text format. I do not recommend this.
- R Studio comes with a built-in text editor, which knows about, and has lots of tools for, working with R Markdown documents.

If this is your first time using a text editor for something serious, I recommend using R Studio. (That's what I'm using to write this.)

Rendering in R Studio

Assuming you have the document you're working on open in the text editor, click the button that says "knit".

Basic Formatting in R Markdown

For the most part, text is just text. One advantage of R Markdown is that the vast majority of your document will be stuff you just type as you ordinarily would.

Paragraph Breaks and Forced Line Breaks

To insert a break between paragraphs, include a single completely blank line.

To force a line break, put *two*
blank spaces at the end of a line.

Headers

- The character `#` at the beginning of a line means that the rest of the line is interpreted as a section header.
- The number of `#`s at the beginning of the line indicates whether it is treated as a section, sub-section, sub-sub-section, etc. of the document.

Italics, Boldface

- Text to be *italicized* goes inside *a single set of underscores* or *asterisks*.
- Text to be **boldfaced** goes inside a **double set of underscores** or **asterisks**.

Quotations

Set-off quoted paragraphs are indicated by an initial `>`:

> In fact, all epistemological value of the theory of probability is based on this: that large-scale random phenomena in their collective action create strict, nonrandom regularity. [Gnedenko and Kolmogorov, *Limit Distributions for Sums of Independent Random Variables*, p. 1]

Bullet Lists

- This is a list marked where items are marked with bullet points.
- Each item in the list should start with a `*` (asterisk) character.
- Each item should also be on a new line.
  - Indent lines and begin them with `+` or `-` for sub-bullets.
  - Sub-sub-bullet aren't really a thing in R Markdown.

Numbered lists

1. Lines which begin with a numeral (0–9), followed by a period, will usually be interpreted as items in a numbered list.
2. R Markdown handles the numbering in what it renders automatically.
3. This can be handy when you lose count or don't update the numbers yourself when editing. (Look carefully at the .Rmd file for this item.)
   a. Sub-lists of numbered lists, with letters for sub-items, are a thing.
   b. They are however a fragile thing, which you'd better not push too hard.

Title, Author, Date, Output Format

You can specify things like title, author and date in the **header** of your R Markdown file. This goes at the very beginning of the file, preceded and followed by lines containing three dashes. Thus the beginning of this file looks like so:

```
---
title: Using R Markdown for Class Reports
author: DJM
date: "06 January, 2019"
---
```

You can also use the header to tell R Markdown whether you want it to render to HTML (the default), PDF, or something else. To have this turned into PDF, for instance, I'd write

```
---
title: Using R Markdown for Class Reports
author: DJM
date: "06 January, 2019"
output: pdf_document
---
```

- This header information is called `YAML` if you need the Google.

Figure 1: A local image

Hyperlinks

- Hyperlinks anchored by URLs are easy: just type the URL, as, e.g., http://mypage.iu.edu/~dajmcdon/ to get to my website
- Hyperlinks anchored to text have the anchor in square brackets, then the link in parentheses.
- A very useful link is the R markdown cheatsheet.

Images

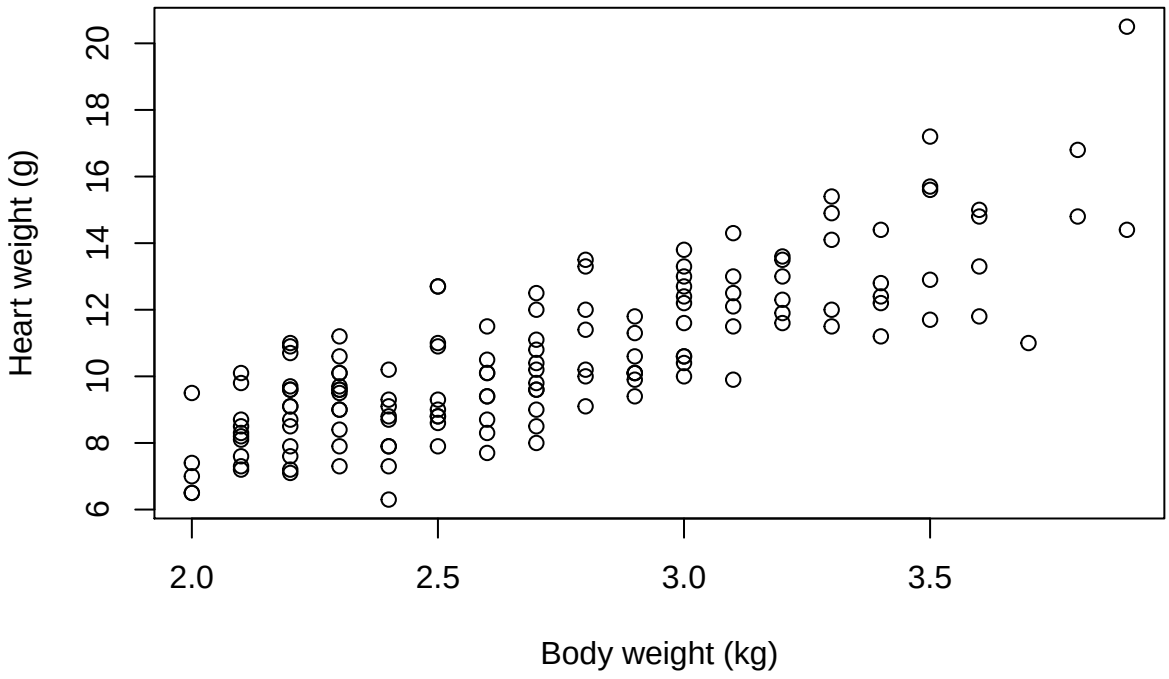
Images begin with an exclamation mark, then the text to use if the image can't be displayed, then either the file address of the image (in the same directory as your document) or a URL. This is an example using an image stored locally.

Including Code

Code Chunks and Their Results

A code **chunk** is simply an off-set piece of code by itself. It is preceded by ```` ```{r} ```` on a line by itself, and ended by a line which just says ```` ``` ````. The code itself goes in between. Here, for instance, is some code which loads a data set from a library, and makes a scatter plot.

```r
library(MASS)
data(cats) #Default dataset
plot(Hwt ~ Bwt, data=cats, xlab="Body weight (kg)", ylab="Heart weight (g)")
```



Inline Code

- This is code not set off on a line by itself, but beginning with `` `r `` and ending with `` ` ``. Using inline code is how this document knows that the `cats` data set contains 144 rows, and that the median weight of the female cats' hearts was 9.1 grams.
- Notice that inline code does *not* display the commands run, just their output.

Seen But Not Heard

- Code chunks (but not inline code) can take a lot of **options** which modify how they are run, and how they appear in the document.
- These options go after the initial `r` and before the closing `}` that announces the start of a code chunk. One of the most common options turns off printing out the code, but leaves the results alone: `` ```{r, echo=FALSE} ``
- Another runs the code, but includes neither the text of the code nor its output. `` ```{r, include=FALSE} ``
- This might seem pointless, but it can be useful for code chunks which do set-up like loading data files, or initial model estimates, etc.
- Another option prints the code in the document, but does not run it: `` ```{r, eval=FALSE} `` This is useful if you want to talk about the (nicely formatted) code.
- You can give chunks names immediately after their opening, like `` ```{r clevername} ``. This name is then used for the images (or other files) that are generated when the document is rendered.

Tables

The default print-out of matrices, tables, etc. from R Markdown is frankly ugly. The `knitr` package contains a very basic command, `kable`, which will format an array or data frame more nicely for display.

Compare:

```
coefficients(summary(lm(Hwt ~ Bwt, data=cats)))
```

```
##               Estimate Std. Error    t value     Pr(>|t|)
## (Intercept) -0.3566624  0.6922770 -0.5152019 6.072131e-01
## Bwt          4.0340627  0.2502615 16.1193908 6.969045e-34
```

with

```
library(knitr) # Only need this the first time!
kable(coefficients(summary(lm(Hwt ~ Bwt, data=cats))))
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-0.3566624	0.6922770	-0.5152019	0.6072131
Bwt	4.0340627	0.2502615	16.1193908	0.0000000

— Of course, R's defaults print out a crazy number of decimal places. So use

```
kable(coefficients(summary(lm(Hwt ~ Bwt, data=cats))),digits=3)
```

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-0.357	0.692	-0.515	0.607
Bwt	4.034	0.250	16.119	0.000

"Caching" Code Chunks (Re-Running Only When Changed)

- By default, R Markdown will re-run all of your code every time you render your document.
- If some of your code is slow, this can add up to a lot of time. You can, however, ask R Markdown to keep track of whether a chunk of code has changed, and only re-run it if it has.
- This is called **caching** the chunk.

```
lm(Hwt ~ Bwt, data=cats)
```

```
##
## Call:
## lm(formula = Hwt ~ Bwt, data = cats)
##
## Coefficients:
## (Intercept)          Bwt
##     -0.3567       4.0341
```

Setting Defaults for All Chunks

You can tell R to set some defaults to apply to all chunks where you don't specifically over-ride them. Here are the ones I generally use:

```
# Need the knitr package to set chunk options
library(knitr)

# Set knitr options for knitting code into the report:
# - Don't print out code (echo)
```

```
# - Save results so that code blocks aren't re-run unless code changes (cache),
# _or_ a relevant earlier code block changed (autodep), but don't re-run if the
# only thing that changed was the comments (cache.comments)
# - Don't clutter R output with messages or warnings (message, warning)
  # This _will_ leave error messages showing up in the knitted report
opts_chunk$set(echo=FALSE,
               cache=TRUE, autodep=TRUE, cache.comments=FALSE,
               message=FALSE, warning=FALSE)
```

- This sets some additional options beyond the ones I've discussed, like not re-running a chunk if only the comments have changed (`cache.comments = FALSE`), and leaving out messages and warnings.
- You can over-ride these defaults by setting options for individual chunks.

Math in R Markdown

Point out the Cheat sheet.

- R Markdown lets you write complex mathematical formulas and derivations, and have them displayed *very* nicely.
- Like code, the math can either be inline or set off (**displays**).
- Inline math is marked off with a pair of dollar signs (`$`), as `$\pi r^2$` πr^2 or `$e^{i\pi}$` $e^{i\pi}$.
- Mathematical displays are marked off with `\[` and `\]`, as in

$$e^{i\pi} = -1$$

Elements of Math Mode

- Lots of details in the **rmarkdownDetails** file in the lectures repo.

Putting It All Together: Writing Your Report in R Markdown

- You have installed the `rmarkdown` package and all its dependencies.
- This is automatic with Rstudio
- You open it up to a new document.
  - You give it a title, an author, and a date.
- You use headers to divide it into appropriate, titled sections, and possibly sub-sections.
  - One common pattern: "Introduction", "Data and Research Questions", "Analysis", "Results", "Conclusion".
  - Another common pattern: "Problem 1", "Problem 2", . . . , "Extra Credit".
- You write text.
- When you need it, you insert math into the text, or even whole mathematical displays.
- When you need it, you insert code into your document.
  - The code runs (as needed) when you render the document.
  - Figures, tables, and other output are automatically inserted into the document, and track changes in your code.
- Every so often, try to render your document.
  - When you (think you) have finished a section is a good time to do so.
  - Another good time is once you've made any non-trivial change to the code or the text.
- Either your document rendered successfully or it didn't.
  - If it did, and you like the results, congratulate yourself and cheerfully go on to your next task.

– If it rendered but you don't like the results, think about why and try to fix it.
– If it didn't render, R will tell you where it gave up, so try to debug from around there.