

Data Science with R

Documenting with Knitr

Graham.Williams@togaware.com

17th August 2014

Visit <http://HandsOnDataScience.com/> for more Chapters.

Transparency and efficiency are important to the data scientists. We need to record our activities for quality assurance and peer review. We also find ourselves repeating our work and so documenting what we do helps when we come back to it at a later time. We introduce here the concept of intermixing the narrative and our code, within the one dataset, known as [literate programming](#).

`knitr` (Xie, 2014) combines the typesetting power of \LaTeX with the statistical power of R (R Core Team, 2014). It is well supported by RStudio, the ESS-mode in Emacs, and by LyX. Each provide actions to process the source document into PDF with considerable ease.

The OnePageR documents themselves are all produced using `knitr`.

The required packages for this chapter include:

```
library(rattle)           # The weather dataset.
library(ggplot2)         # Data visualisation.
library(xtable)          # Format R data frames as LaTeX tables.
library(Hmisc)           # Escape special characters in R strings for LaTeX.
library(diagram)         # Produce a flowchart.
library(dplyr)           # Data munging: tbl_df() for printing data frames.
```

As we work through this chapter, new R commands will be introduced. Be sure to review the command's documentation and understand what the command does. You can ask for help using the `?` command as in:

```
?read.csv
```

We can obtain documentation on a particular package using the `help=` option of `library()`:

```
library(help=rattle)
```

This chapter is intended to be hands on. To learn effectively, you are encouraged to have R running (e.g., RStudio) and to run all the commands as they appear here. Check that you get the same output, and you understand the output. Try some variations. Explore.

Copyright © 2013-2014 Graham Williams. You can freely copy, distribute, or adapt this material, as long as the attribution is retained and derivative work is provided under the same license.



1 Project Report Template

To get started with `knitr`, we create a new text document with a filename extension of `.Rnw` (using `RStudio`). Into this file we enter the \LaTeX commands and text we want included in the document. \LaTeX is a markup language and so we intermix the \LaTeX commands, identifying the different parts of the document, with the actual content of the document. We can edit \LaTeX documents with any text editor—we illustrate the process here using `RStudio`.

The basic template on the next page serves as a starting point. It is a complete \LaTeX document. Copy it into an `RStudio` edit window—in `RStudio` create a new R Sweave document and paste the text of the next page into it. Save the file with a `.Rnw` filename extension.

It is then a simple matter to generate a typeset PDF document by clicking the PDF button in `RStudio`. `RStudio` supports both the older Sweave style documents and the modern `knitr` documents. We need to tell `RStudio` that we have a `knitr` document though—under Tools → Options, choose the Sweave icon and then choose `knitr` as the option for Weaving `.Rnw` files.

The document below is the result of doing this using the template from the next section.

Project Report Template

Graham Williams

17th August 2014

1 Introduction

A paragraph or two introducing the project.

2 Business Problem

Describe discussions with client (business experts) and record decisions made and shared understanding of the business problem.

3 Data Sources

Identify the data sources and discuss access with the data owners. Document data sources, integrity, providence, and dates.

4 Data Preparation

Load the data into R and perform various operations on the data to shape it for modelling.

5 Data Exploration

We should always understand our data by exploring it in various ways. Include data summaries and various plots that give insights.

6 Model Building

Include all models built and parameters tried. Include R code and model evaluations.

7 Deployment

Choose the model to deploy and export it, perhaps as PMML.

2 Sample Template

```
\documentclass[a4paper]{article}
\usepackage[british]{babel}
\begin{document}
```

```
\title{Project Report Template}
\author{Graham Williams}
\maketitle\thispagestyle{empty}
```

```
\section{Introduction}
```

A paragraph or two introducing the project.

```
\section{Business Problem}
```

Describe discussions with client (business experts) and record decisions made and shared understanding of the business problem.

```
\section{Data Sources}
```

Identify the data sources and discuss access with the data owners. Document data sources, integrity, providence, and dates.

```
\section{Data Preparation}
```

Load the data into R and perform various operations on the data to shape it for modelling.

```
\section{Data Exploration}
```

We should always understand our data by exploring it in various ways. Include data summaries and various plots that give insights.

```
\section{Model Building}
```

Include all models built and parameters tried. Include R code and model evaluations.

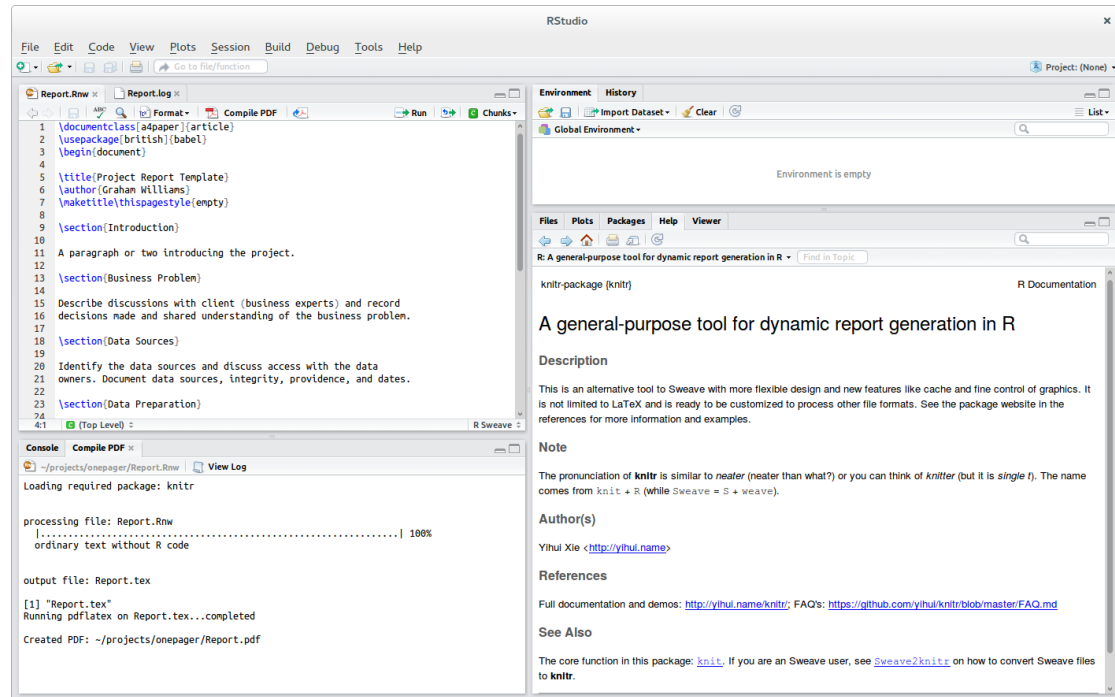
```
\section{Deployment}
```

Choose the model to deploy and export it, perhaps as PMML.

```
\end{document}
```

3 Process into PDF with RStudio

With a Rnw file loaded into RStudio for editing we click the **Compile PDF** button to build and preview the PDF. This will run the R code contained in the document, then process the document with \LaTeX to generate a PDF file, and then display the PDF file .



Be sure to select `knitr` under *Tools* → *Global Options* → *Sweave* → *Weave Rnw files using:*. If you have initially run your document using `Sweave` rather than `knitr` (RStudio defaults to `Sweave` at present) and then attempt to change to `knitr`, you may find the following *Undefined control sequence* error:

```
! Undefined control sequence.
```

```
1.54 \SweaveOpts
```

```
      {concordance=TRUE}
```

The control sequence at the end of the top line of your error message was never `\def`'ed. If you have misspelled it (e.g., ``\hobx'`), type ``I` and the correct spelling (e.g., ``I\hbox'`). Otherwise just continue, and I'll forget about whatever was undefined.

We will find this error message towards the end of the log file obtained by clicking the View Log button. A popup will also tell us that we can convert the file using `Sweave2knitr("Report.Rnw")`

RStudio has automatically inserted a required line for compiling with `Sweave`. As the error message suggests, go to the line containing the undefined sequence (line 54 is noted in this instance, but that is in the automatically generated `.tex` file rather than the source `.Rnw` file that we are editing so we will need to search for the string) and delete the offending \LaTeX control sequence and the bracketed argument. It should then run just fine in `knitr`.

4 Adding R Code

All of our R code can be very easily added to the knitr document and it will be automatically run when we process the document. The output from the R code can be displayed together with the R code itself. We can also include tables and plots.

R code is included in the knitr document surrounded by special markers—starting with a line beginning with double less than symbols (or angle brackets <<), starting in column one. This line ends with double greater than symbols (>>) followed immediately by an equals (=). The block of R code is then terminated with a single “at” symbol (@) starting in column one.

```
<<>>=  
... R code ...  
@
```

Between the angle brackets we place instructions to tell knitr what to do with the R code. We can tell it to simply print the commands, but not to run them, or to run the commands but don't print the commands themselves, and so on. Whilst it is optional, we should provide a label for each block of R code. This is the first element between the angle brackets. Here is an example:

```
<<example_label, echo=FALSE>>
```

The label here is `example_label`, and we ask knitr to not echo the R commands into the output. There might be a plot or two generated in the R chunk or perhaps a table and the output will be captured as a figure or \LaTeX table and inserted into the document at this position.

As an example we can generate some data and then calculate the mean. Here is how this looks in the source `.Rnw` file:

```
<<example_random_mean>>=  
x <- runif(1000) * 1000  
head(x)  
mean(x)  
@
```

This is what it looks like after it is processed by knitr and then \LaTeX :

```
x <- runif(1000) * 1000  
head(x)  
## [1] 605.0 602.3 158.5 940.1 388.1 792.3  
mean(x)  
## [1] 515.6
```

5 Inline R Code

Often we find ourselves wanting to refer to the results of some R command within the text we are writing, rather than as a separate chunk of R code. That is easily done using the `\Sexpr{}` command in \LaTeX .

For example, if we wanted today's date included, we can type the command sequence exactly as `\Sexpr{Sys.Date()}` to get "2014-08-17". Any R function can be called upon in this way. If we wanted a specifically formatted date, we can use R's `format()` function to specify how the date is displayed, as in `\Sexpr{format(Sys.Date(), format="%A, %e %B %Y")}` to produce Sunday, 17 August 2014.

We typically intermix a discussion of the characteristics of our dataset with output from R to support and illustrate the discussion. In the following sentence we do this showing first the output from the R command and then the actual R command we included in the source document. During the knitting phase, `knitr` runs the R commands found in the `.Rnw` file and substitutes the output into the generated `.tex` document. For example, the **weather** dataset from `rattle` (Williams, 2014) has 366 (i.e., `\Sexpr{nrow(weather)}`) observations including observations of the following 4 variables: MinTemp, MaxTemp, Rainfall, Evaporation (i.e., `\Sexpr{paste(names(weather)[3:6], collapse=", ")}`).

\LaTeX treats some characters specially, and we need to be careful to escape such characters. For example, the underscore "_" is used to introduce a subscript. Thus it needs to be escaped if we really want an underscore. If not, \LaTeX will get confused. An example is listing one of the variable names from the **weather** dataset with an underscore in its name: RISK_MM (`\Sexpr{names(weather)[23]}`). We will see an error like:

```
KnitR.tex:230: Missing $ inserted.  
KnitR.tex:230: leading text: ...set with an underscore in its name: RISK_  
KnitR.tex:232: Missing $ inserted.
```

The `Hmisc` (Harrell, 2014) package provides `latexTranslate()` to assist here. It was used above to print the variable name, using `\Sexpr{latexTranslate(names(weather)[23])}`.

There are many more support functions in `Hmisc` that are extremely useful when working with `knitr` and \LaTeX . Look at `library(help=Hmisc)` for a list.

Exercise: Investigate the \LaTeX support functions available in `Hmisc` and include some examples in your template `knitr` document.

6 Including a Table using `kable()`

Including a typeset table is quite simple using `kable()` as provided by `knitr`. Here we will use the larger `weatherAUS` dataset from `rattle`, setting it up as a `tbl_df()` courtesy of `dplyr` (Wickham and Francois, 2014), and choosing some specific columns and a random selection of rows to include in the table.

The text we include in our `.Rnw` file will be:

```
<<example_kable, echo=TRUE, results="asis">>=
library(rattle)
library(dplyr)
set.seed(42)

dsname <- "weatherAUS"
ds      <- tbl_df(get(dsname))
nobs    <- nrow(ds)
obs     <- sample(nobs, 5)
vars    <- 2:7
ds      <- ds[obs, vars]
kable(ds)
@
```

The result (also showing the R code since we specified `echo=TRUE`) is then:

```
library(rattle)
library(dplyr)
set.seed(42)

dsname <- "weatherAUS"
ds      <- tbl_df(get(dsname))
nobs    <- nrow(ds)
obs     <- sample(nobs, 5)
vars    <- 2:7
ds      <- ds[obs, vars]
kable(ds)
```

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
84120	Hobart	15.4	21.3	3.2	7.6	3.2
86166	Launceston	5.3	19.9	0.0	NA	NA
26311	Williamstown	13.6	18.9	NA	NA	NA
76360	PerthAirport	11.5	29.5	0.0	4.8	10.0
59008	GoldCoast	14.3	27.9	0.0	NA	NA

6.1 Formatting Options

There are a few formatting options available for fine tuning how the table is to be presented.

We can remove the row names easily with `row.names=FALSE`:

```
kable(ds, row.names=FALSE)
```

Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
Hobart	15.4	21.3	3.2	7.6	3.2
Launceston	5.3	19.9	0.0	NA	NA
Williamstown	13.6	18.9	NA	NA	NA
PerthAirport	11.5	29.5	0.0	4.8	10.0
GoldCoast	14.3	27.9	0.0	NA	NA

We can limit the digits displayed to avoid an impression of a high level of accuracy or to simplify presentation using `digits=`. By doing so the numeric values are rounded to the supplied number of decimal points.

```
kable(ds, row.names=FALSE, digits=0)
```

Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
Hobart	15	21	3	8	3
Launceston	5	20	0	NA	NA
Williamstown	14	19	NA	NA	NA
PerthAirport	12	30	0	5	10
GoldCoast	14	28	0	NA	NA

6.2 Improved Look Using BookTabs

The `booktabs` package for \LaTeX provides additional functionality that we can make use of with `kable()`. To use this be sure to include the following in the preamble of your `.Rnw` file:

```
\usepackage{booktabs}
```

We can then set `booktabs=TRUE`.

```
kable(ds, row.names=FALSE, digits=0, booktabs=TRUE)
```

Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
Hobart	15	21	3	8	3
Launceston	5	20	0	NA	NA
Williamstown	14	19	NA	NA	NA
PerthAirport	12	30	0	5	10
GoldCoast	14	28	0	NA	NA

Notice also that with more rows, `booktabs=TRUE` will add a small gap every 5 rows.

```
dst <- weatherAUS[sample(nobs, 20), vars]
kable(dst, row.names=FALSE, digits=0, booktabs=TRUE)
```

Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
Portland	15	19	0	4	0
Woomera	17	33	0	70	12
NorahHead	18	28	0	NA	NA
Townsville	15	27	0	6	10
MountGambier	5	14	2	2	10
MelbourneAirport	6	14	2	1	5
Nuriootpa	12	30	0	4	7
Launceston	3	11	0	NA	NA
WaggaWagga	11	29	0	15	10
MelbourneAirport	9	13	0	2	4
Launceston	8	24	0	NA	NA
Darwin	20	33	0	5	11
Newcastle	15	24	0	NA	NA
Melbourne	13	32	0	5	12
Dartmoor	NA	NA	NA	5	12
Hobart	11	22	0	4	11
NorahHead	16	21	0	NA	NA
Darwin	26	34	0	7	9
AliceSprings	17	35	0	11	10
CoffsHarbour	20	24	7	3	NA

Other fine tuning is not readily available through `kable()`. Instead of reinventing the wheel, more sophisticated formatting is available through `xtable`.

7 Including a Table using XTable

Whilst `kable()` provides basic functionality, much more extensive control over the formatting of tables is provided by `xtable` (Dahl, 2013).

```
library(xtable)
xtable(dst)
```

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
47733	Portland	15.10	18.90	0.00	3.80	0.00
67731	Woomera	17.00	33.10	0.00	70.00	11.70
12383	NorahHead	18.00	28.00	0.00		
60411	Townsville	14.90	26.60	0.00	6.40	10.40
64831	MountGambier	5.20	14.30	1.80	1.60	9.70
42089	MelbourneAirport	6.50	13.90	2.40	0.80	5.30
66121	Nuriootpa	12.40	30.00	0.00	3.60	7.30
85940	Launceston	3.30	11.40	0.00		
23486	WaggaWagga	11.00	28.80	0.00	15.00	9.90
42506	MelbourneAirport	9.40	13.40	0.00	2.20	4.10
86428	Launceston	7.80	24.00	0.00		
89941	Darwin	19.50	32.60	0.00	5.00	10.90
10802	Newcastle	14.90	24.00	0.00		
43672	Melbourne	13.00	32.20	0.00	5.40	12.00
51517	Dartmoor				4.80	11.70
83115	Hobart	11.10	22.50	0.00	3.80	11.00
12753	NorahHead	16.10	21.10	0.40		
90915	Darwin	25.50	34.20	0.00	7.20	8.80
87032	AliceSprings	17.00	34.70	0.00	10.60	10.30
7579	CoffsHarbour	19.70	24.40	6.80	2.80	

There are very many formatting options available for fine tuning how the table is to be presented and we cover some of these in the following pages. We also note that some options are provisioned by `xtable()` whilst others are available through `print.xtable()`. For example, `include.rownames=` is an option with `print.xtable()`:

```
print(xtable(ds), include.rownames=FALSE)
```

Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
Hobart	15.40	21.30	3.20	7.60	3.20
Launceston	5.30	19.90	0.00		
Williamtown	13.60	18.90			
PerthAirport	11.50	29.50	0.00	4.80	10.00
GoldCoast	14.30	27.90	0.00		

7.1 Formatting Numbers with XTable

We can limit the number of digits displayed to avoid giving an impression of a high level of accuracy, or to simplify the presentation. Note that the numbers are rounded.

```
print(xtable(ds, digits=1), include.rownames=FALSE)
```

Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
Hobart	15.4	21.3	3.2	7.6	3.2
Launceston	5.3	19.9	0.0		
Williamtown	13.6	18.9			
PerthAirport	11.5	29.5	0.0	4.8	10.0
GoldCoast	14.3	27.9	0.0		

When we have quite large numbers where digits play no role, we can remove them completely. For this next table we invent some large numbers to make the point.

```
dst <- ds
dst[-1] <- sample(10000:99999, nrow(dst)) * dst[-1]
print(xtable(dst, digits=0), include.rownames=FALSE)
```

Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
Hobart	866697	1198743	180093	427720	180093
Launceston	239120	897828	0		
Williamtown	1244590	1729615			
PerthAirport	577588	1481638	0	241080	502250
GoldCoast	1218889	2378112	0		

Take note of how difficult it is to distinguish between the thousands and millions in the table above. We often find ourselves having to count the digits to double check whether 1234566 is 1,234,566 or 123,456. To avoid this cognitive load on the reader we should always use a comma to separate the thousands and millions. This simple principle makes it so much easier for the reader to appreciate the scale, and to avoid misreading data, yet it is so often overlooked.

```
print(xtable(dst, digits=0),
      include.rownames=FALSE,
      format.args=list(big.mark=","))
```

Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
Hobart	866,697	1,198,743	180,093	427,720	180,093
Launceston	239,120	897,828	0		
Williamtown	1,244,590	1,729,615			
PerthAirport	577,588	1,481,638	0	241,080	502,250
GoldCoast	1,218,889	2,378,112	0		

7.2 Adding a Caption and Reference Label

```
print(xtable(ds,
             digits=0,
             caption="Selected observations from \\textbf{weatherAUS}.",
             include.rownames=FALSE)
```

Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
Hobart	15	21	3	8	3
Launceston	5	20	0		
Williamstown	14	19			
PerthAirport	12	30	0	5	10
GoldCoast	14	28	0		

Table 1: Selected observations from **weatherAUS**.

Notice that we wanted to emphasise the name of the dataset in the caption. We can make it bold using the `\textbf{}` command of \LaTeX within the string passed to `caption=`. We need to repeat the backslash because R itself will attempt to interpret it otherwise. That is, we *escape* the backslash.

As well as adding a caption with a table number, we can add a label to the `xtable()` command and then refer to the table within the text using the `\ref{MyTable}` \LaTeX command. Thus, in the source document we use “Table~`\ref{MyTable}`” to produce “Table 2” in the document.

```
print(xtable(ds,
             digits=0,
             caption="Selected observations from \\textbf{weatherAUS}.",
             label="MyTable"),
      include.rownames=FALSE)
```

Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
Hobart	15	21	3	8	3
Launceston	5	20	0		
Williamstown	14	19			
PerthAirport	12	30	0	5	10
GoldCoast	14	28	0		

Table 2: Selected observations from **weatherAUS**.

The references to Table 2 can appear anywhere in the document. We can even refer to its page number by replacing `\ref{MyTable}` with `\pageref{MyTable}` to get Table 2 on Page 11.

7.3 Adding Special Characters to a Caption

Here we illustrate that the string passed to `caption=` can be generated by R. Here we use `paste()` and `Sys.time()` and include some special symbols known to \LaTeX . The result is shown in Table 3 on page 12.

```
print(xtable(ds,
  digits=0,
  caption=paste("Here we include in the caption a sample of \\LaTeX{",
    "symbols that can be included in the string, and note that the",
    "caption string can be the result of R commands, using paste()",
    "in this instance. Some sample symbols include:",
    "$\\alpha$ $\\longrightarrow$ $\\wp$."),
  Sys.time()),
  label="SymbolCaption"),
include.rownames=FALSE)
```

Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
Hobart	15	21	3	8	3
Launceston	5	20	0		
Williamtown	14	19			
PerthAirport	12	30	0	5	10
GoldCoast	14	28	0		

Table 3: Here we include in the caption a sample of \LaTeX symbols that can be included in the string, and note that the caption string can be the result of R commands, using `paste()` in this instance. Some sample symbols include: $\alpha \longrightarrow \wp$. We also get a timestamp from R: 2014-08-17 11:54:07

Exercise: Explore options for formatting the contents of columns and aligning columns differently.

8 Including Figures

Including figures generated by R in our document is similarly easy. We simply include in a chunk the R code to generate the figure. Here for example is some R code to generate a simple density plot of the 3pm temperature in 4 cities over a year. We use `ggplot2` (Wickham and Chang, 2014) to generate the figure.

```
library(rattle) # For the weatherAUS dataset.
library(ggplot2) # To generate a density plot.
cities <- c("Canberra", "Darwin", "Melbourne", "Sydney")
ds <- subset(weatherAUS, Location %in% cities & ! is.na(Temp3pm))
p <- ggplot(ds, aes(Temp3pm, colour=Location, fill=Location))
p <- p + geom_density(alpha=0.55)
p
```

In the source document for this page, the above R code is actually inserted between the chunk begin and end marks:

```
<<myfigure, eval=FALSE>>=
library(rattle) # For the weatherAUS dataset.
library(ggplot2) # To generate a density plot.
cities <- c("Canberra", "Darwin", "Melbourne", "Sydney")
ds <- subset(weatherAUS, Location %in% cities & ! is.na(Temp3pm))
p <- ggplot(ds, aes(Temp3pm, colour=Location, fill=Location))
p <- p + geom_density(alpha=0.55)
p
@
```

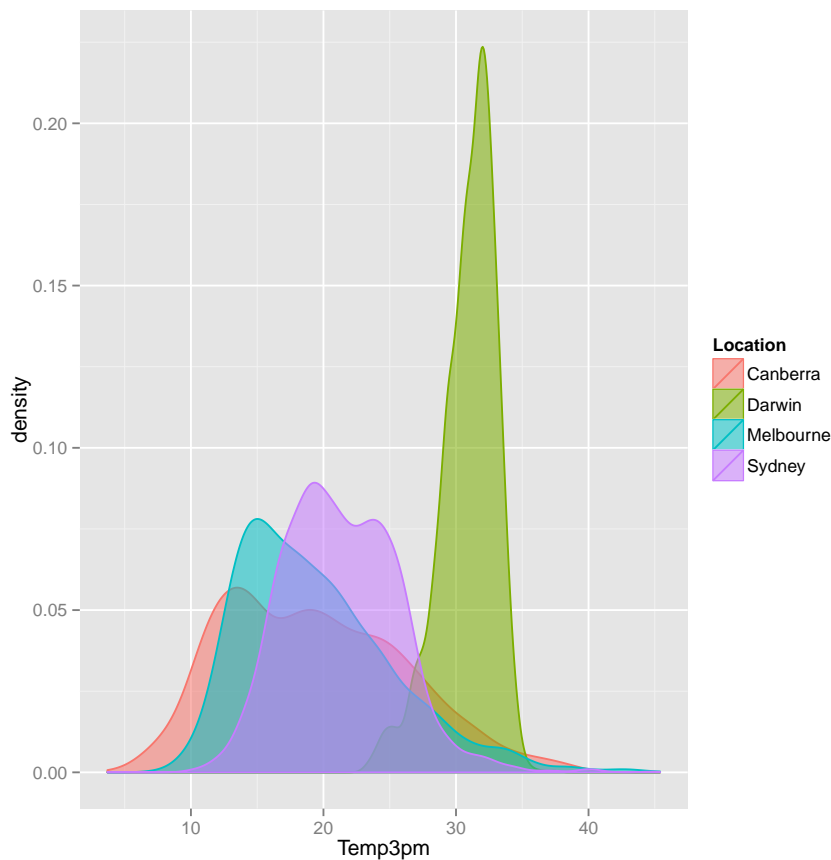
Notice the use of `eval=FALSE`, which allows the R code to be included in the final document, as it is here, but will not generate the plot to be included in the figure, yet. We leave that for the next page.

The R code here begins by loading the requisite packages, loading `rattle` (Williams, 2014) to access the `weatherAUS` dataset, and `ggplot2` (Wickham and Chang, 2014) for the function to generate the actual plot.

The four cities we wish to plot are then identified, and we generate a `subset()` of the `weatherAUS` dataset containing just those cities. We pass the subset on to `ggplot` and identify `Temp3pm` for the x-axis, using `location` to colour and fill the plot.

We then add a layer to the figure containing a density plot with a level of transparency specified as an `alpha=` value. We can see the effect on the following page.

8.1 Displaying the Figure



We include the figure in the final document as above simply by removing the `eval=FALSE` from the previous code chunk. Thus the R code is evaluated and a plot is generated. We have actually, effectively, replaced the `eval=FALSE` with `echo=FALSE` so as not to replicate the R code again.

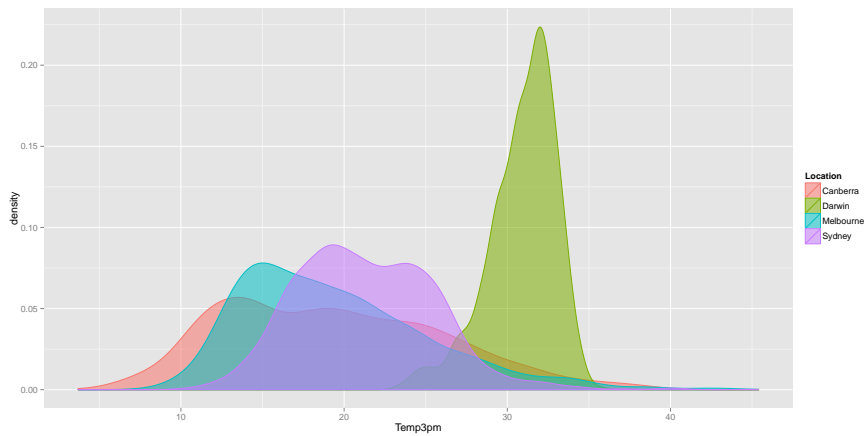
In fact, we do not actually need to rewrite the R code again in a second chunk, given the code has already been provided in the first chunk on the previous page. We use a feature of `knitr` where an empty chunk having the same name as a previous chunk is actually a reference to that previous chunk:

```
<<myfigure, echo=FALSE>>=  
@
```

This is exactly what we included at the beginning of this section in the actual source document for this page. Noticing that we have replaced `eval=FALSE` with `echo=FALSE`, we cause the original R code to be executed, generating the plot which is included as the figure above. Using `echo=FALSE` simply ensures we do not include the R code itself in the final output, this time. That is, the R code is replaced with the figure it generates.

Notice how the figure takes up quite a bit of space on the page.

8.2 Adjusting Aspect



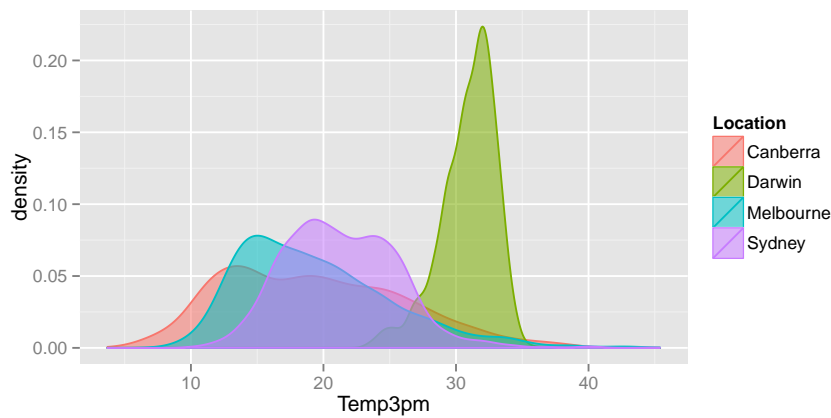
We can fine tune the size of the figure to suit the document and presentation. In this example we have asked R to widen the figure from 7 inches to 14 inches using `fig.width`. The code chunk is:

```
<<myfigure_fig_width, echo=FALSE, fig.width=14>>=  
p  
@
```

Underneath, knitr is using a PDF device on which the plot is generated, and then saved to file for inclusion in the final document. The PDF device `pdf()`, by default, will generate a 7 inch by 7 inch plot (see `?pdf` for details). This is the plot dimensions as we see on the previous page. By setting `fig.width=` (and `fig.height=`) we can change the dimensions. In our example here we have doubled the width, resulting in a more pleasing plot.

Notice that as a consequence of the figure being larger the fonts have remained the same size, resulting them appearing smaller now when we include the figure in the same area on the printed page.

8.3 Choosing Dimensions



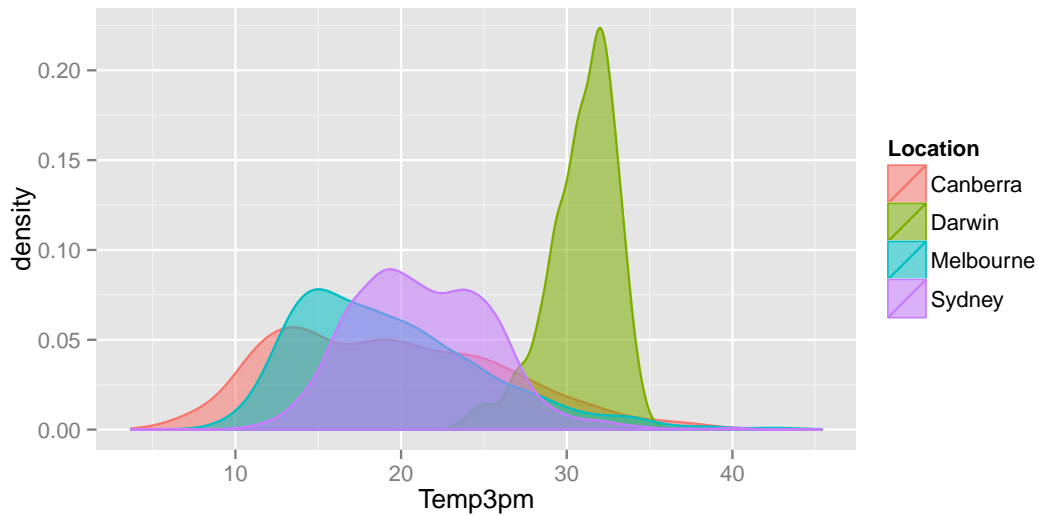
Often a bit of trial and error is required to get the dimensions right. Notice though that increasing the `fig.width=` as we did for the previous plot, and/or increasing the `fig.height=`, effectively also reduces the font size. Actually, the font size remains constant whilst the figure grows (or shrinks) in size. Sometimes it is better to reduce the `fig.width` or `fig.height` to retain a good sized font.

The above plot was generated with the following knitr options.

```
<<myfigure, echo=FALSE, fig.height=3.5>>=
```

```
p  
@
```

8.4 Setting Output Width

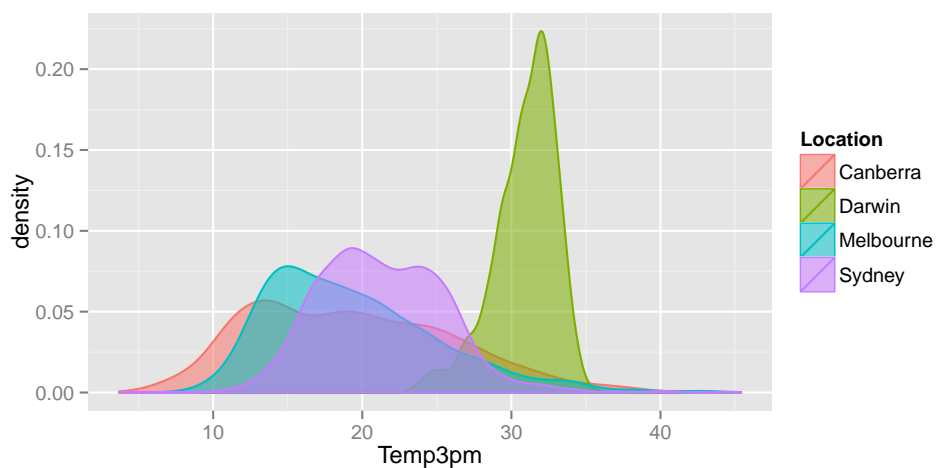


We can use the `out.width=` and `out.height=` to adjust how much space a figure takes up. The above figure is enlarged to fill the `textwidth` of the document using:

```
<<myfigure, fig.align="center", fig.width=14, out.width="\\textwidth">>=  
p  
@
```

If that is too wide, we can reduce it to 90% of the page width with:

```
<<myfigure, fig.align="center", fig.width=14, out.width="0.9\\textwidth">>=  
p  
@
```



9 Add a Caption and Label

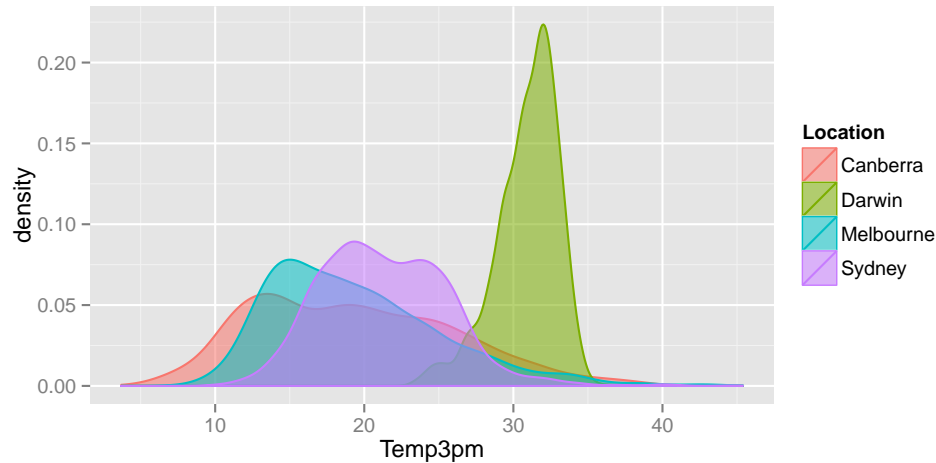


Figure 1: The 3pm temperature for four locations.

Adding a caption (which automatically also adds a label) is done using `fig.cap=`.

```
<<myfigure, fig.cap="The 3pm temperature for four locations.", fig.pos="h",...>=  
p  
@
```

We have also used `fig.pos="h"` which requests placement of the figure “here” rather than letting it float. Other options are to place the figure at the top of a page (“`t`”), or the bottom of a page (“`b`”). We can leave it empty and the placement is done automatically—that is, the figure floats to an appropriate location.

Once a caption is added, a label is also added to the figure so that it can be referred to in the document. The label is made up of `fig:` followed by the chunk label, which is `myfigure` in this example. So we can refer to the figure using `\ref{fig:myfigure}` and `\pageref{fig:myfigure}`, which allows us to refer to Figure 1 on Page 18.

10 Animation: Basic Example

We can generate multiple plots and they then form an animation. For this to work, we use the knitr option `fig.show="animate"`. For the L^AT_EX processing we then need to load the `animate` package:

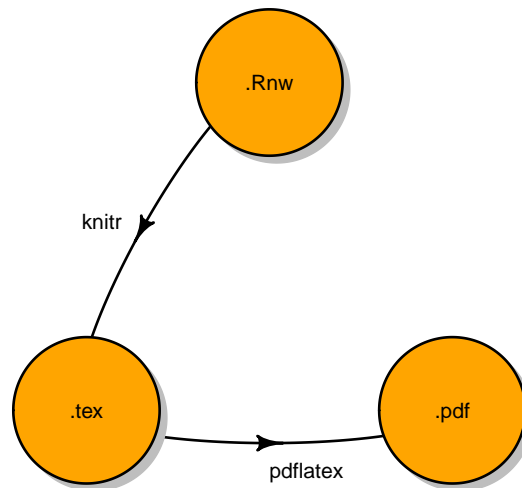
```
\usepackage{animate}
```

To view the animation live we need to use Acrobat Reader to view the PDF file.

This example comes from <http://taiyun.github.io/blog/2012/07/k-means/>.

```
library(animation)
par(mar=c(3, 3, 1, 1.5), mgp=c(1.5, 0.5, 0), bg="white")
cent <- 1.5 * c(1, 1, -1, -1, 1, -1, 1, -1)
x <- NULL
for (i in 1:8) x <- c(x, rnorm(25, mean=cent[i]))
x <- matrix(x, ncol=2)
colnames(x) <- c("X1", "X2")
kmeans.ani(x, centers=4, pch=1:4, col=1:4)
```

11 Adding a Flowchart



Here we use the `diagram` (Soetaert, 2014) package to draw a flow chart, something that is a common requirement in documentation. The simple example here shows the process of converting a `.Rnw` file, using `knitr`, to a `LaTeX` file, which can then be processed by `pdflatex` to generate the `.pdf` file.

```
library(diagram)

names <- c(".Rnw", ".tex", ".pdf")

connect <- c(0,      0,      0,
             "knitr", 0,      0,
             0,      "pdflatex", 0)

M <- matrix(nrow=3, ncol=3, byrow=TRUE, data=connect)

pp <- plotmat(M, pos=c(1, 2), name=names, box.col="orange")
```

There are many more possibilities provided by `diagram`. See `demo(plotmat)` and `demo(plotweb)`.

12 Adding Bibliographies

knitr supports the automatic generation of a bibliography for the packages loaded into R. We can take advantage of this by specifying a L^AT_EX bibliography package like `natbib`. I prefer the (author, year) style and so I include the following in the preamble of my L^AT_EX documents.

```
\usepackage[authoryear]{natbib}
```

If we load the `rattle` package, we might like to cite it with the following L^AT_EX command:

```
\citep{R-rattle}
```

This will produce a citation like (Williams, 2014). The “p” in `\citep` places the parentheses around the citation, without which we get Williams (2014).

Then we can ask knitr to generate bibliographic entries for each loaded package:

```
write_bib(sub("^.*/", "", grep("^/", searchpaths(), value=TRUE)),  
          file="mydoc.bib")
```

Note that the bibliography is saved to a file named `mydoc.bib`.

At the end of the L^AT_EX document, where we want the bibliography to appear, we add the following:

```
\bibliographystyle{jss}  
\bibliography{mydoc}
```

See Page 28 to see what the bibliography will look like.

13 Referencing Chunks in L^AT_EX

We may like to reference code chunks from other parts of our document. L^AT_EX handles cross references nicely, and there are packages that extend this to user defined cross references. The `amsthm` package provides one such mechanism.

To add a cross referencing capability we can tell L^AT_EX to use the `amsthm` package and to create a new theorem environment called `rcode` by including the following in the document preamble (prior to the `\begin{document}`):

```
\usepackage{amsthm}
\newtheorem{rcode}{R Code}[section]
```

We then tell `knitr` to add an `rcode` environment around the chunks. The following chunk achieves this by adding some code to a hook function for `knitr`. This will typically also be in the preamble of the document, though not necessarily.

```
<<setup, include=FALSE>>=
knit_hooks$set(rcode=function(before, options, envir)
{
  if (before)
    sprintf('\begin{rcode}\label{%s}\hfill}', options$label)
  else
    '\end{rcode}'
})
@
```

Once we've done that, we add `rcode=TRUE` for those chunks we wish to refer to. Here is a chunk as an example:

```
<<demo_chunk_ref, rcode=TRUE>>=
seq(0, 10, 2)
@
```

The output from the chunk is:

```
R Code 1.
seq(0, 10, 2)
## [1] 0 2 4 6 8 10
```

We can then refer to the R code chunk with `\ref{demo_chunk_ref}`, which prints the R Code reference number as [1](#), and `\pageref{demo_chunk_ref}`, which prints the page number as [22](#).

14 Truncating Long Lines

The output from knitr will sometimes be longer than fits within the limits of the page. We can add a hook to knitr so that whenever a line is longer than some parameter, it is truncated and replaced with "...". The hook extends the `output` function. Notice we take a copy of the current `output` hook and then run that after our own processing.

```
opts_chunk$set(out.truncate=80)
hook_output <- knitr_hooks$get("output")
knitr_hooks$set(output=function(x, options)
{
  if (options$results != "asis")
  {
    # Split string into separate lines.
    x <- unlist(stringr::str_split(x, "\n"))
    # Truncate each line to length specified.
    if (!is.null(m <- options$out.truncate))
    {
      len <- nchar(x)
      x[len>m] <- paste0(substr(x[len>m], 0, m-3), "...")
    }
    # Paste lines back together.
    x <- paste(x, collapse="\n")
    # Continue with any other output hooks
  }
  hook_output(x, options)
})
```

This is useful to avoid ugly looking long lines that extend beyond the limits of the page. We can illustrate it here by first not truncating at all (`out.truncate=NULL`):

```
paste("This is a very long line that is truncated",
      "at character 80 by default. We change the point",
      "at which it gets truncated using out.truncate=")
## [1] "This is a very long line that is truncated at character 80 by default. We change the point"
```

Now we use the default to truncate it.

```
paste("This is a very long line that is truncated",
      "at character 80 by default. We change the point",
      "at which it gets truncated using out.truncate=")
## [1] "This is a very long line that is truncated at character 80 by default..."
```

Here is another example, with `out.truncate=40` included in the knitr options.

```
paste("This is a very long line that is truncated",
      "at character 80 by default. We change the point",
      "at which it gets truncated using out.truncate=")
## [1] "This is a very long line that..."
```


15 Truncating Too Many Lines

Another issue we sometimes want to deal with is limiting the number of lines of output displayed from knitr. We can add a hook to knitr so that whenever we have reached a particular line count for the output of any command, we replace all the remaining lines with "...". The hook again extends the output function. Notice we take a copy of the current output hook and then run that after our own processing.

```
opts_chunk$set(out.lines=4)
hook_output <- knitr_hooks$get("output")
knitr_hooks$set(output=function(x, options)
{
  if (options$results != "asis")
  {
    # Split string into separate lines.
    x <- unlist(stringr::str_split(x, "\n"))
    # Trim to the number of lines specified.
    if (!is.null(n <- options$out.lines))
    {
      if (length(x) > n)
      {
        # Truncate the output.
        x <- c(head(x, n), "...\\n")
      }
    }
    # Paste lines back together.
    x <- paste(x, collapse="\\n")
  }
  hook_output(x, options)
})
```

We can then illustrate it:

```
weather[2:8]

##      Location MinTemp MaxTemp Rainfall Evaporation Sunshine WindGustDir
## 1   Canberra     8.0    24.3     0.0         3.4         6.3         NW
## 2   Canberra    14.0    26.9     3.6         4.4         9.7         ENE
## 3   Canberra    13.7    23.4     3.6         5.8         3.3         NW
....
```

Now we set `out.lines=2` to only include the first two lines.

```
weather[2:8]

##      Location MinTemp MaxTemp Rainfall Evaporation Sunshine WindGustDir
## 1   Canberra     8.0    24.3     0.0         3.4         6.3         NW
....
```

16 Selective Lines of Code

Another useful formatting trick is to include only the top few and bottom few lines of a block of code. We can again do this using hooks. This time it is through a source hook.

```
opts_chunk$set(src.top=NULL)
opts_chunk$set(src.bot=NULL)
knit_hooks$set(source=function(x, options)
{
  # Split string into separate lines.
  x <- unlist(stringr::str_split(x, "\n"))
  # Trim to the number of lines specified.
  if (!is.null(n <- options$src.top))
  {
    if (length(x) > n)
    {
      # Truncate the output.
      if (is.null(m <- options$src.bot)) m <- 0
      x <- c(head(x, n+1), "\n...\n", tail(x, m+2))
    }
  }
  # Paste lines back together.
  x <- paste(x, collapse="\n")
  hook_source(x, options)
})
```

Now we repeat this code chunk in the source of this current document, but we set `src.top=4` and `src.bot=4`:

```
opts_chunk$set(src.top=NULL)
opts_chunk$set(src.bot=NULL)
knit_hooks$set(source=function(x, options)
{
  # Split string into separate lines.
  ....
  }
}
# Paste lines back together.
x <- paste(x, collapse="\n")
hook_source(x, options)
})
```

17 Knitr Options

Below is a list of common knitr options. We can set the options using `opts_chunk$set()`. The arguments to the function can be any number of named options with their values. For example:

```
opts_chunk$set(size="footnotesize", message=FALSE, tidy=FALSE)
```

Once this is run, the options remain in force as the default values until they are again changed using `opts_chunk$set()`. They can be overridden per chunk in the usual way.

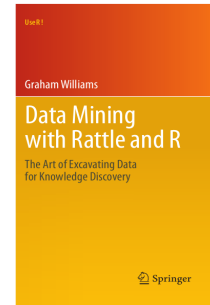
```
background="#F7F7F7"      # The background colour of the code chunks.
cache.path="cache/"      #
comment=NA               # Suppresses "\verb|##|" in R output.
echo=FALSE               # Do not show R commands---just the output.
echo=3:5                 # Only echo lines 3 to 5 of the chunk.
eval=FALSE               # Do not run the R code---its just for display.
eval=2:4                 # Only evaluate lines 2 to 4 of the chunk.
fig.align="center"       #
fig.cap="Caption..."   #
fig.keep="high"          #
fig.lp="fig:"            # Prefix for the label assigned to the figure.
fig.path="figures/plot"  #
fig.scap="Short cap."    # For the table of figures in the contents.
fig.show="animate"       # Collect figures into an animation.
fig.show="hold"          #
fig.height=9             # Height of generated figure.
fig.width=12             # Width of generated figure.
include=FALSE            # Include code but not output/picture.
message=FALSE            # Do not display messages from the commands.
out.height=".6\\textheight" # Figure takes up 60% of the page height.
out.width=".8\\textwidth"  # Figure takes up 80% of the page width.
results="markup"         # The output from commands will be formatted.
results="hide"           # Do not show output from commands.
results="asis"           # Retain R command output as \LaTeX{} code.
size="footnotesize"     # Useful for Beamer slides.
tidy=FALSE               # Retain my own formatting used in the R code.
```

New options defined in this Chapter and used in this book:

```
out.lines=4              # Number of lines of \R{} output to show.
out.truncate=80          # Truncate \R{} output lines beyond this.
src.bot=NULL             # Number of lines of output at top to show.
src.top=NULL             # Number of lines of output at bottom to show.
```

18 Further Reading and Acknowledgements

The [Rattle Book](#), published by Springer, provides a comprehensive introduction to data mining and analytics using Rattle and R. It is available from [Amazon](#). Other documentation on a broader selection of R topics of relevance to the data scientist is freely available from <http://datamining.togaware.com>, including the [Datamining Desktop Survival Guide](#).



This chapter is one of many chapters available from <http://HandsOnDataScience.com>. In particular follow the links on the website with a * which indicates the generally more developed chapters.

Other resources include:

- The home of knitr is <http://yihui.name/knitr/>. Here we can find some documentation on knitr and some discussion forums.
- The home of RStudio is <http://www.rstudio.org/>. The RStudio software can be downloaded from here, as well as finding documentation and discussion forums.
- The knitr author's presentation on knitr <http://yihui.name/slides/2012-knitr-RStudio.html>. The presentation provides a basic introduction to knitr.
- <http://bcb.dfci.harvard.edu/~aedin/courses/ReproducibleResearch/ReproducibleResearch.pdf> This lecture provides some insights into literate programming in R.
- \LaTeX supports many characters and symbols but finding the right incantation is difficult. This web site, <http://detexify.kirelabs.org/classify.html>, does a great job. Draw the character you want and it will show you how to get it ☺.

19 References

- Dahl DB (2013). *xtable: Export tables to LaTeX or HTML*. R package version 1.7-1, URL <http://CRAN.R-project.org/package=xtable>.
- Harrell Jr FE (2014). *Hmisc: Harrell Miscellaneous*. R package version 3.14-4, URL <http://CRAN.R-project.org/package=Hmisc>.
- R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <http://www.R-project.org/>.
- Soetaert K (2014). *diagram: Functions for visualising simple graphs (networks), plotting flow diagrams*. R package version 1.6.2, URL <http://CRAN.R-project.org/package=diagram>.
- Wickham H, Chang W (2014). *ggplot2: An implementation of the Grammar of Graphics*. R package version 1.0.0, URL <http://CRAN.R-project.org/package=ggplot2>.
- Wickham H, Francois R (2014). *dplyr: dplyr: a grammar of data manipulation*. R package version 0.2, URL <http://CRAN.R-project.org/package=dplyr>.
- Williams GJ (2009). “Rattle: A Data Mining GUI for R.” *The R Journal*, **1**(2), 45–55. URL http://journal.r-project.org/archive/2009-2/RJournal_2009-2_Williams.pdf.
- Williams GJ (2011). *Data Mining with Rattle and R: The art of excavating data for knowledge discovery*. Use R! Springer, New York. URL http://www.amazon.com/gp/product/1441998896/ref=as_li_qf_sp_asin_tl?ie=UTF8&tag=togaware-20&linkCode=as2&camp=217145&creative=399373&creativeASIN=1441998896.
- Williams GJ (2014). *rattle: Graphical user interface for data mining in R*. R package version 3.1.4, URL <http://rattle.togaware.com/>.
- Xie Y (2014). *knitr: A general-purpose package for dynamic report generation in R*. R package version 1.6, URL <http://CRAN.R-project.org/package=knitr>.

This document, sourced from KnitRO.Rnw revision 498, was processed by KnitR version 1.6 of 2014-05-24 and took 4.9 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04.1 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-08-17 11:54:11.