Literate Computing

Dan MacLean

May 2022

Table of contents

Motivation

If we didn't have to do it over and over, it wouldn't be called *re*-search.

Developing a data analysis is hard, it can involve many mis-steps and changes of mind from redoing of little bits here and there that weren't quite right the first time, to introducing new ideas or removing whole sections that didn't work out. This iterative process is completely in-line with all other aspects of research and means that we have a personal need to be able to record exactly what we've done with high accuracy, and high reproducibility. It is also our scientific responsibility and an aspect of scientific integrity that we are clear and open about the methods we use as they are key in the interpretations and understanding of the results that we get. In the jargon of the field we think of this as 'keeping a proper lab book', but when it comes to using a computer, what we need to record, when is not often clear nor is it sometimes easy to do so. As a result the methods sections of many reports, theses and papers report scientific computing in a vague and uninterpretable way, making statements like 'tests were done in Excel' or 'GenStat was used to perform *t*-tests', or 'a custom R script was used'. These nebulous reports are useless for anyone trying to understand exactly what was done and reports using them are unreproducible. That they pass reviewers so often is a clear indication of the failure of reviewing of methods. In practice these sorts of write ups are no better and no more informative than announcing that statistical analyses were done with a magic spell.

A major failing of computer graphical interfaces is that they do not make it easy for us to repeat actions, which is ironic as computers are excellent at repeating instructions very quickly. Scripts and programs are required to get the best reproducibility out of our computers, but scripts in R and Python (and any other computer language) are not easily read by people, even those with a great deal of experience in programming. Very quickly reproducible scripts become unusable lumps of code because users can't tell what is in them and what they are supposed to do, a phenomenon that has it's own acronym - WORN - write once, read never.

Literate programming is the skill of writing code that is readable and understandable, often without the need to read the actual code in any depth. This is a very useful day-to-day skill to have when working in science as multidisciplinary teams abound. It is also useful when switching from project to project as we can understand what we were doing in a new project and get going again quickly. Using literate programming also eases our duty to report clearly and openly what we did in our analysis as the task of writing the code and explaining what we did are accomplished simultaneously.

Figure 1: Artwork by @AllisonHorst

In this course we'll learn how to create a literate programming document in two popular data science languages, R and Python. The two systems share a common core and much of what you learn for one will be applicable to the other.

1 R and RMarkdown

1.1 About this chapter

1. Questions:

- How do I write a reproducible analysis document in R with RStudio?

2. Objectives:

- Learn markdown basics
- Mix markdown and R with Rmarkdown

3. Keypoints:

- RMarkdown documents are excellent tools for creating readable, attractive and dynamic reproducible literate programming documents.

1.2 Markdown

The process of adding annotations, like corrections or notes and comments to a physical or digital document is sometimes called 'marking it up. In processing digital documents, a language that adds tags to text to format them is called a markup language. Web pages are all written in a markup language called HTML (hypertext markup language) and it puts tags around elements to format them. Here for example is the markup for making text bold.

```
I am regular text. But <b>I am bold!</b>
```

and when you put that test through a rendering program like a web browser, it shows up like this:

I am regular text. But I am bold!

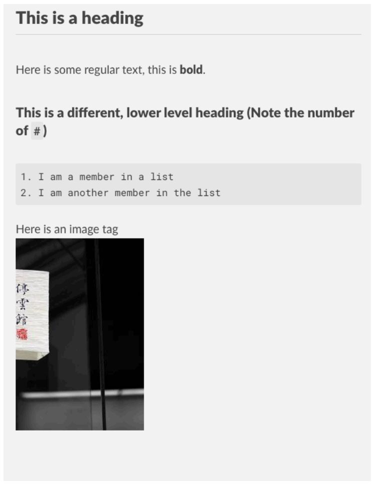
As you can see the bits in the `<b></b>` are formatted as per the tags surrounding them (the `<b><\b>` tags mean bold). Markdown performs a similar job, it is a lightweight markup language (hence mark-*down*), that can do a lot of formatting, and unlike HTML still looks readable in raw text without rendering it. It is a simple base format and can easily be converted into many formats.

1.2.1 Basic markdown tags

Here is a selection of basic markdown tags.

```
## This is a heading

Here is some regular text, this is **bold**.

### This is a different, lower level heading (Note the number of `#`)

    1. I am a member in a list
    2. I am another member in the list

Here is an image tag

![A Random Image](https://picsum.photos/200/300)
```

When rendered, this is going to look like this



This is a heading

Here is some regular text, this is **bold**.

This is a different, lower level heading (Note the number of `#`)

```
1. I am a member in a list
2. I am another member in the list
```

Here is an image tag

More markdown tags are available, hopefully these give you an idea of how markdown works. Here is a helpful 'cheat-sheet' with many more tags you can use, RStudio RMarkdown Cheat Sheet.

1.3 R Markdown

R Markdown is an extension of markdown with R mixed in. The markdown syntax is extended using special blocks that contain R code. When it comes to rendering time, the R is run and the results pushed into the rendered document. By mixing these together we get a tool which we can use to get a literate programmed document that can fulfil all our responsibilities.

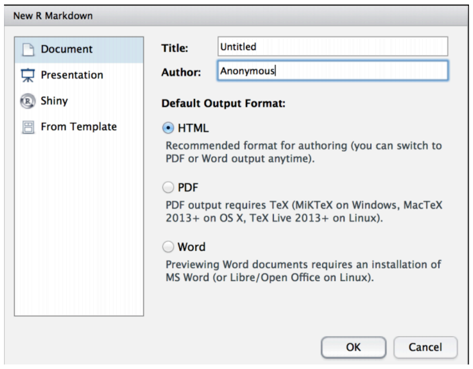
The R code block we mix in with our regular markdown is structured with ``` (backticks, not quotes) and `{r}`, in practice a block looks like this

~~~~
```{r}

print("Hello, World!")

```
~~~~

Any R code can go in the blocks, and the program remembers state between blocks so that stuff you do in earlier blocks is remembered in later ones.

1.3.1 Using R Markdown in RStudio

RStudio provides a lot of tools for creating RMarkdown documents easily. To get a basic template document you can use the menu's `File -> New File -> R Markdown`. On doing this you see a dialogue box, usually you want to set the options as in the figure below (you can put your name and a title in the relevant boxes)



Title: Untitled
Author: Anonymous

Default Output Format:

[✓ HTML]
Recommended format for authoring (you can switch to PDF or Word output anytime).

PDF
PDF output requires TeX (MiKTeX on Windows, MacTeX 2013+ on OS X, TeX Live 2013+ on Linux).

Word
Previewing Word documents requires an installation of MS Word (or Libre/Open Office on Linux).

When you do this you get a new draft document.

For you to do

Using RStudio, create a new RMarkdown document. Read it and compare it with what goes below.

Figure 1.1: Artwork by @AllisonHorst

1.3.1.1 Header Information

The first bit of the draft document is a bit of YAML markup that captures some information about the document. You don't need to change anything here if you don't want to, though it can be useful to use the information at this link to set the date automatically Dates in RMarkdown

```
---
title: "My Document"
author: "Dan MacLean"
date: "20/04/2021"
output: html_document
---
```

1.3.1.2 Setup Block

The next part is the automatically created set up block

~~~
```{r setup, include=FALSE}

knitr::opts_chunk$set(echo = TRUE)

```
~~~

This is a standard R block with some options set. The first bit '```{r setup,} is the standard block start, but this one is named `setup`. You don't need to name each block, but it can be helpful when bugs start to appear. The next bit `include=FALSE` is an option that tells the RMarkdown renderer *not* to include the code or it's output in the final document, this is because this is just setup code that we don't want messing up our nice output. Then we have the code that this block runs `knitr::opts_chunk$set(echo = TRUE)` which set up some formatting options. You can use this block for any bits of setup code that you don't want in the rest of your document.

1.3.1.3 Markdown and R blocks

The other blocks are the document's standard code and text blocks. Read them then perform this task

For you to do

Using RStudio, render the document using the `Knit` button at the top of the editor pane. Note how the code and results, including plots are mixed in the resulting document. Make some text edits and change the plot titles, then re-knit the document.

1.3.2 Conclusion

This all there really is to RMarkdown documents, they are a very easy to use tool for keeping your analyses readable and reproducible. Inspect the cheat sheet a little further for tips on what more you can do in RMarkdown, RStudio RMarkdown Cheat Sheet.

Roundup

RMarkdown documents are excellent tools for creating readable, attractive and dynamic reproducible literate programming documents.

2 Python and Jupyter Notebooks

2.1 About this chapter

1. Questions
    - How do I write a reproducible analysis document in Python with Jupyter??
2. Objectives
    - Mix markdown and Python with Rmarkdown
    - Run an interactive analysis document
3. Keypoints
    - Jupyter notebooks are excellent tools for creating readable, attractive and dynamic reproducible literate programming documents.

2.2 Jupyter Notebooks

A Jupyter Notebook is really quite similar to an RMarkdown document in that it mixes code and markdown in different blocks and allows the user to create a reproducible document. Jupyter varies from RMarkdown slightly in that it prioritises interactive work, instead of rendering a whole document, each block is run individually. Making the whold process a bit more step-by-step than in R Markdown.

2.3 Trying Out Jupyter on The Web

Because Jupyter notebooks are a bit fiddly to get started, there is an online test notebook set up by the Jupyter authors - Classic Jupyter Notebook. It can be a bit busy so may take a few reloads to get started.

For you to do

```
* Go to the link above and start a classic Jupyter notebook.
* Work through the `Notebook Basics` section.
* Start a new workbook and play around mixing Python Code and Markdown.
* Save as an html file when you are done
```

2.4 Starting Jupyter

2.4.1 macOS and Linux

To get Jupyter running on your machine, you need to have Python 3, and the `jupyter` and `ipython` packages installed. That is all listed in the set up of this course.

Once those are installed you can start a new notebook server by typing `jupyter notebook` at the terminal. Depending on the particulars of your local set up you may need to select an appropriate conda environment beforehand. A new server should start showing the filesystem in your web browser, use the buttons in the web page to start a new notebook.

2.4.2 Windows

There should be a Start Menu item that will let you kick off a new Jupyter notebook server. Simply click on that.

For you to do

```
* Start a new workbook and play around mixing Python Code and Markdown.
* Pay attention to the block type options dropdown in the interface (under the `Widgets` me
* Save as an html file when you are done
```

Roundup

Jupyter Notebooks are excellent tools for creating readable, attractive and dynamic reproducible literate programming documents.

Prerequisites

Knowledge prerequisites

The materials in this book assume that you already know something (but not necessarily a great deal) of the languages R and Python, so there won't be any introduction to the languages themselves. The rest of this chapter will help you set up the software you need to practice with those tools.

Software prerequisites

You need to install the following stuff for this book:

1. R
2. RStudio
3. Some R packages: `rmarkdown`, `knitr`
4. Python 3 via Anaconda
5. A reasonably recent web-browser

Installing R

Follow this link and install the right version for your operating system https://www.stats.bris.ac.uk/R/

Installing RStudio

Follow this link and install the right version for your operating system https://www.rstudio.com/products/rstudio/download/

Installing R packages in RStudio

Standard packages

In the RStudio console, type

```
install.packages(c("rmarkdown", "knitr"))
```

Installing Python 3 with Anaconda

Follow this link and install Python *3.x* for your operating system. https://www.anaconda.com/distribution/

Note for macOS users

Accept all of the defaults during installation

Here is a video tutorial https://www.youtube.com/watch?v=TcSAln46u9U

Note for Windows users

Install Python 3 using all of the defaults for installation except make sure to check **Add Anaconda to my PATH environment variable**.

Here is a video tutorial https://www.youtube.com/watch?v=xxQ0mzZ8UvA

Note for Linux Users

You'll need to be able to use the command-line to install with Anaconda. If you aren't comfortable with this, ask for assistance from the local support team.

1. Open https://www.anaconda.com/download/#linux with your web browser.
2. Download the Python 3 installer for Linux.
3. Open a terminal window. 4.Type `bash Anaconda3-`and then press Tab. The name of the file you just downloaded should appear. If it does not, navigate to the folder where you downloaded the file, for example with: `cd Downloads`. Then, try again.
4. Press `enter`. You will follow the text-only prompts. To move through the text, press the `spacebar`.
5. Type `yes` and press `enter` to approve the license.
6. Press `enter` to approve the default location for the files.

7. Type `yes` and press `enter` to prepend Anaconda to your `PATH` (this makes the Anaconda distribution the default Python).
8. Close the terminal window.

Starting a Jupyter Notebook

macOS

1. Start the `Terminal` application in `Applications -> Utilities`
2. Type `jupyter notebook`, it should start in your web browser

Windows

1. From the Start menu, search for and open `Anaconda 3` or `Jupyter Notebook`. You should be able to start a notebook directly by clicking the `Jupyter Notebook` icon.

Linux

1. Open the terminal application. It is *usually* in the task bar or dock
2. Type `jupyter notebook`, it should start in your web browser

Installing Python Packages with `conda`

You can use `conda` to install new Python packages using the Terminal by typing `conda install <package_name>`.

You can install the required packages with the following commands:

```
conda install jupyter
```

Accept all defaults when the system asks a question.