Authoring in R

Colin Farquhar

17/06/2019

Contents

1 Learning Objectives

- Know how RMarkdown and Pandoc extend Markdown
- Create documents in multiple output formats
- Understand the syntax needed to create them
- Knit a document from the command line

Duration - 60 minutes

In the last lesson we looked at how we can use markup languages to enable more consistent formatting of our documents. We won't draw a line under it there though; now we're going to look at some of the tools RStudio has which we can use to deliver different styles of presentation.

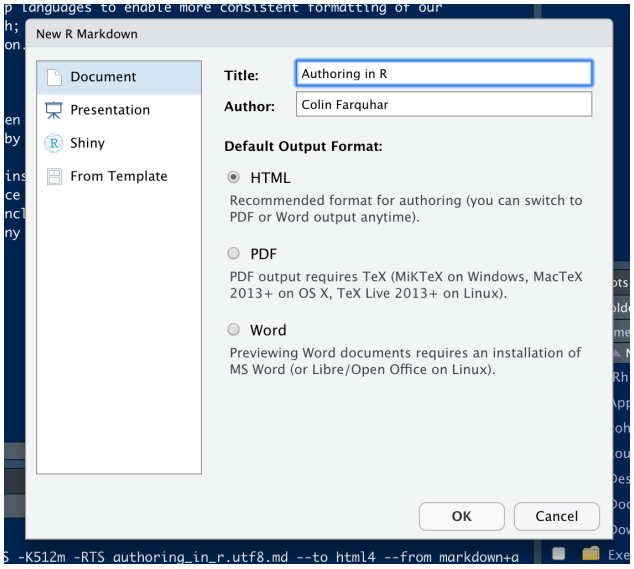


Figure 1: Creating a New RMarkdown file

Figure 2: Inline Blocks

2 RMarkdown

You've been using RMarkdown a lot already without even knowing it – most of your course notes have been written using it! You can tell if a file is using RMarkdown by its `.Rmd` extension.

The good news is that you don't need to do any more installation or setup; RStudio comes with a version of RMarkdown included. It's easy to get started with as well, since it is simply an extension of the markdown syntax we were just looking at. Now, though, we also have the power to include snippets of code in our document which can be run when we generate our output, which can itself take one of many different presentational forms.

Creating a new RMarkdown file is essentially the same as creating any other type of file: `File > New File > R Markdown`. The image above shows the GUI window for new file creation, which includes different options for the default output format. For now we'll leave this set to HTML; we'll look at the other options later in the lesson.

2.1 Embedding R Code

Often we'll want to include the results of evaluating some expression in R in our code. The easiest way is to do it *inline* by denoting the start and end of an expression using backticks – `. We've been doing this a lot already, and it's inherited from markdown. With only the backticks, `we can highlight text like this`.

In RMarkdown we can add a marker to say that the block should be evaluated as R code, and it will be when we compile the document. We can do that simply by adding the letter `r` after the first backtick. Take a look at the picture below to see how I can tell you that my favourite number is 5.

```
n tell you that my favourite number is `r 2 + 3`.
```

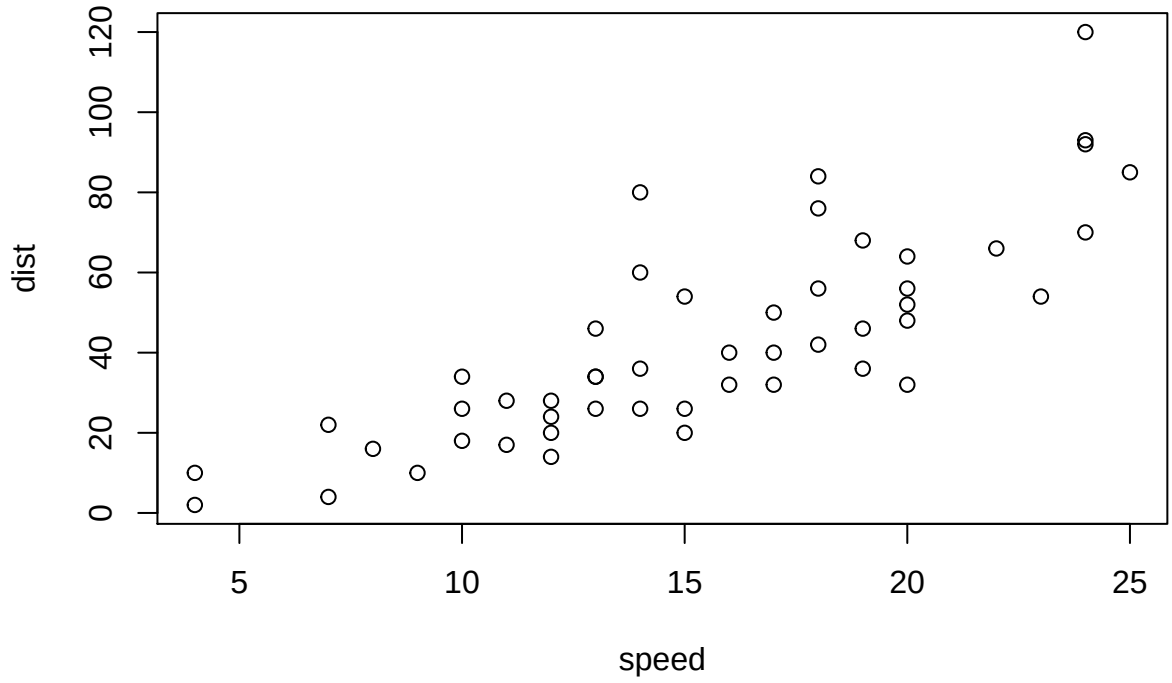
Figure 3: Inline Blocks with Syntax Highlighting

2.2 Code Blocks

Inline blocks are fine for brief snippets of code, but less useful if we want to evaluate larger or more complex expressions. The principle is the same, although now we use triple instead of single backticks (“‘). We can still highlight that code should be evaluated using R, but now we need to wrap it in braces: `{r}`.

The default behaviour is for the code to be evaluated at compile time and then included alongside the result. Let’s generate a plot using one of RStudio’s built-in datasets as an example:

```
plot(cars)
```



That’s great if we want to show the user how we generated our output, but it’s not always practical. Cluttering a presentation with details of calculations can diminsh it’s impact and that effect is magnified when the calculations are particularly complex or repetitive. Instead we can hide the details by setting the *echo* flag to be `false`. Doing this doesn’t affect the running of the code, but stops it being included in the final document.

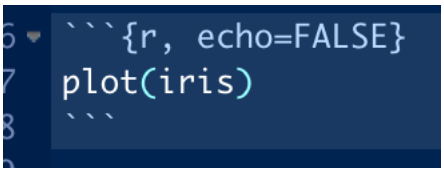


Figure 4: The code to generate the plot below

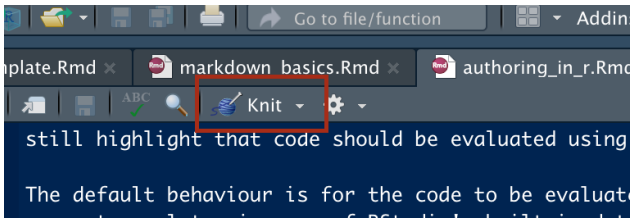
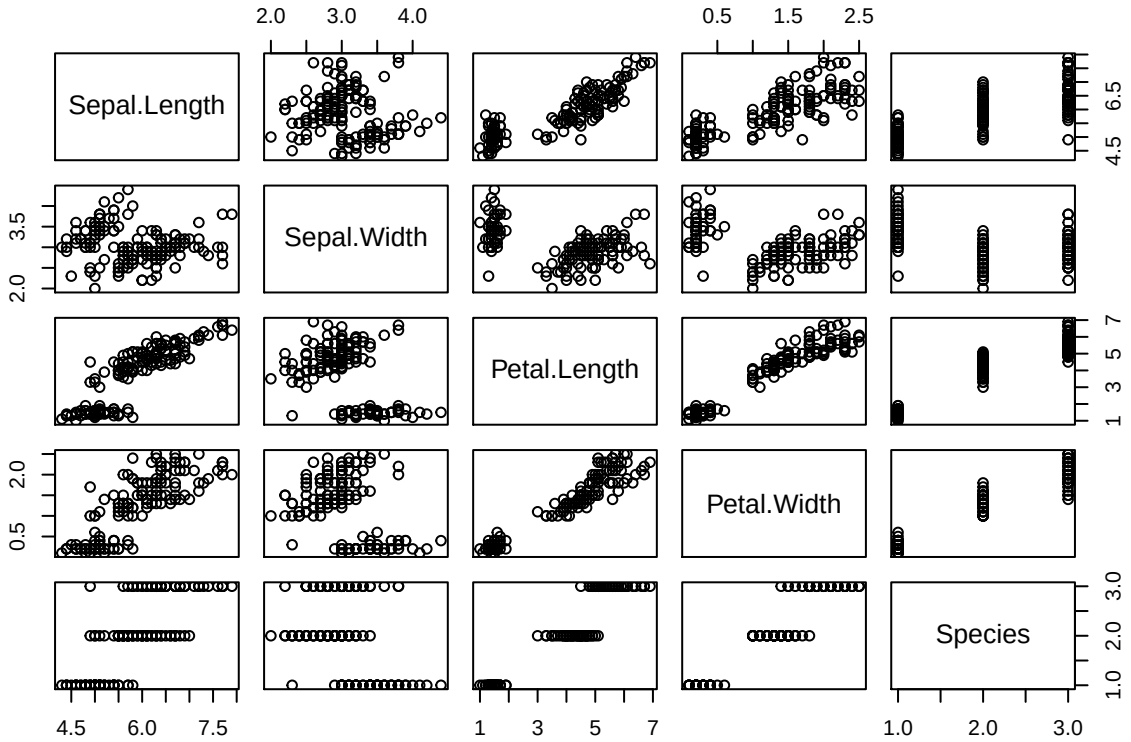


Figure 5: The knit button



3 Pandoc & Knitr

To generate an output we need to tell RStudio to *knit* our document. We use a package called *Knitr* which comes pre-installed to do this, using the handy button in the toolbar:

This carries out step one of the process by generating a markdown file from our R Markdown, eg. creating

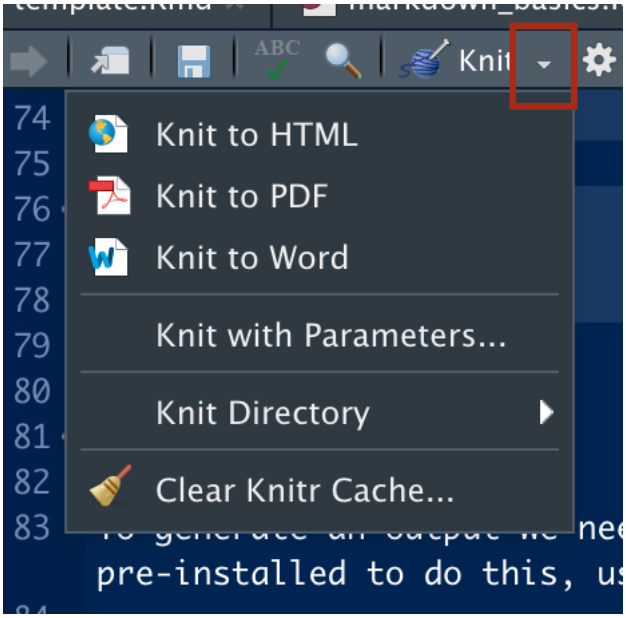


Figure 6: Knitting output options

`my_file.md` from `my_file.Rmd`. This is an important step, but doesn't really get us any closer to a presentable document. That stage is handled by *Pandoc* – a package which converts the markdown file we get from Knitr into the format of our choosing. When we create a document we can choose the default file type for our output – for these notes we chose HTML – but we can change the output at any time by selecting an alternative from the dropdown next to the knit button.

If we had selected "presentation" as our document type when we created it we would also have options to generate different types of slides in the dropdown.

Note that creating a presentation in one of the many slide formats can re-introduce many of the formatting problems that we're trying to avoid by using markdown! If creating slides for anything other than personal use or printing for handouts, check which formats will be supported by the tech available for your presentation. If in doubt, go with pdf!

3.1 YAML

If you look at the top of an RMarkdown file (such as this one) you'll see a block of code which looks like it bears some relation to the output document, but it may not obvious how. What you're seeing are **key-value pairs**, a common structure across many programming languages where we want to set some parameter (a key) to a given value, eg. setting the document's `title` key to the value `"Authoring in R"`.

The language used here is **YAML** which (depending on who you ask) stands for either ***Y**et **A**nother **M**arkup **L**anguage* or ***Y**AML **A**in't **M**arkup **L**anguage*. Yes, really. The important thing is that it is made up of key-value pairs, with it being possible to nest further pairs inside a value (like the value of the `output` key in the above image). That means that we can customise our output, for example by adding styling using CSS (more on that next lesson).

4 Different Types of Output

The final appearance of our document and the options available to customise it will vary according to the output type selected.

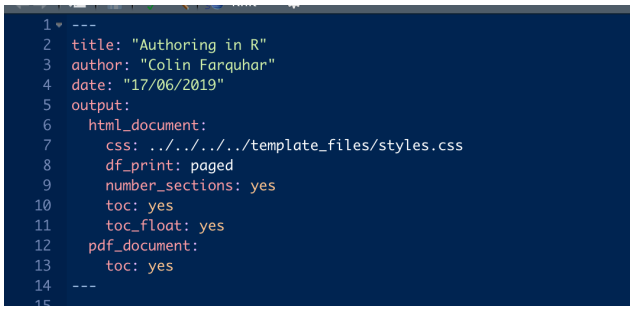


Figure 7: YAML configuration

Note that creating a presentation in one of the many slide formats can re-introduce many of the formatting problems that we're trying to avoid by using markdown! If creating slides for anything other than personal use or printing for handouts, check which formats will be supported by the tech available for your presentation. If in doubt, go with pdf!

Figure 8: No sign of our styling...

4.1 HTML

The document you're reading right now has been prepared using HTML! We can specify a wide range of options and import custom styling from a CSS file. We can also include HTML tags in our document and they will be interpreted correctly when the output is generated, such as the blue box we saw earlier with the notes about slide formatting.

4.2 PDF

Knitting to pdf will generate slightly different results to HTML. All of the content will still be there, but some of it may not be interpreted in the same way. Look at our blue box from earlier, for example:

This has come about because Pandoc compiles pdfs using LaTex, which doesn't recognise the HTML `<div>` tags we used. The tags won't be included in the output since the angle brackets have a similar purpose in LaTeX, but it doesn't know "div" means and so ignores the tags while printing the content. We have similar issues with trying to apply CSS styling. Trying to style LaTeX could be a whole course by itself and so we won't go into it here, but since it is widely used in academic writing the default formatting already has a professional appearance, plus almost guarantees consistency with other documents prepared in this way.

4.3 Word

Most of us will have, at some time, prepared a document using a word processor. One of the options Pandoc gives us is preparing our RMarkdown to look like a Word document, specifically one with a `.docx` extension. It has the advantage of looking like a document format that most people are familiar and comfortable with, but the same disadvantages as knitting to pdf in that our styling disappears.

Note: Pandoc generates `.docx` files, which is the file extension used by Microsoft Word 2010+. We're using MacBooks for this course, which means we might find some unusual interactions when when opening them in anything other than a viewer.

4.4 Powerpoint

5 Knitting From the Terminal

1. `Rscript -e "library(knitr); knit('my_file_name.Rmd')"`
2. `pandoc -f source_type -t output_type -o output_file_name.ext target_file.ext`

This can also be done by writing a script to handle both commands at once, but that is beyond the scope of this course.

6 Recap

- How to we prevent a code snippet being included when knitting a document?
  Answer By setting the `echo=FALSE` flag in the code block.
- Why do we include YAML in our files?
  Answer To store configuration information.

7 Additional Resources

- RMarkdown Cheatsheet
- Pandoc Documentation