

# Reproducible Research

A brief overview of  
Literate Programming via R and `knitr`  
and  
Version control via git

Peter DeWitt

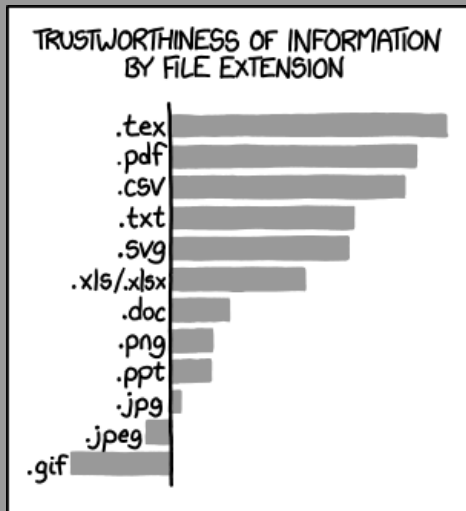
University of Colorado Anschutz Medical Campus  
Department of Biostatistics

29 January 2014

# Presentation Outline

- 1 Introduction
- 2 Hello World
- 3 Design
- 4 Options
  - Chunk Options
  - Language Options
  - Collaborations
- 5 Editors
- 6 Examples
- 7 git

# A little humor

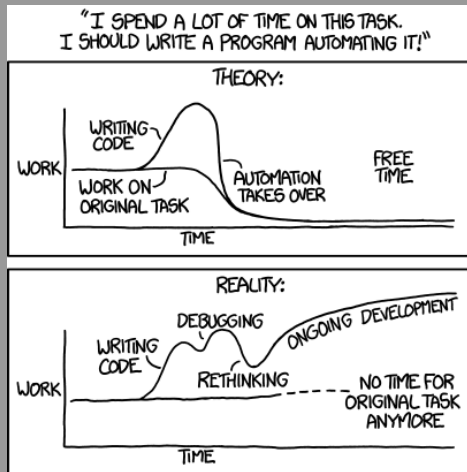


[www.xkcd.com/1301](http://www.xkcd.com/1301)

Alt Text: I have never been lied to by data in a .txt file which has been hand-aligned.

The image itself is a .png

# A little humor



[www.xkcd.com/1319](http://www.xkcd.com/1319)

Alt Text: 'Automating' comes from the roots 'auto; meaning 'self-', and 'mating', meaning 'screwing'.

# Literate Programming

- Originally intended for software development
- Mix source code (for the computer) and documentation (for the humans) together
- Sweave built on this paradigm but with a different focus: reproducible data analysis and statistical reports.
- **knitr**, developed by Yihui Xie, expands on the concept of Sweave.
- ‘designed to give the user access to every part of the process of dealing with a literate programming document’
- package homepage <http://yihui.name/knitr/>

# Dynamic Report Writing

- Pros:
  - Reproducible reports
  - Contextual commenting
  - `knitr` is flexible enough to allow for multiple analysis languages and multiple markup languages.
- Cons:
  - Not all analysis languages are ideally suited for this paradigm.
  - Collaborations with others using WYSIWYG editors requires some additional work and breaks automation/reproducibility.
- Additional Tools: Version control, e.g., git or svn.
- Every project is different; use the tools that will work best.

# Why knitr?

- Incorporate both the analysis code and the manuscript writing into one file.
- Contextually commented analysis code.
- Fully documented and reproducible reports.

## Example: Code for the next two frames

```
\begin{frame}[fragile]

<<"cars", fig.width = 3.5, fig.height = 3.25, results = "asis">>=
fit <- lm(dist ~ speed, data = cars)
latex(cbind(coef(fit), confint(fit)),
      file = "", title = "", ctable = TRUE,
      caption = "Regression Estimates",
      digits = 3, colhead = c("Est", "LCL", "UCL"))

@

The expected stopping distance for a car during the 1920s
increased by \Sexpr{round(coef(fit)[2], 2)} feet for every
additional mph increase in speed.

\end{frame}

\begin{frame}[fragile]

<<"cars_plot", fig.width = 3.5, fig.height = 2.75>>=
qplot(speed, dist, data = cars) + geom_smooth(method = "lm")

@

\end{frame}
```



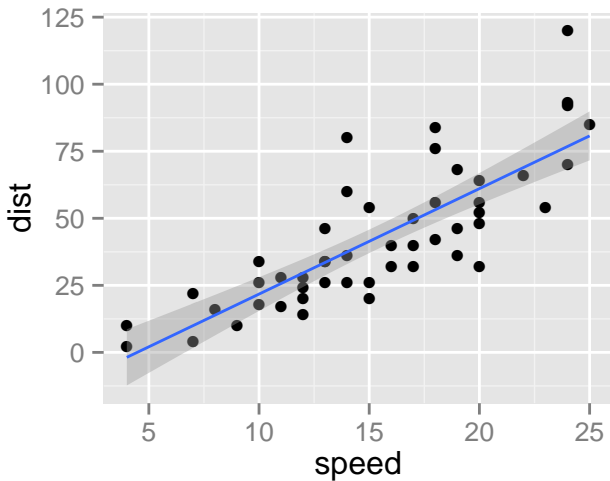
```
fit <- lm(dist ~ speed, data = cars)
latex(cbind(coef(fit), confint(fit)),
      file = "", title = "", ctable = TRUE,
      caption = "Regression Estimates",
      digits = 3, colhead = c("Est", "LCL", "UCL"))
```

Table: Regression Estimates

	Est	LCL	UCL
(Intercept)	-17.58	-31.2	-3.99
speed	3.93	3.1	4.77

The expected stopping distance for a car during the 1920s increased by 3.93 feet for every additional mph increase in speed.

```
qplot(speed, dist, data = cars) + geom_smooth(method = "lm")
```



# How does `knitr` work?

- One input file with
  - an analysis language (R, Python, awk, SAS, ...) and
  - an output markup language (L<sup>A</sup>T<sub>E</sub>X, html, Markdown, ...)
- `knitr` determines the appropriate set of patterns (regular expression to extract analysis language and options from the input file)
- The input file is knitted..., analysis language is evaluated and the appropriate output markup results are placed into a `.tex`, `.html`, `.md`, ..., file.
- Final document (the `.tex`, `.html`, `.md`, ...) is ready for release or post processing as needed.

# Customizing the behavior of knitr

For Rnw files:

```
<<"chunk_label", echo = FALSE, results = "asis">>=  
@
```

- Chunk options must be a single line; no line breaks.
- Options must be valid R expressions.
- Chunk options can be specified for each individual chunk.
- Global options are set via `opts_chunk$set()`

# Chunk Options

Full details for all the chunk options see  
<http://yihui.name/knitr/options>

- Code Evaluation
- How to display results
- Code Decoration
- Cache
- Plots
- Animation
- Chunk References
- Child Documents
- Language Engines
- Extracting source code

# Input/Analysis Language

- R is the ‘default’ language for analysis
- Other options are available, including SAS. The chunk option ‘engine’ allows for different languages to be used.

*engine: (‘R’; character) the language name of the code chunk; currently other possible values are ‘python’ and ‘awk’/‘gawk’; the object `knit_engines` in this package can be used to set up engines for other languages<sup>1</sup>*

- Pick the right language for the job. R is great, but every now and then SAS would be preferable.

---

<sup>1</sup><http://yihui.name/knitr/options>

# Markup Language

html

- Pros:
  - easy to send to others,
  - comments in html,
  - great for tutorials or anything that will be published online.
- Cons:
  - Clunky (in my opinion),
  - not apt for large data analysis reports.
- html code can be placed natively in markdown, ergo, markdown has supplanted html.

## Language Options

# Markup Language

## Markdown

- Pros:
  - Easy to learn
  - Simple and versatile
  - Growing user community
  - Via pandoc, easy to convert to many other file formats such as  $\text{\LaTeX}$ , html, or .docx.
- Cons:
  - no native comments.
  - ‘too minimal’
  - pandoc may require additional scripting to format output as needed for .docx or other file types.



# Markup Language

## L<sup>A</sup>T<sub>E</sub>X

- Pros:
  - Intended use: technical report writing and typesetting.
  - Comments.
  - Many tools exist for formatting R output well in L<sup>A</sup>T<sub>E</sub>X files.
  - Cross referencing, citations.
  - pdf<sub>l</sub>atex produces a file that is almost platform independent.
- Cons:
  - Difficulties will be encountered when collaborating with Microsoft Office users.
  - R is small, Tex Live, MacTeX, and proTeX are not.
  - Steepest learning curve

## Language Options

## Markup Language

## My Preferences

- $\text{\LaTeX}$  is my preferred markup language for data analysis reports and presentations (via beamer).
- Markdown is my preferred markup language for developing a web page. Easier to work in than html and more flexible.

$$.Rmd \xrightarrow{\text{knitr}} .md \xrightarrow{\text{pandoc}} \begin{cases} .html & \text{okay} \\ .doc(x) & \text{might require VB scripting} \\ .pdf & \text{Use } \text{\LaTeX} \end{cases}$$

## Collaborations

## That's great, but ...

- L<sup>A</sup>T<sub>E</sub>X pdfs are not always easy to transfer to MS Word
- Local html pages are easy to break.
- Misconceptions about viewing local files versus remote files in a web browser.
- .doc(x) are ‘accidentally’ editable.
- What about Google docs?
- What about track changes in Office?

# Suggested Development Environments

- I prefer the Vim editor and with the vim-r-plugin.
  - Vim Editor: [www.vim.org/index.php](http://www.vim.org/index.php)
  - vim-r-plugin:  
[www.vim.org/scripts/script.php?script\\_id=2628](http://www.vim.org/scripts/script.php?script_id=2628)
  - Completely customizable.
- If you are not a Vim or Emacs user, then RStudio is the premier R IDE for you.
  - Download and info: [www.rstudio.com](http://www.rstudio.com)
  - Built in tools for version control, projects, knitting...
- RStudio is the better R development environment.
- Vim is a better text editor.
- A pseudo WYSIWYG editor for  $\text{\LaTeX}$  which will work well with `knitr` is LyX.

# Reproducible examples

These slides, and the following examples, can be downloaded/cloned, from

<https://github.com/dewittpe/knitrexamples>

- A simple data analysis report using R, and three different markup languages,  $\text{\LaTeX}$ , html, and Markdown.
- An example of using SAS within `knitr`.
- An example of a more complex and extensive data analysis report.

# Version Control with git

- Distributed version control: everything is local and can be synced with a remote server.
- Simple to use.
  - Built in GUI with RStudio
  - Only need six commands to be able to use git.
    - 1 git status
    - 2 git log
    - 3 git add
    - 4 git commit
    - 5 git pull – only if working with a remote
    - 6 git push – only if working with a remote
  - Other very helpful commands: tag, fetch, merge, mergetool, and branch

# Do you need a server?

- The repositories are local, so servers are not needed.
- Servers to host a central repository are helpful for collaborations.
- Read/Write permissions can be set for different users.
- There are options to put a git enterprise on a server behind our firewall, but it's expensive.
- github.com has free public repos, pay for private repos
- bitbucket.org has free public and private repos.
- Visit <http://git-scm.com/book> for documentation on git.