R Markdown Tutorial

Sam Channon-Wells

30 October 2019

Introduction

Welcome to this R Markdown Tutorial! We will use a simple dataset (cars - provided with R) to show off some of the basic features of R Markdown. The aim of R Markdown is to allow you to keep your analysis code AND your writeup all in the same place, to make it easy to produce pretty, professional looking documents at the end of your work. R Markdown does this using code chunks, and text elements (what I am currently writing in now). Let's start off though with some basics.

First-off we will demonstrate some simple formating:

- To create a bullet point just use a double new-line, followed by either a dash (-), plus-sign (+), or an asterisk (*)
- To italize text just place one asterisk (*) either side of a *piece of text*
- To bold text just place two asterisks either side of a **piece of text**

To create a new paragraph you should put two spaces at the end of the sentence in the previous paragraph

We can also have multiple different levels of header, using the # symbol:

One dash is a LARGE header

Two dashes is slightly smaller

Three dashes is a bit smaller still

Four dashes is getting even smaller (do you get the picture…?)

There are many other things we can do, although the idea behind R Markdown is to have clean text, without too much over-formating. Afterall, we want to focus on the analysis and the science!

For more ideas look at the R Studio R Markdown cheatsheet

Section one - getting used to code-chunks

A code chunk allows us to write code, which will be executed when we "knit" our document. Knitting is the process used to turn your R Markdown document into the end product - typically an html document, but could also be a PDF document, or even a word document.

Code chunks start with three back-ticks, followed by '{r chunk_name, options}'

The simplest way to see this is with the below code chunk. It is called intro_chunk, and has only one option, echo = TRUE, which tells R Markdown to print the code as well as the results of the code:

```r
x <- rnorm(1000, mean = 2, sd = 1)
summary(x)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -1.401   1.289   1.966   1.964   2.635   5.133
```

The code above takes 1000 samples from a normal distribution with mean 2 and standard deviation 1. It then calls the function summary on the vector of samples. This is a great function, which quickly tells you the minimum and maximum values, the mean and median values, and the upper and lower quartiles, of a set of values. Pretty neat!

We now move on to something a little more interesting though.

Section two - looking at some data

We use the cars data which comes with R. I have moved this into a csv file, called “cars.csv”, which will be accessible in the github repository for this tutorial, available by clicking on the previous link.

We first read in the data, with read.csv. By including this simple function in our R Markdown document we ensure that people reading this know where the data comes from, making your work more **Reproducible**.

We then call the function “head” on the data just to get a quick look at it.

```
cars <- read.csv("cars.csv")
head(cars)
```

```
##   X speed dist
## 1 1     4    2
## 2 2     4   10
## 3 3     7    4
## 4 4     7   22
## 5 5     8   16
## 6 6     9   10
```

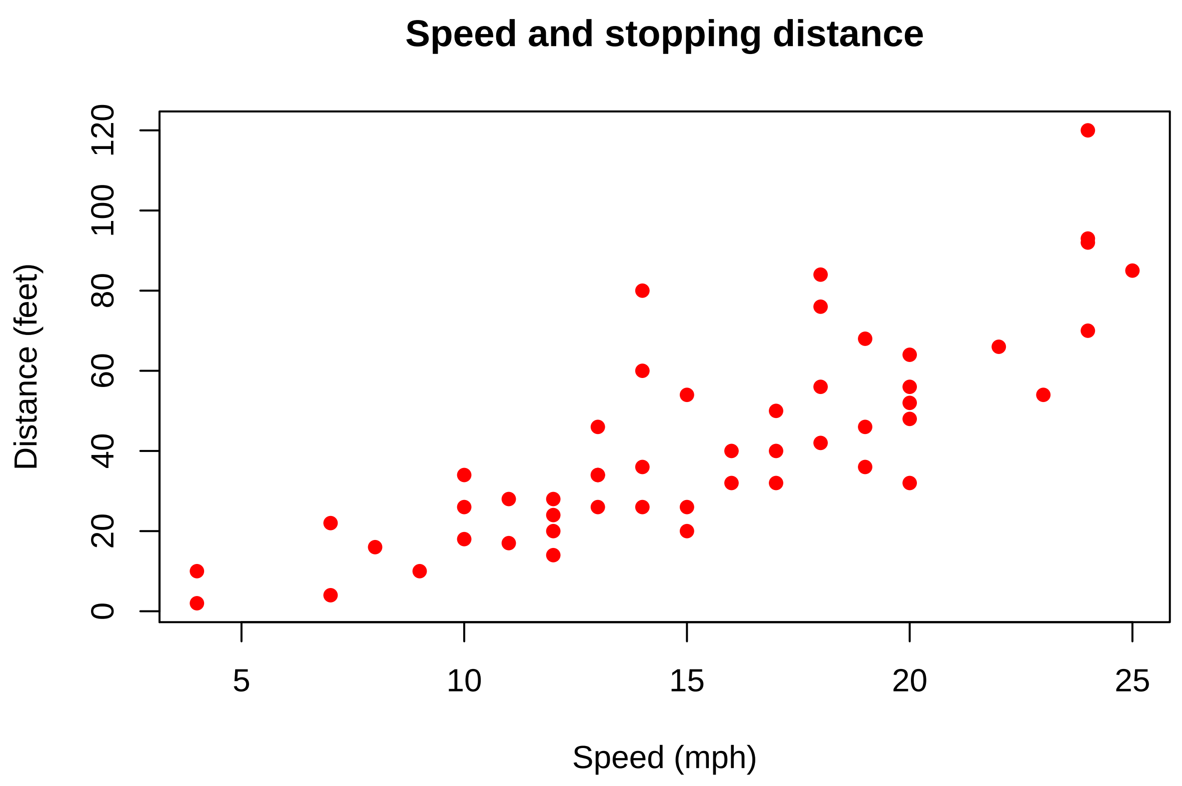
Section three - plotting data

Having looked at the data structure with the “head” function we always want to get a better feel for our data by plotting it. **Plot Plot Plot** to ensure you know what your data looks like, and what the errors are in it.

The code chunk below plots the two main variables in the cars dataframe - speed and distance. As you may have guessed, this data is about cars, and gives the speed of cars and the distances taken to stop. Note that the data were recorded in the 1920s!

A simple plot shows us that the two are probably related, so we fit a linear model to the data and then show the print out from this model.

```
plot(cars$speed, cars$dist, main = "Speed and stopping distance", xlab = "Speed (mph)", ylab = "Distanc
```



```r
# This looks correlated

fit <- lm(data = cars, speed ~ dist)

summary(fit)
```

```
##
## Call:
## lm(formula = speed ~ dist, data = cars)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.5293 -2.1550  0.3615  2.4377  6.4179
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  8.28391    0.87438   9.474 1.44e-12 ***
## dist         0.16557    0.01749   9.464 1.49e-12 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.156 on 48 degrees of freedom
## Multiple R-squared:  0.6511, Adjusted R-squared:  0.6438
## F-statistic: 89.57 on 1 and 48 DF,  p-value: 1.49e-12
```

```r
cor.coeff <- cor(cars$speed, cars$dist)
```

We also calculated the correlation coefficient in the code chunk above. We saved this as cor.coeff, so you won't have seen the result. However, we can access the variables from a previously run code chunk from inside our text elements, just by using **inline code**. We do this by writing the variable name in our text, with the letter r preceeding the variable name, and we wrap the text in backwards ticks. For example, to print the value of our correlation coefficient in our document we write: 0.8068949

Of course, if you're reading this document outside of R Markdown you will only see the value of the correlation coefficient, not the text. Unsurprisingly, the correlation coefficient is pretty high!

Section four - chunk options

Well that was interesting. We saw that speed and stopping distance are correlated. However, the code was a bit ugly, and we probably don't always want to include this in our final documents. This brings me on to chunk options.

Chunk options tell R Markdown what to do with code chunks. There are many many options, which gives R Markdown lots of flexibility. However, the main options you should definitely know about are:

- echo: set this equal to TRUE to print the code as well as the output of the code
- include: set equal to FALSE if you don't want the code OR output of the code to appear in your document
- eval: set equal to FALSE if you don't want the code to be evaluated (run) when knitting your document
- warning: set equal to FALSE to NOT print warnings to your document when code is evaluated (use with caution!)
- error: set equal to FALSE to NOT print error messages to your document (use with caution!)

We run the very first code chunk again, but now with echo = FALSE. In our end-document this will only show the summary print out of the 1000 normally distributed values, and not the actual code used to produce it.

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -1.094   1.330   2.028   2.010   2.659   5.348
```

You may have been wondering what the very first code chunk was doing. Take a look back at it now. It calls the function `knitr::opts_chunk$set`, and has the option echo = TRUE. What does this do? Well, it makes our life easier, by setting the default options for all subsequent code chunks to be echo = TRUE. This means we DONT have to keep writing echo = TRUE in the options section of each code chunk. This is called setting **global options**

Section five - knitting

We have covered a lot, but we haven't yet made any pretty documents like I promised! Let us fix that. R Markdown uses the `knitr` functions to generate html, PDF and word documents. However, if you use R Studio you don't really ever need to use these functions explicitly, since at the top of your R Studio workspace you will see the Knit button. This comes with a drop-down list, with a few options. However, at the moment we just want to knit this into a simple html document. If you click Knit it will do just that!

Clicking this button will run the code in your code chunks (as long as eval is not equal to FALSE…), and will then create a html document with code and output displayed alongside the text in your document. It's that simple! You can also knit to PDF and Word documents, although this can sometimes require downloading additional programs. A quick google is normally enough to help you with this.

The resulting document will be generated in the current working directory, which is usually the same place you have saved your R Markdown file to. For example, my current R Markdown document is called "RMarkdownTutorial.Rmd", and when I click Knit a second file appears in my directory, now called "RMarkdownTutorial.html". We can now share this with whoever we want to demonstrate how awesome our work is!

Conclusion

I hope you enjoyed this whistle-stop tour of R Markdown. It's a useful tool, and is pretty simple to learn. It's also very versatile, and includes options for writing code chunks in other languages (e.g. Python). The best place to look for guidance on other things not covered here is on the R Studio R Markdown cheat sheet page (link above).

Remember, the overall aim is to make sharing your work easier, to make it more reproducible, and to help you focus on analysing your data, NOT on thinking about formating and other less interesting bits of your work.

If you have any comments or feedback please drop me an email at: samuel.channon@cantab.net

Thanks for reading!