Data Science with R
Documenting with KnitR

Graham.Williams@togaware.com

15th February 2013

A high level of transparency and efficiency is required for the work we perform as data miners. All of our activity should be recorded for quality assurance, efficiency, and general sharing of ideas. We will find ourselves repeating the work we do, including preparing and transforming our data in different ways, so we want to streamline that process. We also need to document what we have done so that we can come back to it at a later time and pick up from where we left off, or even explain and justify what we did. The concept of intermixing a narrative and the code of our analyses comes from the idea of literate programming

A particularly useful tool to support this is knitr (Xie, 2012), which combines the typesetting power of LaTeX with the statistical power of R (R Core Team, 2012). Within a single document (or multiple documents if appropriate), ideally maintained under version control, we interact directly with R to load our data, explore and analyse the data, and then present the results. knitr is well supported within RStudio, within the ESS-mode in Emacs, and with LyX, and is highly recommended as the mechanism to perform and record all activities of a project.

The OnePageR documents themselves are all developed using knitr .

The required packages for this module include:

```
library(rattle)  # Weather dataset
library(ggplot2) # Beautiful plots
library(xtable)  # LaTeX tables
library(Hmisc)   # LaTeX string preparation
```

As we work through this module, 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 the *help=* option of `library()`:

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

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

1 Data Mining Report Template

To get started with knitr , we create a new text document with a filename extension of .Rnw (perhaps 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.

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 properly 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 on the left, and then choose knitr as the option to use to Weave Rnw files.

Give it a try. The document below is the result of doing this using exactly the template we see next.

Data Mining Project Report Template

Graham Williams

15th February 2013

1 Introduction

A paragraph or two introducing the project.

2 Business Problem

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

3 Data Sources

Data is available from many sources and we need to identify these 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 tried and built. Include R code. Include model evaluations.

7 Deployment

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

2 Sample Template

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

\title{Data Mining 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 owner) and record
decisions made and shared understanding of the business problem.

\section{Data Sources}

Data is available from many sources and we need to identify these 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 tried and built. Include R code. Include model
evaluations.

\section{Deployment}

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

\end{document}
```

3 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 a special marker. The R code is enclosed between a line beginning with double less than symbols (or angle brackets <<), starting in column one. The 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 show 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, results=hide, echo=FALSE>>
```

The label here is *example_label*, and we ask knitr to hide the results from this code chunk and to not echo the commands. 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.

An actual example is to generate some data and to the 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] 659.321 477.024 954.585   2.782 308.794 564.671
```

```
mean(x)
```

```
## [1] 512
```

4 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 use `\Sexpr{Sys.Date()}` to get "2013-02-15". Any functions can be called upon in this way. If we wanted a specifically formatted date, I can use `format()` as in `\Sexpr{format(Sys.Date(), format="%A, %e %B % Y")}` to produce Friday, 15 February 2013.

We can also talk about the characteristics of the dataset. For example, the **weather** dataset has 366 (`\Sexpr{nrow(weather)}`) observations of the variables: MinTemp, MaxTemp, Rainfall, Evaporation, Sunshine (`\Sexpr{paste(names(weather)[3:7], 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, the LaTeX will get confused. An example is listing one of the variables in 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 (Jr *et al.*, 2012) package provides `latexTranslate()` to assist here. I used it 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.

Experiment with these in your own template knitr document.

5 Including a Table

Including a properly typeset table is quite simple using the xtable (Dahl, 2012) package. We use the **weather** dataset from rattle (Williams, 2013).

```
<<example_table, echo=FALSE, results="asis">>=
library(xtable)
library(rattle)
set.seed(42)
obs <- sample(1:nrow(weatherAUS), 20)
vars <- 2:7
xtable(weatherAUS[obs, vars])
@
```

The result (also showing the R code) is then:

```
library(xtable)
library(rattle)
set.seed(42)
obs <- sample(1:nrow(weatherAUS), 5)
vars <- 2:7
xtable(weatherAUS[obs, vars])
```

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
60992	Hobart	5.60	13.00	7.60	1.60	3.10
62476	Launceston	7.40	13.50	8.80		
19077	Williamtown	18.30	29.10	3.20	1.00	7.00
55366	PerthAirport	9.80	21.90	0.00	3.60	9.80
42784	GoldCoast	23.40	30.40	0.00		

There are very many formatting options available and we cover some of these in the following pages.

6 Table: Formatting Numbers

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

```
ds <- weatherAUS[obs, vars]
xtable(ds, digits=1)
```

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
60992	Hobart	5.6	13.0	7.6	1.6	3.1
62476	Launceston	7.4	13.5	8.8		
19077	Williamtown	18.3	29.1	3.2	1.0	7.0
55366	PerthAirport	9.8	21.9	0.0	3.6	9.8
42784	GoldCoast	23.4	30.4	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.

```
ds[-1] <- sample(10000:99999, nrow(ds)) * ds[-1]
xtable(ds, digits=0)
```

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
60992	Hobart	317621	737334	431057	90749	175826
62476	Launceston	564561	1029942	671370		
19077	Williamtown	404778	643663	70781	22119	154833
55366	PerthAirport	677445	1513881	0	248857	677445
42784	GoldCoast	1718800	2232971	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 makes it so much easier for the reader to appreciate the scale, and to avoid misreading data.

```
print(xtable(ds, digits=0), format.args=list(big.mark=","))
```

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
60992	Hobart	317,621	737,334	431,057	90,749	175,826
62476	Launceston	564,561	1,029,942	671,370		
19077	Williamtown	404,778	643,663	70,781	22,119	154,833
55366	PerthAirport	677,445	1,513,881	0	248,857	677,445
42784	GoldCoast	1,718,800	2,232,971	0		

7 Table: Adding a Caption

```
xtable(weatherAUS[obs, vars], digits=1,
       caption="A sample of variables from the \\textbf{weatherAUS} dataset.")
```

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
60992	Hobart	5.6	13.0	7.6	1.6	3.1
62476	Launceston	7.4	13.5	8.8		
19077	Williamtown	18.3	29.1	3.2	1.0	7.0
55366	PerthAirport	9.8	21.9	0.0	3.6	9.8
42784	GoldCoast	23.4	30.4	0.0		

Table 1: A sample of variables from the **weatherAUS** dataset.

Notice that I wanted to emphasise the name of the dataset I was displaying, within the caption. We can make it bold using the `\textbf{}` command of LaTeX. Within the string that I pass to `xtable()`'s `caption=` option, I need to repeat the backslash because R itself will attempt to interpret it otherwise. That is, we *escape* the backslash.

8 Table: Adding a Reference Label

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 documnet.

```
xtable(weatherAUS[obs, vars], digits=1,
       caption="A sample of variables from the \\textbf{weatherAUS} dataset.",
       label="MyTable")
```

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
60992	Hobart	5.6	13.0	7.6	1.6	3.1
62476	Launceston	7.4	13.5	8.8		
19077	Williamtown	18.3	29.1	3.2	1.0	7.0
55366	PerthAirport	9.8	21.9	0.0	3.6	9.8
42784	GoldCoast	23.4	30.4	0.0		

Table 2: A sample of variables from the **weatherAUS** dataset.

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 8.

9 Table: Adding Special Characters to a Caption

Here we simply illustrate that the string passed to the `caption=` option can be generated by R (e.g., by using `paste()` and `Sys.time()` and can include special symbols known to LaTeX. The result is shown in Table 3 below.

```
xtable(weatherAUS[obs, vars], digits=1,
       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:",
         "$\\alpha$ $\\longrightarrow$ $\\wp$.",
         "We also get a timestamp from R:",
         Sys.time()),
       label="SymbolCaption")
```

	Location	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine
60992	Hobart	5.6	13.0	7.6	1.6	3.1
62476	Launceston	7.4	13.5	8.8		
19077	Williamtown	18.3	29.1	3.2	1.0	7.0
55366	PerthAirport	9.8	21.9	0.0	3.6	9.8
42784	GoldCoast	23.4	30.4	0.0		

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: 2013-02-15 16:50:09

10 Table: Other Format Options

This page is under development.

Other capabilities include:

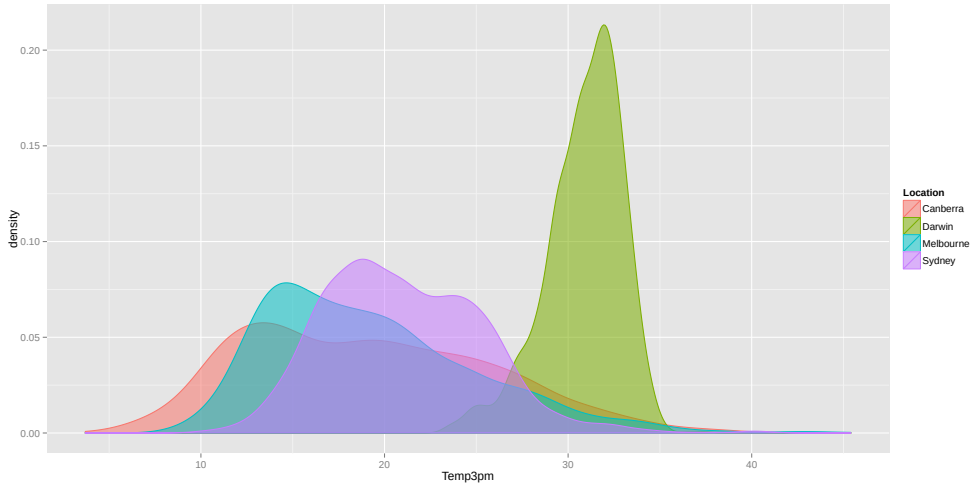
- formatting contents of columns
- aligning columns differently

11 Including a Figure

Including figures generated by R in our document is quite easy. The knitr chunk simply needs to generate a figure and knitr will take care of including it in the document. Here we use ggplot2 (Wickham and Chang, 2012) to generate our plot.

```
<<example_figure, fig.align="center", fig.width=14, out.width="0.9\\textwidth">>=
library(ggplot2)
library(rattle)
cities <- c("Canberra", "Darwin", "Melbourne", "Sydney")
g <- ggplot(subset(weatherAUS, Location %in% cities & ! is.na(Temp3pm)),
            aes(Temp3pm, colour = Location, fill = Location))
g <- g + geom_density(alpha = 0.55)
print(g)
@
```

The result is then a figure that gets included in the document in place of the chuck of R code (turning off the inclusion of the R code itself with `echo=FALSE` included in the knitr options):



For this figure I also use the options fig.align, fig.width, and out.width to tune the figure to fit nicely in the space available on this page. It was a bit of trial and error to get the right dimensions.

12 Figure: Add a Caption and Label

Adding a caption (and then automatically also adding a label) is done using `fig.cap=`.

```
<<myfigure, fig.cap="The 3pm temperature for four locations.", fig.pos="h",...
print(g)
@
```

The result is then:

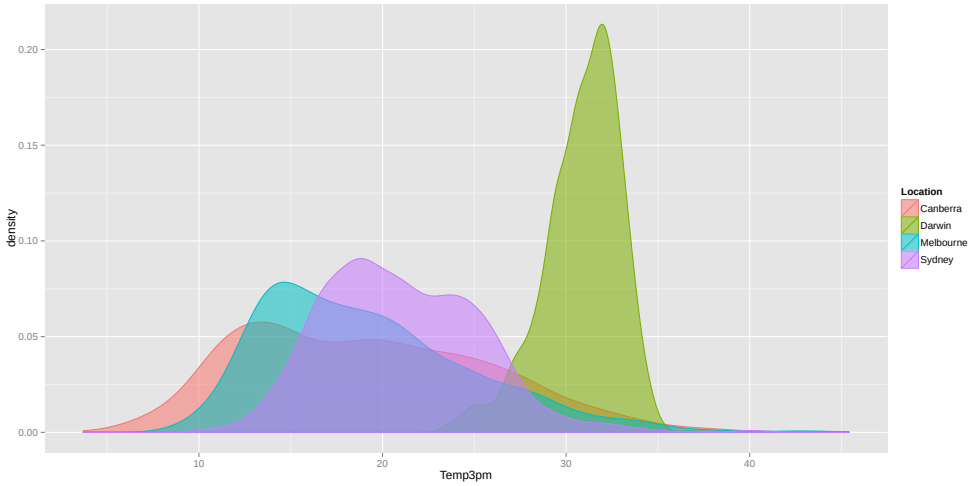


Figure 1: The 3pm temperature for four locations.

Notice the use of the `fig.pos="h"` to aim to place 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 I can refer to the figure using `\ref{fig:myfigure}` and `\pageref{fig:myfigure}`, as in Figure 1 on Page 12.

13 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 LaTeX bibliography package like `natbib`. I prefer the (author, year) style and so I include the following in the preamble of my LaTeX document.

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

If we load the rattle package, we might like to cite it with the following LaTeX command:

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

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.

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

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

14 Advanced Topic — Referencing Chunks in LaTeX

We may like to reference code chunks from other parts of our document. LaTeX 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 LaTeX 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 14.

15 Advanced Topic — 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$set(out.truncate=80)
hook_output <- knit_hooks$get("output")
knit_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 poin
```

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...
```

16 Advanced Topic — 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$set(out.lines=4)
hook_output <- knit_hooks$get("output")
knit_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
....
```

17 Advanced Topic — 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 repaet 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)
{

....

  # Paste lines back together.
  x <- paste(x, collapse="\n")
  hook_source(x, options)
})
```

18 Knitr Options

Here is a list of common knitr options:

`cache.path="cache/"`	
`comment=NA`	Suppresses "`##`" in R output.
`echo=FALSE`	Do not show the R command that is run—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="Figure caption."`	
`fig.keep="high"`	
`fig.lp="fig:"`	
`fig.path="figures/plot"`	
`fig.scap="Short figure caption."`	For the table of figures in the contents.
`fig.show="hold"`	
`include=FALSE`	Include code but not picture which might get included later.
`message=FALSE`	Do not display messages from the commands.
`out.width=".48\\textwidth"`	
`results="hide"`	Do not show output from commands.
`results="asis"`	The output of the R commands is, for example, LaTeX code.
`tidy=FALSE`	Retain my own formatting used in the R code.

knitr provides `opts_chunk$set()` to set the values of the above options. The arguments to the function can be any number of named options with their values. For example:

```
opts\_chunk$set(results="asis", message=FALSE, tidy=FALSE)
```

19 Further Reading

- 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, ans 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 ☺.

20 References

Dahl DB (2012). *xtable: Export tables to LaTeX or HTML*. R package version 1.7-0, URL http://CRAN.R-project.org/package=xtable.

Jr FEH, with contributions from Charles Dupont, many others (2012). *Hmisc: Harrell Miscellaneous*. R package version 3.10-1, URL http://CRAN.R-project.org/package=Hmisc.

R Core Team (2012). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Wickham H, Chang W (2012). *ggplot2: An implementation of the Grammar of Graphics*. R package version 0.9.3, URL http://had.co.nz/ggplot2/.

Williams G (2013). *rattle: Graphical user interface for data mining in R*. R package version 2.6.26, URL http://rattle.togaware.com/.

Xie Y (2012). *knitr: A general-purpose package for dynamic report generation in R*. R package version 0.9, URL http://CRAN.R-project.org/package=knitr.