Advanced R Markdown

Day 2: Customization and Extensions

Yihui Xie and Hao Zhu

2019/01/16 @ rstudio::conf, Austin, TX


6
7

Slides: http://bit.ly/arm-xie

Examples: http://bit.ly/arm-exm

Outline

- Welcome to the command-line world
  - Parameterized reports
- How R Markdown works: knitr + Pandoc
  - Pandoc's Markdown
  - knitr: Things you may not know
- R Markdown output formats
- Custom templates and formats
  - **rticles**: LaTeX journal articles
  - **memor**: LaTeX customization
- Shiny and HTML widgets
- (Optional) knitr hooks and language engines

Using R Markdown via command line

rmarkdown::render()

- Under the hood, it calls `knitr::knit()` (`.Rmd -> .md`) and Pandoc (`.md` to other formats)
- **knitr** processes code chunks and inline R expressions
- Pandoc converts Markdown to other output formats
- Click the Knit button (in RStudio), and get one output document
- If you run a loop, you can easily get a thousand reports

```r
for (year in 1001:2000) {
  rmarkdown::render('input.Rmd', 'pdf_document',
    output_file = paste0('report-', year, '.pdf'))
}
```

```md
# Report for Year `r year`

More content of input.Rmd
```

Understanding the `envir` argument

- `rmarkdown::render()` has an `envir` argument for the environment in which the R code in the R Markdown document is evaluated
- The default is `parent.frame()`, which is usually the global environment of your workspace, unless you are calling this function inside other functions

A quick example

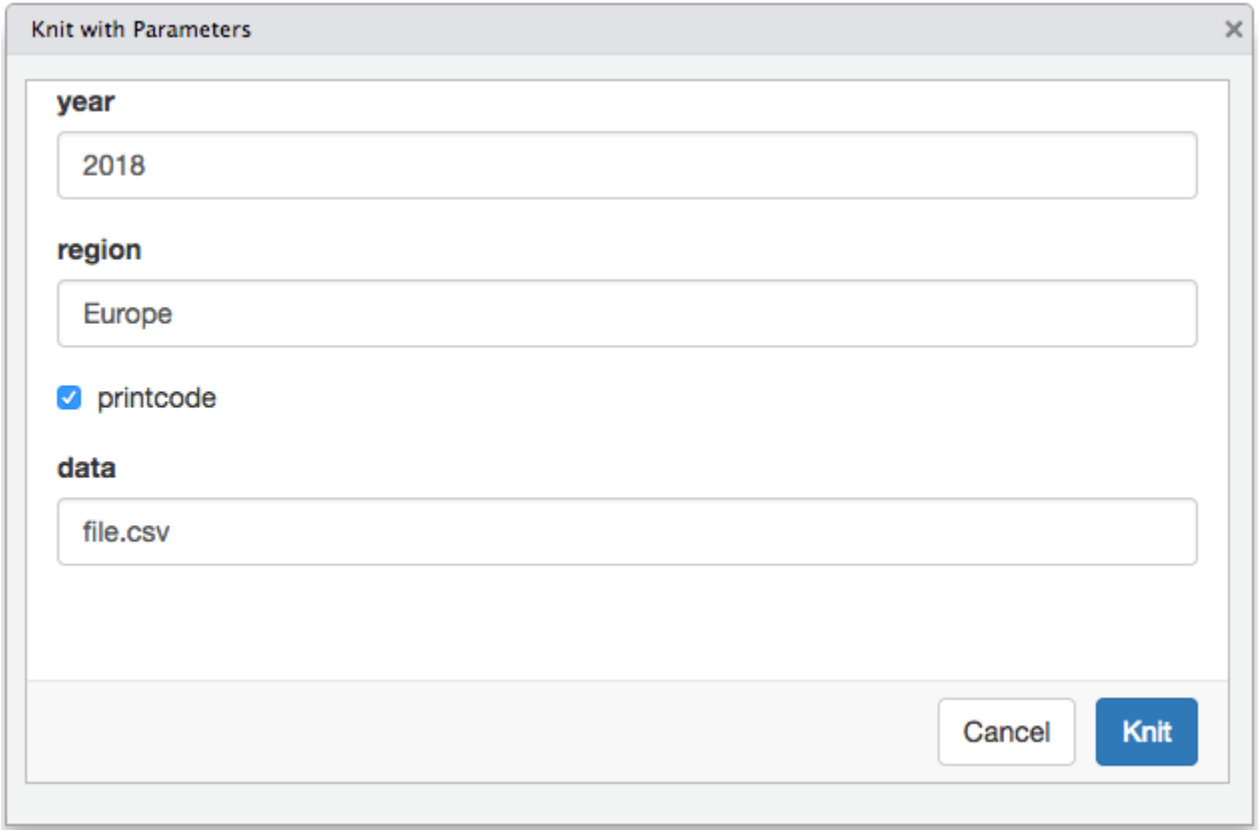
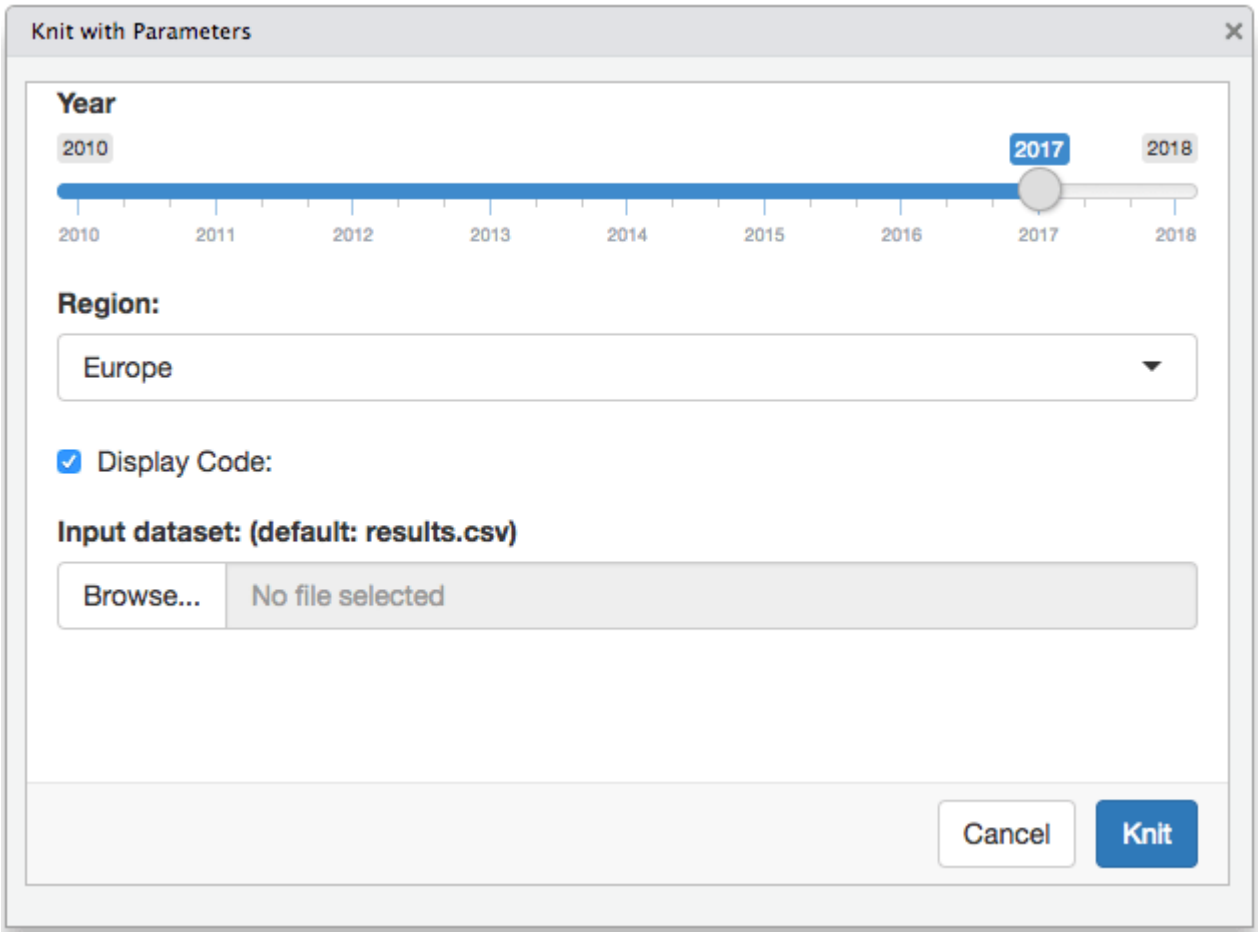
A custom render function:

```
my_render = function(x) {
  rmarkdown::render('input.Rmd')
}
```

The source of `input.Rmd`:

```
# A simple report

```{r}
head(x)
```
```

Call the custom render function:

```
my_render(iris)     # x will be `iris` in input.Rmd
my_render(mtcars)   # use `mtcars` as x now
```

Parameterized reports via `params`

- The `envir` argument is extremely flexible, but it may be too technical (it is not trivially easy to understand R's environments)
- R Markdown introduced a special object to help you parameterize your reports
- You can use either the `params` argument of `render()`, or define `params` in YAML, e.g.,

```
title: "My Report"
params:
  year: 2001
```

or command line:

```
rmarkdown::render('input.Rmd', params = list(year = 2001))
```

Command-line `params` will override `params` in YAML; `params` may contain multiple parameters.

Using `params` inside R Markdown

Typically `params` is a list, so you can extract its elements via `$` (or `[[]]`).

```
# Report for Year `r params$year`

More content of input.Rmd
```

Render reports by a changing parameter through a loop:

```
for (year in 1001:2000) {
  rmarkdown::render('input.Rmd', params = list(year = year))
}
```

Input parameters interactively



`Knit with Parameters` in RStudio. Section 15.3.3 of the R Markdown book.

More input controls



Example: 01-params-controls.Rmd

Parameterized reports on RStudio Connect

- https://www.rstudio.com/products/connect/
- Input parameters through the web interface of RStudio Connect
- View reports built previously
- Automated emails
- Example

Render & download a report in a Shiny app

- Example: http://shiny.rstudio.com/gallery/download-knitr-reports.html
- Source: https://github.com/rstudio/shiny-examples/tree/master/016-knitr-pdf

Debugging R Markdown documents

- For non-trivial debugging tasks (e.g., debugging complicated functions), you have to call `rmarkdown::render()` interactively.
    - Inside the R Markdown document, you may use usual debugging techniques such as `debug()` or inserting `browser()` in functions.
- To debug the Pandoc conversion, try `rmarkdown::render(..., clean = FALSE)`. Then intermediate files (such as `.md`) will be preserved, so you can check what's possibly wrong there.

How R Markdown works

> Good morning, #rstats friends! I mentioned in class how learning R is a lifelong process, there isn't always a "right" answer, & our community is kind & supportive of beginners. In the spirit of being vulnerable, what's one thing in R you don't yet quite understand?
>
> --- Jesse Mostipak (@kierisi)

> Good morning, #rstats friends! I mentioned in class how learning R is a lifelong process, there isn't always a "right" answer, & our community is kind & supportive of beginners. In the spirit of being vulnerable, what's one thing in R you don't yet quite understand?
>
> --- Jesse Mostipak (@kierisi)
>
> Anything about the inner workings of rmarkdown/knitr/pandoc. I press knit, a document appears, and I believe that anything happening in between could be actual magic.
>
> --- Allison Horst (@allison_horst)

https://twitter.com/AlexisLNorris/status/1082039311820836864

The Knit button

- It calls `rmarkdown::render()`
- R Markdown ≈ knitr (R) + Pandoc (Markdown)
- `rmarkdown::render()` ≈ `knitr::knit()` + a `system()` call to `pandoc`
- R Markdown (`.Rmd`) -> `knit()` -> Markdown (`.md`) -> `pandoc` ->
  - `.html`
  - `.pdf` (LaTeX)
  - `.docx`
  - `.epub`
  - `.rtf`
  - ...

A minimal R Markdown document

```
---
title: "A Simple Regression"  #----
author: "Yihui Xie"           #    |--> Pandoc variables
date: "2019-01-02"            #----
output:                       #----
  html_document:              #    |--> Passed to rmarkdown
    toc: true                 #----
---
```

```
We built a linear regression model.

```{r}
fit <- lm(dist ~ speed, data = cars)
b   <- coef(fit)
plot(fit)
```

The slope of the regression is `r b[1]`.
```

Markdown output after knitting

```
---
title: "A Simple Regression"  #----
author: "Yihui Xie"           #    |--> Pandoc variables
date: "2019-01-02"            #----
output:                       #----
  html_document:              #    |--> Passed to rmarkdown
    toc: true                 #----
---
```

```
We built a linear regression model.

```r
fit <- lm(dist ~ speed, data = cars)
b   <- coef(fit)
plot(fit)
```

![a plot](input_files/figure-html/unnamed-chunk-1.png)

The slope of the regression is -17.57909.
```

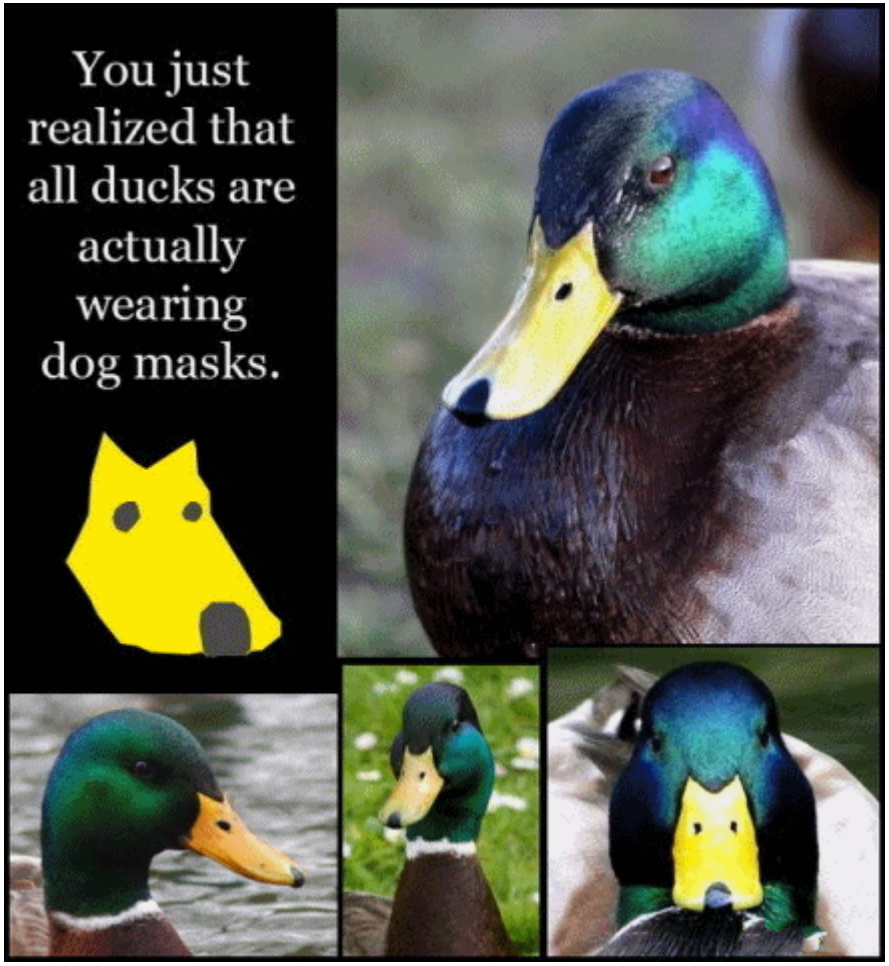
After Pandoc conversion (HTML output)

```html
<html>
  <head>
    <title>A Simple Regression</title>
    <meta name="author" content="Yihui Xie" />
    <meta name="date" content="2019-01-02" />
  </head>

  <body>
    <p>We built a linear regression model.</p>

    <pre>fit &lt;- lm(dist ~ speed, data = cars)
    b   &lt;- coef(fit)
    plot(fit)
    </pre>

    <img src="input_files/figure-html/unnamed-chunk-1.png"
      alt="a plot" />

    <p>The slope of the regression is -17.57909.</p>
  </body>
</html>
```

After Pandoc conversion (LaTeX output)

```
\documentclass{article}
\title{A Simple Regression} \author{Yihui Xie} \date{2019-01-02}
\begin{document}
\maketitle

We built a linear regression model.

\begin{verbatim}
fit <- lm(dist ~ speed, data = cars)
b   <- coef(fit)
plot(fit)
\end{verbatim}

\begin{figure}
\includegraphics{input_files/figure-html/unnamed-chunk-1.png}
\caption{a plot}
\end{figure}

The slope of the regression is -17.57909.
\end{document}
```

Same ducks, different masks



The (R) Markdown philosophy

Similar to KISS

Keep the Duck Simple and Stupid

and wear a mask as fancy as you want

Pandoc's Markdown

- You should read the Pandoc Manual at least once to learn the possibilities of Pandoc's Markdown: https://pandoc.org/MANUAL.html#pandocs-markdown
- Original Markdown (John Gruber)
  - primarily for HTML
  - paragraphs, `# headers`, `> blockquotes`
  - `**bold**`, `_italic_`
  - `- lists`
  - `[text](url)`
  - `![text](image)`
  - code blocks (indent by four spaces)

Pandoc's Markdown

- Markdown extensions
  - YAML metadata
  - LaTeX math `$\sum_{i=1}^n \alpha_i$` = $\sum_{i=1}^n \alpha_i$
  - syntax highlighting of code blocks (three backticks followed by the language name, e.g. ```` ```r ```)
  - tables
  - footnotes `^[A footnote here.]`
  - citations `[@joe2014]` (database can be BibTeX or in YAML)
  - raw HTML/LaTeX

Pandoc's Markdown

- Types of output documents
  - LaTeX/PDF, HTML, Word (MS Word, OpenOffice)
  - beamer, ioslides, Slidy, reveal.js
  - E-books
  - ...

https://pandoc.org

Command-line usage of Pandoc

Some examples:

```
pandoc test.md -o test.html
pandoc test.md -s --mathjax -o test.html
pandoc test.md -o test.odt
pandoc test.md -o test.rtf
pandoc test.md -o test.docx
pandoc test.md -o test.pdf
pandoc test.md --pdf-engine=xelatex -o test.pdf
pandoc test.md -o test.epub
```

To run system commands in R, use functions `system()` or `system2()`.

The **rmarkdown** package provides a helper function `rmarkdown::pandoc_convert()` to convert Markdown documents to other formats using Pandoc.

When you click the Knit button in RStudio, you will see the actual (usually very long) command that is executed.

Example: Markdown in the eyes of Pandoc

The Pandoc abstract syntax tree (AST)

Let's explore a Markdown file with R:

```
f1 = tempfile()

# pandoc -f markdown -t json ...
rmarkdown::pandoc_convert(
  "2019-rstudio-arm/02-markdown-data.md",
  to = "json", from = "markdown", output = f1, wd = "."
)

# read JSON into R
x = jsonlite::fromJSON(f1, simplifyVector = FALSE)
```

```r
str(x)  # original Markdown data
```

```
List of 3
 $ blocks            :List of 4
  ..$ :List of 2
   .. ..$ t: chr "Header"
  .. ..$ c:List of 3
   .. .. ..$ : int 2
  .. .. ..$ :List of 3
  .. .. .. ..$ : chr "a-header"
  .. .. .. ..$ : list()
  .. .. .. ..$ : list()
  .. .. ..$ :List of 3
  .. .. .. ..$ :List of 2
  .. .. .. .. ..$ t: chr "Str"
  .. .. .. .. ..$ c: chr "A"
  .. .. .. ..$ :List of 1
  .. .. .. .. ..$ t: chr "Space"
  .. .. .. ..$ :List of 2
  .. .. .. .. ..$ t: chr "Str"
  .. .. .. .. ..$ c: chr "header"
  ..$ :List of 2
  .. ..$ t: chr "Para"
  .. ..$ c:List of 3
  .. .. ..$ :List of 2
  .. .. .. ..$ t: chr "Str"
  .. .. .. ..$ c: chr "A"
```

How to change ## to #? Or in general, level-N headers to level-(N-1) headers?

```
# a recursion into the list to modify header levels
raise_header = function(x) {
  lapply(x, function(el) {
    if (!is.list(el)) return(el)
    if (identical(el[["t"]], "Header")) {
      lvl = el[["c"]][[1]]
      if (lvl <= 1)
        stop("I don't know how to raise the level of h1")
      el[["c"]][[1]] = as.integer(lvl - 1)
    }
    raise_header(el)
  })
}

x = raise_header(x)
```

```r
str(x)  # modified Markdown data
```

```
List of 3
 $ blocks            :List of 4
  ..$ :List of 2
   .. ..$ t: chr "Header"
  .. ..$ c:List of 3
   .. .. ..$ : int 1
  .. .. ..$ :List of 3
  .. .. .. ..$ : chr "a-header"
  .. .. .. ..$ : list()
  .. .. .. ..$ : list()
  .. .. ..$ :List of 3
  .. .. .. ..$ :List of 2
  .. .. .. .. ..$ t: chr "Str"
  .. .. .. .. ..$ c: chr "A"
  .. .. .. ..$ :List of 1
  .. .. .. .. ..$ t: chr "Space"
  .. .. .. ..$ :List of 2
  .. .. .. .. ..$ t: chr "Str"
  .. .. .. .. ..$ c: chr "header"
  ..$ :List of 2
  .. ..$ t: chr "Para"
  .. ..$ c:List of 3
  .. .. ..$ :List of 2
  .. .. .. ..$ t: chr "Str"
  .. .. .. ..$ c: chr "A"
```

```r
f2 = tempfile()  # to write out (the modified) JSON
f3 = tempfile()  # to write out Markdown

xfun::write_utf8(jsonlite::toJSON(x, auto_unbox = TRUE), f2)

rmarkdown::pandoc_convert(
  f2,
  to = "markdown", from = "json", output = f3,
  options = "--atx-headers", wd = "."
)
xfun::file_string(f3)
```

```
# A header

A paragraph.

-   One point.

-   Another point.

# Another header
```

```r
unlink(c(f1, f2, f3))
```

More power (and speed) with Lua filters

Rewrite the previous R function with a Lua filter `raise-header.lua`:

```
function Header(el)
  if (el.level <= 1) then
    error("I don't know how to raise the level of h1")
  end
  el.level = el.level - 1
  return el
end
```

Run it:

```
pandoc -t markdown --lua-filter=raise-header.lua 02-markdown-data.md
```

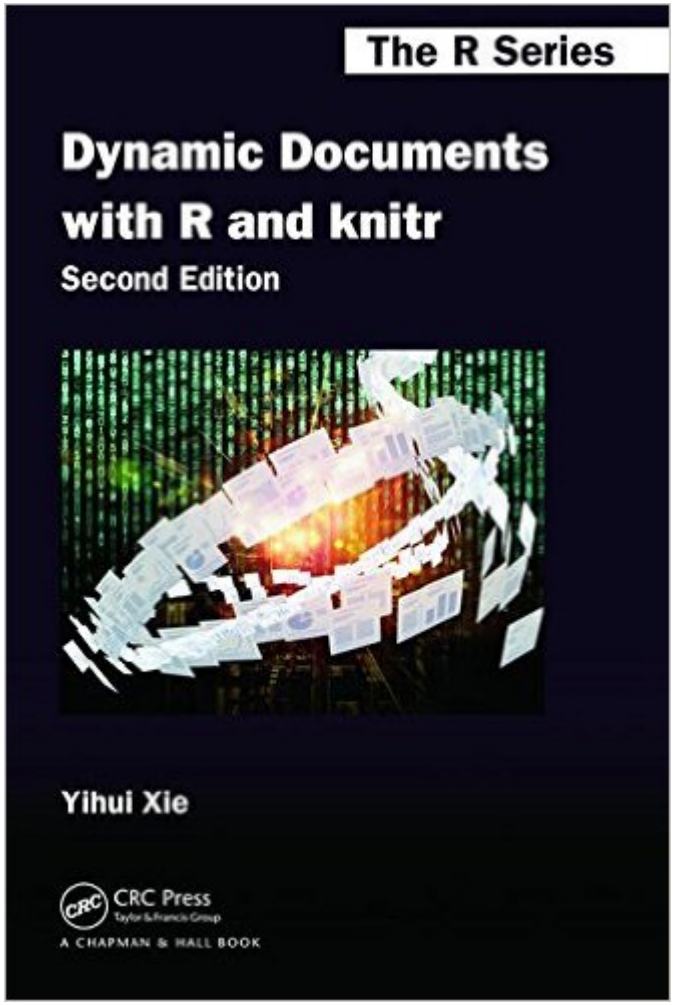
More about Lua filters: https://pandoc.org/lua-filters.html

The Pandoc version

- RStudio has bundled a version of Pandoc, so you don't need to install Pandoc separately if you use RStudio
- If you install Pandoc by yourself, **rmarkdown** will use the highest version of Pandoc that it can find
- Check `rmarkdown::pandoc_version()`
- RStudio 1.1.x included Pandoc 1.19.x; RStudio 1.2.x will include Pandoc 2.x
    - Pandoc 2.x is not fully compatible with 1.x, but we have solved these issues in the **rmarkdown** package and other R packages we maintain (e.g., `--latex-engine` was renamed to `--pdf-engine`)

knitr

the other cornerstone of R Markdown



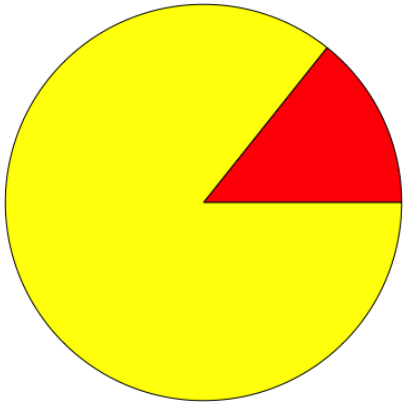
The **knitr** book is a comprehensive guide, but is unfortunately not free. Stay tuned for a free book this year.

knitr is not only for R

- It contains many, many other language engines: https://bookdown.org/yihui/rmarkdown/language-engines.html
- For example, Shell/Bash scripts, SQL, Python, C, C++, Fortran, Stan, ...
- Demo of two engines: `python` and `asis`.

```
```{python}
x = 42
print(x)
```
```

```
```{asis, echo=identical(knitr:::pandoc_to(), 'html')}
Here is _some text_ that you want to display only
when the output format of R Markdown is **HTML**.

You can write arbitrary Markdown content in this chunk.
```
```

knitr is not only for Markdown, either

R Markdown may be the most popular document format, but you could also use other authoring languages such as LaTeX, HTML, AsciiDoc, and reStructuredText.

Demo: `*.Rnw`, `*.Rhtml`

knitr works on R scripts, too

- Most of time you may be using `knitr::knit()`, but sometimes you may want `knitr::spin()`.
- `knitr::spin()` first converts an R script to R Markdown (or other document formats that `knitr::knit()` supports, such as `*.Rnw`).
- If you use RStudio, you can click the button "Compile Report" on the toolbar.
- Demo: https://github.com/yihui/knitr/blob/master/inst/examples/knitr-spin.R

The chunk option include=FALSE

Have you ever used these chunk options?

```
```{r, echo=FALSE, results='hide'}
```
```

or

```
```{r, echo=FALSE, results='hide', message=FALSE, warning=FALSE}
```
```

or even

```
```{r, echo=FALSE, results='hide', message=FALSE, warning=FALSE, fig.
```
```

You probably only need a single chunk option `include=FALSE`: https://yihui.name/en/2017/11/knitr-include-false/.

Conditional evaluation/inclusion

Include a chunk in the output only if the output format is `html`:

```
```{r, include=identical(knitr::pandoc_to(), 'html')}
# blabla
```
```

Helper functions `knitr::is_latex_output()` (`latex` or `beamer`) and `knitr::is_html_output()` (`html`, `ioslides`, `slidy`, ...). Evaluate a code chunk only if the output format is LaTeX:

```
```{r, eval=knitr::is_latex_output()}
# blabla
```
```

BTW, the **tufte** package makes heavy use of these functions so that its functions work for both HTML and LaTeX output, e.g., `tufte::newthought()`.

Live-preview HTML output documents

- Tired of clicking the Knit button to view your results?
- Just use `xaringan::inf_mr('your.Rmd')`.
  - `install.packages('xaringan')`
  - You can also use the RStudio addin "Infinite Moon Reader".
- Demo
- For more info, see
  - https://bookdown.org/yihui/rmarkdown/compile.html
  - https://bookdown.org/yihui/rmarkdown/xaringan-preview.html
  - https://yihui.name/en/2017/08/why-xaringan-remark-js/


DB 3584 AY

knitr::knit_watch()

Watch an input file continuously, and knit it when it is updated, e.g.,

```
library(knitr)
knit_watch('foo.Rnw', knitr::knit2pdf)

knit_watch('foo.Rmd', rmarkdown::render)
```

This function works for any documents with any output formats, but unlike `xaringan::inf_mr()`, it does not automatically refresh the output page. However, if the output format is PDF, your PDF viewer might be able to automatically refresh the page when the PDF has been updated.

Caching

- The chunk option `cache=TRUE`
- Basic idea: if nothing has changed from the previous run, just load the results instead of executing the code chunk again.

```
```{r cache=TRUE}
Sys.sleep(10)  # pretend this is a time-consuming code chunk
```
```

- Further reading (why caching is one of the two hard things in computer science): https://yihui.name/en/2018/06/cache-invalidation/

You can generate animations from R plots

- Requires `install.packages('gifski')`
- Demo

```
```{r, animation.hook='gifski'}
for (i in 0:1) {
  pie(c(i, 6), col = c('red', 'yellow'), labels = NA)
}
```
```

You can generate animations from R plots

- Requires `install.packages('gifski')`
- Demo

```
```{r, animation.hook='gifski'}
for (i in 0:1) {
  pie(c(i, 6), col = c('red', 'yellow'), labels = NA)
}
```
```



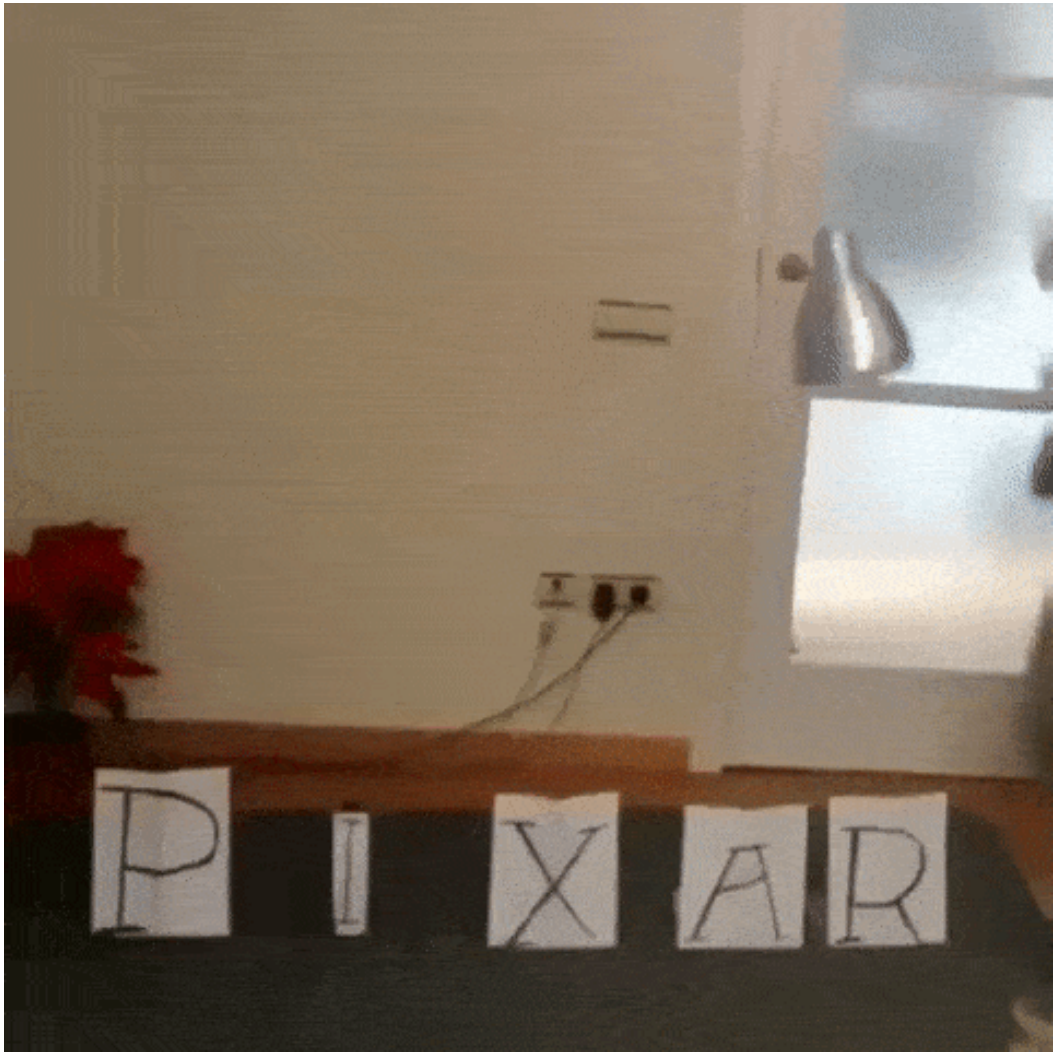
- You may also use FFmpeg (https://ffmpeg.org) (easy to install for macOS users using Homebrew: `brew install ffmpeg`)

  ```{r, animation.hook='ffmpeg', ffmpeg.format='gif', dev='jpeg'}
  for (i in 0:1) {
    pie(c(i, 6), col = c('red', 'yellow'), labels = NA)
  }
  ```

  The animation format is specified by the chunk option `ffmpeg.format`. It could be `gif`, `mp4`, `webm`, or any other formats that FFmpeg supports.

- You may use `cache=TRUE` when the animation takes long time to generate.

- The **gganimate** package works out of the box with **knitr** (`ggplot(...) + transition_*(...)` in a code chunk).


PIXAR

Reuse a code chunk

- If you want to reuse the code from a chunk, don't copy and paste.
- Three ways:
  1. Use the same label, but leave the chunk empty. Useful when you want to run the same code twice with different chunk options.
  2. Use the `ref.label` option, and leave the chunk empty; `ref.label` can be a vector of chunk labels.
  3. Use the `<<chunk-label>>` syntax to embed one chunk in another.
- Demo
- More info: https://yihui.name/knitr/demo/reference/

Child documents

Don't want to write everything in a single document? You can use child documents, and include them in the main document via the `child` option, e.g.,

```
```{r, child=c('one.Rmd', 'another.Rmd')}
```
```

You can also be creative, e.g., conditionally include child documents:

```
```{r, child = if (p.value < 0.05) 'one.Rmd' else 'another.Rmd'}
```
```

Remember: **knitr**'s chunk options can be arbitrary valid R code, so feel free to use `if`-statements.

knitr::knit_expand()

```r
library(knitr)
knit_expand(text = "The value of pi is {{pi}}.")

knit_expand(
  text = "The value of a is {{a}}, so a + 1 is {{a+1}}.",
  a = rnorm(1)
)
```

More info:
https://cran.rstudio.com/web/packages/knitr/vignettes/knit_expand.html

knitr::knit_expand() with file templates

A (child) template document `template.Rmd`:

```
# Regression on {{i}}

```{r lm-{{i}}}
lm(mpg ~ {{i}}, data = mtcars)
```
```

Build linear regression models using all variables against `mpg` in the `mtcars` dataset:

```
Below are ten regression models:

```{r, echo=FALSE, results='asis'}
src = lapply(names(mtcars)[-1], function(i) {
  knitr::knit_expand('template.Rmd')
})
res = knitr::knit_child(text = unlist(src))  # knit the source
cat(res, sep = '\n')
```
```

knitr::fig_chunk()

- When you draw a plot in a code chunk, but want to show it elsewhere (not in the code chunk), `knitr::fig_chunk()` gives you the path to the plot file.

```
```{r cars-plot, fig.show='hide'}
plot(cars)
```

Bla bla...

![a figure moved here](`r knitr::fig_chunk('cars-plot', 'png')`)
```

- More info: https://yihui.name/en/2017/09/knitr-fig-chunk/

knitr::write_bib()

```
knitr::write_bib(c('knitr', 'shiny'))
```

```
@Manual{R-knitr,
  title = {knitr: A General-Purpose Package for Dynamic Report Genera
  author = {Yihui Xie},
  note = {R package version 1.21.6},
  url = {https://yihui.name/knitr/},
  year = {2019},
}
@Manual{R-shiny,
  title = {shiny: Web Application Framework for R},
  author = {Winston Chang and Joe Cheng and JJ Allaire and Yihui Xie
  year = {2018},
  note = {R package version 1.2.0},
  url = {https://CRAN.R-project.org/package=shiny},
}
```

Normally you want to write citation entries to a file (the default is to write to the R console), e.g., `write_bib(..., file = 'references.bib')`.

What I often do:

```
---
title: "My Report"
bibliography: references.bib
---

Cite @R-knitr or [@R-shiny].

```{r, include=FALSE}
knitr::write_bib(c(.packages(), 'shiny'), file = 'references.bib')
```
```

knitr::knit_print()

- Visible objects in code chunks are printed through this S3 generic function
- You can register custom printing methods
- See the vignette for details: https://cran.rstudio.com/web/packages/knitr/vignettes/knit_print.html
- The **printr** package
- Example `03-knit_print.Rmd`

R Markdown output formats

R Markdown output formats

- An output format is an abstraction in **rmarkdown** as a uniform (programming) interface to deal with
  - **knitr** options (chunk options, hooks, package options, ...)
  - pandoc options (`--from`, `--to`, `--bibliography`, ...)
  - pre/post-processors
  - and other options (e.g., whether to keep the intermediate .md)
- Can be created via `rmarkdown::output_format()`
- Note the `base_format` argument: output formats are *extensible*. If you only want to modify a few options of an existing format, you can use it as the base, e.g., you can add a custom post-processor on top of the existing one.

Built-in formats

- `beamer_presentation`
- `github_document`
- `html_document`
- `ioslides_presentation`
- `latex_document`
- `md_document`
- `odt_document`
- `pdf_document`
- `powerpoint_presentation`
- `rtf_document`
- `slidy_presentation`
- `word_document`

YAML options for output formats

The YAML metadata

```
---
output:
  html_document:
    toc: true
    theme: "united"
    fig_height: 6
---
```

will be translated to

```
rmarkdown::render(
  'input.Rmd',
  html_document(
    toc = TRUE,
    theme = "united",
    fig_height = 6
  )
)
```

Example: html_document()

```
str(rmarkdown::html_document())
```

```
List of 11
 $ knitr                  :List of 5
  ..$ opts_knit    : NULL
  ..$ opts_chunk   :List of 5
  .. ..$ dev       : chr "png"
  .. ..$ dpi       : num 96
  .. ..$ fig.width : num 7
  .. ..$ fig.height: num 5
  .. ..$ fig.retina: num 2
  ..$ knit_hooks   : NULL
  ..$ opts_hooks   : NULL
  ..$ opts_template: NULL
 $ pandoc                 :List of 6
  ..$ to          : chr "html"
  ..$ from        : chr "markdown+autolink_bare_uris+ascii_identifiers+tex_ma
  ..$ args        : chr [1:10] "--email-obfuscation" "none" "--self-contained
  ..$ keep_tex    : logi FALSE
  ..$ latex_engine: chr "pdflatex"
  ..$ ext         : NULL
 $ keep_md                : logi FALSE
```

Example: html_document()

Some options:

- `theme`: you can set it to `NULL` to reduce the HTML file size significantly (because of Bootstrap)
- `css`: tweak the styles of certain elements
- `template`: a custom Pandoc template

Pandoc templates

- Official Pandoc templates: https://github.com/jgm/pandoc-templates
- **rmarkdown**'s templates: https://github.com/rstudio/rmarkdown/tree/master/inst/rmd

A minimal HTML template

```
<html>
  <head>
    <title>$title$</title>
    $for(css)$
    <link rel="stylesheet" href="$css$" type="text/css" />
    $endfor$
  </head>

  <body>
  $body$
  </body>
</html>
```

A minimal LaTeX example

```latex
\documentclass{article}
\begin{document}
$body$
\end{document}
```

Simple customization

There are many options you can set in YAML. Two types of options:

- Options for Pandoc: make sure you read the Pandoc manual to know the possible options (e.g., for LaTeX output: https://pandoc.org/MANUAL.html#variables-for-latex).

```
fontsize: 12pt
documentclass: book
monofont: "Source Code Pro"  # for XeLaTeX output
```

- Options for an R Markdown output format in the `output` field in YAML: consult the specific R help page.

You can certainly create your own template, but it may not be necessary to do so if your problem can be solved by setting a few options in YAML.

A crash course on HTML/CSS/JavaScript?

```
output:
  html_document:
    css: ["style.css", "another.css"]
```

or

```
```{css, echo=FALSE}
p {
  color: red;
}
```
```

Learn to use the Developer Tools of your web browser. They are very powerful!

Custom Word/PPT templates

Idea: generate an arbitrary document with Pandoc first, customize the style of this document, and use it as the "reference document".

```
output:
  word_document:
    reference_docx: "word-template.docx"
  powerpoint_presentation:
    reference_doc: "powerpoint-template.pptx"
```

PowerPoint output requires Pandoc 2.x, which has been bundled in RStudio 1.2.x (currently a preview version).

Deeper customization

A common use case: inject a snippet of code to the HTML `<head>` (e.g., JS/CSS code), or the LaTeX preamble (e.g., load some LaTeX packages before `\begin{document}`).

```
output:
  html_document:
    includes:
      in_header: "header.html"
      before_body: "before.html"
      after_body: "after.html"
  pdf_document:
    includes:
      in_header: "preamble.tex"
```

Even deeper customization? Sure, write a package with custom output formats! Let's study a few relatively simple examples in **rmarkdown** first.

Take a deep breath and read some source code!

Example: latex_fragment

```
rmarkdown::latex_fragment
```

```
function (...)
{
    latex_document(..., template = rmarkdown_system_file("rmd/fragment/defaul
}
<bytecode: 0x7fcc33b405f0>
<environment: namespace:rmarkdown>
```

- https://github.com/rstudio/rmarkdown/blob/b209cdc/R/pdf_document.R#L252-L256
- The key: use a custom template https://github.com/rstudio/rmarkdown/blob/master/inst/rmd/fragment/default.tex
- Similarly: https://github.com/rstudio/rmarkdown/blob/master/R/html_fragment.R and https://github.com/rstudio/rmarkdown/blob/master/inst/rmd/fragment/default.html

Example: powerpoint_presentation

A minimal example of the PowerPoint output format (not really `rmarkdown::powerpoint_presentation`):

```
powerpoint_presentation = function(pandoc_args = NULL) {
  rmarkdown::output_format(
    knitr = list(),
    pandoc = rmarkdown::pandoc_options(
      to = 'pptx',
      args = pandoc_args
    )
  )
}
```

Example: rtf_document

- https://github.com/rstudio/rmarkdown/blob/master/R/rtf_document.R
- pre-processor (protect raw RTF content)
- post-processor (restore raw RTF content)
- raw RTF looks like this

```
{\rtf1\ansi{\fonttbl\f0\fswiss Helvetica;}\f0\pard
This is some {\b bold} text.\par
}
```

Custom Templates and Formats

Hao Zhu's session

https://arm.rbind.io/days/day2/

Shiny documents

Shiny documents vs Shiny apps

- R Markdown + `runtime: shiny` in YAML
- In a Shiny document, you render output wherever you need it in the document. No need to write a UI. A Shiny app requires both a UI and the server logic (`shiny::shinyApp(ui = ..., server = ...)`).
- In other words, the R Markdown document itself is the *implicit* UI.

Render output inline

- I assume most people are familiar with using `shiny::renderXXX()` in code blocks.
- You can also `renderXXX()` in an inline R expression in R Markdown.
- https://shiny.rstudio.com/gallery/inline-output.html
- Source: https://github.com/rstudio/shiny-examples/blob/master/026-shiny-inline/index.Rmd

Render output inline

- I assume most people are familiar with using `shiny::renderXXX()` in code blocks.
- You can also `renderXXX()` in an inline R expression in R Markdown.
- https://shiny.rstudio.com/gallery/inline-output.html
- Source: https://github.com/rstudio/shiny-examples/blob/master/026-shiny-inline/index.Rmd
- Potential application: a recipe website? I really need this for making moon cakes.

Delayed rendering

- Wrap your `renderXXX()` in `rmarkdown::render_delayed()` to delay rendering output until the document has been compiled.
- Useful when the Shiny output takes long time to render.
- Demo

```
```{r, echo = FALSE}
numericInput("rows", "How many cars?", 5)

rmarkdown::render_delayed({
  # Sys.sleep(6)
  renderTable({
    head(cars, input$rows)
  })
})
```
```

HTML widgets

HTML widgets

- (Often interactive) JavaScript applications created from R and displayed on HTML pages
- Can be viewed (1) as a standalone page when printed in the R console (2) in R Markdown output documents (HTML) (3) in Shiny apps
- You can pretty much think them like normal R plots
- See Chapter 16 of the R Markdown book

The three components

- R binding: pass data and options from R to JS
- JS binding: receive data from R and create the widget
- A YAML configuration file to specify HTML/JS/CSS dependencies

A self-contained minimal example

```
blink = function(text, interval = 1) {
  htmlwidgets::createWidget(
    'blink',
    x = list(text = text, interval = interval),
    dependencies = htmltools::htmlDependency(
      'blink', '0.1', src = c(href = ''), head = '<script>
HTMLWidgets.widget({
  name: "blink", type: "output",
  factory: function(el, width, height) {
    return {
      renderValue: function(x) {
        setInterval(function() {
          el.innerText = el.innerText == "" ? x.text : "";
        }, x.interval * 1000);
      },
      resize: function(width, height) {}
    };
  }
});
</script>'))}
```

Example: the sigma package

- Source: https://github.com/jjallaire/sigma
- sigma.js: http://sigmajs.org
- Basic file structure:

```
R/
| sigma.R

inst/
|-- htmlwidgets/
|    |-- sigma.js
|    |-- sigma.yaml
|    |-- lib/
|    |    |-- sigma-1.0.3/
|    |    |    |-- sigma.min.js
|    |    |    |-- plugins/
|    |    |    |    |-- sigma.parsers.gexf.min.js
```

sigma.yaml

```yaml
dependencies:
  - name: sigma
    version: 1.0.3
    src: htmlwidgets/lib/sigma-1.0.3
    script:
      - sigma.min.js
      - plugins/sigma.parsers.gexf.min.js
```

sigma.R

```r
sigma = function(gexf, drawEdges = TRUE, drawNodes = TRUE,
                 width = NULL, height = NULL) {

  # read the gexf file
  data = paste(readLines(gexf), collapse = "\n")

  # create a list that contains the settings
  settings = list(drawEdges = drawEdges, drawNodes = drawNodes)

  # pass the data and settings using 'x'
  x = list(data = data, settings = settings)

  # create the widget
  htmlwidgets::createWidget(
    "sigma", x, width = width, height = height
  )
}
```

sigma.js

```
HTMLWidgets.widget({
  name: "sigma",
  factory: function(el, width, height) {
    // create our sigma object and bind it to the element
    var sig = new sigma(el.id);
    return {
      renderValue: function(x) {
        // apply settings
        for (var name in x.settings)
          sig.settings(name, x.settings[name]);
        // update the sigma object
        sigma.parsers.gexf(
          x.data, sig,
      ...
```

Demo

```
remotes::install_github("jjallaire/sigma")

sigma::sigma(
  system.file("examples/ediaspora.gexf.xml", package = "sigma")
)
```

Shiny output wrappers

```r
# UI wrapper
sigmaOutput = function(outputId, width = "100%", height = "400px") {
  htmlwidgets::shinyWidgetOutput(
    outputId, "sigma", width, height, package = "sigma"
  )
}

# use in the server logic
renderSigma = function(expr, env = parent.frame(), quoted = FALSE) {
  if (!quoted) { expr = substitute(expr) } # force quoted
  htmlwidgets::shinyRenderWidget(
    expr, sigmaOutput, env, quoted = TRUE
  )
}
```

HTML widgets for non-HTML output

- HTML widgets are for HTML output formats (of course!). What if we embed a widget in a PDF document? In this case, **knitr** will take a screenshot of the widget automatically if you have installed **webshot** and PhantomJS:

```
install.packages("webshot")
webshot::install_phantomjs()
```

- Demo

```
```{r}
DT::datatable(iris)
```
```

Misc topics (time permitting)

knitr hooks

- Chunk hooks: you can run extra code before/after each code chunk
- Output hooks: you have control over every single piece of the output (text, plots, messages)
- https://yihui.name/knitr/hooks/

Chunk hooks

A chunk hook is a function with three arguments (the latter two are optional). Register the hook function via `knitr::knit_hooks$set()`:

```
knitr::knit_hooks$set(HOOK_NAME = function(before, options, envir) {
  if (before) {
    # code to run before a chunk
  } else {
    # code to run after a chunk
  }
})
```

Chunk hooks are triggered when the corresponding chunk option is not `NULL`, e.g.,

```
```{r include=FALSE}
knitr::knit_hooks$set(small_mar = function(before, options, envir) {
  if (before) par(mar = c(4, 4, .1, .1))
})
```

```{r small_mar=TRUE}
plot(cars)
```
```

Of course, you can set the option globally so that the hook is executed for all chunks:

```
knitr::opts_chunk$set(small_mar = TRUE)
```

Use your imagination.

```r
knitr::knit_hooks$set(tweet = function(before, options, envir) {
  if (before) {
    rtweet::post_message("I have started the computation, my lord.")
  } else {
    rtweet::post_message("The MCMC has converged!")
  }
})
```

Output hooks

Hook names preserved for output:

```
names(knitr::knit_hooks$get())
```

```
##  [1] "source"          "output"
##  [3] "warning"         "message"
##  [5] "error"           "plot"
##  [7] "inline"          "chunk"
##  [9] "text"            "evaluate.inline"
## [11] "evaluate"        "document"
```

```
knitr::knit_hooks$get('inline')
```

```
## function(x) {
##       fmt = pandoc_to()
##       fmt = if (length(fmt) == 1L) 'latex' else 'html'
##       .inline.hook(format_sci(x, fmt))
##     }
## <bytecode: 0x7fcc33df1620>
## <environment: 0x7fcc1ea5a7a8>
```

Example: truncate long text output

Example 052: https://github.com/yihui/knitr-examples/

```
knitr::knit_hooks$set(output = local({
  # the default output hook
  hook_output = knitr::knit_hooks$get('output')

  function(x, options) {
    if (!is.null(n <- options$out.lines)) {
      x = knitr:::split_lines(x)
      if (length(x) > n) {
        # truncate the output
        x = c(head(x, n), '....\n')
      }
      x = paste(x, collapse = '\n') # paste first n lines together
    }
    hook_output(x, options)
  }
}))
```

```r
1:100  # normal and full output
```

```
  [1]   1   2   3   4   5   6   7   8   9  10  11
 [12]  12  13  14  15  16  17  18  19  20  21  22
 [23]  23  24  25  26  27  28  29  30  31  32  33
 [34]  34  35  36  37  38  39  40  41  42  43  44
 [45]  45  46  47  48  49  50  51  52  53  54  55
 [56]  56  57  58  59  60  61  62  63  64  65  66
 [67]  67  68  69  70  71  72  73  74  75  76  77
 [78]  78  79  80  81  82  83  84  85  86  87  88
 [89]  89  90  91  92  93  94  95  96  97  98  99
[100] 100
```

```r
```{r out.lines=3}
1:100  # truncated output
```
```

```
  [1]   1   2   3   4   5   6   7   8   9  10  11
 [12]  12  13  14  15  16  17  18  19  20  21  22
 [23]  23  24  25  26  27  28  29  30  31  32  33
....
```

knitr's language engines

See Section 2.7 of the R Markdown book for some examples.

```
names(knitr::knit_engines$get())
```

```
##  [1] "awk"       "bash"      "coffee"
##  [4] "gawk"      "groovy"    "haskell"
##  [7] "lein"      "mysql"     "node"
## [10] "octave"    "perl"      "psql"
## [13] "Rscript"   "ruby"      "sas"
## [16] "scala"     "sed"       "sh"
## [19] "stata"     "zsh"       "highlight"
## [22] "Rcpp"      "tikz"      "dot"
## [25] "c"         "fortran"   "fortran95"
## [28] "asy"       "cat"       "asis"
## [31] "stan"      "block"     "block2"
## [34] "js"        "css"       "sql"
## [37] "go"        "python"    "julia"
## [40] "py"        "upper"
```

For curious hackers: https://github.com/yihui/knitr/blob/master/R/engine.R

A minimal Python engine

You can execute Python code via the command line `python -c 'YOUR CODE'`.

```
knitr::knit_engines$set(py = function(options) {
  code = paste(options$code, collapse = '\n')
  out = system2('python', c('-c', shQuote(code)), stdout = TRUE)
  knitr::engine_output(options, code, out)
})
```

Now you can use the new engine `py`, e.g.,

```
```{py}
print(1 + 1)
```
```

Use your imagination. Language engines don't have to involve command-line tools. I give you the code and chunk options. You do whatever you like.

```
knitr::knit_engines$set(upper = function(options) {
  if (!options$eval) return()  # don't run this chunk
  code = paste(options$code, collapse = '\n')
  toupper(code)
})
```

```
```{upper}
Hello, knitr engines!
```
```

HELLO, KNITR ENGINES!

Thank you!

All materials can be found at https://arm.rbind.io

You will receive an email request to fill out a workshop feedback survey at the end of the day. We will truly appreciate it if you could fill it out to help us improve our workshops in the future.