Advanced Data Analysis in R

Advanced `rmarkdown`

Michael DeWitt

2018-02-09 (Updated 2019-02-10)

Why is `rmarkdown` important?

It’s all about communication and documentation!

1. We used have notebooks to document our work:

2. Our reports change change with our data

Reproducibility

- Tying our analysis to our output documentation
- No more of this https://youtu.be/s3JldKoA0zw

Sold? So how do I do it?

- Rmarkdown integrates R and the Markdown language into a single method
- Rmarkdown documents end in `.Rmd` extension
- Can be created from within the R Studio Integrated Development Environment (IDE)

Some technical details...

- `rmarkdown` (and `bookdown`) compile their outputs to pandoc
- Depending on the output specified
    - LaTeXfor pdf style outputs (and beamer)
    - html
    - html + javascript
    - epub
    - ...
- Specific commands can be issued depending on the output used (LaTeXand/or html tags)

Building Your Documents

Three Components to an Rmarkdown Documents

1. yml header than contains metadata and build instructions
2. Markdown mark-up conventions
3. Code chunks with language and output instructions

Yet Another Markup Language

Parts of An R Markdown Documents

`yml` header instructs to `pandoc` engine how to build the documents

```
---
title:
subtitle:
author:
abstract:
date:
output:
---
```

You can access R code from within the yml

Utilising back ticks and the letter “r” you can include R code into your ‘yml‘

```
---
title: "This is a quick example"
subtitle: "Just to illustrate a point"
author: "William Gosset"
date: "2018-01-23 Updated(2019-02-10)"
abstract: "Just a little exloration of things. We looked at `nro
bibliography: my_bib.bib
output:
  pdf_document
---
```

A Little More About Outputs. . .

Rmarkdown Outputs

- pdf_document
- word_document
- html_document

Presentations

- io_presentation
- beamer_presentation

My Preferred

- bookdown::pdf_document2[1]

[1] Almost identical to pdf_output but provides additional control over code chuck references. Requires the `bookdown` package

Markdown

Rmarkdown = R + Markdown

- Markdown developed as an easy way to implement `html` style formatting
- Keyboard symbols to generate basic `html` outputs
- Additionally allows you to interleave plain text with code

Markdown Examples

```
_italics_ or *italics*

__bold__ or **bold**

sub~script~

super^script^
```

$\rightsquigarrow$

Markdown Examples

italics or *italics*

bold or **bold**

sub$_{\text{script}}$

super$^{\text{script}}$

Markdown Examples

R Studio hosts a ton of great example Rmarkdown reports here

Code Chunk

Using Code Chunks

Rmarkdown documents are powerful because of code chunks

They can be inserted into a document by `CMD/CTRL + OPTION + I`

Best practice is to name each chunk to help with debugging

Writing R Code

Write code just like you would with any R scripts:

- call libraries
- write code
- any code is executed and printed as in the console

```
rnorm(1, 10, 1)
## [1] 11.90716
```

Compiling the Documents

A document will not compile if a chunk has an error!

Use Chunk Options to Control the Outputs

Some times you don't want everything printed when you compile (knit) the document

Different messages that can be set for each code chunk:

- `Echo` - If `false` then no code is printed
- `Warnings` - If `false` no warning messages are printed
- `Messages` - If `false` no messages are printed
- `Include` - If `false` no echo, no warnings, no messages printed
- `Eval` - If `false` the chunk won't be evaluated
- `Error` - If `true` allows chunk to display an error

Use Chunk Global Options

Chunk options can be set locally or globally

Global options are over-written by local options

Accessing global options

```
knitr::opts_chunk$set(echo = FALSE,
                      warning = FALSE,
                      message = FALSE)
```

Accessing *Other* Languages

```
names(knitr::knit_engines$get())
##  [1] "awk"       "bash"      "coffee"    "gawk"
##  [5] "groovy"    "haskell"   "lein"      "mysql"
##  [9] "node"      "octave"    "perl"      "psql"
## [13] "Rscript"   "ruby"      "sas"       "scala"
## [17] "sed"       "sh"        "stata"     "zsh"
## [21] "highlight" "Rcpp"      "tikz"      "dot"
## [25] "c"         "fortran"   "fortran95" "asy"
## [29] "cat"       "asis"      "stan"      "block"
## [33] "block2"    "js"        "css"       "sql"
## [37] "go"        "python"    "julia"
```

Setting Engine

Generally you will need specify where the executable file is to run other languages

```
knitr::opts_chunk$set(engine.path = list(
  python = '~/anaconda/bin/python',
  ruby = '/usr/local/bin/ruby'
))
```

A Motivating Example

Write my code is `Stan` specifying `output.var = "stan_example"` in the chunk options

```
parameters {
  real y[2];
}
model {
  y[1] ~ normal(0, 1);
  y[2] ~ double_exponential(0, 2);
}
```

Fit the Model

```r
library(rstan)
fit <- sampling(stan_example, cores = 2,
                iter = 50, refresh = 0)
```

Print the Results Directly

```
print(fit)
## Inference for Stan model: 6c400a2ac89dae0e85da9f76
## 4 chains, each with iter=50; warmup=25; thin=1;
## post-warmup draws per chain=25, total post-warmup
##
##        mean se_mean   sd  2.5%   25%   50%   75% 97
## y[1] -0.02    0.14 1.07 -2.39 -0.76  0.09  0.62  2
## y[2]  0.15    0.83 3.80 -7.59 -1.10  0.48  2.44  7
## lp__ -1.99    0.67 1.60 -5.37 -2.92 -1.47 -0.71 -0
##      n_eff Rhat
## y[1]    61 1.00
## y[2]    21 1.12
## lp__     6 1.45
##
```

Controlling Output

Chunk options can also be used to specify

- output size of figure via fig.out, fig.width, fig.height
- captions with fig.caption

Automating Rmarkdowns

An additional `yaml` argument can be passed call `params`

```
---
params:
  team: "Wake Forest"
  sport: "Football"
---
```

Using the Parameters

The parameters can be be used in other code chunks

```
data %>%
filter(team = params$team, sport = params:sport)
```

And now the code will function for any available team/ sport you specify.

A Report for Each Team and Sport!

Further, if you desired a standard report for different combinations of team

`purrr::pwalk`

Making Basic Tables

The basic way to make a table in R is through the `kable` function from `knitr`

```r
knitr::kable(head(mtcars,4)[,1:3])
```

	mpg	cyl	disp
Mazda RX4	21.0	6	160
Mazda RX4 Wag	21.0	6	160
Datsun 710	22.8	4	108
Hornet 4 Drive	21.4	6	258

Additional Table Options

The basic way to make a table in R is through the `kable` function from `knitr`

```
knitr::kable(head(mtcars,4)[,1:3],
             caption = "A Basic Table",
             col.names = c("One", "Two", "Three"))
```

Table 2: A Basic Table

	One	Two	Three
Mazda RX4	21.0	6	160
Mazda RX4 Wag	21.0	6	160
Datsun 710	22.8	4	108
Hornet 4 Drive	21.4	6	258

Extending RMarkdown

LaTex

- You can use LaTex extensively
- This include LaTex templates and cls for control
- Different citation styles

```
---
includes:
  in_header: my_format.tex
---
```

Advanced Packages

Table/ Figure Generation

- `kableExtra`
- `gt`

Templates

- `rticles`
- `papaja`
- `markdowntemplates`

Even More

- `flexdashboard`
- `shinydashboard`

Other Resources

Rmarkdown

R Studio Cheat Sheets for the basic commands Introduction to Rmarkdown for the basic ideas and getting started R Markdown Definitive Guide for the details for how Rmarkdown works

Bookdown

For Writing Books

Blogdown

For writing blogs/ websites

Scripts vs Rmds

Everything in an Rmd File?!

Always

- Final reports/ analysis
- Exploratory work

Except

- Code outnumbers Prose -> scripts
- Can run large scripts from within an Rmd with `source`

But...

`knitr::purl` can make an Rmd into an R file

Summary

Key Points

1. Rmarkdown files are a good way to produce reproducible documents
2. Whatever format you want can be done (but it might be a challenge)
3. You can work in *multiple* languages!