Reproducible Research with knitr

Thomas J. Leeper

Department of Political Science and Government
Aarhus University

October 28, 2014

1 Overview

2 Activity

3 Literate Programming

4 knitr in Depth

5 Wrapup

1 Overview

2 Activity

3 Literate Programming

4 knitr in Depth

5 Wrapup

Teaching/Learning Approach

- Hands-on practice
- Work independently to enhance your own workflow
- You will not learn everything today

Outline for afternoon

- A short activity
- History and philosophy of literate programming
- Work through basics together
- Independent project work
- Wrap up and move forward

1 Overview

2 Activity

3 Literate Programming

4 knitr in Depth

5 Wrapup

Think about your own workflow

- Think about: *How do I get outputs from my data?*
- Draw a map or diagram of your workflow
- Include relevant steps and tools, such as:
    - Tables
    - Figures
    - In-text citations and reference list
    - In-text analysis summaries
    - Cross-referencing (tables, figures, sections)
    - Document layout
- Make notes about areas that are *time-consuming* and/or *difficult*

1 Overview

2 Activity

3 Literate Programming

4 knitr in Depth

5 Wrapup

Literate programming

- Origins in computer program documentation
- Software source code should describe how to use that software
- Early tools
  - WEB by Donald Knuth (author of TeX)
  - noweb by Norman Ramsey (1989)
- Two operations to create two different outputs
  - *Weave*: Nice Documentation
  - *Tangle*: Executable code

Sweave

- Released in 2002 by Friedrich Leisch[1]
- Written for S (the language of R)
- Focused on creating articles
- Two operations to create two different outputs
  - `SWeave`: LaTeX document (and PDF)
  - `STangle`: Executable R code

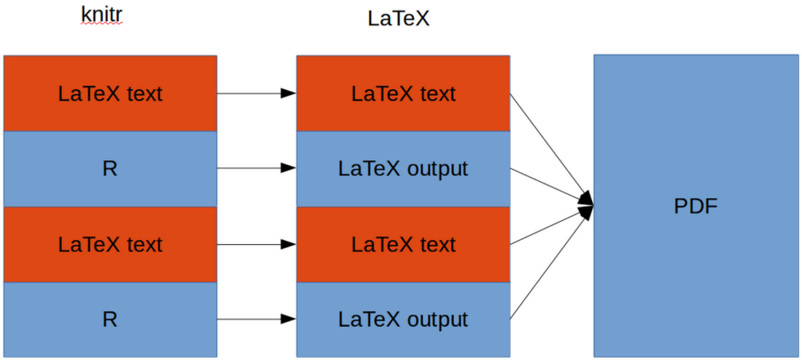
[1] Sweave: Dynamic Generation of Statistical Reports Using Literate Data Analysis

knitr

- Released in 2012 by Yihui Xie[2]
- Conceptual descendant of Sweave
  - Easier than Sweave
  - Much more functionality and flexibility
- Three operations to create two different outputs
  - `knit`: PDF (and LaTeX document)
  - `purl`: Executable R code
  - `spin`: PDF (from pure R code)
- Also create various outputs from non-LaTeX input

[2] knitr Homepage

How knitr Works[3]

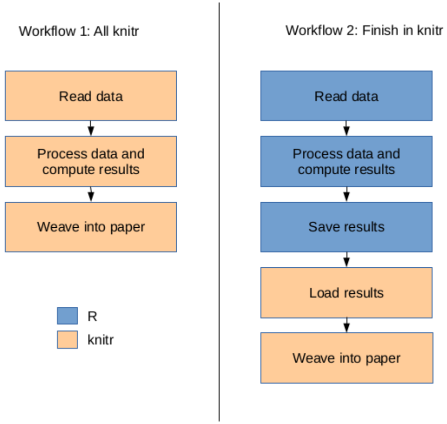


[3]Image by Ari B. Friedman

Workflows for knitr

	Analysis	**Output**
Irreproducible	R	Copy-paste
No knitr	R	Manual includes
Finish in knitr	R	Load and knit
All knitr	knitr	n/a

Workflows for knitr[4]



[4]Image by Ari B. Friedman

1 Overview

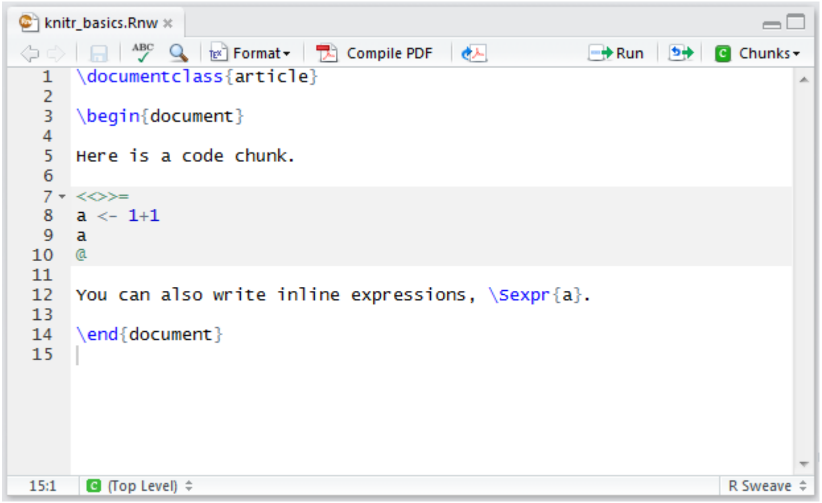
2 Activity

3 Literate Programming

4 knitr in Depth

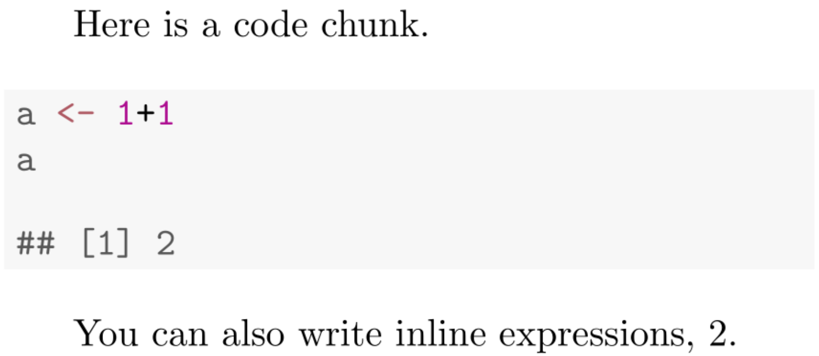
5 Wrapup

knitr Input



```
\documentclass{article}

\begin{document}

Here is a code chunk.

<<>>=
a <- 1+1
a
@

You can also write inline expressions, \Sexpr{a}.

\end{document}

```

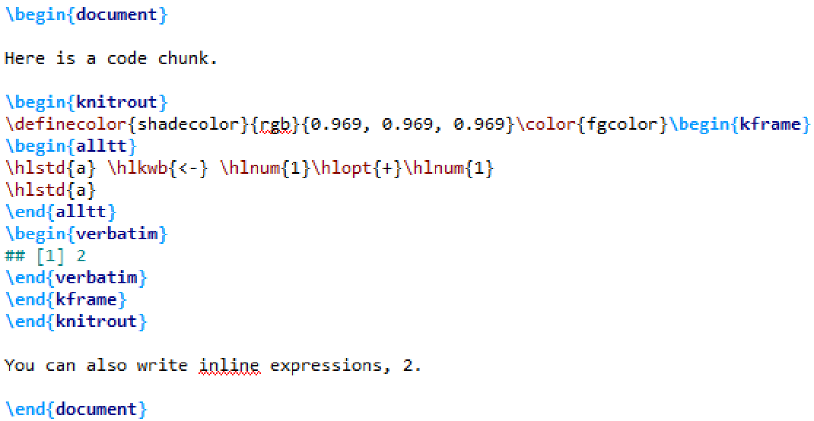
PDF Output


Here is a code chunk.

```
a <- 1+1
a

## [1] 2
```

You can also write inline expressions, 2.

LaTeX Intermediary



```
\begin{document}

Here is a code chunk.

\begin{knitrout}
\definecolor{shadecolor}{rgb}{0.969, 0.969, 0.969}\color{fgcolor}\begin{kframe}
\begin{alltt}
\hlstd{a} \hlkwb{<-} \hlnum{1}\hlopt{+}\hlnum{1}
\hlstd{a}
\end{alltt}
\begin{verbatim}
## [1] 2
\end{verbatim}
\end{kframe}
\end{knitrout}

You can also write inline expressions, 2.

\end{document}
```

Code Chunks

- Code chunks contain three parts
- Label
  - Used for referencing chunks
- Options
  - Control chunk behavior and appearance
- Contents
  - R code to be evaluated

Code Chunks: Anatomy

```
«a,eval=TRUE,echo=FALSE,results='asis'»=
a <- 1+1
a
@
```

Code Chunks: Anatomy

```
<<a,eval=TRUE,echo=FALSE,results='asis'>>=
a <- 1+1
a
@
```

Code Chunks: Anatomy

```
«a,eval=TRUE,echo=FALSE,results='asis'»=
a <- 1+1
a
@
```

Code Chunks: Anatomy

```
«a,eval=TRUE,echo=FALSE,results='asis'»=
a <- 1+1
a
@
```

Code Chunks: Anatomy

```
<<a,eval=TRUE,echo=FALSE,results='asis'>>=
a <- 1+1
a
@
```

Code Chunks: Anatomy

```
<<a,eval=TRUE,echo=FALSE,results='asis'>>=
a <- 1+1
a
@
```

Code Chunks: Anatomy

```
«a,eval=TRUE,echo=FALSE,results='asis'»=
a <- 1+1
a
@
```

Code Chunks: Anatomy

```
«a,eval=TRUE,echo=FALSE,results='asis'»=
a <- 1+1
a
@
```

Code Chunks: Anatomy

```
«a,eval=TRUE,echo=FALSE,results='asis'»=
a <- 1+1
a
@
```

```
«a»=
@
```

Code Chunks: Options

- `echo`
- `eval`
- `results`
- `tidy` and `highlight`
- `warning` and `message`

Code Chunks: Options

- Chunk options can be set for each chunk
- They can also be set globally in a document
- E.g., `opts_chunk$set(echo = FALSE)`

Code Chunks: Inline Code

- In addition to chunks, code can be written in-line
- Anything in `\Sexpr{}` is evaluated
- Useful for in-line reporting of analyses

Code Externalization

- Possible to *externalize* R code

Code Externalization

- Possible to *externalize* R code
- “Child” documents
  - knitr code chunks in separate file

Child knitr Document

- Child Document: `child.Rnw`

```
<<>>=
x <- 1:3
y <- 4:6
@
```

- Parent Document: `knitrdoc.Rnw`

```
<<a, child = 'child.Rnw'>>=
@
```

Code Externalization

- Possible to *externalize* R code
- "Child" documents
  - knitr code chunks in separate file

Code Externalization

- Possible to *externalize* R code
- “Child” documents
  - knitr code chunks in separate file
- Reading code from file
  - Code in specially formatted R script
  - Code remains executable without knitr

External R Script

- R Script: `analysis.R`

  ```
  ## ---- a
  x <- 1:3

  ## ---- b
  y <- 4:6
  ```

- knitr Document: `knitrdoc.Rnw`

  ```
  <<>>=
  read_chunk('analysis.R')
  @
  <<a>>=
  @
  ```

Chunk Caching

- knitr runs every chunk every time
- This is unnecessary if you're making non-code changes
- Can be time-consuming
- The `cache` chunk option changes this

Chunk Caching: How it Works

- Set `cache=TRUE` to *cache* a chunk
- knitr stores the chunk and its results
  - Stored in .RData files in `./cache`
- Cached chunks are only run after changes
  - Substantive and non-substantive changes
- Behavior depends on relations between chunks

Chunk Caching: Chunk Dependencies

- Cached chunks are only rerun if modified
- But chunks might depend on other chunks
  - B depends on cached A
  - Cached B depends on A
  - Cached B depends on cached A
- Specify dependencies with `dependson`
  - Or: `opts_chunk$set(cache=TRUE, autodep=TRUE)`

Figures

- Two ways to include figures:
- Using knitr chunk options for figures
  - Handles lots of details automatically
  - Takes work to customize
- Manually using `\includegraphics{}`
  - Somewhat finer control
  - Requires more LaTeX overhead

Tables

- LaTeX tables are tedious
- Doing them by-hand is irreproducible and a waste of time
- Lots of ways to create tables with knitr
  - `kable`
  - `xtable`
  - `stargazer`

Porting a Project to knitr

- Move existing R code into a knitr framework
- What code chunks and in-line expressions do you need
- How do you create tables and figures?

Package Versioning

- Reproducibility requires knowing software used to conduct analyses
- Including package names using `library` or `require` is not enough
- Your future self (and others) need to know package versions
- How do we handle that?

Package Versioning: Do it Manually

- Record versions and either:
  - Put these in a README
  - Have knitr fail on wrong version
- Manually install package version:
  - devtools
  - repmis
- Tedious

Package Versioning: packrat

- Package developed by RStudio
- Work in an isolated software environment
- Install packages into a local project directory
- Share your **packrat** directory as part of your reproducible directory

Package Versioning: checkpoint

- Package developed by Revolution Analytics
- Register a “checkpoint” (a date) for your analyses
- All packages are drawn from MRAN, a daily snapshot of the R package universe
- No need to store/share a large package directory

Package Versioning: Virtual Machines

- knitr connects analyses and output

Package Versioning: Virtual Machines

- knitr connects analyses and output
- `packrat` or `checkpoint` connect R and analyses

Package Versioning: Virtual Machines

- knitr connects analyses and output
- `packrat` or `checkpoint` connect R and analyses
- What about the connection between OS and R?

Package Versioning: Virtual Machines

- knitr connects analyses and output
- `packrat` or `checkpoint` connect R and analyses
- What about the connection between OS and R?
  - Virtual machines

Package Versioning: Virtual Machines

- knitr connects analyses and output
- `packrat` or `checkpoint` connect R and analyses
- What about the connection between OS and R?
  - Virtual machines
  - Docker

1 Overview

2 Activity

3 Literate Programming

4 knitr in Depth

5 Wrapup

Wrapup

- What questions/concerns do you have?
- How have today's activities helped you think about your own reproducible workflow?

Things we probably didn't cover

- knitr's `spin` function: Creates a PDF from an R script
  - Really useful for teaching assignments
- Language engines: Embed non-R code
  - Python, Bash, Julia, FORTRAN, Stata(?)
- rmarkdown: knit without using LaTeX markup

Other Reproducible Research Tools

- git: Version control
- GitHub and Bitbucket: Git cloud services
    - Good for collaboration[5]
- pandoc: Command-line tool to convert documents between formats
- Tools for R package versioning
    - devtools
    - repmis
    - packrat
    - checkpoint

[5]See “Collaborating with Git and Bitbucket”

knitr Resources

- knitr website
- CRAN Reproducible Research TaskView
- *Dynamic Documents with R and knitr*
- *Reproducible Research with R and RStudio*
- knitr Google Group
- knitr on StackOverflow