1

Statistical Modelling and knitr

CONTENTS

1.1 Incorporating analyses into the markup

1.1.1 Full code in the main document

1.1.1.1 LaTeX

1.1.1.2 Markdown

1.1.2 Showing code & results inline

Sometimes we want to have some R code or output to show up in the text of our documents. We may want to include stylized code in our text when we discuss how we did an analysis. We may want to report the mean of some variable in our text.

1.1.2.1 LaTeX

inline static code

If we just want to include a code snippet in out text we can simply use the LaTeXcommand `\tt`. This sets our text to 'typewriter' font, the standard font for inline code in LaTeX(I use it in this book, as you have probably noticed).

inline dynamic code

If we want to dynamically show the results of some R code in our text we can use the `\Sexpr` command. This is a pseudo LaTeXcommand. Its structure is more like a LaTeXcommand's structure than `knitr` in that we enclose our R code in curly brackets ({}) rather than the usual <<>>`= . . . @` syntax for code chunks.

For example, imagine that we wanted to include the mean–591–in the text of our document. The *rivers* numeric vector, loaded by default in R, has the length of 141 major rivers recorded in miles. We can simply use the `mean` command to find the mean and the `round` command to round it to the nearest whole number:

```
round(mean(rivers), digits = 0)

## [1] 591
```

To have just the output show up inline with the text of our document we would type something like:

```
The mean length of 141 major rivers in North America
is \Sexpr{round(mean(rivers), digits = 0)} miles.
```

This will produce the sentence:

> The mean length of 141 major rivers in North America is 591 miles.

1.1.2.2 Markdown

inline static code

To include static code inline in an R Markdown document we enclose the code in single backticks (``). For example typing ` MeanRiver <- mean(rivers) ` produces `MeanRiver <- mean(rivers)`.

inline dynamic code

To include dynamic code in an R Markdown document we use the backticks as be fore but include a the letter `r` after the first one.

1.1.3 Sourcing R code from another file

There are a number of reasons that you might want to have your R source code located in a separate file from your markup even if you plan to compile them together with *knitr*.

First, it can be unwieldy to edit both your markup and long R source code chunks in the same document, even with RStudio's handy *knitr* code collapsing and chunk management options. There are just too many things going on in one document.

Second, you may want to use the same code in multiple documents–an article and presentation for example. It is nice to not have to copy and paste the same code into multiple places, but have multiple documents link to the same source code. Plus if you make changes to the source code, these changes will automatically be made across all of your presentation documents. You don't need to make the same changes multiple times.

Third, other researchers trying to replicate your work might only be interested in specific parts of your analysis. If you have the analysis broken into separate and clearly labeled files it is easier for these researchers to find the specific bits of code that they are interested compared to digging through long markup files.

1.1.3.1 Source from a local file

Usually in the early stages of research you may want to source analysis files located on your computer. Doing this is simple. The `knitr` syntax is the same as above. The only change is that instead of writing all of our code in the chunk we save it to its own file and use the `source` command in *base* **R** to access it. For example:

1.1.3.2 Source from a non-secure URL (`http`)

Sourcing from your local computer is fine if you are working alone and do not want others to access your code. Once you start collaborating and generally wanting people to be able to replicate your code, you need to use another method.[1]

The simplest solution to these issues is to host the replication code in your **Dropbox** public folder. You can find the file's public URL the same way we did in Chapter 6. Now use the `source` command the same way as before. For example:

[1]You can make the replication code accessible for download and either instruct others to change the working directory to the replication file or have them change the directory information as necessary. However, this usually just adds an extra complicating step that makes replication harder. It is also a pain if you are collaborating and each author has to constantly change the directories.

1.1.3.3 Source from a secure URL (`https`)

If you are using **GitHub** or another service that uses secure URLs the steps are generally the same, but you need to use the `source_url` command in the *devtools* package. For **GitHub** based source code we find the file's URL the same way we did in Chapter 6. Remember to get the URL for the *raw* version of the file.

1.2 Saving output objects for future use

1.3 Including highlighted syntax in the output

1.3.1 LaTeX

1.3.2 Markdown/HTML

1.4 Debugging