Knit Rmd

R Markdown is a powerful tool for combining analysis and reporting into the same document. R Markdown has grown substantially from a package that supports a few output formats, to an extensive and diverse ecosystem that supports the creation of books, blogs, scientific articles, websites, and even resumes.

Nice documentations

- `rmarkdown` package CRAN
  - Package CRAN page
  - Reference manual
- `bookdown` package CRAN
  - Package CRAN page
  - Reference manual
- R markdown: The definitive guide. provides detailed references
- R markdown cookbook: concise and covers essential functions, with examples.
- Authoring Books with R Markdown: with a focus on `bookdown`.

Q: What is the difference between Rmd and R script?
A:

- An R script (`.R`) is used for developing and troubleshooting code; a place where you can store reusable code fragments.
- An R Markdown file (`.Rmd`) is used to integrate R commands with explanatory text and output, making it useful for creating reports.

Quick takeaways:

- Can still use horizontal separator `ctrl + shift + S` for dashed lines and `ctrl + shift + =` for equals
- Headers must have one empty line above and below to separate it from other text

---

YAML metadata

Q: What is YAML?
A: YAML is a human-friendly data serialization language for all programming languages. YAML stands for "Yet Another Markup Language."

Q: What does YAML do?
A: It is placed at the very beginning of the document and is read by each of Pandoc, **rmarkdown**, and **knitr**.

- Provide metadata of the document.
- located at the top of the file.
- adheres to the YAML format and is delimited by lines containing three three dashes (`---`).

YAML also called header and front matter.

See HERE for commonly used YAML metadata (header) in different R Markdown output formats.

YAML can set values of the template variables, such as `title`, `author`, and `date` of the document.

- The `output` field is used by rmarkdown to apply the output format function `rmarkdown::html_document()` in the rendering process.

  There are two types of output formats in the **rmarkdown** package: documents (e.g., `pdf_document`), and presentations (e.g., `beamer_presentation`).

  Supported output format examples: `html_document`, `pdf_document`.

  R Markdown documents (`html_documents`) and R Notebook documents (`html_notebook`) are very similar; in fact, an R Notebook document is a special type of R Markdown document. The main

difference is using R Markdown document (`html_documents`) you have to knit (render) the entire document each time you want to preview the document, even if you have made a minor change. However, using an R Notebook document (`html_notebook`) you can view a preview of the final document without rendering the entire document.

Troubleshooting

Issue: `bookdown` always output html, even if specified to pdf.
Cause: If it produces HTML, the output format must have been provided somewhere.
Fix: Check if you have a _output.yml under the root directory of your book project. If you do, you may delete it. Then bookdown will use the output field that you specified in the YAML frontmatter of your Rmd document.

`bookdown` wrappers of base markdown format

`bookdown` output formats allow numbering and cross-referencing figures/tables/equations. It takes the format `html_document2`, in general, `markdown_document2` is a wrapper for the base format `markdown_document`. With the `bookdown` output format, you can cross-reference sections by their ID's using the same syntax when sections are numbered.

Other bookdown output format examples: `pdf_document2`, `beamer_presentation2`, `tufte_html2`, `word_document2`. See Page 12 of the reference manual for a complete list of supported format by `bookdown`.

- Many aspects of the LaTeX template used to create PDF documents can be customized using **top-level** YAML metadata (note that these options do **NOT** appear underneath the `output` section, but rather appear at the top level along with `title`, `author`, and so on). For example:

```
---
title: "Crop Analysis Q3 2013"
output: pdf_document
fontsize: 11pt
geometry: margin=1in
---
```

  A few available metadata variables are displayed in the following (consult the Pandoc manual for the full list):

Top-level YAML Variable	Description
`lang`	Document language code
`fontsize`	Font size (e.g., `10pt`, `11pt`, or `12pt`)
`papersize`	Defines the paper size (e.g., `a4paper`, `letterpaper`)
`documentclass`	LaTeX document class (e.g., `article`, `book`, and `report`)
`classoption`	A list of options to be passed to the document class, e.g., you can create a two-column document with the `twocolumn` option.
`geometry`	Options for `geometry` package (e.g., `margin=1in` set all margins to be 1 inch)
`mainfont`, `sansfont`, `monofont`, `mathfont`	Document fonts (works only with `xelatex` and `lualatex`)
`linkcolor`, `urlcolor`, `citecolor`	Color for internal links (cross references), external links (link to websites), and citation links (bibliography)
`linestretch`	Options for line spacing (e.g. 1, 1.5, 3).

Pandoc User's Guide: https://www.uv.es/wiki/pandoc_manual_2.7.3.wiki?21

`classoption`

- **`onecolumn, twocolumn`** - Instructs LaTeX to typeset the document in one column or two columns.
- **`twoside, oneside`**: Specifies whether double or single sided output should be generated. The classes' article and report are single sided and the book class is double sided by default.

  Note that this option concerns the style of the document only. The option two side does *NOT* tell the printer you use that it should actually make a two-sided printout.

  The difference between single-sided and double-sided documents in LaTeX lies in the layout of the page margins and the orientation of the text on the page.

  * Single-sided documents are printed on only one side of the page, with the text and images aligned to the right-hand side of the page. This type of layout is often used for brochures, flyers, and other types of promotional materials.
  * Double-sided documents are printed on both sides of the page, with the text and images alternating between right-hand and left-hand margins. This type of layout is often used for **books**, reports, and other types of long-form documents.

    A `twoside` document has different margins and headers/footers for odd and even pages.

  The layout of a `twoside` book

  Q: Why Inner margin is narrow?
  A: The reason for this is that with two pages side by side, you actually have only THREE margins - the left, right and middle. The middle margin is made up from the inside margins of both pages, and so these are smaller because they add together to make the middle margin. If they were bigger, then you would end up with too much whitespace in the middle.

```
o - outside margin
i - inside margin
b - binding offset

Before binding:
-----------------    -----------------
|oooo|~~~~~~|ii|b|    | |  |~~~~~~|   |
|    |~~~~~~|  | |    | |  |~~~~~~|   |
|    |~~~~~~|  | |    | |  |~~~~~~|   |
|    |~~~~~~|  | |    | |  |~~~~~~|   |
|    |~~~~~~|  | |    | |  |~~~~~~|   |
|    |~~~~~~|  | |    | |  |~~~~~~|   |
-----------------    -----------------

After binding:
-------------------------------
|oooo|~~~~~~|ii|ii|~~~~~~|oooo|
|oooo|~~~~~~|ii|ii|~~~~~~|oooo|
|oooo|~~~~~~|ii|ii|~~~~~~|oooo|
|oooo|~~~~~~|ii|ii|~~~~~~|oooo|
|oooo|~~~~~~|ii|ii|~~~~~~|oooo|
|oooo|~~~~~~|ii|ii|~~~~~~|oooo|
-------------------------------
```

- **`landscape`** - Changes the layout of the document to print in landscape mode.
- **`openright, openany`** - Makes chapters begin either only on right hand pages or on the next page

available. This does not work with the article class, as it does not know about chapters. The report class by default starts chapters on the next page available and the book class starts them on right hand pages.

- In PDFs, you can use code, typesetting commands (e.g., `\vspace{12pt}`), and specific packages from LaTeX.

  1. The `header-includes` option loads LaTeX packages.

```
---
output: pdf_document
header-includes:
  - \usepackage{fancyhdr}
---

\pagestyle{fancy}
\fancyhead[LE,RO]{Holly Zaharchuk}
\fancyhead[LO,RE]{PSY 508}

# Problem Set 12
```

Common `header-includes`:

  – Chinese/Japanese support

```
---
output: pdf_document
header-includes:
  - \usepackage{ctex}
---
```

  2. Alternatively, use `extra_dependencies` to list a character vector of LaTeX packages. This is useful if you need to load multiple packages:

```
---
title: "Untitled"
output:
  pdf_document:
    extra_dependencies: ["bbm", "threeparttable"]
---
```

If you need to specify options when loading the package, you can add a second-level to the list and provide the options as a list:

```
---
title: "Untitled"
output:
  pdf_document:
    extra_dependencies:
      caption: ["labelfont={bf}"]
      hyperref: ["unicode=true", "breaklinks=true"]
      lmodern: null
---
```

Here are some examples of LaTeX packages you could consider using within your report:

  – pdfpages: Include full PDF pages from an external PDF document within your document.
  – caption: Change the appearance of caption subtitles. For example, you can make the figure title italic or bold.
  – fancyhdr: Change the style of running headers of all pages.

- Some output options are passed to Pandoc, such as `toc`, `toc_depth`, and `number_sections`. You should consult the Pandoc documentation when in doubt.

```
---
output:
  pdf_document:
    toc: true
    keep_tex: true
---
```

  - `keep_tex: true` if you want to keep intermediate TeX. Easy to debug. Defaults to `false`.

  To learn which arguments a format takes, read the format's help page in R, e.g. **`?html_document`**.

Parameters

We can include variables and R expressions in this header that can be referenced throughout our R Markdown document. For example, the following header defines `start_date` and `end_date` parameters, which will be reflected in a list called `params` later in the R Markdown document.

```
---
title: My RMarkdown
author: Yihui Xie
output: html_document
params:
  start_date: '2020-01-01'
  end_date: '2020-06-01'
---
```

To access a parameter in our R code, call `params$<parameter name>`, e.g., `params$start_date` and `params$end_date`.

Should I use quotes to surround the values?

- Whenever applicable use the unquoted style since it is the most readable.
- Use quotes when the value can be misinterpreted as a data type or the value contains a `:`.

```
# values need quotes
foo: '{{ bar }}' # need quotes to avoid interpreting as `dict` object
foo: '123'       # need quote to avoid interpreting as `int` object
foo: 'yes'           # avoid interpreting as `boolean` object
foo: "bar:baz:bam" # has colon, can be misinterpreted as key

# values need not quotes
foo: bar1baz234
bar: 123baz
```

ref:

- R Markdown anatomy, R Markdown Cookbook
- https://rmarkdown.rstudio.com/lesson-6.html

Compile an Rmd

You can have more than output formats for your Rmd. For example, you want both the html and pdf output.

When you render the Rmd with `rmarkdown::render()`, it will use the **first output format you specify in the YAML metadata** (if it is missing, the default is `html_document`).

do not want to use the first one, you can specify the one you want in the second argument, e.g., for an Rmd document `foo.Rmd` with the metadata:

```
output:
  html_document:
    toc: true
  pdf_document:
    keep_tex: true
```

You can render it to PDF via:

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

- RStudio calls the function `rmarkdown::render()` to render the document in **a new R session**.

  RStudio does this to ensure **reproducibility**.

Document dependency

By default, R Markdown produces standalone HTML files with no external dependencies, using `data:`URIs to incorporate the contents of linked scripts, stylesheets, images, and videos. This means you can share or publish the file just like you share Office documents or PDFs. If you would rather keep dependencies in external files, you can specify `self_contained: false`.

Note that even for self-contained documents, MathJax is still loaded externally (this is necessary because of its big size). If you want to serve MathJax locally, you should specify `mathjax: local` and `self_contained: false`.

One common reason to keep dependencies external is for serving R Markdown documents from a website (external dependencies can be cached separately by browsers, leading to faster page load times). In the case of serving multiple R Markdown documents you may also want to consolidate dependent library files (e.g. Bootstrap, and MathJax, etc.) into a single directory shared by multiple documents. You can use the `lib_dir` option to do this. For example:

```
---
title: "Habits"
output:
  html_document:
    self_contained: false
    lib_dir: libs
---
```

Loading LaTeX packages

We can load additional LaTeX packages using the `extra_dependencies` option **within** the `pdf_document` YAML settings.

This allows us to provide a list of LaTeX packages to be loaded in the intermediate LaTeX output document, e.g.,

```
---
title: "Using more LaTeX packages"
output:
  pdf_document:
    extra_dependencies: ["bbm", "threeparttable"]
---
```

If you need to **specify options** when loading the package, you can add a sub-level to the list and provide the options as a list, e.g.,

```
output:
  pdf_document:
    extra_dependencies:
      caption: ["labelfont={bf}"]
      hyperref: ["unicode=true", "breaklinks=true"]
      lmodern: null
```

For those familiar with LaTeX, this is equivalent to the following LaTeX code:

```
\usepackage[labelfont={bf}]{caption}
\usepackage[unicode=true, breaklinks=true]{hyperref}
\usepackage{lmodern}
```

The advantage of using the `extra_dependencies` argument over the `includes` argument introduced in Section 6.1 is that you do not need to include an external file, so your Rmd document can be **self-contained**.

Includes

html output

You can do more advanced customization of output by including additional HTML content or by replacing the core Pandoc template entirely. To include content in the document header or before/after the document body, you use the `includes` option as follows:

```
---
title: "Habits"
output:
  html_document:
    includes:
      in_header: header.html
      before_body: doc_prefix.html
      after_body: doc_suffix.html
---
```

An example `header.html` to load a MathJax extension `textmacros`.

```
<script type="text/x-mathjax-config">
  MathJax.Hub.Config({
    loader: {load: ['[tex]/textmacros']},
    tex: {packages: {'[+]': ['textmacros']}}
  });
</script>
```

---

pdf output

For example, to support Chinese characters.

You can use `includes` and `preamble.tex` (can be any name, contains any pre-loaded latex code you want to run before your main text code, for setting up environment, loading pkgs, define new commands ... Very flexible.)

In the main Rmd:

```
---
output:
  pdf_document:
    includes:
      in_header: preamble.tex
---
```

In `preamble.tex`:

```
\usepackage{xeCJK}
\setCJKmainfont{Noto Sans CJK SC}
```

Alternatively, you can use `header-includes` but with less flexibility to change options:

```
---
output: pdf_document
header-includes:
  - \usepackage{ctex}
---
```

Ref: https://github.com/hao203/rmarkdown-YAML?tab=readme-ov-file#chinesejapanese-support

---

Chunk Options

If you want to set chunk options globally, call `knitr::opts_chunk$set()` in a code chunk (usually the first one in the document), e.g.,

````
```{r, label="setup", include=FALSE}
knitr::opts_chunk$set(
  comment = "#>", echo = FALSE, fig.width = 6
)
```
````

Full list of chunk options: https://yihui.org/knitr/options/

Chunk options can customize nearly all components of code chunks, such as the source code, text output, plots, and the language of the chunk.

Other languages are supported in `Rmd`

You can list the names of all available engines via:

```
names(knitr::knit_engines$get())
##  [1] "awk"          "bash"         "coffee"
##  [4] "gawk"         "groovy"       "haskell"
##  [7] "lein"         "mysql"        "node"
## [10] "octave"       "perl"         "php"
## [13] "psql"         "Rscript"      "ruby"
## [16] "sas"          "scala"        "sed"
## [19] "sh"           "stata"        "zsh"
## [22] "asis"         "asy"          "block"
## [25] "block2"       "bslib"        "c"
## [28] "cat"          "cc"           "comment"
## [31] "css"          "ditaa"        "dot"
## [34] "embed"        "eviews"       "exec"
## [37] "fortran"      "fortran95"    "go"
## [40] "highlight"    "js"           "julia"
## [43] "python"       "R"            "Rcpp"
```

```
## [46] "sass"         "scss"         "sql"
## [49] "stan"         "targets"      "tikz"
## [52] "verbatim"     "theorem"      "lemma"
## [55] "corollary"    "proposition"  "conjecture"
## [58] "definition"   "example"      "exercise"
## [61] "hypothesis"   "proof"        "remark"
## [64] "solution"     "marginfigure"
```

The engines from `theorem` to `solution` are only available when you use the **bookdown** package, and the rest are shipped with the **knitr** package.

To use a different language engine, you can change the language name in the chunk header from `r` to the engine name, e.g.,

````
```python
x = 'hello, python world!'
print(x.split(' '))
```
````

For engines that rely on external interpreters such as `python`, `perl`, and `ruby`, the default interpreters are obtained from `Sys.which()`, i.e., using the interpreter found via the environment variable `PATH` of the system. If you want to use an alternative interpreter, you may specify its path in the chunk option `engine.path`.

For example, you may want to use Python 3 instead of the default Python 2, and we assume Python 3 is at `/usr/bin/python3`

````
```{python, engine.path = '/usr/bin/python3'}
import sys
print(sys.version)
```
````

- All outputs support markdown syntax.
- If the output is html, you can write in html syntax.

The **chunk label** for each chunk is assumed to be unique within the document. This is especially important for cache and plot filenames, because these filenames are based on chunk labels. Chunks without labels will be assigned labels like `unnamed-chunk-i`, where `i` is an incremental number.

- Chunk label doesn't need a `tag`, i.e., you only provide the `value`.
- If you prefer the form `tag=value`, you could also use the chunk option `label` explicitly, e.g.,

  ````
  ```{r, label='my-chunk'}
  # one code chunk example
  ```
  ````

You may use `knitr::opts_chunk$set()` to change the default values of chunk options in a document.

Commonly used chunk options

- Complete list here. Or `?opts_chunk` to get the help page.

Options	Definitions
`echo=TRUE`	Whether to display the **source code** in the output document.Use this when you want to show the output but not the code itself.
`eval=TRUE`	Whether to evaluate the code chunk.

Options	Definitions
`include=TRUE`	Whether to include the chunk **output** in the output document. If `FALSE`, nothing will be written into the output document, but the code is still evaluated and plot files are generated if there are any plots in the chunk, so you can manually insert figures later.
`message=TRUE`	Whether to preserve messages emitted by `message()`
`warning=TRUE`	Whether to show warnings in the output produced by `warning()`.
`results='markup'`	Controls how to display the text results. When `results='markup'` that is to write text output as-is, i.e., write the raw text results directly into the output document without any markups.Useful when priting `stargazer` tables.
`comment='##'`	The prefix to be added before each line of the text output. Set `comment = ''` remove the default `##`.
`collapse=FALSE`	Whether to, if possible, collapse all the source and output blocks from one code chunk into a single block (by default, they are written to separate blocks). This option only applies to Markdown documents.
`fig.keep='high'`	How plots in chunks should be kept. `high`: Only keep high-level plots (merge low-level changes into high-level plots). `none`: Discard all plots. `all`: Keep all plots (low-level plot changes may produce new plots). `first`: Only keep the first plot. `last`: Only keep the last plot. If set to a numeric vector, the values are indices of (low-level) plots to keep.If you want to choose the second to the fourth plots, you could use `fig.keep = 2:4` (or remove the first plot via `fig.keep = -1`).
`fig.align="center"`	Figure alignment.
`fig.pos="H"`	A character string for the figure position arrangement to be used in `\begin{figure}[]`.
`fig.cap`	Figure caption.

`results='markup'` note plural form for result**s**.

- `markup`: Default. Mark up text output with the appropriate environments depending on the output format. For example, for R Markdown, if the text output is a character string `"[1] 1 2 3"`, the actual output that **knitr** produces will be:

  ```
  ```
  [1] 1 2 3
  ```
  ```

  In this case, `results='markup'` means to put the text output in fenced code blocks (“‘).

- `asis`: Write text output as-is, i.e., write the raw text results directly into the output document without any markups.

  ```
  ```{r, results='asis'}
  cat("I'm raw **Markdown** content.\n")
  ```
  ```

  Sometime, you encounter the following error messages when you have R codes within `enumerate` environment.

  > You can't use `macro parameter character #` in horizontal mode.

  By default, knitr prefixes R output with `##`, which can't be present in your TeX file.

  Solution:

  – specify `results="asis"` in code chunks.

- `hold`: Hold all pieces of text output in a chunk and flush them to the end of the chunk.

- `hide` (or `FALSE`): Hide text output.

`collapse=FALSE` Whether to merge text output and source code into a single code block in the output. The default `FALSE` means R expressions and their text output are separated into different blocks.

`collapse = TRUE` makes the output more compact, since the R source code and its text output are displayed in a single output block. The default `collapse = FALSE` means R expressions and their text output are separated into different blocks.

Print Verbatim R code chunks

verbatim in line code

- use `knitr::inline_expr`.

```
---
title: "Test inline expr"
output: html_document
---

To use `chunk_reveal("walrus", title = "## Walrus operator")` inline, you can wrap it in R inline chunk
```

Including verbatim R code chunks inside R Markdown

One solution for including verbatim R code chunks (see below for more) is to insert hidden inline R code (`` `r ''` ``) immediately before or after your R code chunk.

- The hidden inline R code will be evaluated as an inline expression to an empty string by knitr.

Then wrap the whole block within a markdown code block. The rendered output will display the verbatim R code chunk — including backticks.

R code generating the four backticks block:

```
output_code <-
"````markdown
```{r}
plot(cars)
``` \n````"
cat(output_code)
```

Write this code in your R Markdown document:

````
````markdown
`r ''````{r}
plot(cars)
```
````
````

or

````
````markdown
```{r}`r '''
plot(cars)
```
````
````

Knit the document and the code will render like this in your output:
````

```
```{r}
plot(cars)
```
```

This method makes use of **Markdown Syntax** for code.

Q: What is the Markdown Syntax for code?
A:

- Inline code use a pair of backticks, e.g., `` `code` ``. To use n literal backticks, use at least $n+1$ backticks outside. Note that use a space to separate your outside backticks from your literal backtick(s). For example, to generate `` `code` ``, you use ``` ``␣`code`␣`` ``` (i.e., two backticks + space + one backtick + `code` + one backtick + space + two backticks). Note that you need to write sequentially.
- Plain code blocks can be written either
  - After three or more backticks (fenced code blocks), or

    Can also use tildes (~)
  - Indent the blocks by four spaces (indented code blocks)

    Special characters do not trigger special formatting, and all spaces and line breaks are preserved. Blank lines in the verbatim text need not begin with four spaces.
- Note that code blocks must be separated from surrounding text by blank lines.

If the code itself contains a row of tildes or backticks, just use a longer row of tildes or backticks at the start and end:

```
~~~~~~~~~~~~~~~~
~~~~~~~~~~
code including tildes
~~~~~~~~~~
~~~~~~~~~~~~~~~~
```

These begin with a row of three or more tildes (~) and end with a row of tildes that must be at least as long as the starting row.

A trick if you don't want to type more than three tildes or backticks is that you just use different inner and outer symbols.

```
~~~markdown
```r
print ("hello world")
```
~~~
```

Will be rendered as:

```
```r
print ("hello world")
```
```

A shortcut form (without braces) can also be used for specifying the language of the code block:

```
```haskell
qsort [] = []
```
```

This is equivalent to:

```` {.haskell}
qsort [] = []
````

`haskell` is the language class.

You can add more classes, such as `numberLines` for adding line numbers.

This shortcut form may be combined with attributes:

````haskell {.numberLines}
qsort [] = []
````

Which is equivalent to:

```` {.haskell .numberLines}
qsort [] = []
````

and

```
<pre id="mycode" class="haskell numberLines" startFrom="100">
  <code>
  primes = filterPrime [2..] where
  filterPrime (p:xs) =
    p : filterPrime [x | x <- xs, x `mod` p /= 0]
  </code>
</pre>
```

If highlighting is supported for your output format and language, then the code block above will appear highlighted, with numbered lines starting with 100, 101, and go on.

Code chunks within `enumerate`

- Mind the indentation. Rstudio does not automatically adjust indentation for codes.
- specify `results="asis"` if encounter

  > You can't use 'macro parameter character #' in horizontal mode.

- cross references using bookdown (`\@ref{fig:scatter-plot}`) might not work.

  Use latex references `\ref{fig:scatter-plot}` (base latex) or `\autoref{fig:scatter-plot}` (from `hyperref` package)

- markdown language does not work well inside latex environments. A possible workaround is use `1` and indent four spaces for contents that follow.

If it is still a pain in the ass, use this solution.

Basically, just copy the output from R condole and paste in Rmd.

References:

https://yihui.org/en/2017/11/knitr-verbatim-code-chunk/

https://support.posit.co/hc/en-us/articles/360018181633-Including-verbatim-R-code-chunks-inside-R-Markdown

https://themockup.blog/posts/2021-08-27-displaying-verbatim-code-chunks-in-xaringan-presentations/

Pandoc's Markdown: https://pandoc.org/MANUAL.html#fenced-code-blocks