

# A Resource Document for R Markdown

*Alec Wong*

*February 24, 2017*

## Contents

<b>Background Components</b>	<b>2</b>
knitr . . . . .	2
pandoc . . . . .	2
YAML . . . . .	2
<b>Composing an R Markdown Document</b>	<b>3</b>
Markdown . . . . .	3
Text . . . . .	3
Headings . . . . .	3
Code chunks . . . . .	4
Inline code . . . . .	4
Plots . . . . .	5
Plot appearance . . . . .	6
Appearance . . . . .	7
HTML . . . . .	7
.pdf L <sup>A</sup> T <sub>E</sub> X Variables . . . . .	7
<b>L<sup>A</sup>T<sub>E</sub>X Primer</b>	<b>8</b>
Equations . . . . .	8
Equation Structure . . . . .	8
Symbols . . . . .	9
Distributions . . . . .	9
Fractions . . . . .	10
Parentheses . . . . .	10
Spacing . . . . .	10
L <sup>A</sup> T <sub>E</sub> X ‘split’ environment . . . . .	10
Summations and Products . . . . .	10
<b>Conclusion</b>	<b>11</b>

R Markdown is a union of several components - namely **knitr**, **pandoc**, and of course **R** - that allow the simultaneous comprehension of varied syntaxes and conventions, including Markdown, HTML & CSS, and  $\text{\LaTeX}$ . Composing an R Markdown document would proceed the same way this document is written.

*Very* generally, changes to the structure of .pdf outputs are handled by  $\text{\LaTeX}$  commands, and HTML outputs by HTML, and these are not generally mixed except in the case of equations.

## Background Components

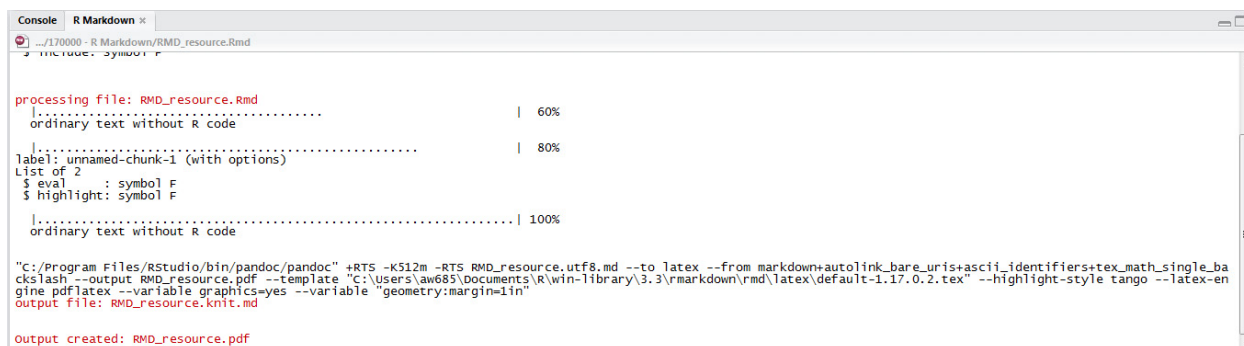
### knitr

**knitr** is the central component for handling R code input/output within the R Markdown ecosystem. It was inspired by **Sweave**, which is still an available authoring system within Rstudio, but instead of operating exclusively with  $\text{\LaTeX}$  syntax as **Sweave** does, **knitr** allows use of the simpler markdown syntax within the R Markdown authoring ecosystem.

Code chunks are the most obvious contribution of **knitr**, and they allow verbose or quiet evaluation of code, or just illustration without evaluation, among a myriad of other such “chunk options”. These chunks are updated whenever you specify, making reproducible research easy. **knitr** facilitates dynamic report generation, with code chunks being seamlessly integrated into the report.

### pandoc

**pandoc** is a tool that handles the conversion of the .md (markdown format) produced by **knitr** into the output format of your specification, be it HTML, .pdf, Word doc, or the new R Notebook. When you go to knit a file, **knitr** handles evaluation and formatting of code chunks into markdown format, and **pandoc** processes the **knitr** output into the final document - you can see its activity in RStudio in the R Markdown tab next to the console:



```
processing file: RMD_resource.Rmd
|.....| 60%
ordinary text without R code
|.....| 80%
label: unnamed-chunk-1 (with options)
List of 2
 $ eval      : symbol F
 $ highlight : symbol F
|.....| 100%
ordinary text without R code

"C:/Program Files/RStudio/bin/pandoc/pandoc" +RTS -K512m -RTS RMD_resource.utf8.md --to latex --from markdown+autolink_bare_uris+ascii_identifiers+tex_math_single_backslash --output RMD_resource.pdf --template "C:/Users/aw685/Documents/R/win-library/3.3/rmarkdown/rmd/latex/default-1.17.0.2.tex" --highlight-style tango --latex-engine pdflatex --variable graphics=yes --variable "geometry:margin=1in"
output file: RMD_resource.knit.md

output created: RMD_resource.pdf
```

Figure 1: Pandoc’s output begins where pandoc is called on the local machine. The text after the double dashes ‘--’ are pandoc arguments, and these can accept arguments from within the YAML - next.

### YAML

There isn’t much typing that goes to the **YAML** (**YAML Ain’t Markup Language**) header. It’s mostly used to set metadata, like titles, authors, date, and more usefully, passing arguments to **pandoc** and **knitr**, as well as other things.

The top of the accompanying .rmd document is an example, and the basic **YAML** is automatically generated when an R Markdown document is created. Some additional arguments are specified in this document, such

as including a table of contents with `toc: true`, the table of content depth to be 3 levels `toc_depth: 3`. Notice the indenting structure used to nest `YAML` arguments appropriately.

---

## Composing an R Markdown Document

A cheat-sheet for many concerns about R Markdown can be found [here](#). As with many other things, Google is your #1 friend when finding out how to do something.

### Markdown

Markdown conventions can be accessed in many places, such as the cheatsheet provided earlier, the [pandoc README page](#)

#### Text

Simple operations like *italicizing*, or **boldening** font can be accomplished by wrapping text in single, and double-asterisks respectively.

New lines are only recognized after adding a white space (two carriage returns) between paragraphs. A single carriage return is ignored, as are multiple spaces (these are treated as one space).

Bullet points are accomplished using tab spaces, asterisks, pluses, and dashes

- \* To make
  - + nested
    - lists
- To make
  - nested
    - \* lists

Numbered lists proceed in a similar fashion although the number you type does not necessarily correspond to the number that prints, instead the order that prints is the order of appearance after the first number you type.

1. Therefore, a list starting with number 1
2. Will continue sequentially regardless of what is typed here (I typed 10)

#### Headings

Markdown puts an emphasis on organizing text into sections. Headers are created using hashes #, and sub-sections are created simply by adding additional hashes ##...

For additional flexibility, the structure of the document can be altered using HTML (if publishing to an HTML document) and L<sup>A</sup>T<sub>E</sub>X (if publishing to a .pdf). You can accomplish a top-level header any which way you want:

---

### Markdown

# Section

**LaTeX**  
`\section{Section}`

**HTML** (will be ignored if passed to .pdf)  
`<h1>`  
Section  
`</h1>`

---

These are all ways to format a heading, and sub-headings follow suit in the respective languages (i.e. secondary headers use two hashes, a `\subsection` in LaTeX, or `<h2>` in HTML).

## Code chunks

Code chunks are the component of R Markdown that distinguishes it from any other editor, and they provide great flexibility in composing documents with code.

A code chunk is initiated using triple graves followed by `{r}`, and code chunk options are added after the `r` within the brackets as follows:

---

```
```{r, ARGS}
code
```
```

---

If you only want to print the results of code, say a 3x3 matrix of 1:9, you can use the `echo = F` argument in the code chunk `ARGS` area shown previously. (Reference the .rmd for the code, as it isn't displayed here.)

```
      [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
```

Or, if you only want to print the code for reference without evaluating it, you can use the `eval = F` argument in the code chunk; the code will be printed without its results, particularly useful if the code operates for a long time.

Plenty of additional arguments can be passed to individual code chunks, or set at the beginning in a single code chunk containing `knitr::opts_chunk$set(...)`.

A list of available options can be found [here](#).

## Inline code

Code can also be displayed in-line by surrounding the code with graves, or backticks 'like this'.

Knitr also handles inline *evaluation* of code. Say I set a an object `A = seq(from = 1, to = 10)` in my R environment; I can perform any operation on `A` by putting the letter 'r' before the code *within the backticks*:

```
A = seq(from = 1, to = 10)
```

```
`r A`
```

will print the contents of A:

1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Again, any operation can be performed in-line:

```
`r = sum(A)`
```

will print the sum of the contents of A:

55

And so on. This feature, along with the code chunks, make reproducible reports extremely simple; any analysis can be included in the code chunks and new reports on new data can be produced without much additional work.

## Plots

Including graphics is just as simple as generating plots in R. Using the code chunk functionality, plots can be created and the output will be printed below the chunk.

```
library(ggplot2)

x = seq(0,10,0.1)

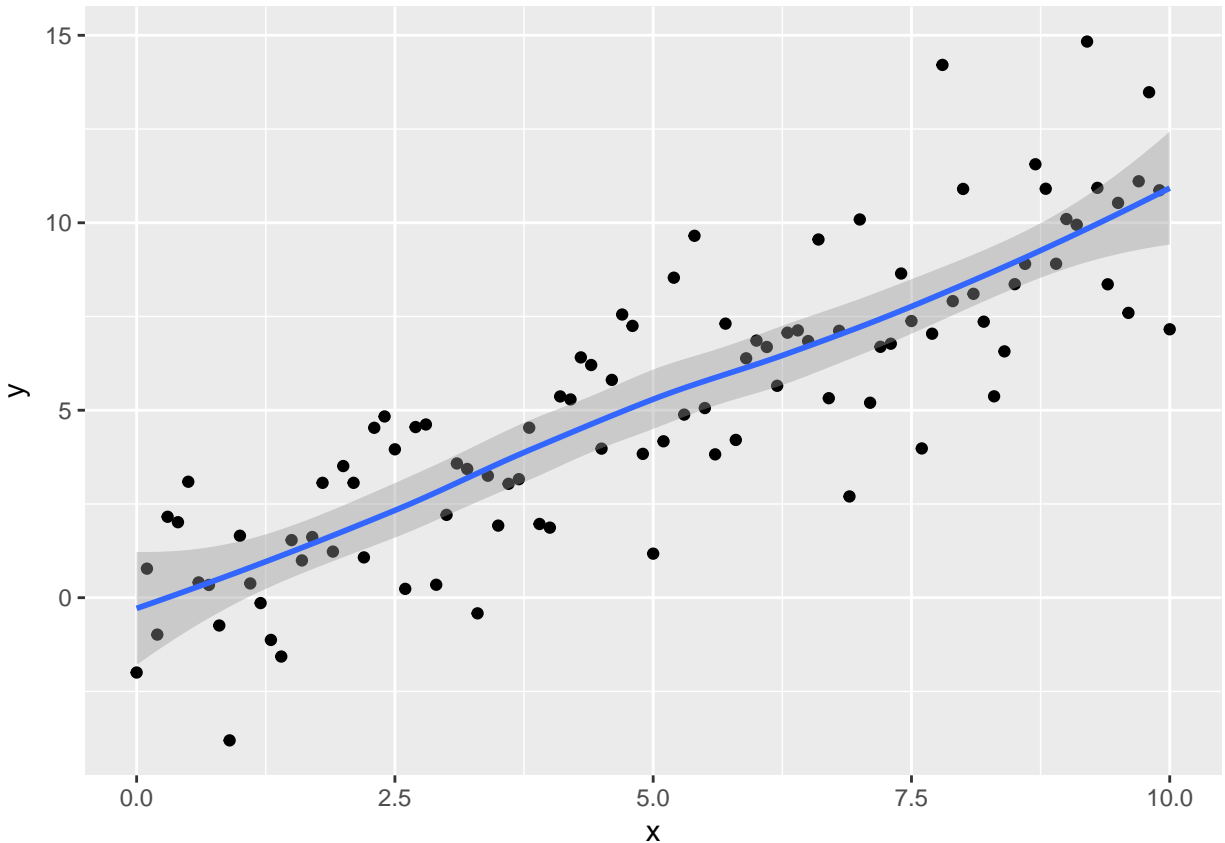
y = vector(mode = "numeric", length = length(x))
for(i in 1:length(x)){
  y[i] = x[i] + rnorm(n = 1, mean = 0, sd = 2)
}

xy = data.frame("x" = x, "y" = y)

showplot = ggplot(data = xy) +
  geom_point(aes(x = x, y = y)) +
  geom_smooth(aes(x = x, y = y))

showplot

`geom_smooth()` using method = 'loess'
```



This is also a good example of reproducibility - since the data simulated has a random normal component, the plot (and any analysis on the data, for that matter) will change each time the code chunk is evaluated. If the data supplied were real, this plot would update based on any (appropriately formatted) input data, say for different years' analysis.

### Plot appearance

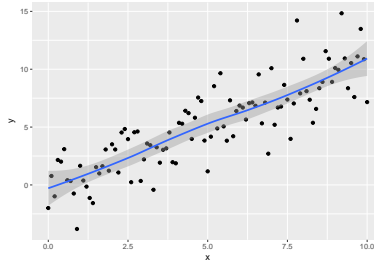
Chunk options exist for plots - for example, the following chunk options manipulate the figure width & height to be very small (not particularly useful, but good for example) and centered.

---

```
```{r, out.width='200px', out.height='100px', fig.align='center'}  
showplot  
```
```

---

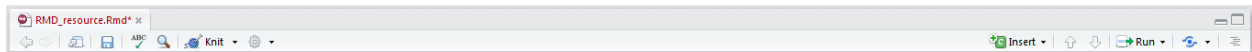
```
`geom_smooth()` using method = 'loess'
```



See the [code chunk link](#) for more figure options.

## Appearance

An easy way to manipulate the output appearance without too much work is to use one of the pre-set themes (if using HTML), accessed through the gear icon in the tab bar up top:



Code highlighting styles can also be set there.

## HTML

For HTML documents, if you know CSS, you can [supply your own style tags at the top of your R Markdown document](#).

## .pdf $\text{\LaTeX}$ Variables

Various setup parameters can be adjusted when making a .pdf using  $\text{\LaTeX}$  variables. These are passed to pandoc through the YAML header. A complete list can be found [here](#).

### Fonts

For pdf documents, the font can be changed by supplying a `fontfamily:` argument in a new unindented line to the YAML header; if you're using TeX Live, a list of available fonts can be found at [this website](#).. Try setting the `fontfamily:` `arev` to see the effect (I picked a sans-serif font so the difference is exaggerated).

Fonts can be colored individually using the `\fontcolor{color}{text}` command.

### Page setup

Page setup can be handled by  $\text{\LaTeX}$  variables in the header.

Margins can be altered by including

```
geometry: margin=#in
```

in a new unindented line, with the margin units in inches, or centimeters.

A table of contents can be included as done in this document

```
toc: true
```

creates a table of contents based upon your headers (but note that a pure  $\text{\LaTeX}$  grammar would use [sectioning commands](#) and the hierarchy specified therein). The depth of the sections displayed in the table of contents is specified by `toc_depth: #`.

See the [pandoc readme page](#) for more details.

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

R Markdown can be used with a minimal knowledge of L<sup>A</sup>T<sub>E</sub>X , but having access to its functionality opens up many options when it comes to formatting the document - the previous section is an example.

### Equations

One foremost use of L<sup>A</sup>T<sub>E</sub>X in R Markdown is its syntax in formatting equations. Equations are flexible to construct in R Markdown. For inline equations, wrap the L<sup>A</sup>T<sub>E</sub>X formatted equation in single dollar signs: `E = mc^2` prints  $E = mc^2$ . These don't have to be strictly equations; single letters can be used to reference variables in text with minimal effort. R Studio will display a preview of the equation when the cursor is within the equation boundary.

Displayed equations are created most simply using the double-dollar sign, although you may wrap the equation in `\[` and `\]` as well:

```
$$  
F = ma  
$$
```

$$F = ma$$

To number the equations, I have resorted to writing the equations in L<sup>A</sup>T<sub>E</sub>X completely, as R Markdown doesn't seem to handle numbered equations well. Additionally, referencing equations is easiest if you stay in L<sup>A</sup>T<sub>E</sub>X.

```
\begin{equation}  
\label{eq:equation1}  
KE = 1/2mv^2  
\end{equation}
```

$$KE = 1/2mv^2 \tag{1}$$

I can reference equation [1](#) with this structure; L<sup>A</sup>T<sub>E</sub>X automatically handles the order of the equations depending on when they appear in the body, eliminating the need to manually enter references to the equation number. The reference in the previous sentence was accomplished by entering `\ref{eq:equation1}`, and more generally one enters `\ref{eq:EQUATIONLABEL}`, with EQUATIONLABEL being whatever you decide to name it after the `\label{eq:EQUATIONLABEL}` command.

Note that RStudio will not preview input of pure L<sup>A</sup>T<sub>E</sub>X equations.

### Equation Structure

Formatting equations can be complicated at first glance:

$$\begin{aligned} Y &\sim \text{Binomial}(K, p_{ij}) \\ p_{ij} &= p_0 \exp\left(-\frac{\text{dist}(x_j, s_i)}{\sigma^2}\right) \\ s_i &\sim \text{Uniform}(\mathcal{S}) \end{aligned} \tag{2}$$



Is formatted as follows:

---

```
\begin{equation}
\begin{split}
Y &\sim \text{Binomial}(K, p_{ij}) \\
p_{ij} &= p_0 * \text{exp}\left(-\frac{\text{dist}(x_j, s_i)}{\sigma^2}\right) \\
s_i &\sim \text{Uniform}(\mathcal{S})
\end{split}
\end{equation}
```

---

There's a lot going on in the code above, and we'll break it down in the sections below.

## Symbols

Simple symbols like Greek letters typically have their own commands in L<sup>A</sup>T<sub>E</sub>X. Here's a small table (see the .rmd for table syntax):

| Symbol    | Command              |
|-----------|----------------------|
| $\lambda$ | <code>\lambda</code> |
| $\theta$  | <code>\theta</code>  |
| $\pi$     | <code>\pi</code>     |
| $\omega$  | <code>\omega</code>  |

Capital Greek letters can sometimes be obtained by capitalizing the first letter of the Greek letter's name in the command:

| Symbol    | Command              |
|-----------|----------------------|
| $\Lambda$ | <code>\Lambda</code> |
| $\Theta$  | <code>\Theta</code>  |
| $\Pi$     | <code>\Pi</code>     |
| $\Omega$  | <code>\Omega</code>  |

Some capital letters are the same in Roman and Greek alphabets, such as capital chi (X) or capital tau (T), and so normal `\text{text}` can handle those.

See [this page](#) for the complete list of L<sup>A</sup>T<sub>E</sub>X symbols.

## Distributions

For things like statistical distributions, you will want to use the `\text{text}` command. For example, you would write

```
$N \sim \text{Poisson}(\lambda)$
```

to produce  $N \sim \text{Poisson}(\lambda)$  (notice the `\sim` command for the “distributed as” symbol)

## Fractions

In my experience, complex fractions are best accomplished by using the `\frac{numerator}{denominator}` command. So, one-half would trivially be `\frac{1}{2} = \frac{1}{2}`

## Parentheses

Sometimes using fractions makes a large term that isn't pretty to enclose in regular-size parentheses. For these, you must wrap the term in `\left(` followed by `\right)`. Notice the difference between:

`$$\left(\frac{\text{numerator}}{\text{denominator}}\right)$$`

$$\left(\frac{\text{numerator}}{\text{denominator}}\right)$$

and

`$$\left(\frac{\text{numerator}}{\text{denominator}}\right)$$`

$$\left(\frac{\text{numerator}}{\text{denominator}}\right)$$

## Spacing

You may find spacing between terms a little tight:

`$p_0 \text{exp}(x)$`

$p_0 \exp(x)$

You can resolve this in one of several ways, shown in [this page](#)

Typically the easiest is simply to add a single backslash, but other modifications are possible. Do note that spacebar spaces are ignored!

`$p_0 \ \text{exp}(x)$`

$p_0 \exp(x)$

## L<sup>A</sup>T<sub>E</sub>X ‘split’ environment

In order to stack equations, you need a few extra commands, starting with the `\begin{split}` command and wrapping it with the `\end{split}` command.

Additionally, after each line you must signify a new line with a double backslash, and the equations are centered at the `&` character; I put the ampersands on the tildes and equal signs.

## Summations and Products

If you want to write a summation, the structure is as follows:

`\sum_{FROM}^{TO} SUMMAND`

for example,

`\sum_{i = 1}^{n} x_n`

produces

$$\sum_{i=1}^n x_n$$

Products and other series follow in this fashion.

---

$\text{\LaTeX}$  commands are extremely numerous and can't all be described here, but these are some typical implementations that I've used *commonly* in the past.

## Conclusion

### What good is R Markdown?

R Markdown is useful for illustrating scientific, mathematic, and computational ideas in a legible, cohesive, presentable format. Integrating code into a pure  $\text{\LaTeX}$  document can *probably* be done, but not with the utility and ease of R Markdown.

Reports based on R analyses can be compiled easily and quickly once the template is constructed. These can be shared as .pdfs, or published to web pages directly with the HTML output.

Annotation of code is easy in RStudio now that code chunks can be evaluated in-line, with the output printed directly below your chunk or equation.

All these promote shareable, understandable science, and can aid communication of results.

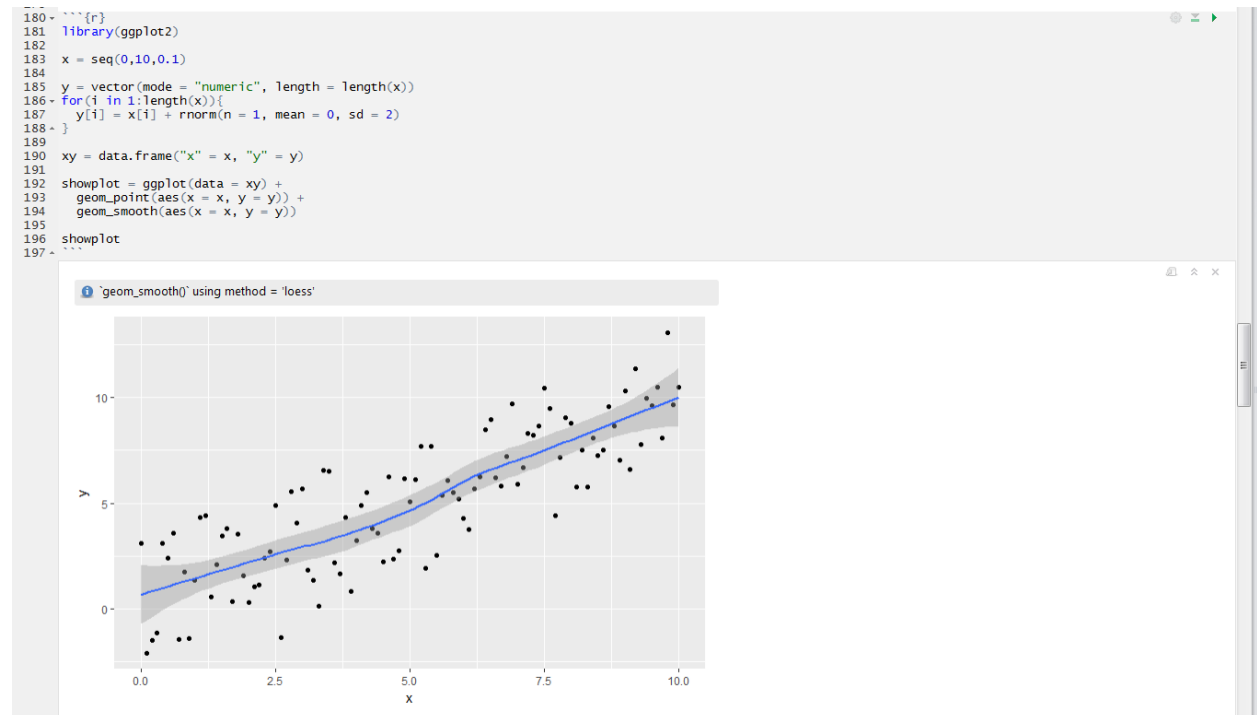


Figure 2: View of the RStudio editor with code evaluated in-line