Intro to RMarkdown

Dante

2024-07-03

Text

You should already be familiar with writing code in files with the `.R` extension like you have been doing for koans. In those files, everything not commented with `#` is recognized to be lines of code.

Now we will learn how to write documents using a light-weight markup language called *R Markdown*. Files with the extension `.Rmd` are recognized by RStudio as R Markdown documents. Everything you write will be recognized as either *text*, *formatting*, *math notation*, or *code*.

This document is written in markdown. Markdown is designed to facilitate *literate programming* which (briefly) means that writing in plain english comes first, then code. Notice how everything I have written so far (except for the formatting at the top) is compiled as type-set text instead of code?

Formatting

Markdown provides simple shorthand for your basic formatting that you would find in a menu of options in word processors like Microsoft Word.

Headings

Heading Level 1

Heading Level 2

Heading Level 3

Heading Level 4

Emphasize and link text

Italicized Text

Bold Text

Bold and Italicized Text

Hyperlinked Text

Lists

Enumerated List:

1. First item
2. Second item
3. Third item

Bulleted List:

- Item
- Another Item
- Parent Item
  - Indented child item
- Last item

Horizontal rules

R Code

Unlike in an `.R` file, R Markdown will not recognize text as R code unless you tell it. Text inside pairs of backticks (‘_‘, or “‘_“‘) is recognized as code.

Inline code

In markdown, I can say something like “2 + 2 = 4”.

Code chunks

I can also add a code chunk with the green ‘+C’ button in Rstudio or the hotkey, `Command Alt I`.

```
# everything inside a chunk works like in a .R file
library(tidyverse)
```

```
## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become error
```

```
library(gapminder)
head(gapminder)
```

```
## # A tibble: 6 x 6
##   country     continent  year lifeExp      pop gdpPercap
##   <fct>       <fct>     <int>   <dbl>    <int>     <dbl>
## 1 Afghanistan Asia       1952    28.8  8425333      779.
## 2 Afghanistan Asia       1957    30.3  9240934      821.
## 3 Afghanistan Asia       1962    32.0 10267083      853.
## 4 Afghanistan Asia       1967    34.0 11537966      836.
## 5 Afghanistan Asia       1972    36.1 13079460      740.
## 6 Afghanistan Asia       1977    38.4 14880372      786.
```

I can also define an object in an `R` object,

```
meanGDP = mean(gapminder$gdpPercap)
```

which I can call upon with the `r` syntax to print a result in text: The average GDP per capita is 7215.3270812.

Chunk options

Inside of the r chunk heading, you can specify a number of options depending on what you want to appear in your final document.

- If you don't want the code to be *evaluated* and the results printed, you can set `eval = FALSE`

```
sample_data <- tibble(
  X = runif(n = 10, min = 0, max = 10),
  Y = 1 * X + rnorm(n = 10, mean = 0, sd = 0.1)
)
lm(data = sample_data, Y ~ X)
```

- If you don't want the code itself to show up in the document, set `echo = FALSE`

```
##
## Call:
## lm(formula = Y ~ X, data = sample_data)
##
## Coefficients:
## (Intercept)            X
##    -0.02839      1.00655
```

There are other options like `warning`, `error`, `message`, etc. which you can read about here.

Math equations

Markdown includes syntax which comes from the LaTeX set of macros for the TeX type-setting language.

LaTeX is its own world which you should read more about if you're interested, but for this class we just need the basics of writing equations.

Inline math equations

Just like backticks are used to denote code, $ are used to denote math equations.

I can write math in the middle of a line of text such as $Y = 3X^2 + \pi$ which will be compiled by RStudio as a nice-looking equation. You can write superscripts such as R^2, and subscripts such as Y_i.

The summation notation has its own command, `\sum`, which can be used to write $\sum_{i=1}^{N} X_i$. There are also `\sqrt` for $\sqrt{\cdot}$, `\int` for $\int$, and `\frac` for fractions $\frac{a}{b}$, and many more!

Some characters like {, }, will be recognized as LaTeX formatting unless escaped with the \ symbol. Notice the difference between x_1, x_2, x_3 and $\{x_1, x_2, x_3\}$.

$$E = mc^2$$

Greek letters

`\alpha` becomes α, `\beta` becomes β, `\gamma` becomes γ, etc.

'Hat' and 'Bar' notation

- `\hat{Y}` becomes $\hat{Y}$
- `\bar{X}` becomes $\bar{X}$

Display math mode

Equations inside double pairs of $ will be printed on their own line.

Writing OLS equations

$$Y_i = \hat{\beta}_0 + \hat{\beta}_1 X_i + \hat{u}_i$$

$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X}$$

$$\hat{\beta}_1 = \frac{\sum_{i=1}^{n}(Y_i - \bar{Y})(X_i - \bar{X})}{\sum_{i=1}^{n}(X_i - \bar{X})^2}$$

Learning LaTeX

Feel free to use the examples in these notes as guidance by comparing the source code to the compiled html output.

You can also copy the code for any equation in lecture slides by finding the appropriate .qmd file in Andrew's github repository.

- ChatGPT knows LaTeX surprisingly well, but you have to be able to tell it what you want it to write and then edit the code it gives you for mistakes.
- Overleaf is an online LaTeX editor which offers a free trial you can use to play around with writing LaTeX documents. They also have great tutorials and template galleries.

Learning RMarkdown

Consult the official cheatsheet!