RESOURCE REVIEW

Knitting Statistics Notes: Creating Teaching Materials with Data Analysis Code and Results Embedded

Nicola Reeve, University of Leicester, Leicester, UK. Email: nfr5@le.ac.uk

Abstract

R is a widely used, free, open source software package for data analysis and graphics. Along with core functionality, there are many additional packages to enable specialist usage. One of these packages 'knitr' allows the integration of R and LaTeX code. This means that code, output and narrative are all included in one source document. Output is generated automatically from the code. If any changes are made to the data, all output is updated automatically. From a teaching perspective, this is particularly useful if you want to create notes with various graphs, statistics etc and you also want to provide the code that shows the student how to do this analysis in R. It is possible to hide either the code or the output. For example, if you are making introduction to statistics worksheets for non-statistics students, you might not want to include R code. If you are including exercises for students, you might not want to include the output and instead have them answer some questions. You can also set these options globally then by changing one option, create two versions of the document - one with solutions, one without, just from one source document.

Keywords: R, knitr, teaching notes, embedded code, individualised assessments.

1. Background

Literate programming is a programming paradigm conceived by Donald Knuth (Knuth 1984), originally applied to writing software. When developing software, large amounts of documentation are produced, including information for the development team, for maintenance engineers, and for end users, in addition to code comments. The idea of literate programming is to keep the source code for software together with its documentation. Rather than writing instructions to a computer, and then writing documentation to explain the code to a human, literate programming suggests writing a single document that is both the code and its natural explanation and presentation. This is achieved by writing a narrative that interweaves segments of code with text which explains their purpose and operation.

Here we apply the same idea to writing documents such as teaching materials. The narrative is kept together with all code used to produce the output in the document. This means any user (including your future self!) can easily see what data was used, what analysis was done and how (though there are options to hide various elements in the final document if required). This has obvious links to reproducible research - the authors' methods are clearly given along with the output.

2. R

This article is about the knitr package (Xie 2016), which applies literate programming to R (R Core Team 2015). R is a programming environment for data analysis and graphics. While it is a programming environment and is not 'point-and-click', all but advanced functionality can be achieved using simple commands, so it is relatively easy for a beginner to get started. There is

also a big, active internet community meaning there are many free resources and forums available to provide help and guidance.

R is a widely used, free, open source software package and is multi-platform (i.e. it can be used on Windows, Mac or Linux). This means there is very little restriction to its usage in terms of either cost or availability - anyone with access to an internet connection can begin using it right away. There are also no recurring costs such as annual licence renewal fees. Updates are simply downloaded when convenient to the user - you do not find the software stops working on an arbitrary date when the licence runs out.

'Free' really does mean free - there is no subscription, advertising, hidden spyware or malware. Open source means that anyone with the technical know-how can see exactly what the code does so there is complete transparency of its contents and functionality. Users are able to confirm for themselves exactly what a function does and to suggest improvements to the code.

The standard R installation has core functionality, which is sufficient for a beginner and for common or basic analysis. For more advanced functionality, such as advanced graphics, or specific types of statistical analysis, such as spatial statistics or missing data analysis, R has thousands of specialist packages available to download as required, making it very flexible. Users are also able to create their own packages, which after submission to the R development team for checking, can be made available to all users.

R also includes many datasets as standard. A list of available datasets can be seen by typing 'data()' into the R console. These are freely available for use and are often used in examples in the R documentation to illustrate how to use a function.

R and its additional packages are available to download from the Comprehensive R Archive Network (CRAN, (R Core Team (2015)). The CRAN website also provides manuals and introductory guides.

The standard installation of R has a basic interface, but there are several additional graphical interfaces which can be more user friendly. Those available as freeware include Tinn-R, RKWard and RStudio. RStudio is particularly helpful for knitr and is discussed in more detail in Section 3. These require R to be installed first, then a separate install for your chosen interface, in much the same way as you would install LaTeX on your system, then your chosen tex editor. Knitr can then be installed using the 'install.packages()' function in R.

3. RStudio

In addition to being freeware, RStudio (RStudio, 2016) is multi-platform (runs on Windows, Mac, Linux) and open source. Use of RStudio for working with R generally, or the knitr package specifically, is optional but has several benefits including:

- multiple windows displaying various aspects (discussed below)
- syntax highlighting
- version control
- LaTeX integration

The RStudio interface consists of several windows (see Figure 1):

- Bottom left: console window (also called command window). Here you can type simple commands after the '>' prompt and R will then execute your command.
- Top left: editor window (also called script window). Collections of commands (scripts) can be edited and saved. If you don't see this window initially, you can open it with File -> New -> R script. Typing a command in the editor window will not run it. If you want to run a line from the script window (or the whole script), you can click Run or press CTRL+ENTER to send it to the command window.
- Top right: workspace / history window. In the workspace window you can see which data and values R has in its memory. The history window shows what commands have been typed before.
- Bottom right: files / plots / packages / help window. Here you can open files, view plots, install and load packages or use the help function. You can change the size of the windows by dragging the grey bars between the windows. All the plots you generate will be kept in the plots window and can be exported as pdf, jpg etc.

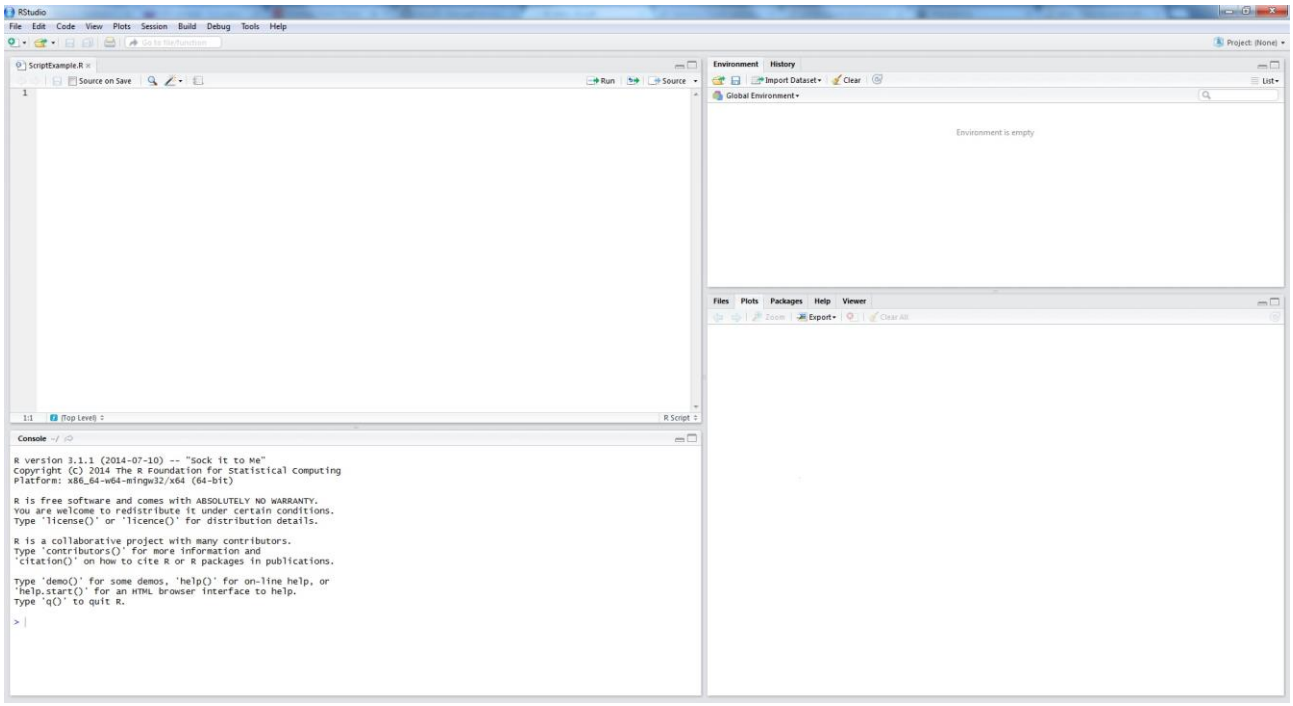


Figure 1. RStudio start up.

4. The knitr Package

The knitr package in R allows the integration of R and LaTeX, employing full LaTeX functionality. This means that R code can be embedded within the LaTeX code, so code, output and narrative are all included within the same document. From a teaching perspective, this is particularly useful if you want to create notes with various graphs, statistics etc and you also want to provide the code that shows the student how to do this analysis in R. If you want to create individualised tasks for students for coursework purposes, you can use random number generation in your R code so each student has different data. You then create the document multiple times, one for each student. Each copy of the document can be created by a single button click (see Section 5.3 for details). All

plots, statistics etc in the document would be created automatically using the individualised data. It is also possible to automate the creation of multiple, individualised copies by using simple script, so if you have 100 students, you do not have to create 100 copies yourself. Using a specified random seed means that while data is created/altered randomly, you would be able to recreate it at a later date and see what results a particular student should have.

5. How To

5.1. An Example

R code embedded in LaTeX code is saved as a .Rnw file, essentially just a text file. Xie (2015) provides a minimal example:

```
\documentclass{article}
\begin{document}
\title{A Minimal Example}
\author{Yihui Xie}
\maketitle

We examine the relationship between speed and stopping distance using a linear regression model:
$Y=\beta_0 + \beta_1 x + \epsilon$.

<<model, fig.width=4, fig.height=3, fig.align='center'>>=
par(mar=c(4,4,1,1), mgp=c(2,1,0),cex=0.8)
plot(cars, pch=20, col = 'darkgray')
fit<- lm(dist ~ speed, data = cars)
abline(fit, lwd = 2)
@

The slope of a simple linear regression is
\Sexpr{coef(fit)[2]}.
\end{document}
```

In R, running the ‘knit' function on the .Rnw file produces a .tex file, which can be compiled in the usual way to produce a PDF. If using RStudio, this process can be reduced to a single button click (see Section 5.3). For the minimal example, this produces the document shown in Figure 2. The first few lines of code will be familiar to LaTeX users. The R code lies between the ’<<>>=' and ‘@' markers - these define where the R code starts and ends. Sections of code are referred to as ‘chunks'. In this example, the R code sets some graphical parameters, plots the ‘cars' data (supplied with R), fits a linear model, then adds the fitted regression line to the plot. Options for the chunk are given between the ‘<<' and ‘>>=' markers. Here they give it a name: ‘model' so it can be referred to elsewhere, and set the size and alignment of figures produced by the chunk. The penultimate line embeds inline code, giving the regression coefficient in this case.

5.2. Updating Documents and Further Options

If I want to change the dataset that has been used, or if the dataset has been updated to a more recent version, I simply make the appropriate change to the ‘data' argument within the R code chunk and re-run knitr to create the updated output document. If I was using other statistical and word processing packages, for example SPSS and Microsoft Word, I would likely need to manually re-do the analysis and change all of the figures and results. Similarly if I want to make a change to a figure, I don't have to re-make the figure, save or copy it, and insert it into the document. I just make the change in the R code chunk and re-run knitr. When the data or analyses are changed, it is easy for errors to occur in the document as a consequence of some results not being updated to the latest version. Here, this issue does not arise as everything is updated automatically.

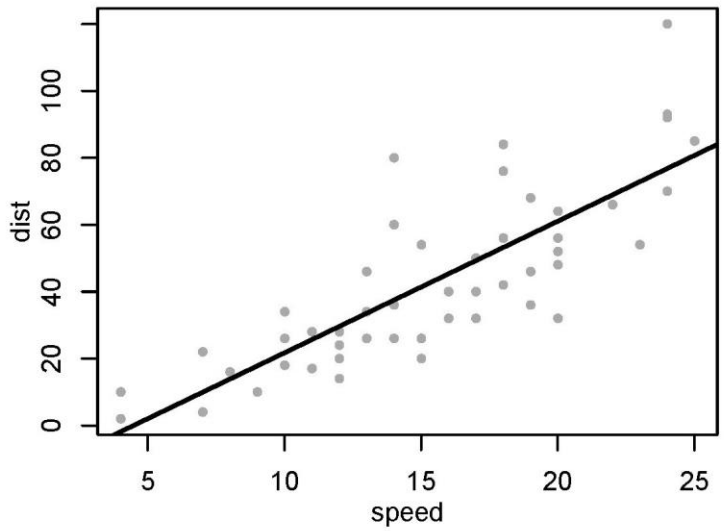
A Minimal Example

Yihui Xie

July 1, 2016

We examine the relationship between speed and stopping distance using a linear regression model: $Y = \beta_0 + \beta_1 x + \epsilon$.

```
par(mar=c(4,4,1,1), mgp=c(2,1,0),cex=0.8)
plot(cars, pch=20, col = 'darkgray')
fit<- lm(dist ~ speed, data = cars)
abline(fit, lwd = 2)
```



The slope of a simple linear regression is 3.9324088.

Figure 2. Output of minimal example

You can use the chunk options to either show or hide code and output for different audiences or uses. For example, if you are making introduction to statistics worksheets for non-statistics students, you might not want to include R code. If you are including exercises for students, you might not want to include the output and instead have them answer some questions. To hide the R code, use the option 'echo=FALSE', to hide the output, use the option 'eval=FALSE'. It is also possible to set these options globally so they apply to the whole document, therefore by changing one option, you can have two different versions of the document - one with code/output and one without. This could be useful if you are creating worksheets and want to create a version with the solutions included.

While it is the focus of this article, a PDF is not the only possible output from knitr. For users wanting to create webpage content, Markdown can be used instead of LaTeX, with R code

embedded in the same way. Instead of saving as a .Rnw file, code is saved as a .Rmd file and the output is Markdown which can be used to produce HTML. See Xie (2015) for further details.

5.3. Setting Up RStudio for knitr

The first step is to install R, the additional R package, knitr and RStudio (RStudio is optional but this section assumes you will be using it). Then you need to set up RStudio for knitr. Go to tools-> Global options, click the 'sweave' tab on the left (Figure 3) and change 'weave Rnw files using' to knitr. These steps only need to be completed once when first setting up your system. Having done this, when you have a .Rnw file open in RStudio, you should see a 'compile PDF' button on the toolbar. Clicking this will run the knit function on the file, then run pdflatex on the resultant .tex file, giving you the final PDF output. It will also run bibtex if you are using it for references within your document. So knitting the .Rnw file, compiling the .tex file and running bibtex is all achieved with one button click!

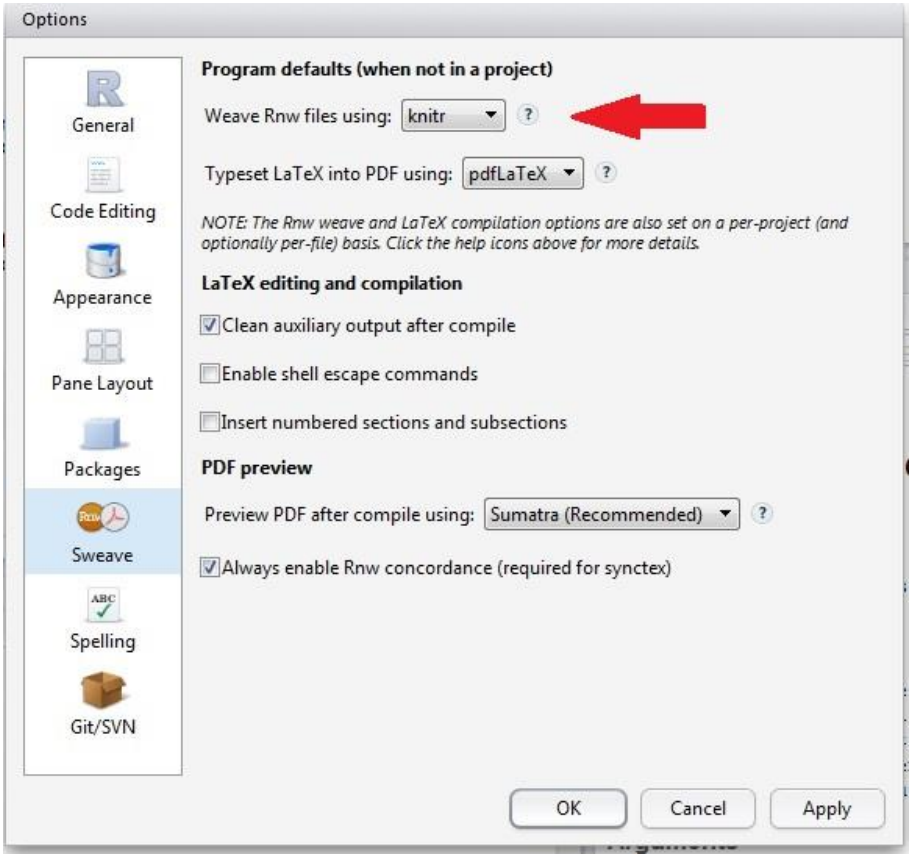


Figure 3. Setting RStudio options for knitr

6. More Examples

Let's look at some more examples using the cars dataset from R. This dataset contains the speed and stopping distances of 50 cars, recorded in the 1920s. The following code will produce a histogram. We just include the code within a code chunk and the histogram will automatically be produced and included in the final document.

```
hist(cars$speed, main="Speed of cars in the 1920s", xlab="Speed (mph)")
```

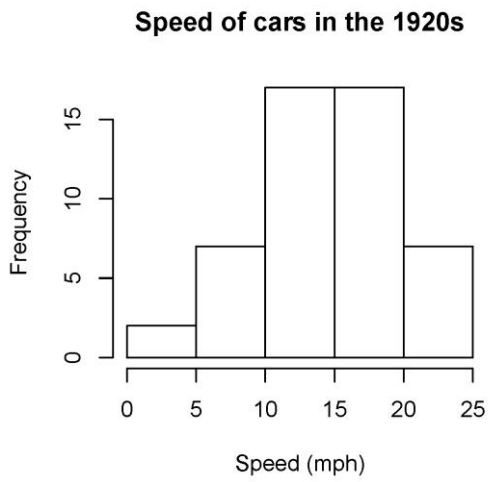


Figure 4. Automatically generated histogram

Some descriptive statistics:

```
colMeans(cars)
 ##   speed   dist
 ##   15.40   42.98
apply(cars,2,sd)
 ##   speed        dist
 ##   5.287644     25.769377
```

Again, R code is included in a code chunk and the output is produced automatically, preceded by '##'. For cars in the 1920s, the average speed was 15.4, with a standard deviation of 5.29 and the average stopping distance was 42.98, with a standard deviation of 25.77. Here, as in Xie's minimal example, we use the command Sexpr{} to embed inline code: rather than writing out the means and standard deviations, we include the code for it, so these numbers would automatically update if the data were changed. Another clear advantage is that this reduces the risk of typing errors: we simply indicate where we want the numbers to be and the software inserts them for us.

Many functions are available either in the core R packages, or in additional packages. However, sometimes it is helpful or necessary to be able to write your own. As always when using knitr, we can include the code within a code chunk and the output will be automatically generated and included in the document. For example, this very simple function will square whatever it is given as input.

```
simple<-function(x){  #creates a function named 'simple' which takes x as input
  y<-x^2              #everything inside the curly braces is what the function does
  return(y)           #the function will return the value of y
}
simple(2)             #calls the function, giving a value of 2 for x
## [1] 4
y                     #note that R does not know what y is outside of the function
## Error in eval(expr, envir, enclos): object 'y' not found
```

This is a very simple function and it was not really necessary to create a function: we could simply have written:

```
2^2
## [1] 4
```

But we can extend this function to include more lines of code, more complicated code, anything we want.

```
bigger<-function(x){
  y<-x^2
  z<-y+2
  w<-z^2
  return(c(x,y,z,w))      #note the use of c to return a vector
}

bigger(2)
## [1] 2 4 6 36
```

Additional packages can be installed using the install.packages function, for example, for the advanced graphics package ‘ggplot2':

```
install.packages("ggplot2")
```

Packages only need to be installed once per machine, but need to be loaded into the session each time you use it, using the ‘library' function:

```
library(ggplot2)
```

Along with the use of some other packages, ggplot2 can be used to make some appealing graphics, including a world map. As with the previous examples, including the following code in a code chunk will automatically generate the plot.

```
library(ggmap)

cities <- c("London ", "Sydney", "Paris", "Washington", "Moscow")
ll.cities <- geocode(cities) #find the longitude and latitude
cities.x <- ll.cities$lon
cities.y <- ll.cities$lat

#Using ggplot, plot the world map
mp <- NULL
mapWorld <- borders("world", colour="gray30", fill="lightblue")
mp <- ggplot() + mapWorld
#Add city locations as blue dots
mp <- mp+ geom_point(aes(x=cities.x, y=cities.y) ,color="blue", size=1)
mp
```

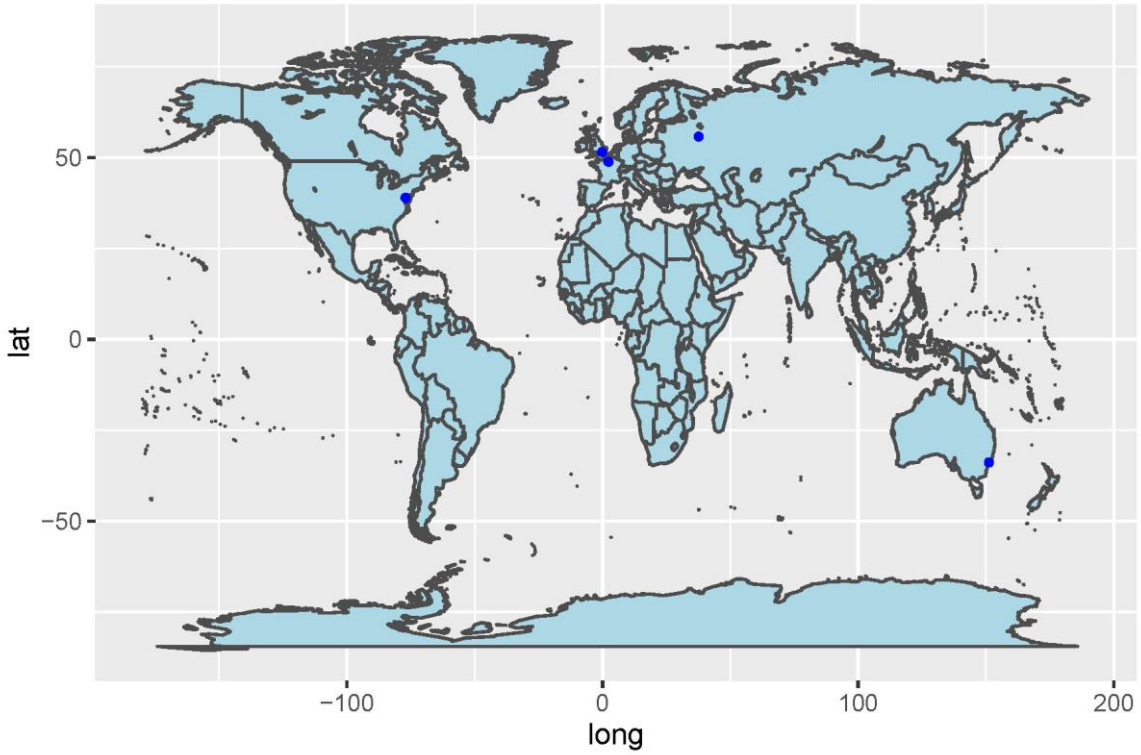


Figure 5. Automatically generated map

7. Further Reading and Information

For further information about knitr, examples and details of available options, see Yihui Xie's book (Xie, 2015) and website (http://yihui.name/knitr/). For examples of knitr reports created by users, see http://rpubs.com.

8. References

Knuth, D.E., 1984. Literate programming. *The Computer Journal*, *27*(2), pp.97-111.

RStudio, 2016. RStudio. www.rstudio.com.

R Core Team, 2015. R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. Available at: https://www.R-project.org/ .

Xie, Y., 2015. *Dynamic Documents with R and knitr*. Boca Raton: CRC Press.

Xie, Y., 2016. knitr: A General-Purpose Package for Dynamic Report Generation in R. R package version 1.15.1.