Short Tutorial of R Markdown :lab:math6312:R:R_Markdown:

Overview

R Markdown has two advantages that are of interest to a researcher. The first is it allows the results of R code to be directly inserted into formatted documents. The second advantage is it is incredibly easy to use. This ease is a result of R Markdown only using a small set of features and this reduces the complexity of the needed commands. This set of features supports the most commonly used formatting, resulting in the ability to create most documents. These features make R Markdown documents easy to write and the process less error prone.

This guide assumes that you are using R Studio, and the source `rmd` file lives at lab2.rmd. After you download it, please rename it to `lab2.rmd`, and open it in R Studio.

Reproducible research

A minimal standard for data analysis and other scientific computations is the reproducibility. That is, the method, code and data are assembled in a way so that another group can recreate all of the results.

Recommendations for reproducible research

1. Encapsulate the full project into one directory that is supported with version control.
2. Release your code and data.
3. Document everything (method, workflow, when/how you downloaded data, software version, etc).
4. Make figures, tables, and statistics the results of scripts.
5. Write code that uses relative paths.
6. Always set your random seed.

Markup and Markdown

The essential idea in a markup language is that it consists of ordinary text, plus signs which indicate how to change the formatting or meaning of the text. Markdown is a lightweight markup language with plain text formatting syntax. It is intended to be as easy-to-read and easy-to-write as is feasible. Markdown is designed so that it can be converted to HTML and dozens of output formats, like PDFs, Word files, slideshows, and more.

R Markdown

R Markdown is a file format for making dynamic documents with R. An R Markdown document is written in Markdown and contains chunks of embedded R code. R Markdown provides an unified authoring framework for data science, combining your code, its results, and your prose commentary.

Prerequisites

You need `knitr` and `rmarkdown` packages, but you don't need to explicitly install them or load them, as R Studio automatically does both when needed.

Rendering Output

To render an R Markdown document into it's final output format, click the "Knit" button will render the document and display a preview of it.

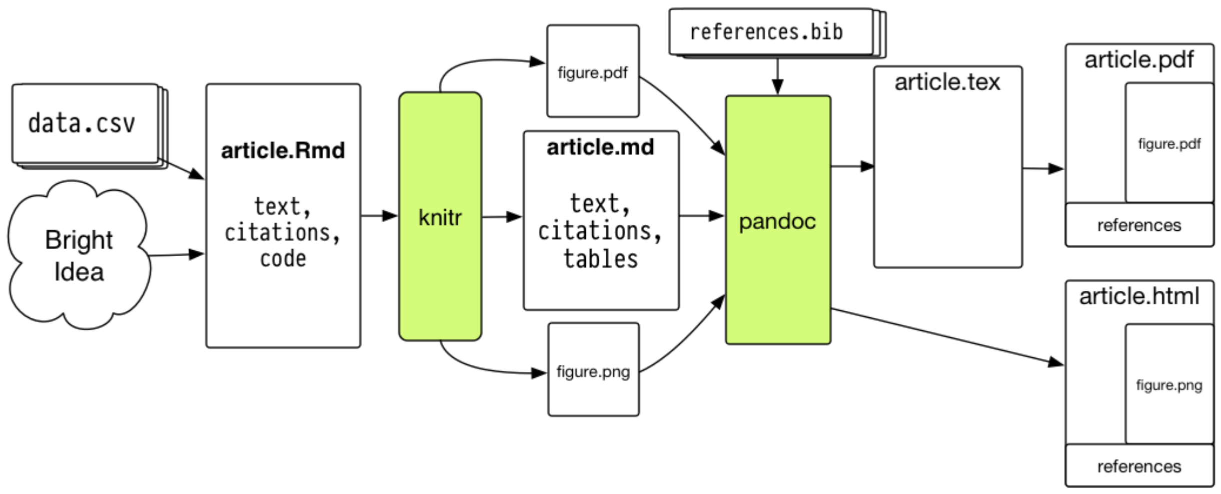

Figure 1: img

Basic Formatting in R Markdown

Prose in `.rmd` files is written in Markdown, a lightweight set of conventions for formatting plain text files. Markdown is designed to be easy to read and easy to write. It is also very easy to learn. The guide below shows how to use Pandoc's Markdown, a slightly extended version of Markdown that R Markdown understands.

```
Text Formatting
------------------------------------------------------------

*italic*

**bold**

~~strikethrough~~

superscript^2^ and subscript~2~

> quotation

`code`

endash: --

emdash: ---

ellipsis: ...

Headings
------------------------------------------------------------

# 1st Level Header

## 2nd Level Header
```

```
### 3rd Level Header

Lists
------------------------------------------------------------
* unordered list
* item 2
    + sub-item 1
    + sub-item 2

1. ordered list
2. item 2
    1. sub-item 1
    2. sub-item 2

The numbers are incremented automatically in the output.

Links and images
------------------------------------------------------------

<http://example.com>

[linked phrase](http://example.com)

![optional caption text](path-to-image)

Tables
------------------------------------------------------------

First Header  | Second Header
------------- | -------------
Content Cell  | Content Cell
Content Cell  | Content Cell
```

Paragraph Breaks and Forced Line Breaks

To insert a break between paragraphs, include a single completely blank line.

To force a line break, put *two* blank spaces at the end of a line.

```
To insert a break between paragraphs, include a single completely blank line.

To force a line break, put *two* blank
spaces at the end of a line.
```

Title, Author, Date, Output Format, Table of Contents

You caan also use the header to tell R Markdown whether you want it to render to HTML (the default), PDF, or something else. See http://rmarkdown.rstudio.com/lesson-9.html for more about the output format.

```
---
title: Your Title
author: Your Name
date: Today
output: output_format
---
```

R notebooks

An R Notebook is an R Markdown document with chunks that can be executed independently and interactively, with output visible immediately beneath the input. You can create a new notebook in RStudio with the menu command *File -> New File -> R Notebook*, or by using the `html_notebook` output type in your **header**.

```
---
title: "My Notebook"
output: html_notebook
---
```

Adding a table of contents is done as an option to the output type.

```
---
output:
  html_document:
    toc: true
---
```

You can specify the `toc_float` option to float the table of contents to the left of the main document. The floating table of contents will always be visible even when the document is scrolled.

```
---
output:
  html_document:
    toc: true
    toc_float: true
---
```

Including Code

The real point of R Markdown is that it lets you include your code, have the code run automatically when your document is rendered, and seemlessly include the results of that code in your document. The code comes in two varieties, code **chunks** and **inline** code.

Code Chunks and Their Results

A code **chunk** is simply an off-set piece of code by itself. It is preceded by ` ```{r} ` on a line by itself, and ended by a line which just says ` ``` `. The code itself goes in between. Here, for instance, is some code which loads a data set from a library, and makes a scatter plot.

First, notice how the code is included, nicely formatted, in the document. Second, notice how the output of the code is also automatically included in the document. If your code outputs numbers or text, those can be included too:

Interactive HTML

R Markdown document can be rendered into the interactive HTML document. You can run the code chunk below by setting `eval=TRUE` to see the interactive table and plot. To run this chunk, you need to install `DT`, `ggplot2` and `plotly` packages.

Chunk name

Chunks can be given an **optional** name: ```` ```{r by-name} ````. You can more easily navigate to specific chunks using the drop-down code navigator in the bottom-left of the script editor. This name is then used for the images (or other files) that are generated when the document is rendered.

Chunk options

Code chunks (but not inline code) can take a lot of **options** which modify how they are run, and how they appear in the document. These options go after the initial `r` and before the closing `}`. You can see the full list at http://yihui.name/knitr/options/.

- `eval = FALSE` prevents code from being evaluated. This is useful for displaying example code, or for disabling a large block of code without commenting each line.
- `include = FALSE` runs the code, but doesn't show the code or results in the final document. Use this for setup code that you don't want cluttering your report.
- `echo = FALSE` prevents code, but not the results from appearing in the finished file. Use this when writing reports aimed at people who don't want to see the underlying R code.
- `message = FALSE` or `warning = FALSE` prevents messages or warnings from appearing in the finished file.

Chunk options are seperated by comma. If there is no individual chunk options, the global chunks options will be used.

Inline Code

Code output can also be seamlessly incorporated into the text, using **inline code**. This is code not set off on a line by itself, but beginning with `` `r `` and ending with `` ` ``.

```
For example, the sample correlation coefficient between =speed= and =dist= is 0.8068949.
```

For example, the sample correlation coefficient between `speed` and `dist` is 0.8068949.

Changing Image Sizes and Alignments

There are a bunch of options for adjusting the placement of the figures which R produces. `fig.align` controls the horizontal **alignment** (left, right, or center). The options `fig.height` and `fig.width` let you specify the desired height or width of the figure.

The figure size can be globally defined in the **header** of your R Markdown file.

```
---
output:
    fig_height: 9
    fig_width: 6
---
```

Tables

The default print-out of matrices, tables, etc. from R Markdown is frankly ugly.

kable

If you want to make a somewhat nicer table, the `knitr` package contains a very basic command, `kable`, which will format an array or data frame more nicely for display.

pander

Another good option is the pander package. It allows more customization, and if you give it the output of lm(), it will automatically produce the table of regression coefficients that we're interested in.

Caching

If document rendering becomes time consuming due to long computations or plots that are expensive to generate, you can ask R Markdown to keep track of whether a chunk of code has changed, and only re-run it if it has. This is called **caching** the chunk. Let's see the effect of caching by executing an inline function. The Fibonacci numbers (F_n) are defined by $F_0 = 0$, $F_1 = 1$, and $F_n = F_{n-1} + F_{n-2}$ for $n \geq 1$.

The below code will take several seconds to compute at the first run. On the second run, `knitr` will use the cached output without evaluating the code (unless you change it!)

However, be careful with **caching**. One issue is that a chunk of code which hasn't changed itself might call on results of earlier, modified chunks, and then we *would* want to re-run the downstream chunks. There are options for manually telling R Markdown "this chunk depends on this earlier chunk", but it's generally easier to let it take care of that, by setting the `autodep=TRUE` option.

1. If you load a package with the `library()` or `require()` commands, R Markdown isn't smart enough to check whether the package has changed (or indeed been installed, if you were missing it). So that won't trigger an automatic re-running of a cached code chunk.
2. To manually force re-running all code chunks, the easiest thing to do is to delete the directory R Markdown will create (named something like /filename/=_cache=) which it uses to store the state of all code chunks.

Rcpp Code Chunks

This section is optional. Those who have a C compiler installed can turn on the evaluation of three chunks below. If you don't have the C compiler, please ignore this section. Sometimes R code just isn't fast enough. You've used profiling to figure out where your bottlenecks are, and you've done everything you can in R, but your code still isn't fast enough. You may want to consider the `Rcpp` package to accelerate computations, which permit to rewrite functions in `C++`.

The improved performance by using the `Rcpp` function over the naive R function can be seen by benchmarking these two functions. To understand performance, you can also profile execution of two functions using `Rprof` or `profvis`.

If you want use an Rcpp function in R console, you must compile it or install it using a local R package. For more details, please see http://www.rcpp.org.

Setting Defaults for All Chunks

You can tell R to set some defaults to apply to all chunks where you don't specifically over-ride them. Here are the ones I generally use:

This sets some additional options beyond the ones I've discussed, like not re-running a chunk if only the comments have changed (`cache.comments = FALSE`), and leaving out messages and warnings. I would typically give this set-up chunk itself the option `include=FALSE`.

Individual chunk options will overide the global chuck options.

Tangle

You can extract the source code chunk out of the R Markdown document. This procedure is called "tangling", which can be done by using `knit()` function with `tangle=TRUE` option in the R console. Then, you can execute the "tangled" code by `source()` to get the results without rendering the R Markdown document.

Math in R Markdown

With R Markdown, you can embed math equations directly into your document. R Markdown gives you the syntax to render complex mathematical formulas and derivations, and have them displayed *very* nicely. Like code, the math can either be inline or set off (**displays**).

Inline math is marked off witha pair of dollar signs (`$`), as $\pi r^2$ or $e^{i\pi}$.

```
Inline math is marked off witha pair of dollar
signs (`$`), as $\pi r^2$ or $e^{i\pi}$.
```

Mathematical displays are marked off with `\[` and `\]`, as in

$$e^{i\pi} = -1$$

```
Mathematical displays are marked off with `\[` and `\]`, as in
\[
e^{i \pi} = -1
\]
```

Prerequisites

If your output format is `html`, equations are displayed using the MathJax JavaScript library. Note that this library is loaded from the MathJax website so readers of your document must be online to see the rendered equations. On the other hand, if your output format is `pdf`, then you need to install a program called `LaTeX`. I recommend to install TeX Live for Linux or Window machines and MacTeX for OS X.

Elements of Math Mode

- Most letters will be rendered in italics (compare: a vs.=a= vs.a; only the last is in math mode). The spacing between letters also follows the conventions for math, so don't treat it as just another way of getting italics. (Compare *speed*, in simple italics, with $speed$, in math mode.)
- Greek letters can be accessed with the slash in front of their names, as `\alpha` for α. Making the first letter upper case gives the upper-case letter, as in `\Gamma` for Γ vs.= = for γ. (Upper-case alpha and beta are the same as Roman A and B, so no special commands for them.)
- There are other "slashed" (or "escaped") commands for other mathematical symbols:
  - `\times` for $\times$
  - `\cdot` for $\cdot$
  - `\leq` and `\geq` for $\leq$ and $\geq$
  - `\subset` and `\subseteq` for $\subset$ and $\subseteq$
  - `\leftarrow`, `\rightarrow`, `\Leftarrow`, `\Rightarrow` for $\leftarrow$, $\rightarrow$, $\Leftarrow$, $\Rightarrow$
  - `\approx`, `\sim`, `\equiv` for $\approx$, $\sim$, $\equiv$
  - See, e.g., http://web.ift.uib.no/Teori/KURS/WRK/TeX/symALL.html for a fuller listing of available symbols. (http://tug.ctan.org/info/symbols/comprehensive/symbols-a4.pdf lists *all* symbols available in `LaTeX`, including many non-mathematical special chracters)
- Subscripts go after an underscore character, `_`, and superscripts go after a caret, `^`, as `\beta_1` for β_1 or `a^2` for a^2.

- Curly braces are used to create groupings that should be kept together, e.g., `a_{ij}` for a_{ij} (vs.=a__ij= for a_ij).
- If you need something set in ordinary (Roman) type within math mode, use `\mathrm`, as `t_{\mathrm{in}}^2` for t_{in}^2.
- If you'd like something set in an outline font ("blackboard bold"), use `\mathbb`, as `\mathbb{R}` for $\mathbb{R}$.
- For bold face, use `\mathbf`, as

  `{\mathbf{x}^T \mathbf{x}}^{-1}\mathbf{x}^T\mathbf{y}`

  for ${\mathbf{x}^T \mathbf{x}}^{-1}\mathbf{x}^T\mathbf{y}$
- Accents on characters work rather like changes of font: `\vec{a}` produces $\vec{a}$, `\hat{a}` produces $\hat{a}$. Some accents, particularly hats, work better if they space out, as with `\widehat{\mathrm{Var}}` producing $\widehat{\mathrm{Var}}$.
- Function names are typically written in romans, and spaced differently: thus $\log x$, not $log x$. `LaTeX`, and therefore `R Markdown`, knows about a lot of such functions, and their names all begin with `\`. For instance: `\log`, `\sin`, `\cos`, `\exp`, `\min`, etc. Follow these function names with the argument in curly braces; this helps `LaTeX` figure out what exactly the argument is, and keep it grouped together with the function name when it's laying out the text. Thus `\log{(x+1)}` is better than `\log (x+1)`.
- Fractions can be created with `\frac`, like so:

  `\frac{a+b}{b} = 1 + \frac{a}{b}`

  produces $\frac{a+b}{b} = 1 + \frac{a}{b}$. Sums can be written like so:

  `\sum_{i=1}^{n}{x_i^2}`

  will produce $\sum_{i=1}^{n}{x_i^2}$. The lower and upper limits of summation after the `\sum` are both optional. Products and integrals work similarly, only with `\prod` and `\int`:

  $$n! = \prod_{i=1}^{n}{i}$$

  $$\log{b} - \log{a} = \int_{a}^{b}{\frac{1}{x}dx}$$

  `\sum`, `\prod` and `\int` all automatically adjust to the size of the expression being summed, producted or integrated.
- "Delimiters", like parentheses or braces, can automatically re-size to match what they're surrounding. To do this, you need to use `\left` and `\right`, as

  ```
  \[
  \left( \sum_{i=1}^{n}{i} \right)^2 = \left( \frac{n(n-1)}{2}\right)^2 = \frac{n^2(n-1)^2}{4}
  \]
  ```

  renders as

  $$\left(\sum_{i=1}^{n}{i} \right)^2 = \left(\frac{n(n-1)}{2}\right)^2 = \frac{n^2(n-1)^2}{4}$$

  - To use curly braces as delimiters, precede them with slashes, as `\{` and `\}` for $\{$ and $\}$.
- A set of multiple aligned equations can be created using `align*`, as follows.

  ```
  \begin{align*}
  X & \sim  \mathrm{N}(0,1)\\
  Y & \sim  \chi^2_{n-p}
  \end{align*}
  ```

$$X \sim \mathrm{N}(0,1) \tag{1}$$
$$Y \sim \chi^2_{n-p} \tag{2}$$

Notice that `&` is a delimiter separating columns, and each line (except the last) is terminated with `\\`. The left or right hand side of the equation can be blank, and space will be made:

```
\begin{align*}
P(|X-\mu| > k) & = P(|X-\mu|^2 > k^2)\\
& \leq \frac{\mathbb{E}\left[|X-\mu|^2\right]}{k^2}\\
& \leq \frac{\mathrm{Var}[X]}{k^2}
\end{align*}
```

$$\begin{aligned}
P(|X-\mu| > k) & = P(|X-\mu|^2 > k^2)\\
& \leq \frac{\mathbb{E}\left[|X-\mu|^2\right]}{k^2}\\
& \leq \frac{\mathrm{Var}[X]}{k^2}
\end{aligned}$$

- A matrix can be represented using `pmatrix` in a similar fashion. `matrix` or `bmatrix` gives a matrix with a different surrounding.

  ```
  \begin{pmatrix}
  1 & 0 & 0 \\
  0 & 1 & 0 \\
  0 & 0 & 1
  \end{pmatrix}
  ```

10	0
01	0
00	1

Troubleshooting

- "I am able to do *knit to HTML*, but *knit to PDF* fails.": First, make sure if you installed `LaTeX`. Go to the line that the error message pointed. You may put some characters (math symbols, greek letters, etc) that `LaTeX` doesn't know how to process. Correct them and run *knit to PDF* again.
- Do not call `View` or `help` in your document; these are interactive commands which don't work well in scripts.
- "It worked in the console but it wouldn't knit": You have almost certainly done something somewhat different *before* the code chunk that's giving you trouble. Clear your workspace in the console and re-run.
  - R Studio keeps *two* environments or workspaces which it uses to evaluate R expressions, look up function or variable names, etc. One is the "usual" global environment of the console, which builds cumulatively from the start of your session. (Unless you deliberately manipulate it; don't do that unless you know what you're doing.) Every time you knit, however, it re-runs your code in clean workspace, as though you had just started R from scratch. This means knitted code does what you say it should, and *only* that. If your code knits, it should work on any computer; getting something to run in the console which you can't reproduce is just dumb luck.

- “It works when I source it, but it won’t knit”: This is basically the same problem as “it worked in the console”.
- Avoid `attach` in both the console and in your file; using it is a recipe for creating hard-to-find errors. You can still shorten expressions using `with` instead.
- When you need to load data files or source someone else’s code, use full URLs, rather than creating local copies and loading them from your disk.