

One Page R

Graham Williams

Source:

<https://web.archive.org/web/20200609170435/https://togaware.com/onepager/>

Licence:

[Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#)

(CC BY-NC-SA 4.0)

See also:

<https://www.amazon.com/Essentials-Data-Science-Knowledge-Discovery-ebook/dp/B074CDN35H/>

Contents	3
DataScienceO	5
RattleO	22
IntroRO	27
KnitRO	62
BasicRO	91
DataO	110
ReadO	161
CkanO	194
SummaryO	211
VisualiseO	222
TransformO	245
CaseStudiesO	262
WebLogO	287
ModelsO	288
ClustersO	309
ARulesO	366
DTreesO	381
DTreesG	468
EnsemblesO	479
Support Vector Machines	496
Neural Networks	497
BayesO	498
MarsO	505
EvaluateO	514
Scoring	537
PMML	538
StringsO	539
DateTimeO	560
MapsO	589

BigDataO	630
PlotsO	645
FunctionsO	664
ParallelIO	677
EnvironmentsO	694
TextMiningO	703
Social Network Analysis	750
Genetic Programming	751
Time Series Analysis	752
StyleO	753
PackageO	774

Contents

My book, [The Essentials of Data Science: Knowledge Discovery Through R](#), introduces the concept of templates for supporting the data scientist. Below are a variety of templates covering different tasks. The [data.R](#) and [model.R](#) scripts collect together all of the other component scripts.

DATA	CLASSIFICATION	REGRESSION
data.R	model.R	70_model.R
00_setup.R	60_model.R	72_lm.R
10_ingest.R	62_rpart.R	74_rpart
20_observe.R	64_randomForest.R	76_dnn.R
30_prepare.R	66_xgboost.R	
40_meta.R	68_dnn.R	
50_save.R		

Part 1: Data Science

1. Data Mining, Analytics, and Data Science: [Chapter – R – Lecture](#)
2. Rattle to R: [Chapter – R](#)
3. An Introduction to R Programming: [Chapter – R](#)
4. Literate Data Science with KnitR: [Chapter – R – Lecture](#)
5. More Basics of R [Chapter – R](#)

Part 2: Dealing With Data

6. A Template for Preparing Data: [Chapter – R](#)
7. Reading Data into R: *[Chapter – *R](#)
8. Open Access Data via the CKAN API: [Chapter – R](#)
9. Exploring and Summarising Data: *[Chapter – *R](#)
10. Visualising Data with GGPlot2: *[Chapter – *R](#)
11. Transforming Data: *[Chapter – *R](#)
12. Case Study: Analysis of Sea Ports: [Chapter – R](#)
13. Case Study: Web Log Analysis: Chapter – R

Part 3: Building Models

14. A Template for Building Models: [Chapter – R](#)
15. Cluster Analysis: [Chapter – R – Lecture](#)
16. Association Analysis: [Chapter – R – Lecture](#)
17. Decision Trees: *[Lecture – *Chapter – *R – *Rattle](#)
18. Ensembles of Decision Trees: *[Lecture – *Chapter – *R](#)
19. Support Vector Machines
20. Neural Networks
21. Naive Bayes: [Chapter – R](#)
22. Multivariate Adaptive Regression Splines: [Chapter – R](#)
23. Evaluating Models: *[Chapter – *R](#)

24. Scoring (R)
25. PMML (R) Exporting Models for Deployment

Part 4: Advanced R and Analytics

26. Strings: [Chapter](#) – [R](#)
27. Dates and Time: *[Chapter](#) – *[R](#)
28. Spatial Data *[Chapter](#) – *[R](#)
29. Big Data *[Chapter](#) – *[R](#)
30. Exploring Different Plots: [Chapter](#) – [R](#)
31. Writing Functions: [Chapter](#) – [R](#)
32. Parallel Processing: [Chapter](#) – [R](#)
33. Environments: *[Chapter](#) – [R](#)
34. Text Mining: *[Chapter](#) – *[R](#) – Corpus as [tar.gz](#) or [zip](#)
35. Social Network Analysis: Chapter – R
36. Genetic Programming: Chapter – R
37. Time Series Analysis: Chapter – R

Part 5: Appendices

38. Doing R with Style: [Chapter](#) – [R](#)
 39. Packaging (R) Pulling it Together into a Package: [Chapter](#)
-

Other great resources for modular approaches to learning R include:

- [Jared Knowles' R-Bootcamp](#)
-

Other Togaware resources:

- Open Source Machine Learning with R – [FOSSAsia 2017](#) – ([PDF](#))
 - [Introducing Data Mining — Lecture](#)
 - [Ensembles in the ATO – October 2014](#)
 - [International Centre for Free and Open Source Software – May 2015](#) ([PDF](#))
 - [CUNY NSF Workshop – March 2014](#) ([PDF](#))
 - [AusDM-2013 Tutorial – November 2013](#)
 - [IDEAL-2013 Tutorial – October 2013](#)
-

Other resources include:

- [Resources from WhoIsHostingThis](#)

Data Science with R

Getting Started with Rattle

Graham.Williams@togaware.com

21st June 2014

Visit <http://onepager.togaware.com/> for more OnePageR's.

Rattle (Williams, 2014), the R Analytic Tool To Learn Easily, is a graphical data mining application built using the statistical language R (R Core Team, 2014).

Rattle runs under various operating systems, including GNU/Linux, Macintosh OS/X, and MS/Windows. R needs to be installed on your system and then

```
install.packages("rattle")
```

Rattle's user interface steps through the data mining tasks, recording the actual R code as it goes. The R code can be saved to file and used as an automatic script, loaded into R (outside of Rattle) to repeat the data mining exercise. Repeatability is important both in science and in commerce!

This laboratory provides a quick start guide to building our first models using Rattle. Record in a report the tasks you complete, including observations of the data and plots you might generate. This is to be submitted for assessment.

The required packages for this module include:

```
library(rattle)
```

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 Starting Rattle

Rattle is started from R. There are several ways that Rattle might be configured for your particular computer. For example, some installations set up an icon on the desktop from which Rattle is automatically invoked. The most common way though is to start up Rattle from within R.

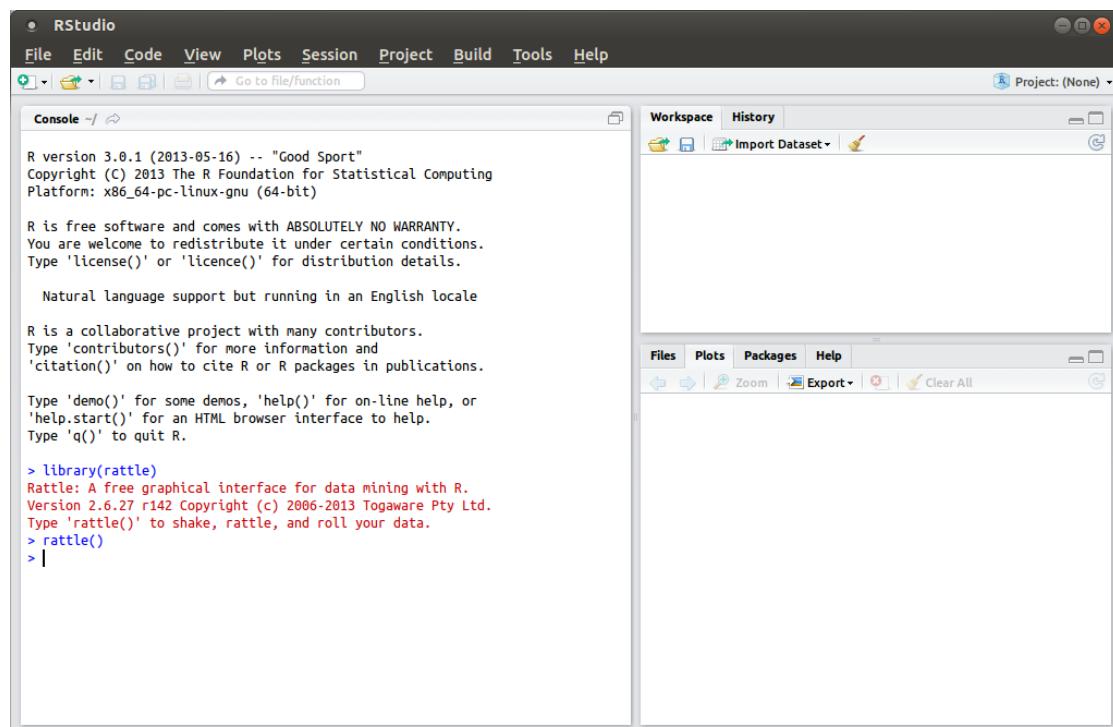
Even starting up R depends on your particular platform. Generally, it is started from a desktop icon or from the Application menu. Alternatively, on Linux it is often started up from a terminal window, like gnome-terminal or xterm. From the terminal we simply type the command R to invoke R itself.

An increasingly popular approach is to use RStudio. RStudio includes an R console. We can see the RStudio application below, with the commands to start up Rattle. Do note that this only works with the Desktop version of RStudio and not the server version of RStudio. The server version runs the interface in a browser on your desktop and communicates to a remote server running R itself. RStudio handles all of the graphical interface. Because Rattle has its own graphical interface, RStudio is unable to capture that interface from the server and display it on your desktop.

We can access the desktop version of RStudio from a server by running an XWindows server, such as xming, on our desktop.

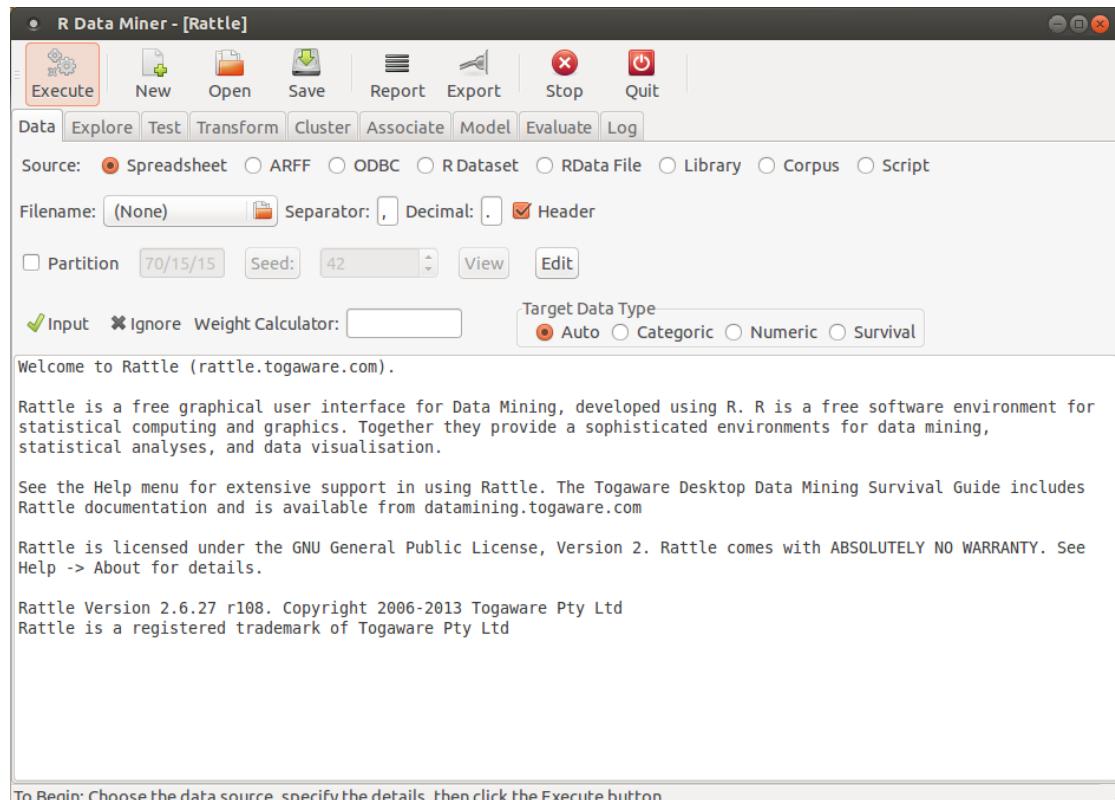
Whichever way we start R, we initiate Rattle with:

```
library(rattle)
rattle()
```



2 Getting Familiar With Rattle

The Rattle interface is based on a set of tabs through which we proceed, left to right. For any tab, once we have set up the required information, we **must** click the Execute button (or F2) to perform the actions. Take a moment to explore the interface a little by clicking through the various tabs. Notice the Help menu and find that the help layout mimics the tab layout.



To Quit from Rattle we simply click on the Quit button in the main Rattle window.

To Quit from RStudio we choose Quit from the File menu.

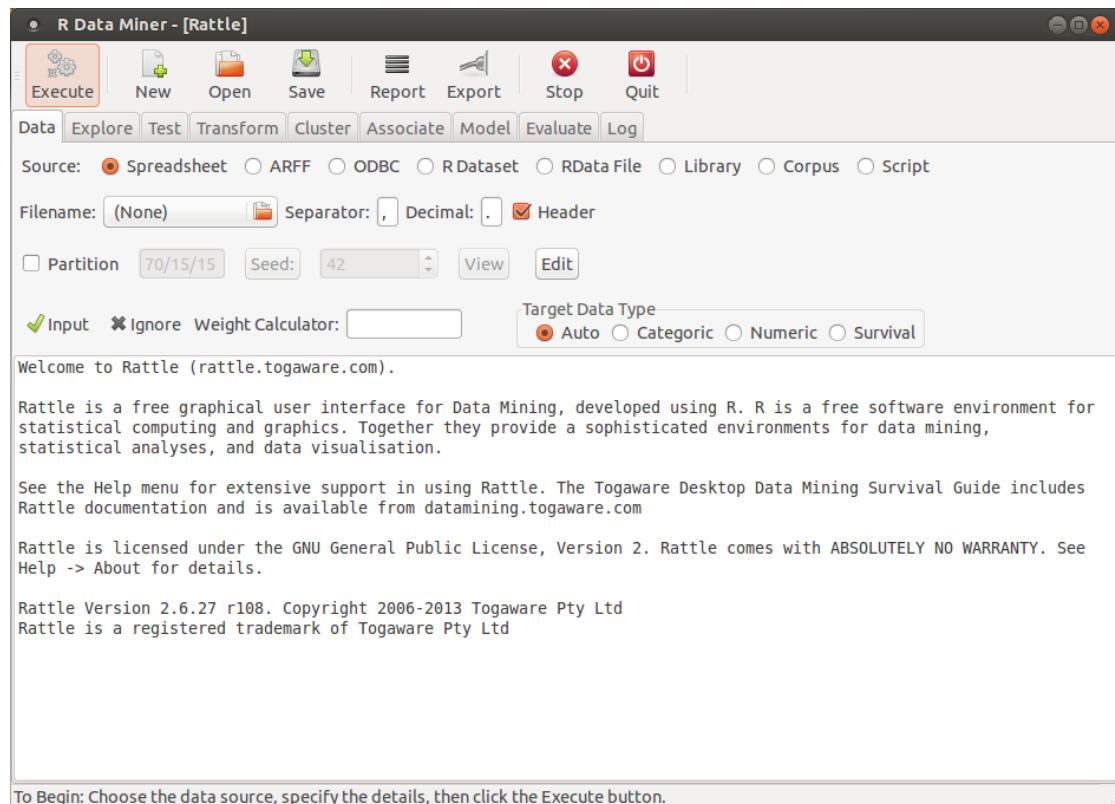
If we are using a terminal to run R then we can press 'Ctrl-D' (i.e. press the 'Control' key and then the 'D' key together).

In most cases we are asked whether to save our workspace. For now (and indeed for most users) we **do not** save the workspace.

3 The Initial Interface

The process that we implement in Rattle and that is reflected in the tabs that we see in the Rattle interface is:

1. Load a **Dataset**;
2. Select **Variables** for exploring and mining;
3. **Sample** the data into training and test datasets;
4. **Explore** the distributions of the data;
5. Perhaps **Test** some of the distributions;
6. Optionally **Transform** our data;
7. Build **Clusters** or **Association Rules** from the data;
8. Build predictive **Models**;
9. **Evaluate** the models;
10. Record the steps in building your model as listed in the **Log**.



4 Load Data, Build Model

Our first familiarisation task is to load the sample weather dataset supplied with Rattle and build a simple model.

1. Start up Rattle.
2. Click the Execute button.
3. Answer Yes to load the example weather dataset.
4. Click the Model tab.
5. Click the Execute button.
6. Click the Draw button.

This is our very first model. It is a decision tree model and can be used to predict the probability that it will rain in Canberra (Australia) tomorrow, given today's conditions in Canberra.

```

R Data Miner - [Rattle (weather.csv)]
Execute New Open Save Report Export Stop Quit
Data Explore Test Transform Cluster Associate Model Evaluate Log
Type: Tree Forest Boost SVM Linear Neural Net Survival All
Target: RainTomorrow Algorithm: Traditional Conditional Model Builder: rpart
Min Split: 20 Max Depth: 30 Priors: Include Missing
Min Bucket: 7 Complexity: 0.0100 Loss Matrix: Rules Draw
Summary of the Decision Tree model for Classification (built using 'rpart'):
n= 256

node), split, n, loss, yval, (yprob)
 * denotes terminal node

1) root 256 41 No (0.83984375 0.16015625)
  2) Pressure3pm>=1011.9 204 16 No (0.92156863 0.07843137)
    4) Cloud3pm< 7.5 195 10 No (0.94871795 0.05128205) *
    5) Cloud3pm>=7.5 9 3 Yes (0.33333333 0.66666667) *
  3) Pressure3pm< 1011.9 52 25 No (0.51923077 0.48076923)
    6) Sunshine>=8.85 25 5 No (0.80000000 0.20000000) *
    7) Sunshine< 8.85 27 7 Yes (0.25925926 0.74074074) *

Classification tree:
rpart(formula = RainTomorrow ~ ., data = crs$dataset[crs$train,
  c(crs$input, crs$target)], method = "class", parms = list(split = "information"),
  control = rpart.control(usesurrogate = 0, maxsurrogate = 0))

Variables actually used in tree construction:
[1] Cloud3pm  Pressure3pm Sunshine

Root node error: 41/256 = 0.16016
The Decision Tree model has been built. Time taken: 0.09 secs

```

5 Audit: Load Dataset

We now switch to the sample Audit dataset provided with `rattle` (Williams, 2014).

1. Click the Data tab.
2. Click the Filename: button where `weather.csv` is currently listed.
3. Choose the `audit.csv` file to load
4. Load the file into Rattle.

Be sure to investigate what the audit dataset is about, and the meaning of each of the variables. You should document this.

No.	Variable	Data Type	Input	Target	Risk	Ident	Ignore	Weight	Comment
1	ID	Numeric	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 2000
2	Age	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 67
3	Employment	Categoric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 8 Missing: 100
4	Education	Categoric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 16
5	Marital	Categoric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 6
6	Occupation	Categoric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 14 Missing: 101
7	Income	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 2000
8	Gender	Categoric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 2
9	Deductions	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 41
10	Hours	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 68
11	IGNORE_Accounts	Categoric	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	Unique: 33 Missing: 43
12	RISK_Adjustment	Numeric	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 310
13	TARGET_Adjusted	Numeric	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 2

Roles noted. 2000 observations and 9 input variables. The target is TARGET_Adjusted. Categorical 2. Classification models enabled.

6 Audit: Explore

Switching to the Explore tab investigate for any interesting patterns in the data. In particular, consider at least the following options.

1. Various summaries, noting any skewness or high values of kurtosis.
2. Anything interesting about missing values?
3. What does the cross tabulation suggest, if anything?
4. Various distribution plots including Benford's Law.
5. Any correlation between variables?

7 Audit: Test

The Test tab provides the opportunity to test out statistical hypotheses.

8 Audit: Transform

9 Audit: Cluster

10 Audit: Associate

11 Audit: Predictive Model

Exercise: Draw a tree and plot the evaluation.

12 Audit: Evaluate

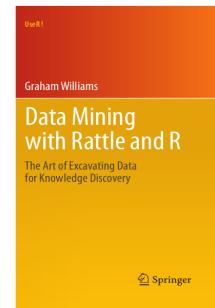
13 Audit: Review the Log

14 Assessment Activity

Now that you are familiar with interacting with a dataset in Rattle, load one of your own datasets, or else a public dataset from the Internet, and repeat the steps above using this dataset. Produce a report of your activities and discoveries. Submit the report as a PDF for assessment.

15 Further Reading

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 module is one of many OnePageR modules available from <http://onepager.togaware.com>. In particular follow the links on the website with a * which indicates the generally more developed OnePageR modules.

Other resources include:

- <http://rattle.togaware.com>
- <http://datamining.togaware.com>
- <http://datamining.togaware.com/survivor/index.html>

16 References

- 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/>.
- 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 3.0.4, URL <http://rattle.togaware.com/>.

This document, sourced from StartO.Rnw revision 436, was processed by KnitR version 1.6 of 2014-05-24 and took 1 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-06-21 20:44:44.

Data Science with R

Beyond Rattle into R

Graham.Williams@togaware.com

13th June 2014

Visit <http://onepager.togaware.com/> for more OnePageR's.

In this chapter we explore extensions to Rattle that allow us to enhance the significant capabilities of Rattle with the full power of R. Many of the extensions presented here could be incorporated into Rattle itself, and may indeed one day find their way into Rattle. However, one of the aims of Rattle is to provide an entree into programming R itself for the data scientist, and so users are encouraged to become comfortable switching between Rattle and R to achieve their goals as data scientists.

The required packages for this module include:

```
library(rattle)      # The Rattle GUI for Data Mining.
```

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 Model — Multiple Models of the Same Type

Rattle can build multiple models of different types in a single run, but not variations of a model type. Here we illustrate in R code how we can do this.

Suppose we want to build several variations of decision trees, with different options. On a small test dataset, run Rattle withing RStudio to build models with different parameter choices. Grab the R code from the Log tab, paste into RStudio, and turn it into a schedule to run on the full dataset overnight.

Here's an example. Run Rattle over the weather dataset, using a small training partition, to build a decision tree with different parameter options. The Rattle Log code for just two runs:

```
crs$rpart <- rpart(RainTomorrow ~ .,
  data=crs$dataset[crs$train, c(crs$input, crs$target)],
  method="class", parms=list(split="information"),
  control=rpart.control(usesurrogate=0, maxsurrogate=0))

crs$rpart <- rpart(RainTomorrow ~ .,
  data=crs$dataset[crs$train, c(crs$input, crs$target)],
  method="class", parms=list(split="information"),
  control=rpart.control(minsplit=5, maxdepth=10,
    cp=0.000010, usesurrogate=0, maxsurrogate=0))
```

We can see that each run overwrites the previous model. So tidying things up a little, we might end up with the following script:

```
ds <- crs$dataset[crs$train, c(crs$input, crs$target)]
form <- formula("RainTomorrow ~ .")

m.rp01 <- rpart(form, data=ds, method="class", parms=list(split="information"),
  control=rpart.control(usesurrogate=0, maxsurrogate=0))

m.rp02 <- rpart(form, data=ds, method="class", parms=list(split="information"),
  control=rpart.control(minsplit=5, maxdepth=10,
    cp=0.000010, usesurrogate=0, maxsurrogate=0))
```

Change `ds` to point to the full (i.e., larger) dataset and set the script running overnight:

```
ds <- crs$dataset[crs$train, c(crs$input, crs$target)]
```

With a bit of care we can then use Rattle to explore the different models! For example:

```
crs$rpart <- m.rp02
```

Then click on Draw within the Model Tab, Decision Tree option, to draw the tree. Of course we can copy the code to draw the tree and run it directly in RStudio:

```
fancyRpartPlot(m.rp02, main="My Very Own Decision Tree")
```

Concept suggested by Richard Calaba (140607) on [Google Code](#).

2 Evaluate — Score — Class and Probability

Rattle (Evaluate Tab, Score Option) can score new data to report the predicted Class or Probability, but not both at the same time. Here we score and collect both and then save into the one file, together with the original dataset being scored.

We begin by saving the dataset to be scored into the data frame `ds`. Notice that because we have a random forest as one of the models, we use `na.omit()` to eliminate rows with missing data. The dataset we are scoring is the test dataset that is created by default as a partition of the dataset loaded into Rattle.

```
ds <- na.omit(crs$dataset[crs$test, c(crs$input, crs$target)])
```

We then simply call `predict()` for each of the models we which to use to score our dataset, appending the results as additional columns of the data frame `ds`.

```
ds$rp.cl <- predict(crs$rpart, newdata=ds, type="class")
ds$rp.pr <- predict(crs$rpart, newdata=ds)[,2]
ds$rf.cl <- predict(crs$rf, newdata=ds)
ds$rf.pr <- predict(crs$rf, newdata=ds, type="prob")[,2]
```

After checking the data frame looks okay using `head()` we save the results to file using `write.csv()`.

```
head(ds)
write.csv(ds, file="scores.csv", row.names=FALSE)
```

Other models can easily be added using the above template. For example, to add an AdaBoost model, build the model in Rattle and then score using the Evaluate tab. From the Rattle Log tab we will see the syntax used to score using the new model. The following lines come directly from the Log Tab:

```
crs$pr <- predict(crs$rpart,
                   newdata=crs$dataset[crs$validate, c(crs$input)],
                   type="class")
crs$pr <- predict(crs$ada,
                   newdata=crs$dataset[crs$validate, c(crs$input)])
```

We simply paste these lines into the above script and change the lines to:

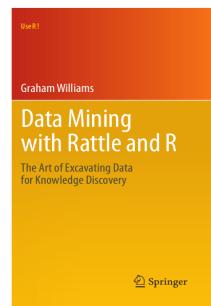
```
ds$ada.cl <- predict(crs$ada, newdata=ds)
ds$ada.pr <- predict(crs$ada, newdata=ds, type="prob")[,2]
```

Concept suggested by Richard Calaba (140607) on [Google Code](#).

3 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 module is one of many OnePageR modules available from <http://onepager.togaware.com>. In particular follow the links on the website with a * which indicates the generally more developed OnePageR modules.



4 References

- 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/>.
- 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 3.0.4, URL <http://rattle.togaware.com/>.

This document, sourced from RattleO.Rnw revision 421, was processed by KnitR version 1.6 of 2014-05-24 and took 1.3 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-06-13 20:59:47.

One Pager Data Science

Programming with Data: Introducing R

Graham.Williams@togaware.com

4th February 2019

Visit <https://essentials.togaware.com/onepagers> for more Essentials.

Data scientists write programs to ingest, fuse, clean, wrangle, visualise, analyse, and model data. Programming over data is a core task for the data scientist. We will primarily use R as our programming language ([R Core Team, 2018a](#)) and assume basic familiarity of R as may be gained from the many resources available on the Intranet, particularly from <https://cran.r-project.org/manuals.html>.

Programmers of data develop sentences or *code*. Code instructs a computer to perform specific tasks. A collection of sentences written in a language is what we might call a *program*. Through *programming by example* and *learning by immersion* we will share programs to deliver insights and outcomes from our data.

R is a large and complex ecosystem for the practice of data science. There is much freely available information on the Internet from which we can continually learn and borrow useful code segments that illustrate almost any task we might think of. We introduce here the basics for getting started with R, libraries and packages which extend the language, and the concepts of functions, commands, and operators. We also introduce the powerful concept of pipes as a foundation for building sophisticated data processing pipelines in R.

Through this guide new R commands will be introduced. The reader is encouraged to review the command's documentation and understand what the command does. Help is obtained using the `? command as in:`

```
?read.csv
```

Documentation on a particular package can be obtained using the `help=` option of `library()`:

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

This chapter is intended to be hands on. To learn effectively you are encouraged to run R (e.g., RStudio or Emacs with ESS mode) and to replicate the commands. Check that output is the same and that you understand how it is generated. Try some variations. Explore.

Copyright © 2000-2019 Graham Williams. This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#) allowing this work to be copied, distributed, or adapted, with attribution and provided under the same license.



1 Tooling For R Programming

We will first need to have available to us the computer software that interprets our programs that we write as sentences in the R language. Usually we will install this software, called the R Interpreter, on our own computer and instructions for doing so are readily available on the Internet. The [R Project¹](#) is a good place to start. Most GNU/Linux distributions freely provide R packages from their application stores and a commercial offering is available from Microsoft who also provide direct access within SQL Server and through Cortana Analytics on the Azure cloud. Now is a good time to install R on your own computer if that is required.

The open source and freely available [RStudio²](#) software is recommended as a modern **integrated development environment** (IDE) for writing R programs. It can be installed on the common desktop or server computers running GNU/Linux, Microsoft/Windows, or Apple/MacOS. A browser version of RStudio also allows a desktop to access R running on a back-end cloud server.

The server version of RStudio runs on a remote computer (e.g., a server in the cloud) with the graphical interface presented within a web browser on our own desktop. The interface is much the same as the desktop version but all of the commands are run on the server rather than on our own desktop. Such a setup is useful when we require a powerful server on which to analyse very large datasets. We can then control the analyses from our own desktop with the results sent from the server back to our desktop whilst all the computation is performed on the server itself which may be a considerably more powerful computer than our usually less capable personal computers.

¹<https://www.r-project.org>

²<https://www.rstudio.com>

2 Introducing RStudio

The RStudio interface includes an **R Console** on the left through which we directly communicate with the R software. The window pane on the top right provides access to the **Environment** within which R is operating. This includes data and datasets that are defined as we interact with R. Initially it is empty. A **History** of all the commands we have asked R to run is available on another tab within the top right pane.

The bottom right pane provides direct access to an extensive collection of **Help**. On another tab within this same pane we can access the **Files** on our hard disk. Other tabs provide access to **Plots**, **Packages**, and a **Viewer** to view documents that we might generate.

In the **Help** tab displayed in the bottom right pane of Figure 1 we can see numerous links to R documentation. The **Manual** titled *An Introduction to R* within the **Help** tab is a good place to start if you are unfamiliar with R. There are also links to learning more about RStudio and these are recommended if you are new to RStudio.

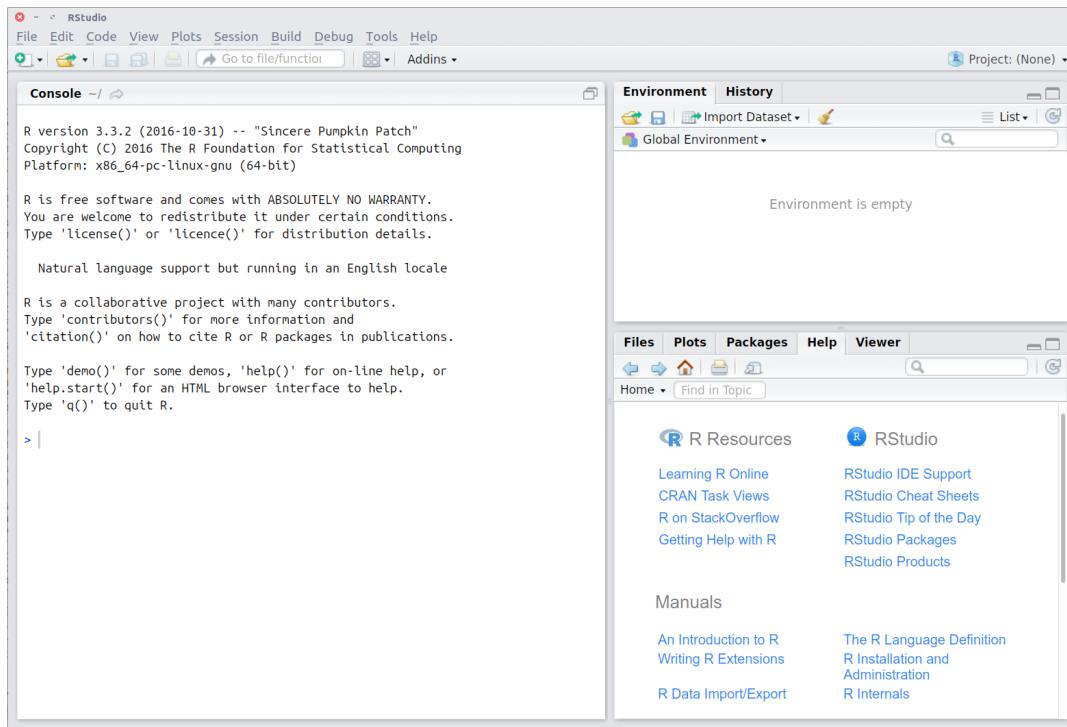


Figure 1: The initial layout of the RStudio window panes showing the **Console** pane on the left with the **Environment** and **Help** panes on the right.

3 Editing Code

Often we will be interacting with R by writing code and sending that code to the R Interpreter so that it can be run (locally or remotely). It is always good practice to store this code into a file so that we have a record of what we have done and are able to replicate the work at a later time. Such a file is called an R Script.

We can create a new R Script by clicking on the first icon on the RStudio toolbar and choosing the first item in the resulting menu as in Figure 2. A keyboard shortcut is also available to do this: **Ctrl+Shift+N** (hold the **Ctrl** and the **Shift** keys down and press the **N** key).

A sophisticated file editor is presented as the top left pane within RStudio. The tab will initially be named **Untitled1** until we actually save the script to a file on the disk. When we do so we will be asked to provide a name for the file.

The editor is where we write our R code and compose the programs that instruct the computer to perform particular tasks. The editor provides numerous features that are expected in a modern program editor. These include syntax colouring, automatic indentation to improve layout, automatic command completion, interactive command documentation, and the ability to send specific commands to the R Console to have them run by the R Interpreter.

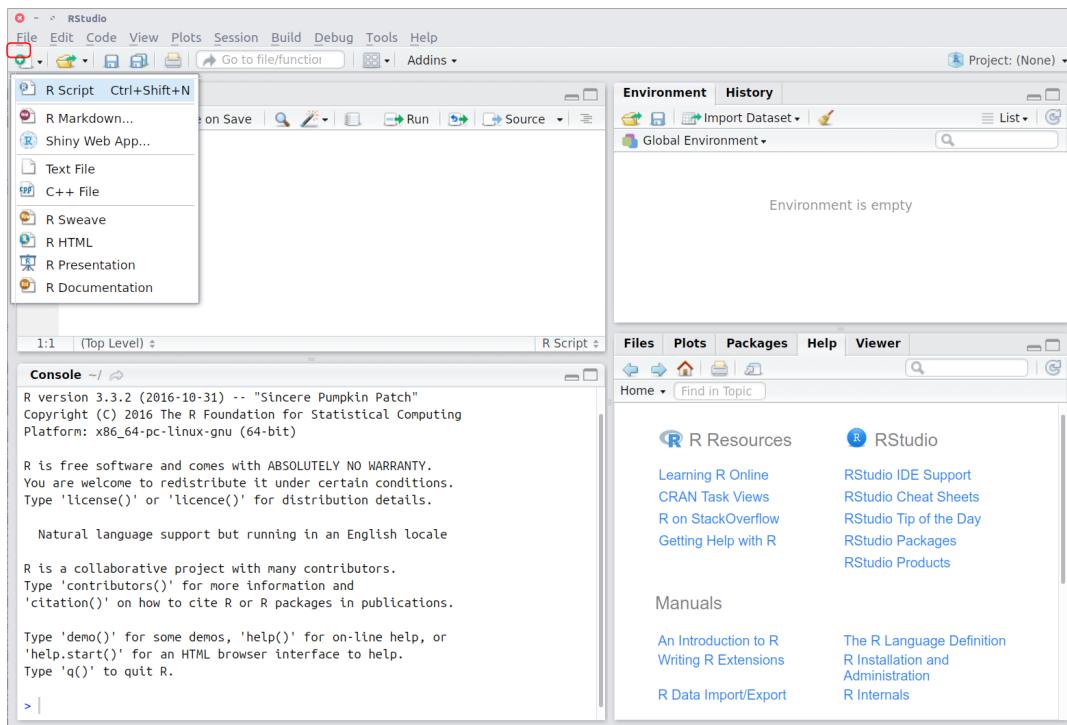


Figure 2: Ready to edit R scripts in RStudio using the **Editor** pane on the top left created when we choose to add a new R Script from the toolbar menu displayed by clicking the icon highlighted in red.

4 Executing Code

To send R commands to the R Console we can use the appropriate toolbar buttons. One line or a highlighted region can be sent using the Run button. Having opened a new R script file we can enter the commands below. The four commands make up a program which begins with an `install.packages()` to ensure we have installed two requisite software packages for R. The second and third commands use `library()` to load those packages into the current R session. The fourth command produces a plot of the `weatherAUS` dataset. This dataset contains observations of weather related variables across almost 50 weather stations in Australia over multiple years.

```
install.packages(c("ggplot2", "rattle"))
library(ggplot2)
library(rattle)
qplot(data=weatherAUS, x=MinTemp, y=MaxTemp)
```

`qplot()` allows us to quickly code a plot. Using the `weatherAUS` dataset we plot the minimum daily temperature (`MinTemp`) on the x-axis against the maximum daily temperature (`MaxTemp`) on the y-axis. The resulting plot is a **scatter plot** as we see the plot in the lower right pane.

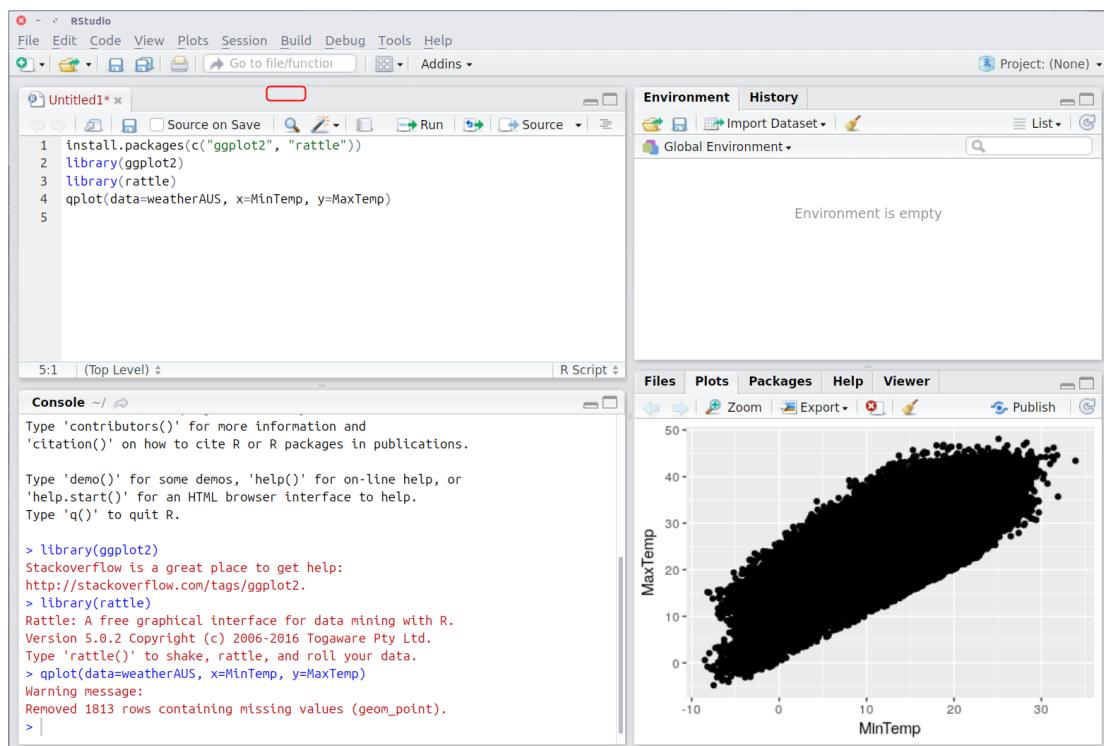


Figure 3: Running R commands in RStudio. The R programming code is written into a file using the editor in the top left pane. With the cursor on the line containing the code we click the Run button to pass the code on to the R Console to have it run by the R Interpreter to produce the plot we see in the bottom right pane.

5 RStudio Review

Figure 3 shows the RStudio editor as it appears after we have typed the above commands into the R Script file in the top left pane. We have sent the commands to the R Console to have it run by R. We have done this by ensuring the cursor within the R Script editor is on the same line as the command to be run and then clicking the Run button. We will notice that the command is sent to the R Console in the bottom left pane and the cursor advances to the next line within the R Script editor.

After each command is run any text output by the command is displayed in the R Console whilst graphic output is displayed in the Plots tab of the bottom right pane.

This is now our first program in R. We can now provide our first observations of the data. It is not too hard to see from the plot that there appears to be quite a strong relationship between the minimum temperature and the maximum temperature: with higher values of the minimum temperature recorded on any particular day we see higher values of the maximum temperature. There is also a clear lower boundary that might suggest, as logic would dictate, that the maximum temperature can not be less than the minimum temperature. If we were to observe data points below this line then we would begin to explore issues with the quality of the data.

As data scientists we have begun our observation and understanding of the data, taking our first steps toward immersing ourselves in and thereby beginning to understand the data.

6 Packages and Libraries

The power of the R ecosystem comes from the ability to extend the language by its nature as **open source software**. Anyone is able to contribute to the R ecosystem and by following stringent guidelines they can have their contributions included in the comprehensive R archive network, abbreviated as CRAN³. Such contributions are collected by an author into what is called a *package*. Over the decades many researchers and developers have contributed very many *packages* to CRAN.

A **package** is how R collects together *commands* for a specific collection of tasks. A **command** is a verb in the computer language used to tell the computer to do something. There are over 13,000 packages available for R from almost as many different authors. Hence there are very many verbs available to build our sentences to command R appropriately. With so many packages there is bound to be a package or two covering essentially any kind of processing we could imagine though we will also find packages offering the same or similar commands (verbs) perhaps even with very different meanings.

Beginning with Chapter 7 we will list at the beginning of each chapter the R *packages* that are required for us to be able to replicate the examples presented in that chapter. Packages are installed from the Internet (from the securely managed CRAN⁴ package repository) into a local *library* on our own computer. A **library** is a folder on our computer's storage which contains sub-folders corresponding to each of the installed packages.

To install a package from the Internet we can use the command `install.packages()` and provide to it as an *argument* the name of the package to install. The package name is provided as a **string** of **characters** within quotes and supplied as the `pkgs= argument` to the command as in the following code. Here we choose to install a package called `dplyr`—a very useful package for data manipulations.

```
# Install a package from a CRAN repository.  
  
install.packages(pkgs="dplyr")
```

Once a package is installed we can access the commands provided by that package by prefixing the command name with the package name as in `ggplot2::qplot()`. This is to say that `qplot()` is provided by the `ggplot2` (Wickham *et al.*, 2018a) package.

³CRAN: <https://cran.r-project.org>.

⁴<https://cran.r-project.org>

7 A Glimpse of the Dataset

Another example of a useful command that we will find ourselves using often is the `glimpse()` command from the `dplyr` (Wickham *et al.*, 2018b) package. This command can be accessed in the R console as `dplyr::glimpse()` once the `dplyr` package has been installed. This particular command accepts an *argument x=* which names the dataset we wish to glimpse. In the following R example we use the `weatherAUS` dataset from the `rattle` (Williams, 2018) package.

```
# Review the dataset.

dplyr::glimpse(x=rattle::weatherAUS)

## Observations: 145,463
## Variables: 24
## $ Date      <date> 2008-12-01, 2008-12-02, 2008-12-03, 2008-12-04, 200...
## $ Location   <fct> Albury, Albury, Albury, Albury, Albury, Albu...
## $ MinTemp    <dbl> 13.4, 7.4, 12.9, 9.2, 17.5, 14.6, 14.3, 7.7, 9.7, 13...
## $ MaxTemp    <dbl> 22.9, 25.1, 25.7, 28.0, 32.3, 29.7, 25.0, 26.7, 31.9...
....
```

We can see here the command that was run and the output from running that command. As a convention used in this book the output from running R commands is prefixed with “## ”. The “#” introduces a comment in an R script file and tells R to ignore everything that follows on that line. We use the “## ” convention throughout the book to clearly identify output produced by R. When we run these commands ourselves in R this prefix is not displayed.

Long lines of output are also truncated for our presentation here. The ... at the end of the lines and the at the end of the output indicate that the output has been truncated for the sake of keeping our example output to a minimum.

8 Attaching a Package

We can **attach** a package to our *current* R session from our local *library* for added convenience. This will make the command available during this specific R session without the requirement to specify the package name each time we use the command. Attaching a package tells the R software to look within that package (and then to look within any other attached packages) when it needs to find out what the command should do (the **definition** of the command).

We can see which packages are currently attached using `base::search()`. The order in which they are listed here corresponds to the order in which R searches for the definition of a command.

```
base::search()  
## [1] ".GlobalEnv"      "package:stats"  
## [3] "package:graphics" "package:grDevices"  
## [5] "package:utils"     "package:datasets"  
## [7] "package:methods"   "Autoloads"  
## [9] "package:base"
```

Notice that a collection of *packages* are installed by default among a couple of other special objects (`.GlobalEnv` and `Autoloads`).

A package is attached using the `base::library()` command. The `base::library()` command takes an argument to identify the `package=` we wish to *attach*.

```
# Load packages from the local library into the R session.  
  
library(package=dplyr)  
library(package=rattle)
```

Running these two commands will affect the search path by placing these packages early within the path.

```
base::search()  
## [1] ".GlobalEnv"      "package:rattle"  
## [3] "package:dplyr"    "package:stats"  
## [5] "package:graphics" "package:grDevices"  
## [7] "package:utils"     "package:datasets"  
## [9] "package:methods"   "Autoloads"  
## [11] "package:base"
```

9 Simplifying Commands

By attaching the `dplyr` package we can drop the package name prefix for any commands from the package. Similarly by attaching `rattle` we can drop the package name prefix from the name of the dataset. Our previous `dplyr::glimpse()` command can be simplified.

```
# Review the dataset.

glimpse(x=weatherAUS)

## Observations: 145,463
## Variables: 24
## $ Date           <date> 2008-12-01, 2008-12-02, 2008-12-03, 2008-12-04, 200...
## $ Location       <fct> Albury, Albury, Albury, Albury, Albury, Albu...
## $ MinTemp        <dbl> 13.4, 7.4, 12.9, 9.2, 17.5, 14.6, 14.3, 7.7, 9.7, 13...
## $ MaxTemp        <dbl> 22.9, 25.1, 25.7, 28.0, 32.3, 29.7, 25.0, 26.7, 31.9...
....
```

We can actually simplify this a little more. Often for a command we don't have to explicitly name all of the arguments. In the following example we drop the `package=` and the `x=` arguments as the commands themselves know what to expect implicitly.

```
# Load packages from the local library into the R session.

library(dplyr)
library(rattle)

# Review the dataset.

glimpse(weatherAUS)

## Observations: 145,463
## Variables: 24
## $ Date           <date> 2008-12-01, 2008-12-02, 2008-12-03, 2008-12-04, 200...
## $ Location       <fct> Albury, Albury, Albury, Albury, Albury, Albu...
## $ MinTemp        <dbl> 13.4, 7.4, 12.9, 9.2, 17.5, 14.6, 14.3, 7.7, 9.7, 13...
## $ MaxTemp        <dbl> 22.9, 25.1, 25.7, 28.0, 32.3, 29.7, 25.0, 26.7, 31.9...
....
```

A number of packages are automatically attached when R starts. The first `base::search()` command above returned a vector of packages and since we had yet to attach any further packages those listed are the ones automatically attached. One of those is the `base` (R Core Team, 2018a) package which provides the `base::library()` command.

10 Working with the Library

To summarise then, when we interact with R we can usually drop the package prefix for those commands that can be found in one (or more) of the attached packages. Throughout the running text in this book we retain the package prefix to clarify where each command comes from. However, within the code we tend to drop the package name prefix.

A prefix can still be useful in larger programs to ensure we are using the correct command and to communicate to the human reader where the command comes from. Some packages do implement commands with the same names as commands defined differently and found in other packages and the prefix notation is then useful to specify which command we are referring to.

Each of the following chapters will begin with a list of *packages* to *attach* from the *library* into the R session. Below is an example of attaching five common packages for our work. Attaching the listed packages will allow the examples presented within the chapter to be replicated. In the code below take note of the use of the hash () to introduce a comment which is ignored by R—R will not attempt to understand the comments as commands. Comments are there to assist the human reader in understanding our programs.

```
# Load packages required for this script.

library(rattle)    # The weatherAUS dataset and normVarNames().
library(readr)     # Efficient reading of CSV data.
library(dplyr)      # Data wrangling and glimpse().
library(ggplot2)    # Visualise data.
library(magrittr)   # Pipes %>%, %<>%, %T>%, %$%.
```

In starting up an R session (for example, by starting up RStudio) we can enter the above `library()` commands into an R script file created using the **New R Script File** menu option in RStudio and then ask RStudio to **Run** the commands. RStudio will send each command to the **R Console** which sends the command on to the R software. It is the R software that then runs the commands. If R responds that the package is not available then the package will need to be installed, which we can do from RStudio's **Tools** menu or by directly using `utils::install.packages()` as we saw above.

11 Functions

commands in R instruct the R software to perform particular actions. Formally commands are actually *functions* and we generally use this term to convey the mathematical concept of a function. A function simply takes a set of inputs and produces an output. R provides a wealth of *functions*, and we will generally think of each one as either a *function*, a *command* or an *operator*.

All R *functions* take arguments and return values. When we run a function (as distinct from what we will identify shortly as a *command*) we are interested in the value that it returns. Below is a simple example where we `base::sum()` two numbers.⁵

```
# Add two numbers.

sum(1, 2)
## [1] 3
```

As previously we precede the output from the function by the double hash (`##`). We will however not see the double hash in the R Console when we run the functions ourselves. Instead, the output we see will begin with the `[1]` which indicates that the returned value is a *vector* which starts counting from index 1. A *vector* is a collection of items which we can access through a sequential index—in this case the vector has only a single item.

We can store the result value from running the function (the value being a vector of length 1 containing just the item 3) into a *variable*. A *variable* is a name that we can use to refer to some location in the computer's memory where we store data while our programs are running. To store data in the computer's memory so that we can refer to that data by the specific variable name we use the *assignment operator* `base::<-`.

```
# Add two numbers and assign the result into a variable.

v <- sum(1, 2)
```

⁵It is a good idea to replicate all of the examples presented here in R. Simply open an R script file to edit and type the text `sum(1, 2)` using the editor in RStudio. Then instruct RStudio to Run the command in the R Console.

12 Accessing Stored Values

We can access the value stored in the variable (or actually stored in the computer's memory that the variable name refers to) simply by requesting through the R **Console** that R "run" the command `v`. We do so below with a line containing just `v`. In fact the R software actually runs the `base::print()` function to display the contents of the computer's memory that `v` refers to—this is a convenience that the R software provides for us.

```
# Print the value stored in a variable.  
  
v  
## [1] 3  
  
print(v)  
## [1] 3
```

By indexing the variable with `[1]` we can ask for the first item referred to by `v`. Noting that the variable `v` is a vector with only a single item when we try to index a second item we get an `NA` (meaning not available or a missing value).

```
# Access a particular value from a vector of data.  
  
v[1]  
## [1] 3  
  
v[2]  
## [1] NA
```

13 Commands

For **commands** (rather than *functions*) we are generally more interested in the actions or side-effects that are performed by the command rather than the value returned by the command. For example, the `base::library()` command is run to attach a package from the library (the action) and consequently modifies the search path that R uses to find commands (the side-effect).

```
# Load package from the local library into the R session.

library(rattle)
```

A **command** is also a function and does in fact return a value. In the above example we do not see any value printed. Some functions in R are implemented to return values invisibly. This is the case for `base::library()`. We can ask R to print the value when the returned result is invisible using the function `base::print()`.

```
# Demonstrate that library() returns a value invisibly.

print(library(rattle))

## [1] "readr"      "extrafont"   "Hmisc"       "Formula"     "survival"    "lattice"
## [7] "magrittr"   "xtable"     "rattle"      "ggplot2"    "tidyR"      "stringr"
## [13] "dplyr"      "knitr"      "stats"      "graphics"   "grDevices"  "utils"
## [19] "datasets"   "methods"    "base"
```

We see that the value returned by `base::library()` is a vector of character strings. Each character string names a package that has been attached during this session of R. Notice that the vector begins with item [1] and then item [5] continues on the second line, and so on. We can save the resulting vector into a variable and then index the variable to identify packages at specific locations within the vector.

```
# Load a package and save the value returned.

l <- library(rattle)

# Review one of the returned values.

l[7]
## [1] "magrittr"
```

14 Operators

The third type of function is the *operator*. We use **operators** in the flow of an expression. Technically they are called *infix* operators.

In the following example we use the `base:::+` infix operator.

```
# Add two numbers using an infix operator.  
  
1 + 2  
## [1] 3
```

Internally R actually converts the operator into a functional prefix form to be run in the same way as any other function.

```
# Add two numbers using the equivalent functional form.  
  
`+`(1, 2)  
## [1] 3
```

Note that R also supports using quotes and double quotes instead of the backtick we used in the code above. The backtick is preferred as per page 142 of [Wickham \(2014\)](#).

The key thing to remember is that all commands and operators are functions and we instruct R to take action by running functions.

15 Pipes

A function (Section 11) performs some action on some input and returns some output. We can think of functions as the verbs of the language—the action words of our sentences.

As we learn many new functions useful for the data scientist we will construct longer sentences that string together a sequence of verbs to deliver the results. This is a powerful programming concept, combining dedicated and well designed and implemented functions, each function focused on achieving a specific outcome.

To combine single focus functions into more complex operations we can use *pipes*. Pipes are a powerful concept for combining functions. The concept will be familiar to most who have used Unix and Linux. The idea is to pass the output of one function on to another function as its input. Each function does one task and aims to do that task very well, very accurately, and very simply from a user’s point of view. We can then pipe together many such specialist functions to deliver very complex and quite sophisticated data transformations in an easily accessible manner. Pipes were introduced in R through the `magrittr` (Bache and Wickham, 2014) package.

To illustrate the concept of pipes we will again use the `rattle::weatherAUS` dataset. We review the basic contents of the `rattle::weatherAUS` dataset by printing it.

```
# Review the dataset of weather observations.

weatherAUS

##           Date      Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 1 2008-12-01      Albury     13.4    22.9      0.6          NA        NA
## 2 2008-12-02      Albury      7.4    25.1      0.0          NA        NA
....
```

We might be interested in the distribution of specific numeric variables from the dataset. For that we will `dplyr::select()` a few numeric variables by *piping* the dataset into `dplyr::select()`.

```
# Select variables from the dataset.

weatherAUS %>%
  select(MinTemp, MaxTemp, Rainfall, Sunshine)

##       MinTemp MaxTemp Rainfall Sunshine
## 1     13.4    22.9     0.6      NA
## 2      7.4    25.1     0.0      NA
## 3     12.9    25.7     0.0      NA
## 4      9.2    28.0     0.0      NA
## 5     17.5    32.3     1.0      NA
....
```

Notice how `weatherAUS` by itself will list the whole of the dataset. Piping the whole dataset to the `dplyr::select()` function using the *pipe* operator `purrr::%>%` tells R to send the `rattle::weatherAUS` dataset on the left to the `dplyr::select()` function on the right. We provided `dplyr::select()` with a list of the variables we wish to select. The end result returned as the output of the pipeline is a subset of the original dataset containing just the named columns.

16 Multiple Pipes

Suppose we now want to produce a `base:::summary()` of these numeric variables. We will *pipe* the dataset produced by `dplyr::select()` into `base:::summary()`.

```
# Select variables from the dataset and summarise the result.
```

```
weatherAUS %>%
  select(MinTemp, MaxTemp, Rainfall, Sunshine) %>%
  summary()

##      MinTemp        MaxTemp        Rainfall        Sunshine
##  Min.   :-8.70   Min.   :-4.10   Min.   : 0.00   Min.   : 0.00
##  1st Qu.: 7.40   1st Qu.:17.90   1st Qu.: 0.000  1st Qu.: 4.90
##  Median :11.90   Median :22.50   Median : 0.000  Median : 8.40
##  Mean   :12.04   Mean   :23.14   Mean   : 2.298  Mean   : 7.61
##  3rd Qu.:16.70   3rd Qu.:28.20   3rd Qu.: 0.600  3rd Qu.:10.60
##  Max.   :33.90   Max.   :48.10   Max.   :371.000 Max.   :14.50
....
```

Perhaps we would like to summarise only those observations where there is more than a little rain on the day of the observation. To do so we will `stats::filter()` the observations.

```
# Select specific variables and observations from the dataset.
```

```
weatherAUS %>%
  select(MinTemp, MaxTemp, Rainfall, Sunshine) %>%
  filter(Rainfall >= 1)

##      MinTemp MaxTemp Rainfall Sunshine
##  1       17.5    32.3     1.0      NA
##  2       13.1    30.1     1.4      NA
##  3       15.9    21.7     2.2      NA
##  4       15.9    18.6     15.6     NA
##  5       12.6    21.0     3.6      NA
....
```

We can see that this sequence of functions operating on the original `rattle:::weatherAUS` dataset returns a subset of that dataset where all observations have some rain.

17 Backward Assignment

We saw earlier in this chapter the assignment operator `base::<-` which is used to save a result into the computer's memory and give it a name that we can refer to later. We can use this operator to save the result of the sequence of operations from the pipe.

```
# Select variables/observations and save the result.

rainy.days <-
  weatherAUS %>%
  select(MinTemp, MaxTemp, Rainfall, Sunshine) %>%
  filter(Rainfall >= 1)
```

An alternative that makes sense within the pipe paradigm is to use the forward assignment operator `base::->` to save the resulting data into a variable in a logically consistent way.

```
# Demonstrate use of the forward assignment operator.

weatherAUS %>%
  select(MinTemp, MaxTemp, Rainfall, Sunshine) %>%
  filter(Rainfall >= 1) ->
  rainy.days
```

In this example we see that using the forward assignment operator tends to hide the actual variable into which the assignment is made. The important side effect of the series of commands, the assignment to `rainy.days`, could easily be missed. Compare that to the prior code.

Instead, when using hte forward assignment, be sure to un-indent the final variable name

```
# Best to unindent the final assigned variable for clarity.

weatherAUS %>%
  select(MinTemp, MaxTemp, Rainfall, Sunshine) %>%
  filter(Rainfall >= 1) ->
rainy.days
```

18 Comparing Distributions

Continuing with our pipeline example, we might want a `base::summary()` of the dataset.

```
# Summarise subset of variables for observations with rainfall.

weatherAUS %>%
  select(MinTemp, MaxTemp, Rainfall, Sunshine) %>%
  filter(Rainfall >= 1) %>%
  summary()

##      MinTemp        MaxTemp        Rainfall        Sunshine
##  Min.   :-8.50   Min.   :-4.10   Min.   : 1.000   Min.   : 0.000
##  1st Qu.: 8.40   1st Qu.:15.50   1st Qu.: 2.200   1st Qu.: 2.300
##  Median :12.20   Median :19.30   Median : 4.800   Median : 5.500
##  Mean   :12.71   Mean   :20.13   Mean   : 9.714   Mean   : 5.345
##  3rd Qu.:17.10   3rd Qu.:24.30   3rd Qu.:11.000   3rd Qu.: 8.100
##  Max.   :28.30   Max.   :46.30   Max.   :371.000   Max.   :14.200
....
```

It could be useful to contrast this with a `base::summary()` of those observations where there was virtually no rain.

```
# Summarise observations with little or no rainfall.

weatherAUS %>%
  select(MinTemp, MaxTemp, Rainfall, Sunshine) %>%
  filter(Rainfall < 1) %>%
  summary()

##      MinTemp        MaxTemp        Rainfall        Sunshine
##  Min.   :-8.70   Min.   :-2.10   Min.   :0.00000   Min.   : 0.00
##  1st Qu.: 7.10   1st Qu.:18.90   1st Qu.:0.00000   1st Qu.: 6.10
##  Median :11.80   Median :23.60   Median :0.00000   Median : 9.30
##  Mean   :11.83   Mean   :24.08   Mean   :0.05936   Mean   : 8.33
##  3rd Qu.:16.60   3rd Qu.:29.10   3rd Qu.:0.00000   3rd Qu.:10.90
##  Max.   :33.90   Max.   :48.10   Max.   :0.90000   Max.   :14.50
....
```

Any number of functions can be included in a ***pipeline*** to achieve the results we desire. In the following chapters we will see many examples and some will string together ten or more functions. Each step along the way is of itself generally easily understandable. The power is in what we can achieve by stringing together many simple steps to produce something more complex.

19 Pipeline Syntactic Sugar

For the technically minded we note that what is actually happening here is that the syntax, or how we write the sentences, is changed by R from the normal embedded functional approach in order to increase the ease with which we can read the code. This is an important goal as we need to always keep in mind that we write our code for others (and ourselves later on) to read.

Below we see a pipeline version of a series of commands operating on the dataset and not how it is mapped by R into the functional construct. For many of us it will take quite a bit of effort to parse this traditional functional form of the expression so as to understand what it is doing. The pipeline alternative provides a clearer narrative

```
# Summarise observations with little or no rainfall.
```

```
weatherAUS %>%
  select(MinTemp, MaxTemp, Rainfall, Sunshine) %>%
  filter(Rainfall < 1) %>%
  summary()

## #> #> #> #>
```

Functional form equivalent to the pipeline above.

```
summary(filter(select(weatherAUS,
                     MinTemp, MaxTemp, Rainfall, Sunshine),
                     Rainfall < 1))
```

```

##      MinTemp        MaxTemp       Rainfall      Sunshine
## Min.   :-8.70    Min.   :-2.10   Min.   :0.00000   Min.   : 0.00
## 1st Qu.: 7.10    1st Qu.:18.90   1st Qu.:0.00000   1st Qu.: 6.10
## Median :11.80    Median :23.60   Median :0.00000   Median : 9.30
## Mean   :11.83    Mean   :24.08   Mean   :0.05936   Mean   : 8.33
## 3rd Qu.:16.60    3rd Qu.:29.10   3rd Qu.:0.00000   3rd Qu.:10.90
## Max.   :33.90    Max.   :48.10   Max.   :0.90000   Max.   :14.50

```

Anything that improves the readability of our code is useful. Computers are quite capable of doing the hard work of transforming a simpler sentence into this much more complex looking sentence.

20 Assignment Pipe

There are several variations of the pipe operator available. A particularly handy operator implemented by `magrittr` is the assignment pipe `magrittr::%<>%`. This operator should be the left most pipe of any sequence of pipes. In addition to piping the dataset on the left onto the function on the right the result coming out of the right-hand pipeline is piped back to the original dataset overwriting its original contents in memory with the results from the pipeline.

A simple example is to replace a dataset with the same dataset after removing some observations (rows) and variables (columns). In the example below we `stats::filter()` and `dplyr::select()` the dataset to reduce it to just those observations and variables of interest. The result is piped backwards to the original dataset and thus overwrites the original data (which may or may not be a good thing). We do this on a temporary copy of the dataset and use the `base::dim()` function to report on the dimensions (rows and columns) of the resulting datasets.

```
# Copy the dataset into the variable ds.

ds <- weatherAUS

# Report on the dimensions of the dataset.

dim(ds)
## [1] 145463      24

# Demonstrate an assignment pipeline.

ds %<>%
  filter(Rainfall==0) %>%
  select(MinTemp, MaxTemp, Sunshine)

# Confirm that the dataset has changed.

dim(ds)
## [1] 91879      3
```

Once again this is so-called *syntactic sugar*. The commands are effectively translated by the computer into the following code.

```
# Functional form equivalent to the pipeline above.

ds <- select(filter(weatherAUS, Rainfall==0),
             MinTemp, MaxTemp, Sunshine)
```

21 Tee Pipe

Another useful operation is the tee-pipe `magrittr::%T>%` which causes the command that follows to be run as a side-pipe whilst piping the same data into that command and also into the next following command. The output from the first command is ignored.

Note that a new operator `magrittr::%T>%` from `magrittr` has been introduced here to process the transformed dataset in two ways—once with `dplyr::glimpse()` and then with `base::summary()`. The so-called *tee* pipe will take the flow of the dataset coming out of the previous junction which is the `dplyr::select()` and pipe it in two directions, like a T pipe. The first flow proceeds to the junction where `dplyr::glimpse()` is run. The second flow proceeds directly to the following junction, without going via the `dplyr::glimpse()`, where `base::summary()` is then run. Any further pipes would flow the output of the `base::summary()` command onward.

One use case is to generate a summary of an intermediate dataset whilst continuing to process the dataset.

```
weatherAUS %>%
  select(Rainfall, MinTemp, MaxTemp, Sunshine) %>%
  filter(Rainfall==0) %T>%
  head() %>%
  select(MinTemp, MaxTemp, Sunshine) %>%
  summary()

##      MinTemp        MaxTemp        Sunshine
##  Min.   :-8.70   Min.   :-2.10   Min.   : 0.00
##  1st Qu.: 7.30   1st Qu.:19.60   1st Qu.: 6.80
##  Median :12.00   Median :24.40   Median : 9.60
##  Mean   :12.02   Mean   :24.75   Mean   : 8.69
##  3rd Qu.:16.80   3rd Qu.:29.70   3rd Qu.:11.10
##  Max.   :33.90   Max.   :48.10   Max.   :14.50
##  NA's    :386     NA's    :400     NA's    :44629
```

A common use case is to `base::print()` the result of some data processing steps whilst continuing on to assign the dataset itself to a variable. We will often see the following example.

```
# Demonstrate usage of a tee-pipe.

no.rain <-
  weatherAUS %>%
  filter(Rainfall==0) %T>%
  print()

##           Date       Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 1 2008-12-02       Albury     7.4    25.1      0        NA       NA
## 2 2008-12-03       Albury    12.9    25.7      0        NA       NA
## 3 2008-12-04       Albury     9.2    28.0      0        NA       NA
## 4 2008-12-07       Albury    14.3    25.0      0        NA       NA
## 5 2008-12-08       Albury     7.7    26.7      0        NA       NA
....
```

22 Tee Pipe Alternatives

We could simply `base::print()` the value of the variable `no.rain` after the assignment or else enclose the assignment within round brackets but there is some elegance in including it explicitly as part of the single pipeline.

```
# Equivalent alternatives to the above tee-pipe example.

(
  weatherAUS %>%
    filter(Rainfall==0) ->
  no.rain
)

##           Date      Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 1 2008-12-02      Albury     7.4    25.1       0        NA        NA
## 2 2008-12-03      Albury    12.9    25.7       0        NA        NA
## 3 2008-12-04      Albury     9.2    28.0       0        NA        NA
## 4 2008-12-07      Albury    14.3    25.0       0        NA        NA
## 5 2008-12-08      Albury     7.7    26.7       0        NA        NA
.....
print(no.rain)

##           Date      Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 1 2008-12-02      Albury     7.4    25.1       0        NA        NA
## 2 2008-12-03      Albury    12.9    25.7       0        NA        NA
## 3 2008-12-04      Albury     9.2    28.0       0        NA        NA
## 4 2008-12-07      Albury    14.3    25.0       0        NA        NA
## 5 2008-12-08      Albury     7.7    26.7       0        NA        NA
.....
```

23 Tee Pipe Use Case: Load all CSV Files

Example of usage of the tee operator. Load all `.csv.gz` files in the data folder into a single data frame and include a message for each file loaded to monitor progress.

```
fpath <- "data"
files <- dir(fpath, "*.csv.gz")
ds <- data.frame()
for (i in seq_len(length(files)))
{
  ds <- fpath %>%
    file.path(files[i]) %T>%
    cat("\n") %>%
    readr::read_csv() %>%
    rbind(ds, .)
}
```

This can also be useful when combined with an on the fly function which is introduced with curly braces. We can use the global assignment operator to define variables that can be used later on in the pipeline. This example is a little contrived but illustrates its use. The on the fly function calculates the order of wind gust directions based on the maximum temperature for any day within each group defined by the wind direction. This is then used to order the wind directions and saving that order to a global variable `lvls`. That variable is then used to mutate the original data frame (note the use of the tee pipe) to reorder the levels of the `WindGustDir` variable. This is typically done within a pipeline that feeds into a plot where we want to reorder the levels so that there is some meaning to the order of the bars in a bar plot, for example.

```
# List the levels of a factor.

levels(weatherAUS$WindGustDir)
## [1] "N"     "NNE"   "NE"    "ENE"   "E"     "ESE"   "SE"    "SSE"   "S"     "SSW"   "SW"    "WSW"
## [13] "W"     "WNW"   "NW"    "NNW"

#
weatherAUS %>%
  filter(Rainfall>0) %T>%
  {
    lvls <-> select(., WindGustDir, MaxTemp) %>%
      group_by(WindGustDir) %>%
      summarise(MaxMaxTemp=max(MaxTemp)) %>%
      arrange(MaxMaxTemp) %>%
      pull(WindGustDir)
  } %>%
  mutate(WindGustDir=factor(WindGustDir, levels=lvls)) %$%
  levels(WindGustDir)

## [1] "ESE"   "SW"    "ENE"   "NNE"   "N"     "NE"    "E"     "SE"    "SSE"   "S"     "SSW"   "WSW"
## [13] "W"     "WNW"   "NW"    "NNW"
```

24 Exposition Pipe

We also see here yet another useful pipe operation `%%%`. This is the exposition pipe which is also useful for commands that do not take a dataset as their argument but instead some columns from the dataset. It evaluates the following function within the context of the dataset passed to it, so that the variables of the dataset become available without the need to quote them

```
weatherAUS %>%
  filter(Rainfall==0) %%%
  cor(MinTemp, MaxTemp)

## [1] NA
```

We might otherwise use with.

```
weatherAUS %>%
  filter(Rainfall==0) %>%
  with(cor(MinTemp, MaxTemp))

## [1] NA
```

25 Pipe and Plot

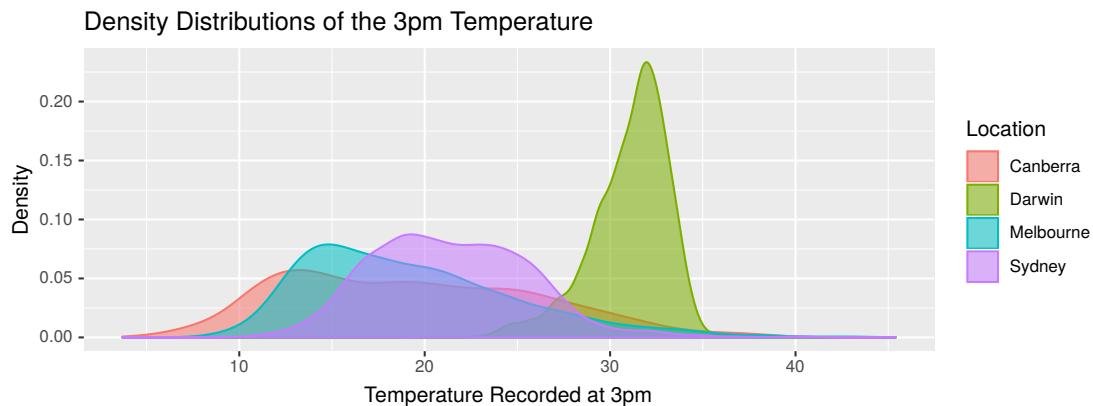
A common scenario for pipeline processing is to prepare data for plotting. Indeed, plotting itself has a pipeline type concept where we build a plot by adding layers to it.

Below the `rattle::weatherAUS` dataset is `stats::filter()`d for observations from four Australian cities. We `stats::filter()` observations that have missing values for the variable `Temp3pm` using an embedded pipeline. The embedded pipeline pipes the `Temp3pm` data through the `base::is.na()` function which tests if the value is missing. These results are then piped to `magrittr::not()` which inverts the true/false values so that we include those that are not missing.

A plot is generated using `ggplot2::ggplot()` into which we pipe the processed dataset. We add a geometric layer using `ggplot2::geom_density()` which consists of a density plot with transparency specified through the `alpha=` argument. We also add a title and label the axes using `ggplot2::labs()`.

```
cities <- c("Canberra", "Darwin", "Melbourne", "Sydney")

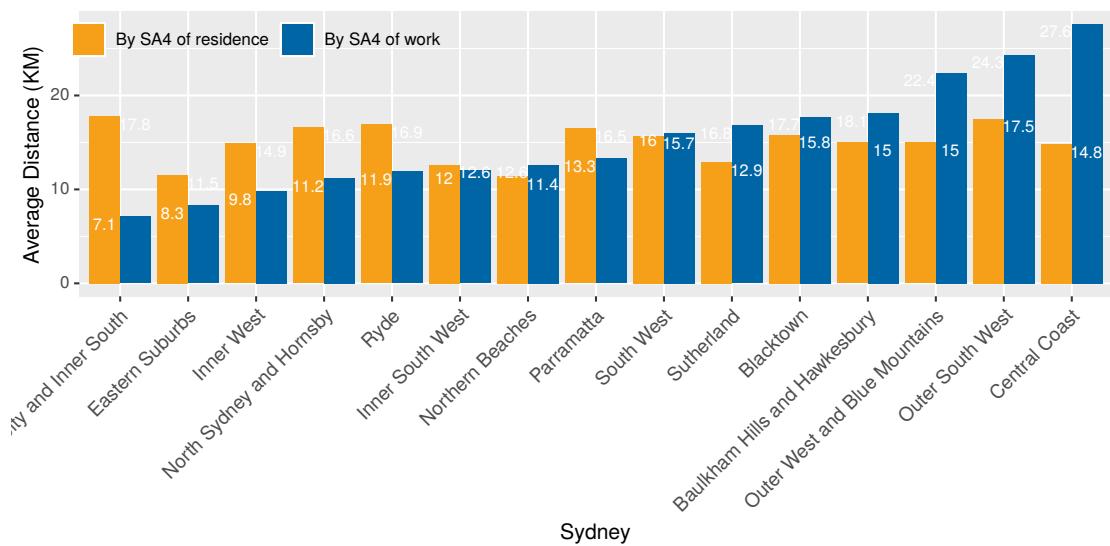
weatherAUS %>%
  filter(Location %in% cities) %>%
  filter(Temp3pm %>% is.na() %>% not()) %>%
  ggplot(aes(x=Temp3pm, colour=Location, fill=Location)) +
  geom_density(alpha=0.55) +
  labs(title = "Density Distributions of the 3pm Temperature",
       x     = "Temperature Recorded at 3pm",
       y     = "Density")
```



We now observe and tell a story from the plot. Our narrative will begin with the observation that Darwin has quite a different and warmer pattern of temperatures at 3pm than Canberra, Melbourne and Sydney.

26 Sophisticated Pipeline

```
avgdis %>%
  filter(GCCSA_CODE_2011=="1GSYD") %>%
  select(0_SA4name, 0_Avg_KM, D_Avg_KM) %>%
  mutate(0_SA4name=str_replace(0_SA4name, "^Sydney - ", ""))
{
  nms <- 0_SA4name[order(0_Avg_KM)] ; .
} %>%
mutate(0_SA4name=factor(0_SA4name, levels=nms)) %>%
gather("OD", "Distance", -0_SA4name) %>%
arrange(0_SA4name) %$%
{
  lbls <- data.frame(x = sort(c(seq(0.75, 14.75), seq(1.2, 15.2))+0.02),
                      y = (Distance-0.75),
                      lbl = round(Distance,1)) ; .
} %>%
ggplot(aes(x=0_SA4name, y=Distance)) +
  geom_bar(stat="identity", position="dodge", aes(fill=OD)) +
  geom_text(data=lbls, aes(x=x, y=y, label=lbl), colour="white",
            size=3, hjust=0.5) +
  theme(axis.text.x=element_text(angle=45, hjust=1, size=10)) +
  theme(legend.position=c(.15, .9)) +
  scale_fill_manual(values=c(rgb(246, 160, 26, max=255),
                            rgb( 0, 101, 164, max=255)),
                    labels=c("By SA4 of residence", "By SA4 of work")) +
  theme(legend.title=element_text(colour="transparent"),
        legend.background=element_rect(fill="transparent"),
        legend.direction="horizontal") +
  labs(x="Sydney", y="Average Distance (KM)")
```



27 Pipeline Identity Operator

A handy trick when building a pipeline is to use what is effectively the identity operator. An identity operator simply takes the data being communicated through the pipeline, and without changing it passes it on to the next operator in the pipeline. An effective identity operator is constructed with the syntax `{.}`. In R terms this is a compound statement containing just the period whereby the period represents the data. Effectively this is an operator that passes the data through without processing it—an identity operator.

Why is this useful? Whilst we are building our pipeline, one line at a time, we will be wanting to put a pipe at the end of each line, but of course we can not do so if there is no following operator. Also, whilst debugging a pipeline, we may want to execute only a part of it, and so the identity operator is handy there too.

As a typical scenario we might be in the process of building a pipeline as here and find that including the `purrr::%>%` at the end of the line of the `dplyr::select()` operation:

```
weatherAUS %>%
  select(Rainfall, MinTemp, MaxTemp, Sunshine) %>%
  {.}

##      Rainfall MinTemp MaxTemp Sunshine
## 1      0.6    13.4   22.9     NA
## 2      0.0     7.4   25.1     NA
## 3      0.0    12.9   25.7     NA
## 4      0.0     9.2   28.0     NA
## 5      1.0    17.5   32.3     NA
....
```

We then add the next operation into the pipeline without having to modify any of the code already present:

```
weatherAUS %>%
  select(Rainfall, MinTemp, MaxTemp, Sunshine) %>%
  summary() %>%
  {.}

##      Rainfall          MinTemp         MaxTemp        Sunshine
##  Min.   : 0.000   Min.   :-8.70   Min.   :-4.10   Min.   : 0.00
##  1st Qu.: 0.000   1st Qu.: 7.40   1st Qu.:17.90   1st Qu.: 4.90
##  Median : 0.000   Median :11.90   Median :22.50   Median : 8.40
##  Mean   : 2.298   Mean   :12.04   Mean   :23.14   Mean   : 7.61
##  3rd Qu.: 0.600   3rd Qu.:16.70   3rd Qu.:28.20   3rd Qu.:10.60
##  Max.   :371.000  Max.   :33.90   Max.   :48.10   Max.   :14.50
##  NA's   :3218     NA's   :1545   NA's   :1335   NA's   :70820
```

And so on. Whilst it appears quite a minor convenience, over time as we build more pipelines, this becomes quite a handy trick.

28 Getting Help

A key skill of any programmer, including those programming over data, is the ability to identify how to access the full power of our tools. The breadth and depth of the capabilities of R means that there is much to learn around both the basics of R programming and the multitude of packages that support the data scientist.

R provides extensive in-built documentation with many introductory manuals and resources accessible through the RStudio Help tab. These are a good adjunct to our very brief introduction here to R. Further we can use RStudio's search facility for documentation on any action and we will find manual pages that provide an understanding of the purpose, arguments, and return value of *functions*, *commands*, and *operators*. We can also ask for help using the `utils::?` operator as in:

```
# Ask for documentation on using the library command.  
?  
library
```

The `utils` package which provides this operator is another package that is attached by default into R. Thus we can drop its prefix as we did here when we run the command in the R Console.

RStudio provides a search box which can be used to find specific topics within the vast collection of R documentation. Now would be a good time to check the documentation for the `base::library()` command.

Beginning with this chapter we will also list at the conclusion of each chapter the functions, commands, and operators introduced within the chapter together with a brief synopsis of the purpose. The following section reviews all of the functions introduced so far.

29 Command Summary

This chapter has introduced, demonstrated and described the following R packages, functions, commands, operators, and datasets:

- `<- operator from base.`** The base R assignment operator. The expression on the right hand side is evaluated and the result is stored into the memory location identified on the left hand side. The left hand side is usually the name of a variable and has associated with it memory where it stores its values.
- `-> operator from base.`** The forward assignment operator. The expression on the left hand side is evaluated and the result is stored into the memory location identified on the right hand side. This assignment is not commonly used in R however it is a natural for use in a pipeline expression where the pipeline processes a dataset and we then store the resulting dataset into a variable.
- `? operator from utils.`** Display the documentation provided for a command. The documentation will be displayed in the lower right pane of the RStudio window and often includes links to other documentation.
- `%<-%` operator from magrittr.** The assignment pipe will pipe a dataset through to a function and the result is piped backwards to overwrite the original dataset.
- `%>% operator from magrittr.`** Pipe a dataset through a series of functions. Each function takes the dataset as its input, processes the dataset in some way, and then returns a processed dataset which is piped onto the next function, if there is another pipe.
- `%T>% operator from magrittr.`** The Tee-Pipe operator pipes a dataset through to the next function and also on to the following function. The results of the immediately following function are ignored and the pipeline then continues with the function following the immediately following function.
- `+` operator from base.** Determine the sum of two arguments. The arguments are usually numbers but we have also seen an example where we add layers to a plot using this operator. This is an infix binary operator in that it is used between the two arguments that are to be added.
- `dim()` function from base.** Returns a count of the number of rows and the number of columns in a dataset—the dimensions.
- `filter()` function from stats.** Choose only those observations from the input dataset that satisfy a specified criteria. The output dataset consists of all of the same variables as the input dataset but a subset of the observations from the input dataset.
- `geom_density()` function from ggplot2.** Add a density plot to a ggplot. A density plot shows the distribution of the different values of the variable over the observations in the dataset.
- `ggplot()` function from ggplot2.** Initiate a new plot and return an object that represents that (currently empty) plot. In order to be able to display the plot we need to add layers. We have illustrated the layering in this chapter where we added density geometries, a title and labels. The `gg` here refers to the *grammar of graphics* which is implemented by ggplot2.
- `glimpse()` command from dplyr.** Summarise a dataset. This is a convenient command for quickly obtaining a view of a potentially very large dataset.

install.packages() command from *utils*. Install a package into a library. Often the package will be downloaded from the Internet though it can also be a locally available file that was separately downloaded. The file will be an archive and this command unpacks the archive and stores the result in a folder that is referred to as a library.

is.na() function from *base*. Test if a value (or for that matter a vector) is “not available” (missing). Returns TRUE or FALSE appropriately or a vector of TRUE/FALSE if the argument is a vector.

labs() function from *ggplot2*. Change the text of the labels used for the x or y axis of a plot built using `ggplot2::ggplot()`.

library() command from *base*. Attach a package from an R library. The name of a package is provided as the argument to this command. That package is searched for within the available libraries on our local computer. When it is found the functions, commands, operators and datasets are made available to this R session.

not() function from *magrittr*. Invert the meaning of TRUE and FALSE, so that TRUE becomes FALSE and FALSE becomes TRUE. This is useful in a pipe as an alternative to the `base::!` operator.

print() command from *base*. Print a representation of the memory referred to by the named object supplied as the argument.

qplot() command from *ggplot2*. Quickly code a plot of the dataset. The generated plot is displayed as a side-effect of the command.

search() function from *base*. List the currently attached packages. This can be useful when we wish to see which packages have been attached to the current R session. The order in which the packages are listed is also the order in which R searches for a function. When the same function name is defined in two packages the first that is found is the one that will be used.

select() function from *dplyr*. Choose specified variables from a dataset. This function returns a smaller dataset than the input dataset consisting of all of the same observations as the input dataset but only of the selected variables.

sum() function from *base*. Determine the sum of two arguments (usually numbers). This is the traditional functional form of the `+` operator.

summary() function from *base*. From the input dataset this function generates a simple statistical summary of each variable.

weather dataset from *rattle*. The weather dataset contains daily observations of numerous weather variables from a weather station at the Canberra airport in Australia. It is a small dataset consisting of just 366 observations of 24 variables. It is used here to illustrate complex processing when size is not relevant for demonstrating the processing.

weatherAUS dataset from *rattle*. The weather dataset contains daily observations of numerous weather variables from nearly 50 weather stations across Australia. This is a moderately sized dataset consisting of 145,463 observations of 24 variables.

30 Exercises

Exercise 1 Exploring RStudio.

1. Install R and RStudio.
2. Start up RStudio and explore the menus, tool bars, and panes and tabs.
3. Explore the various options for learning about R.

Exercise 2 Getting help in RStudio.

1. Attach the `rattle` package in RStudio.
2. Obtain a list of the functions available from the `rattle` package.
3. Review the documentation for `rattle::normVarNames()` and then apply the function to the `base::names()` of the `rattle::weatherAUS` dataset from `rattle`.

Exercise 3 Interacting with RStudio.

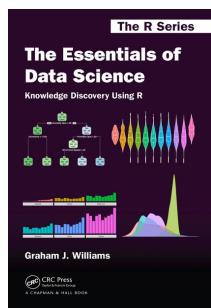
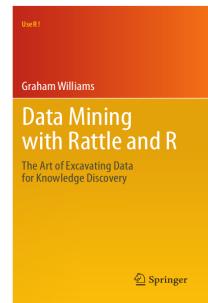
1. Create a new R script file and add a comment at the top with your name and today's date. Then save the file to disk.
2. By interacting with the R script file, attach the `rattle` package.
3. Plot the distribution of `WindSpeed9am` against `Humidity9am` from the `rattle::weatherAUS` dataset.
4. Filter the dataset by different locations and then review and compare the plots.
5. View and write a short report on the `rattle::weatherAUS` dataset. The report should describe the dataset, including a summary of the number of variables and observations. Describe each variable and summarise its distribution. Include some interesting plots.

Exercise 4 Using pipes.

1. Load the `rattle::weatherAUS` dataset into R.
2. Explore the rainfall for a variety of locations, generating different plots similar to the two examples provided in this chapter.
3. Write a short report on what you discovery including the R code you develop (using pipelines) and include sample plots that support your narrative.

31 Further Reading

The [Rattle](#) book ([Williams, 2011](#)), published by Springer, provides a comprehensive introduction to data mining and analytics using Rattle and R. It is available from [Amazon](#). Rattle provides a graphical user interface through which the user is able to load, explore, visualise, and transform data, and to build, evaluate, and export models. Through its Log tab it specifically aims to provide an R template which can be exported and serve as the starting point for further programming with data in R.



The [Essentials of Data Science](#) book ([Williams, 2017](#)), published by CRC Press, provides a comprehensive introduction to data science through programming with data using R. It is available from [Amazon](#). The book provides a template based approach to doing data science and knowledge discovery. Templates are provided for data wrangling and model building. These serve as generic starting points for programming with data, and are designed to require minimal effort to get started. Visit <https://essentials.togaware.com> for further guides and templates.

We list below other resources that augment the material we have presented in this chapter.

- The manual titled *An Introduction to R* is a good place to learn some of the basics of R. Follow the link provided within RStudio.
- <http://www.rstudio.org/> is the home of RStudio. The RStudio software can be downloaded from here as well as finding documentation and discussion forums. For a guide to using RStudio visit <https://support.rstudio.com>.
- Documentation within RStudio through the Help tab in the bottom right pane provides entry points to many of the standard resources for getting started with R and RStudio, as well as more advanced references.
- [Stack Overflow](#)⁶ is a great question and answer resource. There we find crowd sourced answers with crowd sourced ratings. It is always a good first port of call for any questions about using R.

⁶<https://stackoverflow.com>

32 References

- Bache SM, Wickham H (2014). *magrittr: A Forward-Pipe Operator for R*. R package version 1.5, URL <https://CRAN.R-project.org/package=magrittr>.
- R Core Team (2018a). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- R Core Team (2018b). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- Wickham H (2014). *Advanced R*. Chapman & Hall/CRC The R Series. Chapman & Hall.
- Wickham H, Chang W, Henry L, Pedersen TL, Takahashi K, Wilke C, Woo K (2018a). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.1.0, URL <https://CRAN.R-project.org/package=ggplot2>.
- Wickham H, François R, Henry L, Müller K (2018b). *dplyr: A Grammar of Data Manipulation*. R package version 0.7.8, URL <https://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.
- Williams GJ (2017). *The Essentials of Data Science: Knowledge discovery using R*. The R Series. CRC Press.
- Williams GJ (2018). *rattle: Graphical User Interface for Data Science in R*. R package version 5.2.5, URL <https://rattle.togaware.com/>.

This document, sourced from IntroRO.Rnw bitbucket revision 312, was processed by KnitR version 1.21 of 2018-12-10 23:00:03 UTC and took 26.6 seconds to process. It was generated by gjw on Ubuntu 18.04.1 LTS.

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 L^AT_EX 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.

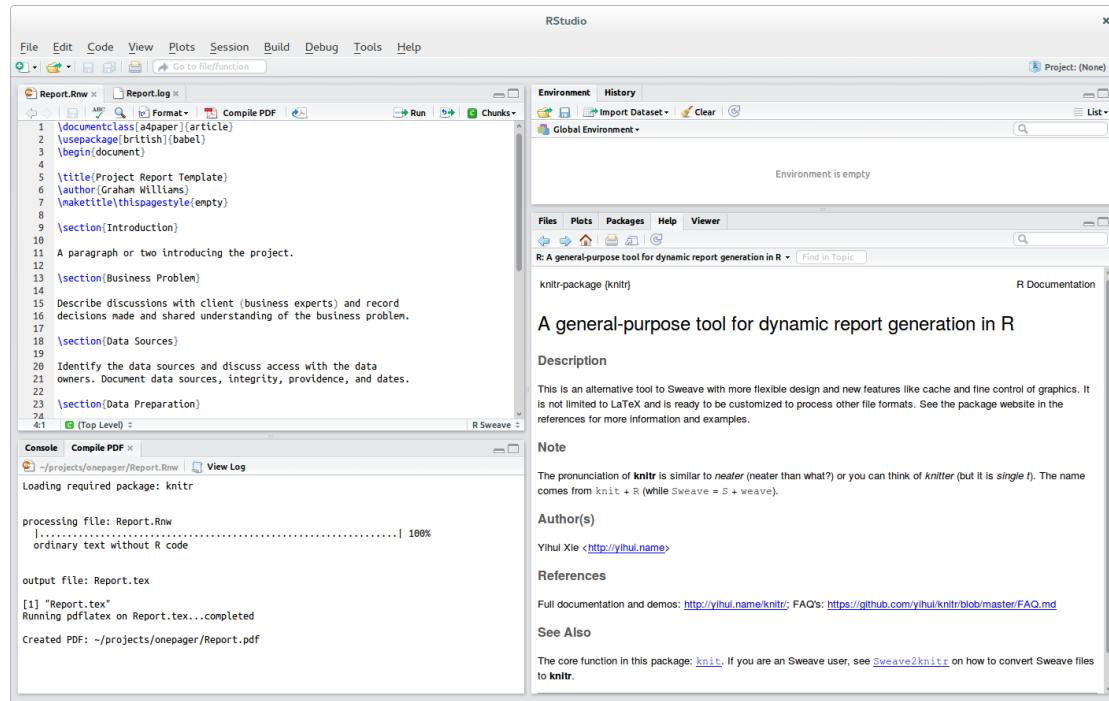
\section{Model Building}
Include all models built and parameters tried. Include R code and model evaluations.

\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 L^AT_EX 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., `\\hbox'), 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 editting so we will need to search for the string) and delete the offending L^AT_EX 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 L^AT_EX 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 L^AT_EX:

```
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 L^AT_EX.

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=", ")}`).

L^AT_EX 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, L^AT_EX 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 L^AT_EX. Look at `library(help=Hmisc)` for a list.

Exercise: Investigate the L^AT_EX 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	Williamtown	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
Williamtown	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
Williamtown	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 L^AT_EX 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
Williamtown	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		
Williamtown	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 L^AT_EX 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}` L^AT_EX 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		
Williamtown	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 L^AT_EX. 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:",
                           "$\\alpha$ $\\longrightarrow$ $\\wp$.",
                           "We also get a timestamp from R:",
                           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 L^AT_EX 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
@
```

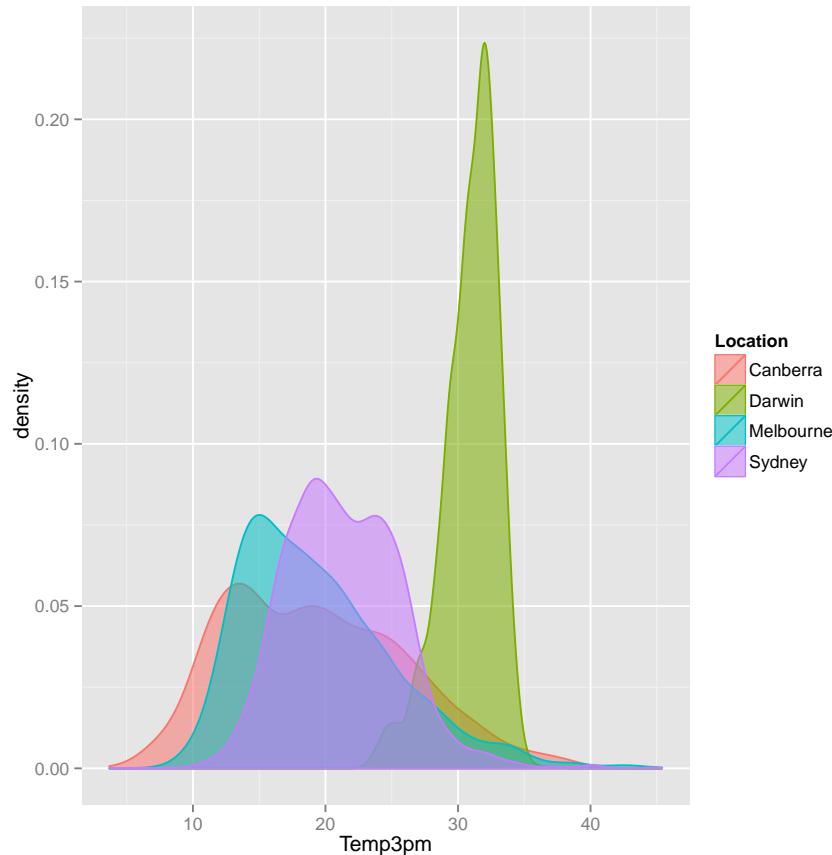
Notice the use of `eval=FALSE`, which allows the R code to be included in the final document, as it is here, but will not generate the plot to be included in the figure, yet. We leave that for the next page.

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.

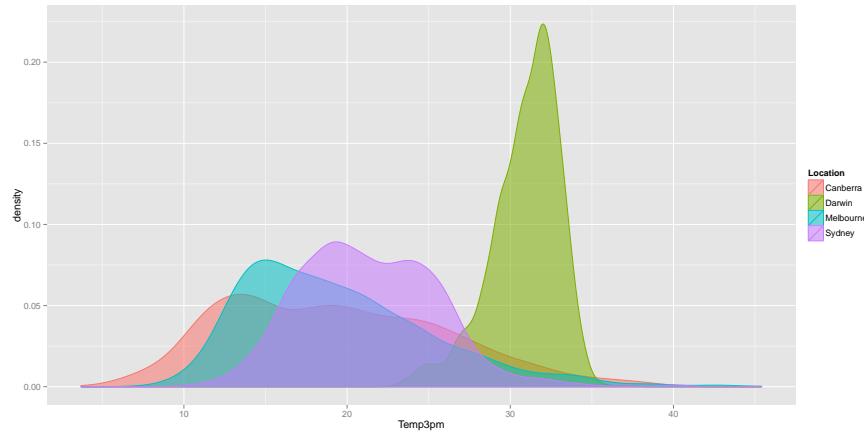
In fact, we do not actually need to rewrite the R code again in a second chunk, given the code has already been provided in the first chunk on the previous page. We use a feature of `knitr` where an empty chunk having the same name as a previous chunk is actually a reference to that previous chunk:

```
<<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



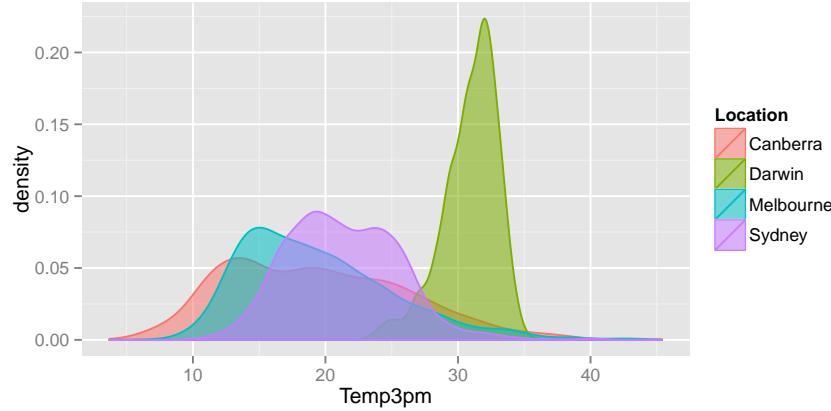
We can fine tune the size of the figure to suit the document and presentation. In this example we have asked R to widen the figure from 7 inches to 14 inches using `fig.width`. The code chunk is:

```
<<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

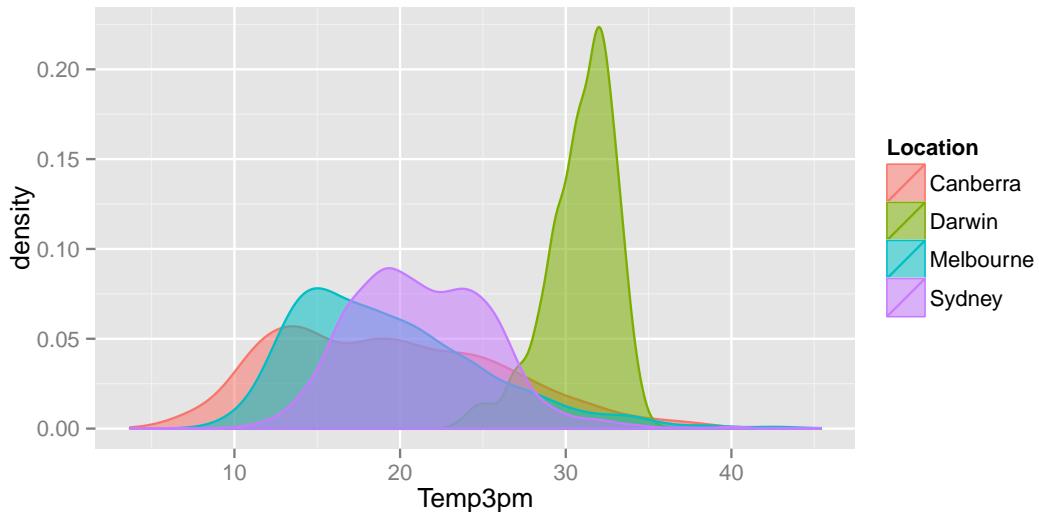


Often a bit of trial and error is required to get the dimensions right. Notice though that increasing the `fig.width=` as we did for the previous plot, and/or increasing the `fig.height=`, effectively also reduces the font size. Actually, the font size remains constant whilst the figure grows (or shrinks)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

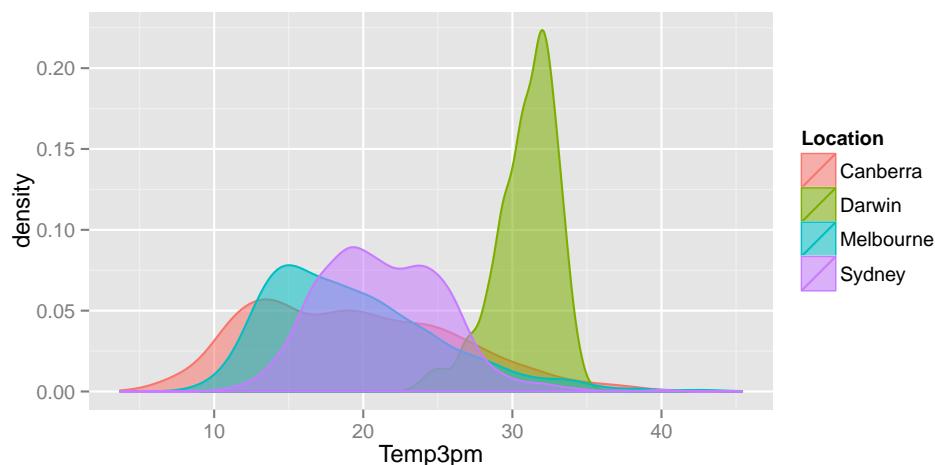


We can use the `out.width=` and `out.height=` to adjust how much space a figure takes up. The above figure is enlarged to fill the `textwidth` of the document using:

```
<<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

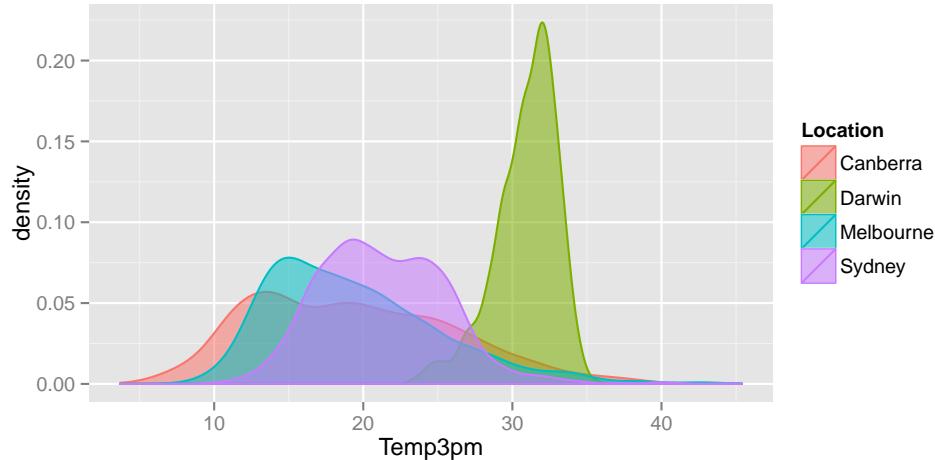


Figure 1: The 3pm temperature for four locations.

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:

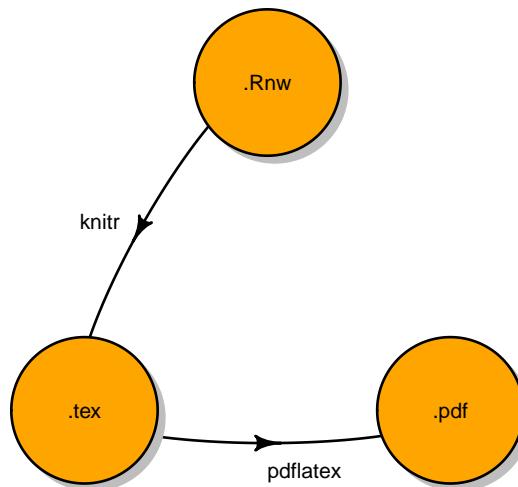
```
\usepackage{animate}
```

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:

```
\citet{R-rattle}
```

This will produce a citation like (Williams, 2014). The “p” in `\citet` 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$set(out.truncate=80)
hook_output <- knit_hooks$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=NULL")
## [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=NULL")
## [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=NULL")
## [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$set(out.lines=4)
hook_output <- knit_hooks$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
....
```

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.
```

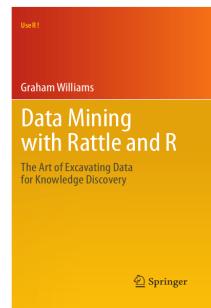
New options defined in this Chapter and used in this book:

```
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.
- L^AT_EX 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.

Data Science with R

The Basics of R

Graham.Williams@togaware.com

16th August 2014

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

In this chapter we introduce some of the basic commands we often use in interacting with R as a Data Scientist. We don't aim to be comprehensive but rather to provide basic familiarity.

The required packages for this module include:

```
library(rattle)  
library(scales)
```

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 Data Types: Numeric

2 Data Types: Integer

3 Data Types: Complex

4 Data Types: Logical

5 Data Types: Character

The class of objects that we generally call strings in character.

```
class("abc")
## [1] "character"
```

We can convert other data types to class character using `as.character()`:

```
n <- 42.134
class(n)

## [1] "numeric"
as.character(n)

## [1] "42.134"
```

Concatenate strings:

```
paste("abc", "def")
## [1] "abc def"

paste("abc", "def", sep="")
## [1] "abcdef"

paste0("abc", "def")
## [1] "abcdef"
```

String formatting:

```
s <- "abc"
sprintf("The length of %s is %d.", s, length(s))
## [1] "The length of abc is 1."
```

Substrings:

```
s <- "Vulpes celeris et fluva salit super ignavum canem."
substr(s, start=8, stop=12)

## [1] "celer"
```

Substitute:

```
s <- "Vulpes celeris et fluva salit super ignavum canem."
sub("Vulpes", "The Fox", s)

## [1] "The Fox celeris et fluva salit super ignavum canem."
```

6 Data Structure: Vector

7 Data Structure: Matrix

8 Data Structure: Data Frame

8.1 Sort a Data Frame

From Stack Overflow.

```
sort.data.frame <- function(x, decreasing=FALSE, by=1, ... )
{
  f <- function(...) order(..., decreasing=decreasing)
  i <- do.call(f, x[by])
  x[i,,drop=FALSE]
}
sort(weather, by="MinTemp")

##           Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 293 2008-08-19 Canberra    -5.3    13.1      0.0        2.2     7.9
## 298 2008-08-24 Canberra    -3.7    14.4      0.0        2.6    10.4
## 314 2008-09-09 Canberra    -3.7    14.7      0.0        3.4    10.9
....
sort(weather, by="MaxTemp")

##           Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 284 2008-08-10 Canberra    -3.5     7.6      0.4        2.4     4.7
## 254 2008-07-11 Canberra     2.9     8.4      1.6        1.4     7.7
## 253 2008-07-10 Canberra     1.8     8.7      0.0        1.8     1.2
....
sort(weather, by=c("MinTemp", "MaxTemp"))

##           Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 293 2008-08-19 Canberra    -5.3    13.1      0.0        2.2     7.9
## 298 2008-08-24 Canberra    -3.7    14.4      0.0        2.6    10.4
## 314 2008-09-09 Canberra    -3.7    14.7      0.0        3.4    10.9
....
sort(weather, decreasing=TRUE, by=c("MinTemp", "MaxTemp"))

##           Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 73 2008-01-12 Canberra    20.9    35.7      0.0       13.8     6.9
## 52 2007-12-22 Canberra   19.9    22.0     11.0       4.4     5.9
## 96 2008-02-04 Canberra   18.2    22.6      1.8       8.0     0.0
....
sort(weather, by=3:4)

##           Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 293 2008-08-19 Canberra    -5.3    13.1      0.0        2.2     7.9
## 298 2008-08-24 Canberra    -3.7    14.4      0.0        2.6    10.4
## 314 2008-09-09 Canberra    -3.7    14.7      0.0        3.4    10.9
....
```

9 Data Structure: List

10 Presentation: Display Large Numbers

By default, large numbers are written in scientific notation if they are too large (have too many digits).

```
22000 * 2500  
## [1] 5.5e+07
```

We can increase the tolerance for large numbers using the `scipen=` argument of `options()`. Many readers find the scientific notation difficult and time consuming to read. So let's write out the number in full:

```
old.opts <- options(scipen=10)  
22000 * 2500  
## [1] 55000000
```

Such numbers remain hard to read though. Is that 5.5 million, 55 million, or 550 million. We have to look carefully to decide. Commas **always** assist, and the simplest way to get commas into the output we produce is to use `comma()` from `scales` (Wickham, 2014).

```
comma(22000 * 2500)  
## [1] "55,000,000"
```

We can restore the original options, if we saved them as above:

```
options(old.opts)  
22000 * 2500  
## [1] 5.5e+07
```

11 Data Frames: Creating

Create empty data frames:

```
data.frame(a=integer(), b=numeric())
## [1] a b
## <0 rows> (or 0-length row.names)

data.frame(c=character(), d=factor(levels=c("y","n")), stringsAsFactors=FALSE)
## [1] c d
## <0 rows> (or 0-length row.names)
```

12 Data Frames: Indexing

13 Data Frames: Subsets Using `subset()`

The function `subset()` was introduced by Peter Dalgaard as a convenience for subsetting data frames rather than using indexing directly. Many now argue that it is the most natural way of subsetting data frames. Brian Ripley noted that he thinks this convenience function is a mistake (R-Devel mailing list 21 October 2013) as people start to make use of them in functions and packages which can lead to issues. As the help page notes, this should only be used as an interactive convenience function, not for programming.

14 Data Frames: Saving Data

A compressed binary data file can be saved, containing one or more R objects. Here we save two R objects:

```
wfname <- "weather.RData"  
save(weather, weatherAUS, file=wfname)
```

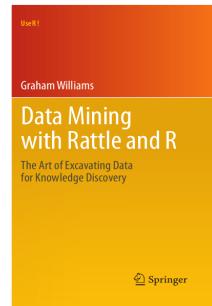
15 Further Reading and Acknowledgements

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:

- http://www.nzdl.org/Books/Books/realistic-books-svn/books/R_ByExample/
- Chi Yau's [R Tutorial](#) is a good place to start with R too.



16 References

- 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/>.
- Wickham H (2014). *scales: Scale functions for graphics*. R package version 0.2.4, URL <http://CRAN.R-project.org/package=scales>.
- 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.

This document, sourced from BasicRO.Rnw revision 484, was processed by Knitr version 1.6 of 2014-05-24 and took 3.5 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 2014-08-16 17:45:28.

One Page R Data Science Data Wrangling

Graham.Williams@togaware.com

8th September 2018

Visit <https://essentials.togaware.com/onepagers> for more Essentials.

20180721

The *business understanding* phase of a data science project aims to understand the business problem and to then liaise with the business data technicians to identify the data available. This is followed by the *data understanding* phase where we work with the business data technicians to access and ingest the data into R. We are then in a position to initiate our journey of discovery driven by the data. By *living and breathing* the data in the context of the business problem we gain our bearings and feed our intuitions as we journey.

In this chapter we present the common series of steps for the data phase of data science. As we progress through the chapter we build a *template* designed to be reused for other journeys. As we foreshadowed in Chapter 1 rather than delving into the intricacies of the R language we immerse ourselves into using R to achieve our outcomes, learning more about R as we proceed.

The template consists of programming code that can be reused with little or no modification on a new dataset. The intention is that to get started with a new dataset only a few lines at the top of the template need to be modified. No or only minimal change is then required for the remainder of the code. In many respects the concept of a template is a stepping stone toward writing functions in R.

Through this guide new R commands will be introduced. The reader is encouraged to review the command's documentation and understand what the command does. Help is obtained using the ? command as in:

```
?read.csv
```

Documentation on a particular package can be obtained using the `help=` option of `library()`:

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

This chapter is intended to be hands on. To learn effectively you are encouraged to run R (e.g., RStudio or Emacs with ESS mode) and to replicate the commands. Check that output is the same and that you understand how it is generated. Try some variations. Explore.

Copyright © 2000-2018 Graham Williams. This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#) allowing this work to be copied, distributed, or adapted, with attribution and provided under the same license.



1 Packages Used

Packages used in this chapter include `dplyr` (Wickham *et al.*, 2018b), `FSelector` (Romanski and Kotthoff, 2018), `ggplot2` (Wickham *et al.*, 2018a), `glue` (Hester, 2018), `lubridate` (Spinu *et al.*, 2018), `randomForest` (Breiman *et al.*, 2018), `readr` (Wickham *et al.*, 2017), `stringi` (Gagolewski *et al.*, 2018), `stringr` (Wickham, 2018), `tidyR` (Wickham and Henry, 2018), `magrittr` (Bache and Wickham, 2014), and `rattle` (Williams, 2018).

```
# Load required packages from local library into R session.

library(rattle)          # normVarNames().
library(readr)            # Efficient reading of CSV data.
library(dplyr)             # Data wrangling, glimpse() andtbl_df().
library(tidyR)             # Prepare a tidy dataset, gather().
library(magrittr)           # Pipes %>% and %T>% and equals().
library(glue)                # Format strings.
library(lubridate)           # Dates and time.
library(FSelector)           # Feature selection, information.gain().
library(stringi)              # String concat operator %s+%.
library(stringr)              # String operations.
library(randomForest)         # Impute missing values with na.roughfix().
library(ggplot2)                # Visualise data.
```

2 Data Source

To begin the data phase of a project we will identify a data source. For our purposes we use the simplest of sources—a text-based **CSV** (comma separated value) file as a typical data source format.

The **weatherAUS** dataset from **rattle** will be used. A binary formatted R dataset is provided by the package but the CSV file for the same dataset is available at <https://rattle.togaware.com/weatherAUS.csv>.

We first identify and record the location of the CSV file to analyse. R is capable of ingesting data directly from the Internet and so we will illustrate how to do so here. The location of the file (the so-called URL or universal resource location) will be saved as a string in a variable called **dspath**—the path to the dataset. The following assignment command does this for us. Simply type this into your R script file within RStudio. The command is then executed in RStudio by clicking the Run button whilst the cursor is situated on the line within the script file.

```
# Note the source location of a dataset to ingest into R.

dspath <- "http://rattle.togaware.com/weatherAUS.csv"
```

The assignment operator **<-** will store the value on the right hand side (which is a string enclosed within quotation marks) into the computer's memory and we can later refer to it as the R variable **dspath**—we retrieve the string simply by reference to the variable **dspath**.

By typing the name of the variable (**dspath**) in the R Console at the **>** prompt R will respond with the value stored in the variable:

```
dspath
## [1] "http://rattle.togaware.com/weatherAUS.csv"
```

If not connected to the Internet we can read the data directly from a local copy of the **CSV** file. The **rattle** package (once the package has been installed) provides a smaller sample **weather.csv**. The location of the CSV file within **rattle** is determined using **base::system.file()**. Knowing that CSV files are located within the **csv** sub-directory of the **rattle** package we generate the string that identifies the file system path to **weather.csv**.

```
dspath <- system.file("csv", "weather.csv", package="rattle") %T>% print()
## [1] "/usr/lib/R/site-library/rattle/csv/weather.csv"
```

This is the path to the CSV file on my file system. Your path may well be different depending on where your system installed the **rattle** package.

Note that this is a considerably smaller subset of the full **weatherAUS** dataset and ingesting this rather than the full dataset will lead to different results to those presented here.

If you have separately downloaded **weatherAUS.csv** then you can identify its location. Here we identify that the downloaded file is located in the current working directory.

```
dspath <- "./weatherAUS.csv"
```

3 Data Ingestion

Having identified the source of the dataset we can ingest the dataset into the memory of the computer using the function `readr::read_csv()`. This function returns an enhanced ***data frame***. A data frame is the basic data structure used to store a dataset within R and the enhanced data frame from the tidyverse adds functionality that improves our interactions with the data frame.

We set up a reference to the data frame's location in the computer's memory by assigning the result of the call to the function `readr::read_csv()` to the R variable `weather`.

```
# Ingest the dataset.

weather <- read_csv(file=dspath)

## Parsed with column specification:
## cols(
##   .default = col_character(),
##   Date = col_date(format = ""),
##   MinTemp = col_double(),
##   MaxTemp = col_double(),
##   Rainfall = col_double(),
##   WindGustSpeed = col_integer(),
##   WindSpeed9am = col_integer(),
##   WindSpeed3pm = col_integer(),
##   Humidity9am = col_integer(),
##   Humidity3pm = col_integer(),
##   Pressure9am = col_double(),
##   Pressure3pm = col_double(),
##   Cloud9am = col_integer(),
##   Cloud3pm = col_integer(),
##   Temp9am = col_double(),
##   Temp3pm = col_double(),
##   RISK_MM = col_double()
## )

## See spec(...) for full column specifications.
```

As a side effect of calling the function `readr::read_csv()` helpful messages are displayed that identify the data types for each of the variables found in the ingested dataset. We should review these to ensure they match our expectations. If they don't, there are optional arguments to `readr::read_csv()` to inform it otherwise.

Note that the `rattle` also provides a smaller `rattle::weather` dataset as an R dataset, also named `weather`. Simply by attaching the `rattle` package from the library a variable called `weather` becomes available. Running the above command will replace the dataset provided by `rattle`. Having done so we can still access the `weather` dataset provided by `rattle` using the package prefix as in `rattle::weather`.

4 Data Frame

A data frame can essentially be thought of as a rectangular table of data consisting of rows (*observations*) and columns (*variables*). We can view the structure of such a table as it stores the weatherAUS dataset. Here we choose to display the first 10 observations of the first 6 variables.

```
# Display the table structure of the ingested dataset.

weather[1:10,1:6] %>% print.data.frame()

##           Date Location MinTemp MaxTemp Rainfall Evaporation
## 1 2008-12-01 Albury    13.4    22.9      0.6        <NA>
## 2 2008-12-02 Albury     7.4    25.1      0.0        <NA>
## 3 2008-12-03 Albury    12.9    25.7      0.0        <NA>
## 4 2008-12-04 Albury     9.2    28.0      0.0        <NA>
## 5 2008-12-05 Albury    17.5    32.3      1.0        <NA>
## 6 2008-12-06 Albury    14.6    29.7      0.2        <NA>
## 7 2008-12-07 Albury    14.3    25.0      0.0        <NA>
## 8 2008-12-08 Albury     7.7    26.7      0.0        <NA>
## 9 2008-12-09 Albury     9.7    31.9      0.0        <NA>
## 10 2008-12-10 Albury    13.1    30.1      1.4        <NA>
```

By choosing to pipe the data through `base::print.data.frame()` we request a raw display of the actual data frame.

Next we select 10 random observations, using `dplyr::sample_n()`, of 5 random variables, orchestrated using `dplyr::select()` of a `dplyr::sample()`, with the support of `base::ncol()`.

```
# Display a random selection of observations and variables.

weather %>%
  sample_n(10) %>%
  select(sample(1:ncol(weather), 5)) %>%
  print.data.frame()

##           WindDir3pm Pressure3pm Temp3pm Evaporation WindDir9am
## 1             NW       1007.3     22.0        <NA>       NNW
## 2             ESE      1023.6     16.6          6         SE
## 3             ENE      1011.0     28.8        <NA>         N
## 4               N      1018.0     23.3        4.8        ENE
## 5               NE      1005.6     30.1        6.2         SE
## 6             NNE        NA      16.1        1.2         N
## 7               W      1019.4     15.6        3.2       WNW
## 8             NNE      1028.3     22.9        <NA>        NE
## 9             ENE      1014.7     27.3        8.2       SSE
## 10              N      1020.8     10.7        <NA>       NNE
```

Observe that this is a tabular form (i.e., it has rows and columns) and that we will generally be working with datasets in such a tabular form.

5 The Shape of the Dataset

Once the dataset is loaded we want to get a basic idea of what it looks like—its shape. Being an extended data frame (what we call a tibble), we can display the data as a tibble simply by printing the data referred to by the variable name.

```
# Print the dataset in a human useful way.

weather

## # A tibble: 145,463 x 24
##   Date      Location MinTemp MaxTemp Rainfall Evaporation Sunshine
##   <date>    <chr>     <dbl>    <dbl>    <dbl> <chr>       <chr>
## 1 2008-12-01 Albury     13.4     22.9     0.6 <NA>       <NA>
## 2 2008-12-02 Albury      7.4     25.1      0 <NA>       <NA>
## 3 2008-12-03 Albury     12.9     25.7      0 <NA>       <NA>
## 4 2008-12-04 Albury      9.2      28       0 <NA>       <NA>
## 5 2008-12-05 Albury     17.5     32.3      1 <NA>       <NA>
## 6 2008-12-06 Albury     14.6     29.7      0.2 <NA>       <NA>
## 7 2008-12-07 Albury     14.3      25       0 <NA>       <NA>
## 8 2008-12-08 Albury      7.7     26.7      0 <NA>       <NA>
## 9 2008-12-09 Albury      9.7     31.9      0 <NA>       <NA>
## 10 2008-12-10 Albury    13.1     30.1      1.4 <NA>       <NA>
## # ... with 145,453 more rows, and 17 more variables: WindGustDir <chr>,
## #   WindGustSpeed <int>, WindDir9am <chr>, WindDir3pm <chr>,
## #   WindSpeed9am <int>, WindSpeed3pm <int>, Humidity9am <int>,
## #   Humidity3pm <int>, Pressure9am <dbl>, Pressure3pm <dbl>, Cloud9am <int>,
## #   Cloud3pm <int>, Temp9am <dbl>, Temp3pm <dbl>, RainToday <chr>,
## #   RISK_MM <dbl>, RainTomorrow <chr>
```

We observe that dataset consists of 145,463 observations of 24 variables. The enhanced nature of the data frame that representing it as a tibble brings to us is that the printout is more informative. The first few observations are shown with a subset of the variables followed by a list of all of the other variables.

6 A Glimpse of the Dataset

A useful alternative to gain some insight into the dataset is through `tibble::glimpse()`.

20180721

```
# A quick view of the contents of the dataset.

glimpse(weather)

## # Observations: 145,463
## # Variables: 24
## # $ Date      <date> 2008-12-01, 2008-12-02, 2008-12-03, 2008-12-04, 200...
## # $ Location   <chr> "Albury", "Albury", "Albury", "Albury", "Albu...
## # $ MinTemp    <dbl> 13.4, 7.4, 12.9, 9.2, 17.5, 14.6, 14.3, 7.7, 9.7, 13...
## # $ MaxTemp    <dbl> 22.9, 25.1, 25.7, 28.0, 32.3, 29.7, 25.0, 26.7, 31.9...
## # $ Rainfall   <dbl> 0.6, 0.0, 0.0, 0.0, 1.0, 0.2, 0.0, 0.0, 0.0, 1.4, 0....
## # $ Evaporation <chr> NA, ...
## # $ Sunshine    <chr> NA, ...
## # $ WindGustDir <chr> "W", "WNW", "WSW", "NE", "W", "WNW", "W", "W", "NNW"...
## # $ WindGustSpeed <int> 44, 44, 46, 24, 41, 56, 50, 35, 80, 28, 30, 31, 61, ...
## # $ WindDir9am   <chr> "W", "NNW", "W", "SE", "ENE", "W", "SW", "SSE", "SE"...
## # $ WindDir3pm   <chr> "WNW", "WSW", "WSW", "E", "NW", "W", "W", "W", "NW", ...
## # $ WindSpeed9am <int> 20, 4, 19, 11, 7, 19, 20, 6, 7, 15, 17, 15, 28, 24, ...
## # $ WindSpeed3pm <int> 24, 22, 26, 9, 20, 24, 24, 17, 28, 11, 6, 13, 28, 20...
## # $ Humidity9am  <int> 71, 44, 38, 45, 82, 55, 49, 48, 42, 58, 48, 89, 76, ...
## # $ Humidity3pm  <int> 22, 25, 30, 16, 33, 23, 19, 19, 9, 27, 22, 91, 93, 4...
## # $ Pressure9am  <dbl> 1007.7, 1010.6, 1007.6, 1017.6, 1010.8, 1009.2, 1009...
## # $ Pressure3pm  <dbl> 1007.1, 1007.8, 1008.7, 1012.8, 1006.0, 1005.4, 1008...
## # $ Cloud9am     <int> 8, NA, NA, NA, 7, NA, 1, NA, NA, NA, 8, 8, NA, N...
## # $ Cloud3pm     <int> NA, NA, 2, NA, 8, NA, NA, NA, NA, 8, 8, 7, N...
## # $ Temp9am      <dbl> 16.9, 17.2, 21.0, 18.1, 17.8, 20.6, 18.1, 16.3, 18.3...
## # $ Temp3pm      <dbl> 21.8, 24.3, 23.2, 26.5, 29.7, 28.9, 24.6, 25.5, 30.2...
## # $ RainToday    <chr> "No", "No", "No", "No", "No", "No", "No", "No"...
## # $ RISK_MM       <dbl> 0.0, 0.0, 0.0, 1.0, 0.2, 0.0, 0.0, 0.0, 1.4, 0.0, 2....
## # $ RainTomorrow <chr> "No", "No", "No", "No", "No", "No", "No", "No", "Yes"...
```

Again we receive a printed summary of the dataset, reporting on the number of observations and variables, but now the table is effectively rotated so that all variables can be listed along with their data type and a selection of their values for the first few observations.

7 Introducing Template Variables

A reference to the original dataset can be created using a template (or generic) variable. The new variable will be called `ds` (short for dataset).

```
# Take a copy of the dataset into a generic variable.

ds <- weather
```

Both `ds` and `weather` will now reference the same dataset within the computer's memory. As we modify `ds` those modifications will only affect the data referenced by `ds`. Effectively, an extra copy of the dataset in the computer's memory will start to grow as we change the data from its original form. R avoids making copies of datasets unnecessarily and so a simple assignment does not create a new copy. As modifications are made to one or the other copy of a dataset then extra memory will be used to store the columns that differ between the datasets.

From here on we no longer refer to the dataset as `weather` but as `ds`. This allows the following analyses and processing to be rather generic—turning the R code into a *template* and so requiring only minor modification when used with a different dataset assigned into `ds`.

Often we will find that we can simply load a different dataset into memory, store it as `ds` and the remaining steps of our analyses and processing will essentially work unchanged.

The first few steps of our template are then captured as creating the reference to the dataset and presenting our initial view of the dataset.

```
# Prepare for a templated analysis and processing.

dsname <- "weather"
ds     <- get(dsname)
glimpse(ds)

## Observations: 145,463
## Variables: 24
## $ Date          <date> 2008-12-01, 2008-12-02, 2008-12-03, 2008-12-04, 200...
## $ Location       <chr> "Albury", "Albury", "Albury", "Albury", "A...
## $ MinTemp        <dbl> 13.4, 7.4, 12.9, 9.2, 17.5, 14.6, 14.3, 7.7, 9.7, 13...
## $ MaxTemp        <dbl> 22.9, 25.1, 25.7, 28.0, 32.3, 29.7, 25.0, 26.7, 31.9...
....
```

We are a little tricky here in recording the dataset name in the variable `dsname` and then using the function `base::get()` to make a copy of the dataset reference and link it to the generic variable `ds`. We could simply assign the data to `ds` directly as we saw above. Either way the generic variable `ds` refers to the same dataset. The use of `base::get()` allows us to be a little more generic in our template.

The use of generic variables within a template for the tasks we perform on each new dataset will have obvious advantages but we need to be careful. A disadvantage is that we may be working with several datasets and accidentally overwrite previously processed datasets referenced using the same generic variable (`ds`). The processing of the dataset might take some time and so accidentally losing it is not an attractive proposition. Care needs to be taken to avoid this.

8 Locating Datasets in Memory

We can see that `ds` and `weather` reference the same dataset in memory using `dplyr::location()` and `dplyr::changes()`.

```
location(weather)

## <0x5622da6a4140>
## Variables:
## * Date:          <0x5622e05a0da0>
## * Location:      <0x5622e06bcf90>
## * MinTemp:       <0x5622e07d9180>
## * MaxTemp:       <0x5622e08f5370>
.....
location(ds)

## <0x5622da6a4140>
## Variables:
## * Date:          <0x5622e05a0da0>
## * Location:      <0x5622e06bcf90>
## * MinTemp:       <0x5622e07d9180>
## * MaxTemp:       <0x5622e08f5370>
.....
changes(weather, ds)

## <identical>
```

This gets rather technical (or geeky), but the strings of digits and characters within the angle brackets are actual memory addresses—that is, they are pointers to a direct location in our computer’s memory. The `0x` at the beginning of each identifies that a hexadecimal scheme is used, thus we see digits 0 to 9 and then the letters a, b, c, d, e, and f being used. That is, 16 digits.

The thing to note is that the addresses recorded for `weather` and `ds`, including the addresses where we find the actual variables (columns) within each dataset, are identical. This is confirmed by the call to `dplyr::changes()`.

9 Changing Datasets in Memory

Let's make a change to the weather dataset by simply changing a single cell, changing the value of `MinTemp` (the third variable) for the first observation to 5.

```
weather[1,3] <- 5
```

Notice the divergence of the two datasets. They still share a lot in common, and hence only one copy of that data, but where they diverge, they now use different memory locations.

```
location(weather)
## <0x5622dc266410>
## Variables:
## * Date:          <0x5622e05a0da0>
## * Location:      <0x5622e06bcf90>
## * MinTemp:        <0x5622e54a1d00>
## * MaxTemp:        <0x5622e08f5370>
.....
location(ds)
## <0x5622da6a4140>
## Variables:
## * Date:          <0x5622e05a0da0>
## * Location:      <0x5622e06bcf90>
## * MinTemp:        <0x5622e07d9180>
## * MaxTemp:        <0x5622e08f5370>
.....
```

Using `dplyr::changes()` makes clear the changes.

```
changes(weather, ds)
## Changed variables:
##           old          new
## MinTemp   0x5622e54a1d00 0x5622e07d9180
##
## Changed attributes:
##           old          new
## row.names 0x5622e3ceb800 0x5622e3cf0760
```

That's an interesting aside, but we now get back to our actual data analysis and processing.

10 Reviewing Variable Names

The names of the variables within the dataset as supplied to us may not be in any particular form and may use different conventions. For example, they may use a mix of upper and lower case letters (`TempToday9AM`) or be very long (`Temperature_Recorded_Today_9am`) or use sequential numbers to identify each variable (`V004` or `V004_rainToday`) or use codes (`XVn34_rain`) or any number of other conventions. Often we prefer to simplify the variable names to ease our processing and thinking and to enforce a standard and consistent naming convention for ourselves.

We use `base::names()` to list the names of the variables within a dataset.

```
# Review the variables to consider normalising their names.

names(ds)

## [1] "Date"          "Location"       "MinTemp"        "MaxTemp"
## [5] "Rainfall"       "Evaporation"    "Sunshine"       "WindGustDir"
## [9] "WindGustSpeed" "WindDir9am"     "WindDir3pm"     "WindSpeed9am"
## [13] "WindSpeed3pm"  "Humidity9am"   "Humidity3pm"   "Pressure9am"
## [17] "Pressure3pm"   "Cloud9am"      "Cloud3pm"      "Temp9am"
## [21] "Temp3pm"       "RainToday"     "RISK_MM"       "RainTomorrow"
....
```

Notice that the names here use a scheme whereby the initial letter is capitalised and each word within the variable name is also capitalised. That's a reasonable naming scheme and is preferred by some.

11 Normalizing Variable Names

A convenient convention that I personally prefer is to map all variable names to lowercase. R is case sensitive so that doing this will result in different variable names as far as R is concerned. Such (so called) normalisation is useful when different upper/lower case conventions are intermixed inconsistently in names like `Incm_tax_PyB1`. Remembering how to capitalize when interactively exploring the data with thousands of such variables can be quite a cognitive load for us. Yet we often see such variable names arising in practise especially when we import data from databases which are often case insensitive.

We can use `rattle::normVarNames()` to make a reasonable attempt of converting variables from a dataset into a preferred standard form. The actual form follows a style that is presented in Appendix 6. The example below shows the transformation into a normalised form. We make extensive use of the function `base::names()` to work with the variable names.

```
# Normalise the variable names.

names(ds) %<>% normVarNames() %T>% print()

## [1] "date"           "location"        "min_temp"        "max_temp"
## [5] "rainfall"       "evaporation"     "sunshine"        "wind_gust_dir"
## [9] "wind_gust_speed" "wind_dir_9am"    "wind_dir_3pm"    "wind_speed_9am"
## [13] "wind_speed_3pm"  "humidity_9am"    "humidity_3pm"    "pressure_9am"
## [17] "pressure_3pm"   "cloud_9am"       "cloud_3pm"       "temp_9am"
## [21] "temp_3pm"        "rain_today"      "risk_mm"         "rain_tomorrow"
....
```

Notice the use of the assignment pipe here as introduced in Chapter 1. We will recall that the `magrittr::%<>%` operator pipes the left-hand data to the function on the right-hand side and then returns the result to the left-hand side overwriting the original contents of the memory referred to on the left-hand side.

12 Effect on Data Storage

When the names of the variables within a dataset are changed R does not make a complete new copy of the dataset. Instead, the actual data in the column remains in tact whilst the variable itself (`ds`) references a new memory location where the new variable names get noted. The underlying data within the table is unaffected.

```
location(weather)
## <0x5622dc266410>
## Variables:
## * Date:          <0x5622e05a0da0>
## * Location:      <0x5622e06bcf90>
## * MinTemp:       <0x5622e54a1d00>
## * MaxTemp:       <0x5622e08f5370>
.....
location(ds)
## <0x5622e581e0c0>
## Variables:
## * date:          <0x5622e05a0da0>
## * location:      <0x5622e06bcf90>
## * min_temp:      <0x5622e07d9180>
## * max_temp:      <0x5622e08f5370>
.....
```

13 Special Case Variable Name Transformations

When reviewing the variables of a dataset we often notice other changes that could be made to the variable names. This might be to simplify the variables or to clarify the meaning of the variable. The string processing functions provided by `stringr` come in handy for such processing.

In the following example we remove the prefix of the variable names where we identify that the prefix consists of all characters up to the first underscore. This is useful where a dataset has prefixed each variable by a sequential number or by some other code and we have no real use of such a prefix in our processing.

```
names(ds) %<-% str_replace("^[^_]*_", "")
```

This will take a variable name like `ab123_tax_payable` and convert it to `tax_payable`.

```
str_replace("ab123_tax_payable", "^[^_]*_", "")  
## [1] "tax_payable"
```

The odd looking characters in the argument to `stringr::str_replace()` are a *regular expression*. Regular expressions are a very powerful concept and can get quite complex. The reader is referred to the many resources on-line that cover regular expressions. The regular expression is a pattern used to match some part of the variable name. The pattern begins with `^` which anchors the match to the beginning of the variable name. This can be followed by zero or more characters (`*`) that do not match the underscore (`[^_]`)—the `*` specifies that the preceding pattern can be repeated zero or more times. The preceding pattern here is actually a list of characters included between square brackets. Since this list begins with `^` the listed characters are excluded from the matching. That is, the pattern preceding the `*` will match any character that is not an underscore. The third component of the match is then an actual underscore. Combined this regular expression matches any sequence (including an empty sequence) of characters (except for an underscore) that is at the beginning of the variable name and followed by an underscore.

The next argument to `stringr::str_replace()` is the replacement string. In this case we are replacing the matched pattern with an empty string.

The example here is simply one example of very many possible transformations we become used to in cleaning our datasets. The aim in transforming the variable names is to make them easier to use and to understand, both for ourselves and for others.

14 Data Review

Having ingested the dataset and an initial review, normalising the variable names, we are now ready to explore more. In particular, what do the data within the dataset look like. We again gain `tibble::glimpse()` into the dataset:

```
# Review the dataset.

glimpse(ds)

## # Observations: 145,463
## # Variables: 24
## # $ date <date> 2008-12-01, 2008-12-02, 2008-12-03, 2008-12-04, 2...
## # $ location <chr> "Albury", "Albury", "Albury", "Albury", "Albury", ...
## # $ min_temp <dbl> 13.4, 7.4, 12.9, 9.2, 17.5, 14.6, 14.3, 7.7, 9.7, ...
## # $ max_temp <dbl> 22.9, 25.1, 25.7, 28.0, 32.3, 29.7, 25.0, 26.7, 31...
## # $ rainfall <dbl> 0.6, 0.0, 0.0, 0.0, 1.0, 0.2, 0.0, 0.0, 0.0, 1.4, ...
## # $ evaporation <chr> NA, NA...
## # $ sunshine <chr> NA, NA...
## # $ wind_gust_dir <chr> "W", "WNW", "WSW", "NE", "W", "WNW", "W", "W", "NN...
## # $ wind_gust_speed <int> 44, 44, 46, 24, 41, 56, 50, 35, 80, 28, 30, 31, 61...
## # $ wind_dir_9am <chr> "W", "NNW", "W", "SE", "ENE", "W", "SW", "SSE", "S...
## # $ wind_dir_3pm <chr> "WNW", "WSW", "WSW", "E", "NW", "W", "W", "W", "NW...
## # $ wind_speed_9am <int> 20, 4, 19, 11, 7, 19, 20, 6, 7, 15, 17, 15, 28, 24...
## # $ wind_speed_3pm <int> 24, 22, 26, 9, 20, 24, 24, 17, 28, 11, 6, 13, 28, ...
## # $ humidity_9am <int> 71, 44, 38, 45, 82, 55, 49, 48, 42, 58, 48, 89, 76...
## # $ humidity_3pm <int> 22, 25, 30, 16, 33, 23, 19, 19, 9, 27, 22, 91, 93, ...
## # $ pressure_9am <dbl> 1007.7, 1010.6, 1007.6, 1017.6, 1010.8, 1009.2, 10...
## # $ pressure_3pm <dbl> 1007.1, 1007.8, 1008.7, 1012.8, 1006.0, 1005.4, 10...
## # $ cloud_9am <int> 8, NA, NA, NA, 7, NA, 1, NA, NA, NA, NA, 8, 8, NA, ...
## # $ cloud_3pm <int> NA, NA, 2, NA, 8, NA, NA, NA, NA, 8, 8, 7, ...
## # $ temp_9am <dbl> 16.9, 17.2, 21.0, 18.1, 17.8, 20.6, 18.1, 16.3, 18...
## # $ temp_3pm <dbl> 21.8, 24.3, 23.2, 26.5, 29.7, 28.9, 24.6, 25.5, 30...
## # $ rain_today <chr> "No", "No", "No", "No", "No", "No", "No", "No", "N...
## # $ risk_mm <dbl> 0.0, 0.0, 0.0, 1.0, 0.2, 0.0, 0.0, 0.0, 1.4, 0.0, ...
## # $ rain_tomorrow <chr> "No", "No", "No", "No", "No", "No", "No", "No", "Y...
```

Observe the variety of data types here, ranging from **Date** (`date`), through **character** (`chr`) and **numeric** (`dbl`).

The data mostly looks as expected though it is odd that `evaporation` and `sunshine` are identified as character. Probably because they seem to be all missing, at least in the first 10 or so observations. We begin question other aspects of the data too. For example, is `date` an ongoing sequence of days as it appears to be here? Does `location` have values other than `Albury`? What is the distribution of the different variables?

These are all questions we will start asking ourselves in the context of “living and breathing” our data. Our aim should be to gleam all we can about the data that we are dealing with. Data science is very much about understanding, not blindly processing. The excitement is in the discovery of patterns in the data and the narrative the data is seeking to tell.

15 Dataset Head and Tail

Datasets can be very large, with many observations (millions) and many variables (thousands). We can't be expected to browse through all of the observations and variables. Instead we might review the contents of the dataset using `utils::head()` and `utils::tail()` to consider the top six (by default) and the bottom six observations.

```
# Review the first few observations.

head(ds) %>% print.data.frame()

##      date location min_temp max_temp rainfall evaporation sunshine
## 1 2008-12-01    Albury     13.4     22.9      0.6        <NA>      <NA>
## 2 2008-12-02    Albury      7.4     25.1      0.0        <NA>      <NA>
## 3 2008-12-03    Albury     12.9     25.7      0.0        <NA>      <NA>
## 4 2008-12-04    Albury      9.2     28.0      0.0        <NA>      <NA>
## 5 2008-12-05    Albury     17.5     32.3      1.0        <NA>      <NA>
## 6 2008-12-06    Albury     14.6     29.7      0.2        <NA>      <NA>
##   wind_gust_dir wind_gust_speed wind_dir_9am wind_dir_3pm wind_speed_9am
## 1             W                  44              W            WNW           20
## 2            WNW                 44             NNW            WSW            4
## 3            WSW                 46              W            WSW           19
....
```

```
# Review the last few observations.

tail(ds) %>% print.data.frame()

##      date location min_temp max_temp rainfall evaporation sunshine
## 1 2018-07-25    Uluru      7.5     26.9      0        <NA>      <NA>
## 2 2018-07-26    Uluru      5.3     29.7      0        <NA>      <NA>
## 3 2018-07-27    Uluru      3.7     21.5      0        <NA>      <NA>
## 4 2018-07-28    Uluru      3.2     21.6      0        <NA>      <NA>
## 5 2018-07-29    Uluru      3.7     21.8      0        <NA>      <NA>
## 6 2018-07-30    Uluru      2.3     25.6      0        <NA>      <NA>
##   wind_gust_dir wind_gust_speed wind_dir_9am wind_dir_3pm wind_speed_9am
## 1             N                  35             ESE            WNW            7
## 2            NW                 67               E            NW            7
## 3            SSW                 30             WSW            WSW            6
....
```

All the time we are building a picture of the data we are looking at. It is beginning to confirm that `location` has multiple values whilst `date` does appear to be a sequence for each location.

16 Random Observations

It is also useful to review some random observations from the dataset to provide a little more insight. Here we use `dplyr::sample_n()` to randomly select six rows from the dataset.

```
# Review a random sample of observations.

sample_n(ds, size=6) %>% print.data.frame()

##      date      location min_temp max_temp rainfall evaporation sunshine
## 1 2010-09-29 AliceSprings    11.7     23.0     0.0         8     11.6
## 2 2018-07-20      Darwin    22.9     33.8     0.0        4.2     8.8
## 3 2014-08-29    Newcastle     9.9     18.0     2.4       <NA>    <NA>
## 4 2013-11-11   Melbourne     9.8     19.4     8.0        5.8     5.3
## 5 2010-04-25    Dartmoor     9.1     17.0     4.2        1.8     9.3
## 6 2013-04-18      Hobart     9.2     13.2     0.8         3     4.5
##   wind_gust_dir wind_gust_speed wind_dir_9am wind_dir_3pm wind_speed_9am
## 1           SE                  48            SE          <NA>            22
## 2           SE                  39            ENE             E              9
## 3        <NA>                  NA            SE            SE              4
....
```

17 Characters

On ingesting the dataset into R we observe the variables identified (automatically) as having **character** `base::class()`. The expected values for such variables are strings of characters. We often call such variables *categoric* variables. Within R these are usually represented as a data type called **factor** and handled specially by many of the modelling algorithms.

We can observe some meta data for each of the character variables. Let's first identify the character variables.

```
# Identify the character variables by index.

ds %>%
  sapply(is.character) %>%
  which() %T>%
  print() ->
chari

##      location    evaporation      sunshine wind_gust_dir  wind_dir_9am
##            2                 6                  7                 8                 10
##  wind_dir_3pm   rain_today rain_tomorrow
##            11                22                 24

# Identify the character variables by name.

ds %>%
  names() %>%
  '['](chari) %T>%
  print() ->
charc

## [1] "location"      "evaporation"    "sunshine"       "wind_gust_dir"
## [5] "wind_dir_9am"  "wind_dir_3pm"   "rain_today"     "rain_tomorrow"
```

We will review each one of these in more detail so as to understand how we make use of them in our analyses. In particular we consider which of the variables might be handled as factors.

Where a **character** variable takes on a limited number of possible values we might convert the variable from **character** into **factor** (*categoric*) so as to take advantage of special handling of factors in R.

In fact, we think of a **factor** as a variable that can only take on a specific number of known distinct values which we call the **levels** of the **factor**.

18 Factors

For datasets that we load into R we will not always have examples of all possible levels of a factor. Consequently it is not always possible to automatically list all of the levels automatically. By default the tidyverse ingests these variables as **character** so that we can take specific action to convert them to **factor** as required.

We first review the number of unique *levels* for each of the factors.

```
# Observe the unique levels.

ds[charc] %>% sapply(unique)

## $location
## [1] "Albury"           "BadgerysCreek"    "Cobar"
## [4] "CoffsHarbour"     "Moree"            "Newcastle"
## [7] "NorahHead"         "NorfolkIsland"   "Penrith"
## [10] "Richmond"          "Sydney"           "SydneyAirport"
## [13] "WaggaWagga"        "Williamtown"      "Wollongong"
## [16] "Canberra"          "Tuggeranong"      "MountGinini"
## [19] "Ballarat"          "Bendigo"          "Sale"
## [22] "MelbourneAirport" "Melbourne"        "Mildura"
## [25] "Nhil"              "Portland"         "Watsonia"
## [28] "Dartmoor"          "Brisbane"         "Cairns"
## [31] "GoldCoast"         "Townsville"       "Adelaide"
## [34] "MountGambier"      "Nuriootpa"        "Woomera"
## [37] "Albany"             "Witchcliffe"     "PearceRAAF"
## [40] "PerthAirport"      "Perth"            "SalmonGums"
## [43] "Walpole"            "Hobart"           "Launceston"
## [46] "AliceSprings"      "Darwin"           "Katherine"
## [49] "Uluru"             ""

## $evaporation
## [1] NA    "12"  "14.8" "12.6" "10.8" "11.4" "11.2" "13"   "9.8"  "14.6"
## [11] "11"  "12.8" "13.8" "16.4" "17.4" "16"   "13.6" "8"    "8.2"  "8.6"
## [21] "14.2" "15.8" "16.2" "13.4" "14.4" "11.8" "15.6" "15.2" "11.6" "9.6"
## [31] "6.6"  "0.6"  "6"    "3"   "2"   "5.2"  "9"   "10.2" "10"  "7.4"
## [41] "8.4"  "9.2"  "9.4"  "12.4" "10.4" "7.2"  "6.8"  "7.6"  "4.4" "6.4"
....
```

If we decide to convert all of these variables from character into factor, then we can do so using `base::factor()`.

```
# Convert all character variables to be factors.

ds[charc] %>% map(factor)
```

We don't actually do so here as we will consider each character variable in turn to decide how to handle it, especially that we might observe that evaporation and sunshine appear to be numeric.

19 Location

From our review of the data so far we start to make some observations about the character variables. The first is `location`. We note that several locations were reported in the above exploration of the dataset. We can confirm the number of locations by counting the number of `data.table::unique()` values the variable has in the original dataset.

```
# How many locations are represented in the dataset.

ds$location %>%
  unique() %>%
  length()

## [1] 49
```

We may not know in general what other locations we will come across in related datasets and we already have quite a collection of 49 locations. We will retain this variable as a character data type.

Here is a list of locations and their frequencies in the dataset.

```
ds$location %>%
  table()

## .
##      Adelaide          Albany        Albury    AliceSprings
##      3193                 3040         3041           3041
##      BadgerysCreek       Ballarat     Bendigo      Brisbane
##      3010                 3041         3041           3194
##      Cairns            Canberra      Cobar      CoffsHarbour
##      3041                 3437         3010           3010
##      Dartmoor           Darwin      GoldCoast     Hobart
##      3010                 3194         3041           3194
##      Katherine         Launceston Melbourne MelbourneAirport
##      1579                 3041         3194           3010
##      Mildura             Moree      MountGambier MountGinini
##      3010                 3010         3041           3041
##      Newcastle           Nhil      NorahHead NorfolkIsland
##      3041                 1579         3005           3010
##      Nuriootpa           PearceRAAF   Penrith      Perth
##      3009                 3009         3040           3193
##      PerthAirport         Portland     Richmond     Sale
##      3009                 3010         3010           3010
##      SalmonGums          Sydney     SydneyAirport Townsville
##      2963                 3345         3010           3041
##      Tuggeranong         Uluru      WaggaWagga  Walpole
##      3040                 1579         3010           3006
##      Watsonia            Williamtown Witchcliffe Wollongong
##      ....
```

20 Evaporation and Sunshine

The next two character variables are: `evaporation`, `sunshine`. It does seem odd that these would be character, expecting both to be numeric values. If we look at the top of the dataset we see they have missing values.

```
# Note the character remaining variables to be dealt with.

head(ds$evaporation)
## [1] NA NA NA NA NA NA
head(ds$sunshine)
## [1] NA NA NA NA NA NA
# Review other random values.

sample(ds$evaporation, 8)
## [1] NA     NA     "5.4"  NA     NA     NA     "1.4"  "6.8"
sample(ds$sunshine, 8)
## [1] "8.3"  NA     "10.3" NA     "10.5" NA     "1.2"  NA
```

The heuristic used to determine the data type when ingesting data only looks at a subset of all the data before it determines the data type. In this case the early observations are all missing and so default to character which is general enough to capture all potential values. We need to convert the variables to numeric.

```
# Identify the variables to process.

cvars <- c("evaporation", "sunshine")

# Check the current class of the variables.

ds[cvars] %>% sapply(class)
## evaporation     sunshine
## "character"    "character"

# Convert to numeric.

ds[cvars] %<>% sapply(as.numeric)

# Review some random values.

sample(ds$evaporation, 10)
## [1] 4.6 1.6 7.8  NA  NA  NA 5.6 4.8 3.0  NA
sample(ds$sunshine, 10)
## [1]  NA 10.7  NA  NA  NA  NA  NA 2.5  NA
```

21 Wind Directions

The three wind direction variables (`wind_gust_dir`, `wind_dir_9am`, `wind_dir_3pm`) are also identified as `character`. We review the distribution of values here with `dplyr::select()` identifying any variable that `tidyselect::contains()` the string `_dir` and then build a `base::table()` over those variables.

```
# Review the distribution of observations across levels.

ds %>%
  select(contains("_dir")) %>%
  sapply(table)

##      wind_gust_dir wind_dir_9am wind_dir_3pm
## E           9179        9245       8461
## ENE          8164        7936       7849
## ESE          7483        7724       8539
## N            9310       11570       8798
## NE           7055        7670       8256
## NNE          6434        7995       6531
## NNW          6511        7787       7741
## NW           8028        8715       8609
## S            9209        8675       9889
## SE           9424        9305      10948
## SSE          9159        9085       9351
## SSW          8760        7533       8219
## SW           8934        8443       9256
## W            9778        8418      10065
## WNW          8265        7436       8867
## WSW          9040        6897       9508
```

Observe all 16 compass directions are represented and it would make sense to convert this into a factor. Notice that the directions are in alphabetic order and conversion to factor will retain that. Instead we can construct an ordered factor to capture the compass order (from N, NNE, to NW and NNW). We note the ordering of the directions here.

```
# Levels of wind direction are ordered compass directions.

compass <- c("N", "NNE", "NE", "ENE",
           "E", "ESE", "SE", "SSE",
           "S", "SSW", "SW", "WSW",
           "W", "WNW", "NW", "NNW")
```

22 Ordered Factor

Given our knowledge that compass directions have an obvious order, we convert the direction variables into an ordered factor. We do so using `ordered=TRUE` with `base::factor()`.

```
# Note the names of the wind direction variables.

ds %>%
  select(contains("_dir")) %>%
  names() %T>%
  print() ->
vnames

## [1] "wind_gust_dir" "wind_dir_9am"   "wind_dir_3pm"

# Convert these variables from character to factor.

ds[vnames] %<>%
  lapply(factor, levels=compass, ordered=TRUE) %>%
  data.frame() %>%
  tbl_df()

# Confirm they are now factors.

ds[vnames] %>% sapply(class)

##      wind_gust_dir wind_dir_9am wind_dir_3pm
## [1,] "ordered"     "ordered"     "ordered"
## [2,] "factor"      "factor"      "factor"
```

We can again obtain a distribution of the variables to confirm that all we have changed is the data type.

```
# Verify the distribution has not changed.

ds %>%
  select(contains("_dir")) %>%
  sapply(table)

##      wind_gust_dir wind_dir_9am wind_dir_3pm
## N         9310       11570       8798
## NNE        6434       7995       6531
## NE         7055       7670       8256
## ENE        8164       7936       7849
## E          9179       9245       8461
## ESE        7483       7724       8539
## SE         9424       9305      10948
## SSE        9159       9085       9351
## S          9209       8675       9889
## SSW        8760       7533       8219
## SW         8934       8443       9256
....
```

23 Rain

The two remaining character variables are: `rain_today`, `rain_tomorrow`. Their distributions are generated by `dplyr::select()`ing from the dataset those variables that start with `rain_` and then build a `base::table()` over those variables. We use `base::sapply()` to apply `base::table()` to the selected columns to count the frequency of the occurrence of each value of a variable within the dataset.

```
# Review the distribution of observations across levels.

ds %>%
  select(starts_with("rain_")) %>%
  sapply(table)

##     rain_today rain_tomorrow
## No      110981        110985
## Yes     31253         31250
```

Noting that `No` and `Yes` are the only values these two variables will take it makes sense to convert them both to factors. We will keep the ordering as alphabetic and so a simple call to `base::factor()` will convert from character to factor.

```
# Note the names of the rain variables.

ds %>%
  select(starts_with("rain_")) %>%
  names() ->
vnames

# Confirm these are currently character variables.

ds[vnames] %>% sapply(class)

##     rain_today rain_tomorrow
## "character"   "character"

# Convert these variables from character to factor.

ds[vnames] %<>%
  lapply(factor) %>%
  data.frame() %>%
  tbl_df()

# Confirm they are now factors.

ds[vnames] %>% sapply(class)

##     rain_today rain_tomorrow
## "factor"       "factor"
```

24 Numeric

Summaries of numeric data are provided using `base:::summary()`. In the following we identify the numeric variables and summarise each. In doing so, as a data scientist, we want to again observe any oddities and to explain them.

```
ds %>%
  sapply(is.numeric) %>%
  which() %>%
  names %T>%
  print() ->
numi

## [1] "min_temp"          "max_temp"          "rainfall"           "evaporation"
## [5] "sunshine"          "wind_gust_speed"   "wind_speed_9am"    "wind_speed_3pm"
## [9] "humidity_9am"      "humidity_3pm"       "pressure_9am"      "pressure_3pm"
## [13] "cloud_9am"         "cloud_3pm"         "temp_9am"          "temp_3pm"
## [17] "risk_mm"

ds[numi] %>%
  summary()

##      min_temp        max_temp        rainfall        evaporation
## Min.   :-8.70   Min.   :-4.10   Min.   : 0.000   Min.   : 0.00
## 1st Qu.: 7.40   1st Qu.:17.90   1st Qu.: 0.000   1st Qu.: 2.60
## Median :11.90   Median :22.50   Median : 0.000   Median : 4.60
## Mean   :12.04   Mean   :23.14   Mean   : 2.298   Mean   : 5.42
## 3rd Qu.:16.70   3rd Qu.:28.20   3rd Qu.: 0.600   3rd Qu.: 7.20
## Max.   :33.90   Max.   :48.10   Max.   :371.000  Max.   :82.40
## NA's   :1545    NA's   :1335    NA's   :3229    NA's   :64843
##      sunshine        wind_gust_speed   wind_speed_9am   wind_speed_3pm
## Min.   : 0.00   Min.   : 2.00   Min.   : 0.00   Min.   : 0.00
## 1st Qu.: 4.90   1st Qu.:31.00   1st Qu.: 7.00   1st Qu.:13.00
## Median : 8.40   Median :39.00   Median :13.00   Median :19.00
## Mean   : 7.61   Mean   :40.01   Mean   :14.01   Mean   :18.64
## 3rd Qu.:10.60   3rd Qu.:48.00   3rd Qu.:19.00   3rd Qu.:24.00
## Max.   :14.50   Max.   :135.00  Max.   :87.00   Max.   :87.00
## NA's   :70820   NA's   :10667   NA's   :2081    NA's   :3407
....
```

Reviewing this information we can make some obvious observations. Temperatures, for example, appears to be in degrees Celsius rather than Fahrenheit. Rainfall looks like millimetres. There are some quite skewed distributions with min and median small but large max values. As data scientists we will further explore the distributions as in Chapter 5.

25 Logical

Above we converted `rain_today` and `rain_tomorrow` to factors. They have just two values as we confirm here, in addition to a small number of missing values (`NA`).

```
ds %>%
  select(rain_today, rain_tomorrow) %>%
  summary()

##   rain_today    rain_tomorrow
##   No    :110981   No    :110985
##   Yes   : 31253   Yes   : 31250
##   NA's:  3229    NA's:  3228
```

As binary valued factors, and particularly as the values suggest, they are both candidates for being considered as logical variables (sometimes called Boolean). They can be treated as `FALSE/TRUE` instead of `No/Yes` and so supported directly by R as class `logical`. Different functions will then treat them as appropriate but not all functions do anything special. If this suits our purposes then the following can be used to perform the conversion to logical.

```
ds %<>%
  mutate(rain_today    = rain_today    == "Yes",
         rain_tomorrow = rain_tomorrow == "Yes")
```

Best to now check that the distribution itself has not changed.

```
ds %>%
  select(rain_today, rain_tomorrow) %>%
  summary()

##   rain_today    rain_tomorrow
##   Mode :logical  Mode :logical
##   FALSE:110319   FALSE:110316
##   TRUE  :31880    TRUE  :31877
##   NA's  :3261    NA's  :3267
```

Observe that the `TRUE` (`Yes`) values are much less frequent than the `FALSE` (`No`) values, and we also note the missing values.

The majority of days not having rain can be cross checked with the rainfall variable. In the previous summary of its distribution we note that rainfall has a median of zero, consistent with fewer days of actual rain. As data scientists we perform various cross checks on the hunt for oddities in the data.

As data scientists we will also want to understand why there are missing values. Is it simply some rare failures to capture the observation, or for example is there a particular location not recording rainfall? We would explore that now before moving on.

For our purposes going forward we will retain these two variables as factors. One reason for doing so is that we will illustrate missing value imputation using `randomForest::na.roughfix()` and this function does not handle logical data but keeping `rain_tomorrow` as character will allow missing value imputation. Of course we could skip this variable for the imputation.

26 Variable Roles

Now that we have a basic idea of the size and shape and contents of the dataset and have performed some basic data type identification and conversion we are in a position to identify the roles played by the variables within the dataset. First we will record the list of available variables so that we might reference them below.

```
# Note the available variables.

vars <- names(ds) %T>% print()

## [1] "date"          "location"       "min_temp"        "max_temp"
## [5] "rainfall"       "evaporation"    "sunshine"        "wind_gust_dir"
## [9] "wind_gust_speed" "wind_dir_9am"   "wind_dir_3pm"    "wind_speed_9am"
## [13] "wind_speed_3pm"  "humidity_9am"   "humidity_3pm"    "pressure_9am"
## [17] "pressure_3pm"    "cloud_9am"      "cloud_3pm"       "temp_9am"
## [21] "temp_3pm"        "rain_today"     "risk_mm"         "rain_tomorrow"
```

By this stage of the project we will usually have identified a business problem that is the focus of attention. In our case we will assume it is to build a predictive analytics model to predict the chance of it raining tomorrow given the observation of today's weather. In this case the variable `rain_tomorrow` is the *target variable*. Given today's observations of the weather this is what we want to predict. The dataset we have is then a *training dataset* of historic observations. The task is to identify any patterns among the other observed variables that suggest that it rains the following day.

```
# Note the target variable.

target <- "rain_tomorrow"

# Place the target variable at the beginning of the vars.

vars <- c(target, vars) %>% unique() %T>% print()

## [1] "rain_tomorrow"   "date"          "location"       "min_temp"
## [5] "max_temp"        "rainfall"       "evaporation"    "sunshine"
## [9] "wind_gust_dir"   "wind_gust_speed" "wind_dir_9am"   "wind_dir_3pm"
## [13] "wind_speed_9am"  "wind_speed_3pm"  "humidity_9am"   "humidity_3pm"
## [17] "pressure_9am"    "pressure_3pm"   "cloud_9am"      "cloud_3pm"
## [21] "temp_9am"        "temp_3pm"       "rain_today"     "risk_mm"
....
```

We have taken the opportunity here to move the target variable to be the first in the vector of variables recorded in `vars`. This is common practice where the first variable in a dataset is the target (dependent variable) and the remainder are the variables (the independent variables) that will be used to build a model to predict that target.

27 Risk Variable

With some knowledge of the data we observe `risk_mm` captures the amount of rain recorded tomorrow. We refer to this as a *risk variable*, being a measure of the impact or risk of the target we are predicting (rain tomorrow). The risk is an output variable and should not be used as an input to the modelling—it is not an independent variable. In other circumstances it might actually be treated as the target variable.

```
# Note the risk variable - measures the severity of the outcome.

risk <- "risk_mm"
```

For this risk variable note that we expect it to have a value of 0 for all observations when the target variable has the value No.

```
# Review the distribution of the risk variable for non-targets.
```

```
ds %>%
  filter(rain_tomorrow == "No") %>%
  select(risk_mm) %>%
  summary()

##      risk_mm
##  Min.   :0.00000
##  1st Qu.:0.00000
##  Median :0.00000
##  Mean   :0.07397
##  3rd Qu.:0.00000
##  Max.   :1.00000
```

Interestingly, even a little rain (defined as 1mm or less) is regarded as no rain. That is useful to keep in mind and is a discovery of the data that we might not have expected. As data scientists we should be expecting to find the unexpected.

A similar analysis for the target observations is more in line with expectations.

```
# Review the distribution of the risk variable for targets.
```

```
ds %>%
  filter(rain_tomorrow == "Yes") %>%
  select(risk_mm) %>%
  summary()

##      risk_mm
##  Min.   : 1.10
##  1st Qu.: 2.40
##  Median : 5.20
##  Mean   :10.19
##  3rd Qu.:11.60
##  Max.   :371.00
```

28 ID Variables

From our observations so far we note that the variable (`date`) acts as an identifier as does the variable (`location`). Given a `date` and a `location` we have an observation of the remaining variables. Thus we note that these two variables are so-called identifiers. Identifiers would not usually be used as independent variables for building predictive analytics models.

```
# Note any identifiers.

id <- c("date", "location")
```

We might get a sense of how this works with the following which will list a random sample of locations and how long the observations for that location have been collected.

```
ds[id] %>%
  group_by(location) %>%
  count() %>%
  rename(days=n) %>%
  mutate(years=round(days/365)) %>%
  as.data.frame() %>%
  sample_n(10)

##      location days years
## 9      Cairns   3041     8
## 13    Dartmoor   3010     8
## 25  Newcastle   3041     8
## 32       Perth   3193     9
## 45   Watsonia   3010     8
....
```

The data for each location ranges in length from 4 years up to 9 years, though most have 8 years of data.

```
ds[id] %>%
  group_by(location) %>%
  count() %>%
  rename(days=n) %>%
  mutate(years=round(days/365)) %>%
  ungroup() %>%
  select(years) %>%
  summary()

##      years
##  Min.   :4.000
##  1st Qu.:8.000
##  Median :8.000
##  Mean   :7.918
##  3rd Qu.:8.000
##  Max.   :9.000
```

29 Ignore IDs and Outputs

The identifiers and any risk variable (which is an output variable) should be ignored in any predictive modelling. Always watch out for treating output variables as inputs to modelling—this is a surprisingly common trap for beginners. We will build a vector of the names of the variables to ignore. Above we have already recorded the `id` variables and (optionally) the `risk`. Here we join them together into a new vector using `data.table::union()` which performs a set union operation—that is, it joins the two arguments together and removes any repeated variables.

```
# Initialise ignored variables: identifiers and risk.

ignore <- union(id, risk) %T>% print()
## [1] "date"      "location"   "risk_mm"
```

We might also check for any variable that has a unique value for every observation. These are often identifiers and if so they are candidates for ignoring. We select the `vars` from the dataset and pipe through to `base::sapply()` for any variables having only unique values. In our case there are no further candidate identifiers, as indicated by the empty result, `character(0)`.

```
# Heuristic for candidate identifiers to possibly ignore.

ds[vars] %>%
  sapply(function(x) x %>% unique() %>% length()) %>%
  equals(nrow(ds)) %>%
  which() %>%
  names() %T>%
  print() ->
ids
## character(0)

# Add them to the variables to be ignored for modelling.

ignore <- union(ignore, ids) %T>% print()
## [1] "date"      "location"   "risk_mm"
```

30 Ignore Missing

We next remove any variable where all of the values are missing. There are none like this in the weather dataset but in general for other datasets with thousands of variables there may be some. Here we first count the number of missing values for each variable and then list the names of those variables that have no values.

```
# Identify variables with only missing values.

ds[vars] %>%
  sapply(function(x) x %>% is.na %>% sum) %>%
  equals(nrow(ds)) %>%
  which() %>%
  names() %T>%
  print() ->
missing
## character(0)

# Add them to the variables to be ignored for modelling.

ignore <- union(ignore, missing) %T>% print()
## [1] "date"      "location"   "risk_mm"
```

It is also useful to identify those variables which are very sparse—that have mostly missing values. We can decide on a threshold of the proportion missing above which to ignore the variable as not likely to add much value to our analysis. For example, we may want to ignore variables with more than 70% of the values missing:

```
# Identify a threshold above which proportion missing is fatal.

missing.threshold <- 0.7

# Identify variables that are mostly missing.

ds[vars] %>%
  sapply(function(x) x %>% is.na() %>% sum()) %>%
  '>'(missing.threshold*nrow(ds)) %>%
  which() %>%
  names() %T>%
  print() ->
mostly
## character(0)

# Add them to the variables to be ignored for modelling.

ignore <- union(ignore, mostly) %T>% print()
## [1] "date"      "location"   "risk_mm"
```

31 Ignore Excessive Level Variables

Another issue we traditionally come across in our datasets are those factors with very many levels. This is more common when we read data as factors rather than as character, and so this step depends on where the data has come from. Nonetheless We might want to check for and ignore such variables.

```
# Identify a threshold above which we have too many levels.

levels.threshold <- 20

# Identify variables that have too many levels.

ds[vars] %>%
  sapply(is.factor) %>%
  which() %>%
  names() %>%
  sapply(function(x) ds %>% extract2(x) %>% levels() %>% length()) %>%
  '>='(levels.threshold) %>%
  which() %>%
  names() %T>%
  print() ->
too.many
## character(0)

# Add them to the variables to be ignored for modelling.

ignore <- union(ignore, too.many) %T>% print()
## [1] "date"      "location"   "risk_mm"
```

32 Ignore Constants

We also ignore variables with constant values as they add no extra information to the analysis.

```
# Identify variables that have a single value.

ds[vars] %>%
  sapply(function(x) all(x == x[1L])) %>%
  which() %>%
  names() %T>%
  print() ->
constants

## character(0)

# Add them to the variables to be ignored for modelling.

ignore <- union(ignore, constants) %T>% print()

## [1] "date"      "location"   "risk_mm"
```

33 Correlated Variables for Numerics

It is often useful to identify highly correlated variables. Such variables will often record the same information but in different ways and often arise when we combine data from different sources.

We identify the numeric variables of a dataset by `base::sapply()`ing the function `base::is.numeric()` to find `base::which()` are numeric. Their integer column positions are stored into the variable `numi`.

```
# Note which variables are numeric.

vars %>%
  setdiff(ignore) %>%
  extract(ds, .) %>%
  sapply(is.numeric) %>%
  which() %>%
  names() %T>%
  print() ->
numc

## [1] "min_temp"      "max_temp"       "rainfall"        "evaporation"
## [5] "sunshine"       "wind_gust_speed" "wind_speed_9am"  "wind_speed_3pm"
## [9] "humidity_9am"   "humidity_3pm"     "pressure_9am"    "pressure_3pm"
## [13] "cloud_9am"      "cloud_3pm"       "temp_9am"        "temp_3pm"
```

34 Calculating Correlations

The correlation is calculated by `dplyr::select()`ing the numeric columns from the dataset and passing that through to `stats::cor()`. This matrix of pairwise correlations is based on only the complete observations so that observations with missing values are ignored.

We set the upper triangle of the correlation matrix to `NA`'s as they are a mirror of the values in the lower triangle and thus redundant. We also set `diag=TRUE` to set the diagonals as `NA` since they will always be perfect correlations.

The processing continues by making all values positive using `base::abs()`. With conversion to `base::data.frame()` then to `dplyr::tbl_df()` the dataset column names need to be reset appropriately using `magrittr::set_colnames()`. We `plyr::mutate()` the dataset with a new column using `plyr::mutate()`, reshape the dataset using `tidy::gather()` from `tidyverse` and then omit missing correlations using `data.table::na.omit()`. Finally the rows are `plyr::arrange()`'d with the highest absolute correlations appearing first.

```
# For the numeric variables generate a table of correlations

ds[numc] %>%
  cor(use="complete.obs") %>%
  ifelse(upper.tri(., diag=TRUE), NA, .) %>%
  abs %>%
  data.frame %>%
  tbl_df %>%
  set_colnames(numc) %>%
  mutate(var1=numc) %>%
  gather(var2, cor, -var1) %>%
  na.omit %>%
  arrange(-abs(cor)) %T>%
  print() ->
mc

## # A tibble: 120 x 3
##   var1      var2       cor
##   <chr>     <chr>     <dbl>
## 1 temp_3pm  max_temp  0.985
## 2 pressure_3pm pressure_9am 0.962
## 3 temp_9am   min_temp  0.908
....
```

35 Dealing with Correlations

From the final result we can identify pairs of variables where we might want to keep one but not the other variable because they are highly correlated. We will select them manually since it is a judgement call. Normally we might limit the removals to those correlations that are 0.90 or more. In our case here the three pairs of highly correlated variables make intuitive sense.

```
# Note the correlated variables that are redundant.

correlated <- c("temp_3pm", "pressure_3pm", "temp_9am")

# Add them to the variables to be ignored for modelling.

ignore <- union(ignore, correlated) %T>% print()

## [1] "date"          "location"       "risk_mm"        "temp_3pm"       "pressure_...
## [6] "temp_9am"
```

36 Removing Ignored Variables

Once we have identified all of the variables to ignore we remove them from our list of variables to use.

```
# Check the number of variables currently.

length(vars)
## [1] 24

# Remove the variables to ignore.

vars <- setdiff(vars, ignore)

# Confirm they are now ignored.

length(vars)
## [1] 18
```

37 Feature Selection

The `FSelector` package provides functions to identify subsets of variables that might be more effective for modelling. We can use this (and other packages) to assist us in reducing the variables that will be useful in our modelling. As we find useful functionality we will add them to our standard template so that for our next dataset we have the functionality readily available.

We first use `FSelector::cfs()` to identify a good subset of variables using correlation and entropy. We then list the variable importance using `FSelector::information.gain()` to advise a useful subset of variables. Note that the `stringi::%s+%` operator is a convenience to concatenate strings together to produce a formula that indicates we will model the target variable using all of the other variables of the dataset.

```
# Construct the formulation of the modelling we plan to do.

form <- formula(target %s+%" ~ .") %T>% print()
## rain_tomorrow ~ .

# Use correlation search to identify key variables.

cfs(form, ds[vars])
## [1] "rainfall"      "sunshine"       "humidity_3pm"   "cloud_3pm"     "rain_today"

# Use information gain to identify variable importance.

information.gain(form, ds[vars])

##                  attr_importance
## min_temp          0.005965086
## max_temp          0.013679399
## rainfall          0.058867687
## evaporation      0.005233566
## sunshine          0.055563157
## wind_gust_dir    0.006082875
## wind_gust_speed  0.027066589
## wind_dir_9am     0.008660761
## wind_dir_3pm     0.004830997
## wind_speed_9am   0.004388310
## wind_speed_3pm   0.005426372
## humidity_9am     0.037671301
## humidity_3pm     0.111123203
## pressure_9am     0.028254633
## cloud_9am         0.035295285
## cloud_3pm         0.051036499
## rain_today        0.047266724
```

The two measures are consistent in this case in that the variables identified by `FSelector::cfs()` are the more important variables identified by `FSelector::information.gain()`.

38 Missing Targets

Sometimes there may be further operations to perform on the dataset prior to modelling. A common task is to deal with missing values. Here we remove observations with a missing target. As with any missing data we should also analyse whether there is any pattern to the missing targets. This may be indicative of a systemic data issue rather than simply randomly missing values.

```
# Check the dimensions to start with.

dim(ds)
## [1] 145463      24

# Identify observations with a missing target.

missing.target <- ds %>% extract2(target) %>% is.na()

# Check how many are found.

sum(missing.target)
## [1] 3228

# Remove observations with a missing target.

ds %<>% filter(!missing.target)

# Confirm the filter delivered the expected dataset.

dim(ds)
## [1] 142235      24
```

39 Missing Values

Missing values for the variables are an issue for some but not all algorithms. For example `randomForest::randomForest()` omits observations with missing values by default whilst `rpart::rpart()` has a particularly well developed approach to dealing with missing values.

We may want to impute missing values in the data (though it is not always wise to do so). Here we do this using `randomForest::na.roughfix()` from `randomForest`. This function provides, as the name implies, a rather basic algorithm for imputing missing values. Because of this we will demonstrate the process but then restore the original dataset—we will not want this imputation to be included in our actual dataset.

```
# Backup the dataset so we can restore it as required.

ods <- ds

# Count the number of missing values.

ds[vars] %>% is.na() %>% sum()
## [1] 306730

# Impute missing values.

ds[vars] %>% na.roughfix()

# Confirm that no missing values remain.

ds[vars] %>% is.na() %>% sum()
## [1] 0
```

As foreshadowed we now restore the dataset with its original contents.

```
# Restore the original dataset.

ds <- ods
```

40 Omitting Observations

An alternative is to remove observations that have missing values. Here `data.table::na.omit()` identifies the rows to omit based on the `vars` to be included for modelling. The list of rows to omit is stored as the `na.action` attribute of the returned object. We then remove these observations from the dataset.

20180726

Notice we keep a copy of the original dataset and then restore it.

```
# Backup the dataset so we can restore it as required.

ods <- ds

# Initialise the list of observations to be removed.

omit <- NULL

# Review the current dataset.

ds[vars] %>% nrow()
## [1] 142235

ds[vars] %>% is.na() %>% sum()
## [1] 306730

# Identify any observations with missing values.

mo <- attr(na.omit(ds[vars]), "na.action")

# Record the observations to omit.

omit <- union(omit, mo)

# If there are observations to omit then remove them.

if (length(omit)) ds <- ds[-omit,]

# Confirm the observations have been removed.

ds[vars] %>% nrow()
## [1] 55650

ds[vars] %>% is.na() %>% sum()
## [1] 0

# Restore the original dataset.

ds <- ods
```

41 Normalise Factors

Some variables will have levels with spaces, and mixture of cases, etc. We may like to normalise the levels for each of the categoric variables. For very large datasets this can take some time and so we may want to be selective.

```
# Note which variables are categoric.

ds %>%
  sapply(is.factor) %>%
  which() ->
catc

# Normalise the levels of all categoric variables.

for (v in catc)
  levels(ds[[v]]) %<>% normVarNames()
```

42 Target as a Factor

We often build classification models. For such models we want to ensure the target is categoric. Often it is 0/1 and hence is loaded as numeric. We could tell our model algorithm of choice to explicitly do classification or else set the target using `base::as.factor()` in the formula. Nonetheless it is generally cleaner to do this here and note that this code has no effect if the target is already categoric.

```
# Ensure the target is categoric.

ds[[target]] %>% as.factor()

# Confirm the distribution.

ds[target] %>% table()

## .
##   no    yes
## 110985 31250
```

We can visualise the distribution of the target variable using `ggplot2`. The dataset is piped to `ggplot2::ggplot()` whereby the target is associated through `ggplot2::aes_string()` (the aesthetics) with the x-axis of the plot. To this we add a graphics layer using `ggplot2::geom_bar()` to produce the bar chart, with bars having `width= 0.2` and a `fill=` color of "grey". The resulting plot can be seen in Figure 1.

```
ds %>%
  ggplot(aes_string(x=target)) +
  geom_bar(width=0.2, fill="grey") +
  theme(text=element_text(size=14))
```

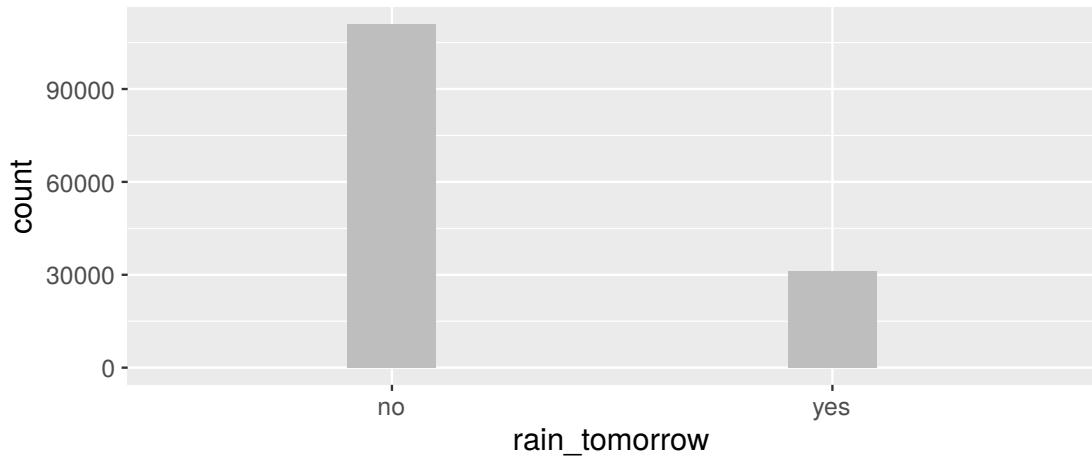


Figure 1: Target variable distribution. Plotting the distribution is useful to gain an insight into the number of observations in each category. As is the case here we often see a skewed distribution.

43 Identify Variable Types

Metadata is data about the data. We now record data about our dataset that we use later in further processing and analysing our data. In one sense the metadata is simply a convenient store.

We identify the variables that will be used to build analytic models that provide different kinds of insight into our data. Above we identified the variable roles such as the target, a risk variable and the ignored variables. From an analytic modelling perspective we identify variables that are the model inputs. We record then both as a vector of characters (the variable names) and a vector of integers (the variable indices).

```
inputs <- setdiff(vars, target) %T>% print()
## [1] "min_temp"          "max_temp"          "rainfall"           "evaporation"
## [5] "sunshine"           "wind_gust_dir"      "wind_gust_speed"   "wind_dir_9am"
## [9] "wind_dir_3pm"       "wind_speed_9am"     "wind_speed_3pm"    "humidity_9am"
## [13] "humidity_3pm"       "pressure_9am"       "cloud_9am"         "cloud_3pm"
## [17] "rain_today"
```

The integer indices are determined from the `base::names()` of the variables in the original dataset. Note the use of `USE.NAMES=` from `base::sapply()` to turn off the inclusion of names in the resulting vector to keep the result as a simple vector.

```
inputi <- sapply(inputs,
                  function(x) which(x == names(ds)),
                  USE.NAMES=FALSE)
inputi
## [1] 3 4 5 6 7 8 9 10 11 12 13 14 15 16 18 19 22
```

For convenience we record the number of observations:

```
nobs <- nrow(ds) %T>% print()
## [1] 142235
```

Here we simply report on the dimensions of various data subsets primarily to confirm the dataset appear as we expect:

```
dim(ds)
## [1] 142235      24
dim(ds[vars])
## [1] 142235      18
dim(ds[inputs])
## [1] 142235      17
```

44 Identify Numeric and Categoric Variables

Identifying numeric and categoric variables may be useful for example for cluster analysis algorithms that only deal with numeric variables. Here we identify them by name (a character string) and by index. When using the index we have to assume the variables remain in the same order within the dataset and all variables are present, otherwise the indices will get out of sync.

```
# Identify the numeric variables by index.

ds %>%
  sapply(is.numeric) %>%
  which() %>%
  intersect(inputi) %T>%
  print() ->
numi
## [1] 3 4 5 6 7 9 12 13 14 15 16 18 19

# Identify the numeric variables by name.

ds %>%
  names() %>%
  extract(numi) %T>%
  print() ->
numc
## [1] "min_temp"          "max_temp"          "rainfall"          "evaporation"
## [5] "sunshine"           "wind_gust_speed"   "wind_speed_9am"    "wind_speed_3pm"
## [9] "humidity_9am"       "humidity_3pm"       "pressure_9am"      "cloud_9am"
## [13] "cloud_3pm"

# Identify the categoric variables by index and then name.

ds %>%
  sapply(is.factor) %>%
  which() %>%
  intersect(inputi) %T>%
  print() ->
cati
## [1] 8 10 11 22

ds %>%
  names() %>%
  extract(cati) %T>%
  print() ->
catc
## [1] "wind_gust_dir" "wind_dir_9am"  "wind_dir_3pm"  "rain_today"
```

45 Save the Dataset

For large datasets we may want to save it to a binary RData file once we have wrangled it into the right shape and collected the metadata. Loading a binary dataset is generally quicker than loading a CSV file—a CSV file with 2 million observations and 800 variables can take 30 minutes to `utils::read.csv()`, 5 minutes to `base::save()`, and 30 seconds to `base::load()`.

```
# Timestamp for the dataset.

dsdate <- "_" %s+%
  format(Sys.Date(), "%y%m%d") %T>% print()
## [1] "_180908"

# Filename for the saved dataset

dsrdata <- dsname %s+%
  dsdate %s+%
  ".RData" %T>% print()
## [1] "weather_180908.RData"

# Save relevant R objects to binary RData file.

save(ds, dsname, dspath, dsdate, nobs,
  vars, target, risk, id, ignore, omit,
  inputi, inputs, numi, numc, cati, catc,
  file=dsrdata)
```

Notice that in addition to the dataset (`ds`) we also store the collection of *metadata*. This begins with items such as the name of the dataset, the source file path, the date we obtained the dataset, the number of observations, the variables of interest, the target variable, the name of the risk variable (if any), the identifiers, the variables to ignore and observations to omit. We continue with the indicies of the input variables and their names, the indicies of the numeric variables and their names, and the indicies of the categoric variables and their names.

Each time we wish to use the dataset we can now simply `base::load()` it into R. The value that is invisibly returned by `base::load()` is a vector naming the R objects loaded from the binary RData file.

```
load(dsrdata) %>% print()
## [1] "ds"      "dsname" "dspath"  "dsdate" "nobs"   "vars"   "target" "risk"
## [9] "id"      "ignore"  "omit"    "inputi"  "inputs"  "numi"   "numc"   "cati"
## [17] "catc"
```

We place the call to `base::load()` within a call to `(` (i.e., we have surrounded the call with round brackets) to ensure the result of the function call is printed. A call to `base::load()` returns its result invisibly since we are primarily interested in its side-effect. The side-effect is to read to R binary data from disk and to make it available within our current R session.

46 A Template for Data Preparation

Through this chapter we have built a template for data preparation. An actual knitr template based on this chapter for data preparation is available as <http://HandsOnDataScience.com/scripts/data.Rnw>. An automatically derived version including just the R code is also available as <http://HandsOnDataScience.com/scripts/data.R>. Notice that we would not necessarily perform all of the steps, such as normalising the variable names, imputing missing values, omitting observations with missing values, and so on. Instead we pick and choose as is appropriate to our situation and specific datasets. Also, some data specific transformations are not included in the template and there may be other transforms we need to perform that we have not covered here. As we discover new tools to support the data scientist we can add them into our own templates.

47 Command Summary

This chapter has introduced, demonstrated and described the following R packages, functions, commands, operators, and datasets:

get *Function from base.* Returns the named dataset.

glimpse *Function from tibble.* Summarise a dataset.

head *Function from utils.* Display the first few rows of a dataset.

names *Function from base.* Column names of a dataset.

normVarNames *Function from rattle.* Normalize variable names.

ncol *Function from base.* Number of columns in a dataset.

nrow *Function from base.* Number of rows in a dataset.

read_csv *Function from readr.* Load data from a CSV file.

sample_n *Function from dplyr.* Random sample of n rows.

sapply *Function from base.* Apply a function to columns of a dataset.

str_replace *Function from stringr.* Replace a string with another.

system.file *Function from base.* Locate a system file.

tail *Function from utils.* Display the last few rows of a dataset.

48 Exercises

Exercise 1 Exploring the Weather

We have worked with the Australian **weatherAUS** dataset throughout this chapter. For this exercise we will explore the dataset further. For each exercise, extend the knitr document with the analyses performed.

1. Create a data preparation script beginning with the template available from <http://HandsOnDataScience.com/scripts/data.Rnw> and replicating the data processing performed in this chapter.
2. Investigate the `dplyr::group_by()` and `plyr::summarise()` functions, combined through a pipeline using `dplyr::%>%` to identify regions with considerable variance in their weather observations. The use of `stats::var()` might be a good starting point.

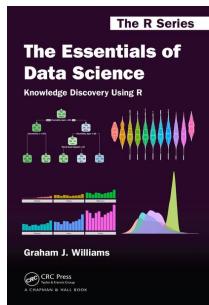
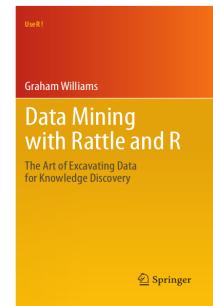
Exercise 2 Understanding Ferries

A dataset of ferry crossings on Sydney Harbour is available as <http://HandsOnDataScience.com/data/ferry.csv>. The original source of the Ferry dataset is <http://www.bts.nsw.gov.au/Statistics/Ferry/default.aspx?FolderID=224>. The dataset is available under a Creative Commons Attribution (CC BY 3.0 AU) license. We will use this dataset to exercise our data template.

1. Create a data preparation script beginning with the template available from <http://HandsOnDataScience/scripts/data.R>.
2. Change the sample source dataset within the template to download the ferry dataset into R.
3. Rename the variables to become normalized variable names.
4. Create two new variables from `sub_route`, called `origin` and `destination`.
5. Convert dates.
6. Convert appropriate variables into factors.
7. Work through the template to explore and prepare the dataset.
8. Develop some visualisations.

49 Further Reading

The [Rattle](#) book ([Williams, 2011](#)), published by Springer, provides a comprehensive introduction to data mining and analytics using Rattle and R. It is available from [Amazon](#). Rattle provides a graphical user interface through which the user is able to load, explore, visualise, and transform data, and to build, evaluate, and export models. Through its Log tab it specifically aims to provide an R template which can be exported and serve as the starting point for further programming with data in R.



The [Essentials of Data Science](#) book ([Williams, 2017](#)), published by CRC Press, provides a comprehensive introduction to data science through programming with data using R. It is available from [Amazon](#). The book provides a template based approach to doing data science and knowledge discovery. Templates are provided for data wrangling and model building. These serve as generic starting points for programming with data, and are designed to require minimal effort to get started. Visit <https://essentials.togaware.com> for further guides and templates.

50 References

- Bache SM, Wickham H (2014). *magrittr: A Forward-Pipe Operator for R*. R package version 1.5, URL <https://CRAN.R-project.org/package=magrittr>.
- Breiman L, Cutler A, Liaw A, Wiener M (2018). *randomForest: Breiman and Cutler's Random Forests for Classification and Regression*. R package version 4.6-14, URL <https://CRAN.R-project.org/package=randomForest>.
- Gagolewski M, Tartanus B, , other contributors; IBM, other contributors; Unicode, Inc (2018). *stringi: Character String Processing Facilities*. R package version 1.2.4, URL <https://CRAN.R-project.org/package=stringi>.
- Hester J (2018). *glue: Interpreted String Literals*. R package version 1.3.0, URL <https://CRAN.R-project.org/package=glue>.
- R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- Romanski P, Kotthoff L (2018). *FSelector: Selecting Attributes*. R package version 0.31, URL <https://CRAN.R-project.org/package=FSelector>.
- Spinu V, Grolemund G, Wickham H (2018). *lubridate: Make Dealing with Dates a Little Easier*. R package version 1.7.4, URL <https://CRAN.R-project.org/package=lubridate>.
- Wickham H (2018). *stringr: Simple, Consistent Wrappers for Common String Operations*. R package version 1.3.1, URL <https://CRAN.R-project.org/package=stringr>.
- Wickham H, Chang W, Henry L, Pedersen TL, Takahashi K, Wilke C, Woo K (2018a). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.0.0, URL <https://CRAN.R-project.org/package=ggplot2>.
- Wickham H, François R, Henry L, Müller K (2018b). *dplyr: A Grammar of Data Manipulation*. R package version 0.7.6, URL <https://CRAN.R-project.org/package=dplyr>.
- Wickham H, Henry L (2018). *tidyverse: Easily Tidy Data with 'spread()' and 'gather()' Functions*. R package version 0.8.1, URL <https://CRAN.R-project.org/package=tidyr>.
- Wickham H, Hester J, Francois R (2017). *readr: Read Rectangular Text Data*. R package version 1.1.1, URL <https://CRAN.R-project.org/package=readr>.
- 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.
- Williams GJ (2017). *The Essentials of Data Science: Knowledge discovery using R*. The R Series. CRC Press.
- Williams GJ (2018). *rattle: Graphical User Interface for Data Science in R*. R package version 5.2.1, URL <https://rattle.togaware.com/>.

This document, sourced from DataO.Rnw bitbucket revision 291, was processed by KnitR version 1.20 of 2018-02-20 10:11:46 UTC and took 28.7 seconds to process. It was generated by gjw on Ubuntu 18.04.1 LTS.

Hands-On Data Science with R

From Data to R

Graham.Williams@togaware.com

13th June 2015

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

Data is available in an enormous variety of formats and stored on our own data stores, from databases, and openly from the Internet through sites like <http://data.gov> in the US, <http://data.gov.uk> in the UK, and <http://data.gov.au> in Australia. A major task we face as Data Scientists is in transferring that data into R. Being open source and with a focus on the freedom of sharing between multiple tools R provides extensive capabilities for reading data.

In this chapter we explore some of the options for reading data into R. We start with a simple and widely used format by reading a dataset from a comma separated values (CSV) file. We will continue with various file formats, move on to accessing data from databases, and discuss the growing need to read data from the Internet.

The required packages for this chapter include:

```
library(xlsx)    # Read Excel spreadsheets.  
library(RCurl)  
library(foreign)  
library(openxlsx)  
library(readxl)  # The definitive Excel reader.  
library(readr)   # Efficient reading of data.
```

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 © 2013-2015 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 To Do

Include Hadley's new haven package as useful for reading SAS, SPSS, and Stata files. <http://blog.rstudio.org/2015/03/04/haven-0-1-0/>.

Include Hadleys' new read.csv() from readr as well as read_excel() from readxl to read Excel spreadsheets.

Might also include read.nexus().

Include SQLite and perhaps make mention of sqldf to manipulate data using SQL although maybe that belongs in the BasicR chapter.

The foreign package may still be useful to access various format datasets.

2 Common Data Storage Formats

Data is available in a variety of forms and from a variety of sources.

.csv Comma Separated Values read.csv()

.

We generally store our data in R as a *data frame*. This is then the starting point for our data wrangling where we tidy our data ready for our analyses.

3 Reading from CSV

One of the simplest ways to load data into R is through the use of `read.csv()` which will read the contents of a CSV file and load them into an R data frame.

To illustrate we will read from a file in the `data` sub-directory of the current working directory. (Download the data file as <http://onepager.togaware.com/heart.csv>.) Check the current working directory using `getwd()`.

```
getwd()
## [1] "/home/gjw/projects/onepager"
```

The file itself will be listed as one of the CSV files in this directory, using `dir()`.

```
dir(path="data", pattern="*.csv")
## [1] "austdm_firecomp.csv"   "cmt_150514.csv"      "dvdtrans.csv"
## [4] "heart.csv"             "ozdata.csv"        "stroke.csv"
## [7] "WDI_Country.csv"       "WDI_CS_Notes.csv"   "WDI_csv.zip"
## [10] "WDI_Data.csv"          "WDI_Description.csv" "WDI_Footnotes.csv"
## [13] "WDI_Series.csv"         "WDI_ST_Notes.csv"    "weatherAUS.csv"
```

We now load the data from the CSV file using `read.csv()`.

```
heart <- read.csv(file=file.path("data", "heart.csv"))
```

For `read.csv()` we do not need to include the string `file=` part of the argument and in the commands below the `x=` and `object=` are also optional and can be dropped, relying on the position within the argument list to identify the formal argument. Once loaded review the data with `dim()`, `head()`, `tail()` and `str()`.

```
dim(x=heart)
## [1] 286 10
head(x=heart)
##   age sex chest_pain rest_bps chol fbs rest_ecg max_hr ex_ang disease
## 1 31 male   asympt     120   270   f  normal    153 yes positive
## 2 33 female   asympt     100   246   f  normal    150 yes positive
## 3 34 male   typ_angina   140   156   f  normal    180 no  positive
## 4 35 male   atyp_angina   110   257   f  normal    140 no  positive
## 5 36 male   atyp_angina   120   267   f  normal    160 no  positive
...
tail(x=heart)
##   age sex chest_pain rest_bps chol fbs rest_ecg max_hr
## 281 45 male   atyp_angina   140   224   t  normal    122
## 282 47 male   asympt     140   276   t  normal    125
## 283 48 female   atyp_angina   120   251   t st_t_wave_abnormality 148
## 284 54 female   atyp_angina   120   230   t  normal    140
## 285 55 male   atyp_angina   120   256   t  normal    137
...
```

```
str(object=heart)
## 'data.frame': 286 obs. of 10 variables:
## $ age      : int 31 33 34 35 36 37 38 38 38 41 ...
## $ sex      : Factor w/ 2 levels "female","male": 2 1 2 2 2 2 2 2 2 2 ...
## $ chest_pain: Factor w/ 4 levels "asympt","atyp_angina",...: 1 1 4 2 2 1 1...
## $ rest_bps  : int 120 100 140 110 120 140 110 120 92 110 ...
## $ chol      : int 270 246 156 257 267 207 196 282 117 289 ...
....
```

3.1 Always Review the Data

We might have noticed some other CSV files in the data folder listed earlier. We now load another of those files (download from <http://onepager.togaware.com/stroke.csv>).

```
stroke <- read.csv(file.path("data", "stroke.csv"))
```

Notice that we have dropped the `file=` part of the argument and rely on the fact that `read.csv()` expects the file to be the first argument.

```
str(read.csv)
## function (file, header=TRUE, sep=",", quote="\\"", dec=".",
##         fill=TRUE, comment.char="", ...)
```

Once loaded, we should always review the data. This is a very good habit to get into. As we have seen previously, `dim()`, `head()`, `tail()` and `str()` come in handy, as does `summary()`.

```
dim(stroke)
## [1] 829     1

head(stroke)
##   SEX.DIED.DSTR.AGE.DGN.COMA.DIAB.MINF.HAN
## 1 1;7.01.1991;2.01.1991;76;INF;0;0;1;0
## 2           1;.;3.01.1991;58;INF;0;0;0;0
## 3 1;2.06.1991;8.01.1991;74;INF;0;0;1;1
## 4 0;13.01.1991;11.01.1991;77;ICH;0;1;0;1
## 5 0;23.01.1996;13.01.1991;76;INF;0;1;0;1
...
tail(stroke)
##   SEX.DIED.DSTR.AGE.DGN.COMA.DIAB.MINF.HAN
## 824 0;.;23.12.1993;62;INF;0;0;0;1
## 825 0;.;26.12.1993;55;INF;0;1;1;1
## 826 0;20.06.1994;29.12.1993;93;INF;0;0;0;0
## 827 0;27.01.1994;31.12.1993;81;INF;1;0;0;0
## 828 0;.;31.12.1993;68;INF;0;0;0;1
...
str(stroke)
## 'data.frame': 829 obs. of  1 variable:
## $ SEX.DIED.DSTR.AGE.DGN.COMA.DIAB.MINF.HAN: Factor w/ 829 levels "0;10.03...
summary(stroke)
##   SEX.DIED.DSTR.AGE.DGN.COMA.DIAB.MINF.HAN
## 0;10.03.1992;1.03.1992;88;ID;1;0;0;1 : 1
## 0;10.03.1992;2.03.1992;87;INF;0;0;0;0 : 1
## 0;10.03.1993;19.02.1991;75;INF;0;0;0;1: 1
## 0;10.03.1993;8.02.1993;74;INF;0;1;0;1 : 1
## 0;.;10.04.1992;65;ID;0;0;0;0 : 1
...
```

Reviewing this data carefully we can see that it is not what we might be expecting. The data appears to have been read in as a single column, with a rather long column name beginning with “SEX.D” and finishing with “F.HAN”. The individual columns have not been extracted.

DRAFT

3.2 Choosing the Separator

If we look at the contents of `stroke.csv` (and based on our observations of the data we have loaded above) we see that there are no commas separating the columns in the file. Instead the columns are separated using a semicolon.

Strictly speaking `stroke.csv` is not the usual kind of CSV (comma separated value) file. Nonetheless, this is readily catered for in R through the argument `sep=` of `read.csv()`. This argument allows us to specify the correct separator.

```
stroke <- read.csv(file.path("data", "stroke.csv"), sep=";")  
dim(stroke)  
  
## [1] 829    9  
  
head(stroke)  
  
##   SEX      DIED      DSTR AGE DGN COMA DIAB MINF HAN  
## 1  1 7.01.1991 2.01.1991  76 INF    0    0    1    0  
## 2  1       . 3.01.1991   58 INF    0    0    0    0  
## 3  1 2.06.1991 8.01.1991   74 INF    0    0    1    1  
## 4  0 13.01.1991 11.01.1991   77 ICH    0    1    0    1  
## 5  0 23.01.1996 13.01.1991   76 INF    0    1    0    1  
....  
  
str(stroke)  
  
## 'data.frame': 829 obs. of  9 variables:  
## $ SEX : int  1 1 1 0 0 1 0 1 0 0 ...  
## $ DIED: Factor w/ 415 levels ".", "10.02.1993", ...: 374 1 179 61 214 61 46 ...  
## $ DSTR: Factor w/ 575 levels "10.01.1993", "10.02.1991", ...: 246 433 542 38 ...  
## $ AGE : int  76 58 74 77 76 48 81 53 73 69 ...  
## $ DGN : Factor w/ 4 levels "ICH", "ID", "INF", ...: 3 3 3 1 3 1 3 3 2 3 ...  
....
```

Note that `read.csv()` requires us to name the `sep=` argument in this instance. According to the manual page for `read.csv()` (and the output of `str()` below) the `sep=` argument is the third argument.

```
str(read.csv)  
  
## function (file, header=TRUE, sep=",", quote="\\"", dec=".",
##           fill=TRUE, comment.char="", ...)
```

We are using the argument in the second position in this call to `read.csv()`. The second argument position is reserved for `header=`.

The following are equivalent and it is useful to take a moment to understand what is happening here:

```
stroke <- read.csv(file.path("data", "stroke.csv"), sep=";")  
stroke <- read.csv(file.path("data", "stroke.csv"), header=TRUE, sep=";")  
stroke <- read.csv(file.path("data", "stroke.csv"), TRUE, ";")
```

The following results in an error, as we see:

```
stroke <- read.csv(file.path("data", "stroke.csv"), ";")  
## Error in !header: invalid argument type
```

DRAFT

3.3 Semicolon Separated Values

The use of semicolons to separate values within a row of a file is a special case of a CSV file. This is often used in countries where the comma is used as the decimal marker. R provides a special version of `read.csv()` called `read.csv2()` which defaults to using the semicolon as the field separator (`sep=";"`) and the comma as the decimal marker (`dec=", "`).

```
stroke <- read.csv2(file.path("data", "stroke.csv"))
```

As always, check the data we have just read to ensure it is as we expected.

```
dim(stroke)
## [1] 829    9

head(stroke)

##   SEX      DIED      DSTR AGE DGN COMA DIAB MINF HAN
## 1  1 7.01.1991 2.01.1991  76 INF    0    0    1    0
## 2  1          . 3.01.1991  58 INF    0    0    0    0
## 3  1 2.06.1991 8.01.1991  74 INF    0    0    1    1
## 4  0 13.01.1991 11.01.1991  77 ICH    0    1    0    1
## 5  0 23.01.1996 13.01.1991  76 INF    0    1    0    1
.....
tail(stroke)

##   SEX      DIED      DSTR AGE DGN COMA DIAB MINF HAN
## 824  0          . 23.12.1993  62 INF    0    0    0    1
## 825  0          . 26.12.1993  55 INF    0    1    1    1
## 826  0 20.06.1994 29.12.1993  93 INF    0    0    0    0
## 827  0 27.01.1994 31.12.1993  81 INF    1    0    0    0
## 828  0          . 31.12.1993  68 INF    0    0    0    1
.....
str(stroke)

## 'data.frame': 829 obs. of  9 variables:
## $ SEX : int  1 1 1 0 0 1 0 1 0 0 ...
## $ DIED: Factor w/ 415 levels ".", "10.02.1993", ...: 374 1 179 61 214 61 46 ...
## $ DSTR: Factor w/ 575 levels "10.01.1993", "10.02.1991", ...: 246 433 542 38...
## $ AGE : int  76 58 74 77 76 48 81 53 73 69 ...
## $ DGN : Factor w/ 4 levels "ICH", "ID", "INF", ...: 3 3 3 1 3 1 3 3 2 3 ...
.....
```

We will often be “pleasantly surprised” like this when using R. If we have a need to do something a little different, the chances will be that R supports it. Have a look at the documentation for `read.csv()` to glean a little of the flexibility of this particular function.

```
?read.csv
```

Notice `read.csv()` is actually just a call to the underlying `read.table()`, with some default options changed to suit CSV standard files.

```
read.csv  
## function (file, header=TRUE, sep=",", quote="\\"", dec=".,",  
##   fill=TRUE, comment.char="", ...)  
## read.table(file=file, header=header, sep=sep, quote=quote,  
##   dec=dec, fill=fill, comment.char=comment.char, ...)  
## <bytecode: 0x2478ba8>  
## <environment: namespace:utils>  
....
```

DRAFT

3.4 Strings as Strings

By default `read.csv` will treat columns consisting of strings as a factor with the levels corresponding to the different strings found in the data file. For data such as peoples names and addresses, we would want to retain these as strings rather than factors.

```
ds <- read.csv("stroke.csv", stringsAsFactors=FALSE)
```

4 White Space

We remove white space around values using `strip.white=TRUE`

DRAFT

5 Viewing the Data

```
View(stroke)  
  
library(RGtk2Extras)  
dfedit(stroke)  
  
library(Deducer)  
date.viewer()
```

DRAFT

6 Identifying Missing Values

A careful review of the stroke data we loaded above will identify that there are periods (i.e., ".") included in the data for DIED. Have a look at the tail of the dataset.

```
tail(stroke)
##      SEX      DIED      DSTR AGE DGN COMA DIAB MINF HAN
## 824    0       . 23.12.1993  62 INF    0    0    0    1
## 825    0       . 26.12.1993  55 INF    0    1    1    1
## 826    0 20.06.1994 29.12.1993  93 INF    0    0    0    0
## 827    0 27.01.1994 31.12.1993  81 INF    1    0    0    0
## 828    0       . 31.12.1993  68 INF    0    0    0    1
....
```

Our guess would be that these are used to indicate missing values (a common practise).

We can tell `read.csv()` about this using the `na.strings=` argument.

```
stroke <- read.csv2(file.path("data", "stroke.csv"), na.strings=".")
```

We review the resulting dataset.

```
dim(stroke)
## [1] 829   9
head(stroke)
##      SEX      DIED      DSTR AGE DGN COMA DIAB MINF HAN
## 1    1 7.01.1991 2.01.1991  76 INF    0    0    1    0
## 2    1 <NA> 3.01.1991  58 INF    0    0    0    0
## 3    1 2.06.1991 8.01.1991  74 INF    0    0    1    1
## 4    0 13.01.1991 11.01.1991  77 ICH    0    1    0    1
## 5    0 23.01.1996 13.01.1991  76 INF    0    1    0    1
....
tail(stroke)
##      SEX      DIED      DSTR AGE DGN COMA DIAB MINF HAN
## 824    0       <NA> 23.12.1993  62 INF    0    0    0    1
## 825    0       <NA> 26.12.1993  55 INF    0    1    1    1
## 826    0 20.06.1994 29.12.1993  93 INF    0    0    0    0
## 827    0 27.01.1994 31.12.1993  81 INF    1    0    0    0
## 828    0       <NA> 31.12.1993  68 INF    0    0    0    1
....
str(stroke)
## 'data.frame': 829 obs. of  9 variables:
## $ SEX : int  1 1 1 0 0 1 0 1 0 0 ...
## $ DIED: Factor w/ 414 levels "10.02.1993","10.03.1992",...: 373 NA 178 60 ...
## $ DSTR: Factor w/ 575 levels "10.01.1993","10.02.1991",...: 246 433 542 38...
## $ AGE : int  76 58 74 77 76 48 81 53 73 69 ...
## $ DGN : Factor w/ 4 levels "ICH","ID","INF",...: 3 3 3 1 3 1 3 3 2 3 ...
```

```
....
```

That looks better.

The value of the argument `na.strings=` can be a character vector, listing all the possibilities that we might come across to represent missing values in our file.

```
stroke <- read.csv2(file.path("data", "stroke.csv"), na.strings=c(".", "?", " "))
```

7 Specifying Data Types

```
sapply(stroke, class)
##      SEX      DIED      DSTR      AGE      DGN      COMA      DIAB
## "integer" "factor" "factor" "integer" "factor" "integer" "integer"
##      MINF      HAN
## "integer" "integer"

classes <- c("factor", "character", "character", "integer", "factor",
           "factor", "factor", "factor", "factor")
stroke <- read.csv2(file.path("data", "stroke.csv"), na.strings=".",
                     colClasses=classes)
sapply(stroke, class)
##      SEX      DIED      DSTR      AGE      DGN      COMA
## "factor" "character" "character" "integer" "factor" "factor"
##      DIAB      MINF      HAN
## "factor" "factor" "factor"
```

8 Reading Multiple Files

Here's some compact code to read all of the CSV files in a directory into one data frame. We assume they all have the same format:

```
fnames <- dir(pattern="\\.csv$")  
ds1    <- lapply(fnames, read.csv)  
ds     <- do.call("rbind", ds1)
```

We use to find all the csv files in a directory, load into a list and then collapse into a single data frame.

9 Reading From the Clipboard

This is an extremely useful tip particularly when following examples from the Internet. You might see something that looks like a well structured table with column headings and aligned rows. You want to get that quickly into R. Select the table and copy it into the clipboard (which happens automatically on Linux or using Ctrl-C or the Copy menu on Windows). Then in R:

```
ds <- read.table("clipboard")
```

DRAFT

10 Reading Microsoft Excel Spreadsheets

To ready Microsoft Excel spreadsheets we use `readxl::read_excel()` from `readxl` (Wickham, 2015). One downside is that it can not select regions to read the data from unlike `read.xlsx`.

Several packages are available, including `xlsx` (Dragulescu, 2014) and `openxlsx` (?). The former depends on Java and the `rJava` (Urbanek, 2013) package, whilst `openxlsx` does not, which may be an advantage given some of the common Java issues. An advantage of `xlsx` is that it can read specific row and column indicies with `RowIndex` and `colIndex`.

DRAFT

11 Writing to CSV

```
write(weather, file=file.path("data", "myweather.csv"), row.names=FALSE)
```

DRAFT

12 Saving RData

```
save(stroke, file=file.path("data", "stroke.RData"))
```

Exercise: Show size in memory and size on disk.

Exercise: Compare with dput() and dget() which convert r objects into an ascii text representation that is generally human readable. dget() recreates the R object.

13 Data from Spreadsheets

Exercise: Illustrate how to read libre office std format, MS/Excel format

DRAFT

14 Loading tab/txt Files

Exercise: Illustrate loading a tab delimited txt file.

DRAFT

15 Loading Fixed Width Files

Exercises: Illustrate reading a fixed width data file.

```
read.fwf()
```

DRAFT

16 Data from Internet Documents

A URL can be supplied to the `read.table()` family of functions, including `read.csv()`.

```
addr <- file.path("http://www.ats.ucla.edu/stat/r/examples/alda/data",
                  "tolerance1_pp.txt")
tolerance <- read.csv(addr)
```

As always, review the data.

```
dim(tolerance)
## [1] 80 6
head(tolerance)

##   id age tolerance male exposure time
## 1  9   11       2.23   0     1.54    0
## 2  9   12       1.79   0     1.54    1
## 3  9   13       1.90   0     1.54    2
## 4  9   14       2.12   0     1.54    3
## 5  9   15       2.66   0     1.54    4
.....
tail(tolerance)

##      id age tolerance male exposure time
## 75 1552 15       1.55   0     1.04    4
## 76 1653 11       1.11   0     1.25    0
## 77 1653 12       1.11   0     1.25    1
## 78 1653 13       1.34   0     1.25    2
## 79 1653 14       1.55   0     1.25    3
.....
str(tolerance)

## 'data.frame': 80 obs. of  6 variables:
## $ id      : int  9 9 9 9 9 45 45 45 45 ...
## $ age     : int  11 12 13 14 15 11 12 13 14 15 ...
## $ tolerance: num  2.23 1.79 1.9 2.12 2.66 1.12 1.45 1.45 1.45 1.99 ...
## $ male    : int  0 0 0 0 0 1 1 1 1 ...
## $ exposure: num  1.54 1.54 1.54 1.54 1.54 1.16 1.16 1.16 1.16 1.16 ...
## $ time    : num  0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 0.0000 ...
.....
summary(tolerance)

##      id           age      tolerance      male
## Min.   :  9.0   Min.   :11   Min.   :1.000   Min.   :0.0000
## 1st Qu.:410.0  1st Qu.:12   1st Qu.:1.220   1st Qu.:0.0000
## Median :673.5  Median :13   Median :1.500   Median :0.0000
## Mean   :762.8  Mean   :13   Mean   :1.619   Mean   :0.4375
## 3rd Qu.:1009.8 3rd Qu.:14   3rd Qu.:1.990   3rd Qu.:1.0000
.....
```

The data identifies a population of adolescents in a youth study. At particular ages their tolerance

to “deviant” behavior is recorded.

Having downloaded the data we may like to save it locally to file. Saving it as a binary R data file will use less disk space than the original CSV file, and retains the meta-data.

```
save(tolerance, file=file.path("data", "tolerance.RData"))
```

17 Data from Google Drive

We can load data into R from a spreadsheet stored on Google Drive by submitting a GET form that retrieves the data in a raw form. We can then parse the data into an R data frame. To access Internet based data we use `RCurl` (Temple Lang, 2015).

The first step is to identify the unique key that is connected to the file on your Google Drive. This can be obtained from Google Drive.

```
key <- "0Aonsf4v9iDjGdHRAWWRFbXdQN1ZvbGx0LWVCeVd0T1E"
```

Next we get the actual raw data, saving it into a variable.

```
tt <- getForm("https://spreadsheets.google.com/spreadsheet/pub",
              hl="en_US", key=key, output="csv")
tt

## [1] "<HTML>\n<HEAD>\n<TITLE>Moved Temporarily</TITLE>\n</HEAD>\n<BODY BGCO...
## attr(,"Content-Type")
##           charset
## "text/html"      "UTF-8"
```

Now we can read the data.

```
ds <- read.csv(textConnection(tt))
dim(ds)

## [1] 8 1

head(ds)

##
## 1
## 2
## 3
## 4
## 5
....
```

18 Data from Google Drive—read.csv() and https

Google changed over to serving all docs up using encryption (perhaps in 2012 or so) and thus automatically rewrites http: as https:. Unfortunately `read.csv()` can not handle encrypted connections (i.e., https:). So the method below will now result in an error.

First we might set up the URL to access:

```
library(RCurl)
key   <- "0AmbQbL4Lrd61dER5Qn13bHo4MkVNRlZ10VdicnZnTHc"
query <- curlEscape("select *")
addr  <- paste0("http://spreadsheets.google.com/tq?", 
               "key=", key,
               "&tq=", query,
               "&tqx=out:csv")
addr
## [1] "http://spreadsheets.google.com/tq?key=0AmbQbL4Lrd61dER5Qn13bHo4MkVNRl..."
```

Here we constructed a URL with the required information. To access the document the URL needs to include the spreadsheet key. The query we pass along uses SQL to select all columns and rows from the spreadsheet, and is constructed for sending through the URL using `curlEscape()`. Finally we generate the appropriate string that is the URL to send out to the Internet.

Pasting the address into a browser will work to download the file `data.csv`. In R though we will see an error:

```
ds <- read.csv(addr)
## Error in file(file, "rt"): cannot open the connection
```

19 Command Summary

This chapter has referenced the following R packages, functions, commands, operators, and datasets:

Complete this list.

`readr::read_csv()` *Function from readr.* Load data from a CSV file. This is fast enough, and so faster than `utils::read.csv()` but not as fast as the options in the `data.table` (?) package.

20 Exercises

Exercise .1 Read data from CSV file from the Internet

Exercise .2 Read data from a SQLite Database

DRAFT

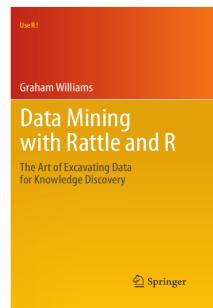
21 Further Reading

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:

- Any further reading?



22 References

- Dragulescu AA (2014). *xlsx: Read, write, format Excel 2007 and Excel 97/2000/XP/2003 files.* R package version 0.5.7, URL <http://CRAN.R-project.org/package=xlsx>.
- R Core Team (2015). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria. URL <http://www.R-project.org/>.
- Temple Lang D (2015). *RCurl: General network (HTTP/FTP/...) client interface for R.* R package version 1.95-4.6, URL <http://CRAN.R-project.org/package=RCurl>.
- Urbanek S (2013). *rJava: Low-level R to Java interface.* R package version 0.9-6, URL <http://CRAN.R-project.org/package=rJava>.
- Wickham H (2015). *readxl: Read Excel Files.* R package version 0.1.0, URL <http://CRAN.R-project.org/package=readxl>.
- 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.

This document, sourced from ReadO.Rnw revision 799, was processed by KnitR version 1.9 of 2015-01-20 and took 3.4 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04.2 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 8 cores and 12.3GB of RAM. It completed the processing 2015-06-13 11:51:19.

Hands-On Data Science with R Using CKAN Data Resources

Graham.Williams@togaware.com

19th December 2015

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

The data that sits behind reports that have been produced to guide policy development in government is an important resource that many governments are beginning to make available through various initiatives like <https://data.gov>, <https://data.gov.uk>, <https://data.gov.au>, and <https://data.gov.sg> to name just a few. In many cases the data is available through the Comprehensive Knowledge Archive Network (**CKAN**) application programming interface (**API**).

In this chapter we explore how to make access this public data accessible via the CKAN API utilizing `ckanr` (Chamberlain, 2015) and then perform a simple analysis of the dataset.

The required packages for this chapter include:

```
library(ckanr)    # Access data from CKAN.  
library(magrittr) # Use pipelines for data processing.  
library(readr)    # Modern data reader: read_csv().  
library(dplyr)    # Data wrangling: select().  
library(jsonlite) # Manage JSON objects.  
library(readxl)   # Read Excel spreadsheets.
```

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 © 2013-2015 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.



Draft Only

1 Introduction

Public and government datasets available through CKAN are a great source of open source data. Some of the available datasets contain data that underlies policy modelling. Having them available allows us to review the actual data that sits behind the policy decision making.

Here we get started using CKAN. After loading the `ckanr` package we can review the list of `ckanr::servers()` known to be offering up a CKAN API to their data holdings.

```
servers()

## [1] "http://catalog.data.gov"
## [2] "http://africaopendata.org"
## [3] "http://annuario.comune.fi.it"
## [4] "http://bermuda.io"
## [5] "http://catalogue.data.gov.bc.ca"
## [6] "http://catalogue.datalocale.fr"
## [7] "http://ckan.gsi.go.jp"
## [8] "http://dados.al.gov.br"
## [9] "http://dados.gov.br"
## [10] "http://dados.recife.pe.gov.br"
## [11] "http://dados.rs.gov.br"
## [12] "http://dadosabertos.senado.gov.br"
## [13] "http://dartportal.leeds.ac.uk"
## [14] "http://data.bris.ac.uk/data"
## [15] "http://data.buenosaires.gob.ar"
## [16] "http://data.cityofsantacruz.com"
## [17] "http://data.edostate.gov.ng"
## [18] "http://data.glasgow.gov.uk"
## [19] "http://data.go.id"
## [20] "http://data.gov.au"
## [21] "http://data.gov.hr"
## [22] "http://data.gov.ie"
## [23] "http://data.gov.ro"
## [24] "http://data.gov.sk"
## [25] "http://data.gov.uk"
....
```

For our purposes we will access a particular server. We can check which server is the default with `ckanr::get_default_url()`. To change to another server we set the environment variable `CKANR_DEFAULT_URL` using `base::Sys.setenv()`.

```
get_default_url()

## [1] "http://data.techno-science.ca/"

Sys.setenv(CKANR_DEFAULT_URL="http://data.gov.au")
get_default_url()

## [1] "http://data.gov.au"
```

The Australian server contains the public dataset that we will use to demonstrate access through the CKAN API.

Draft Only

2 Server Information

Now we can obtain meta data about the server itself and the extensions it supports.

```
ckan_info()  
## $ckan_version  
## [1] "2.3.1b"  
##  
## $site_url  
## [1] "http://data.gov.au"  
##  
## $site_description  
## [1] "Opening up government data"  
##  
## $site_title  
## [1] "data.gov.au"  
##  
## $error_emails_to  
## [1] "alex.sadleir@linkdigital.com.au"  
##  
## $locale_default  
## [1] "en_GB"  
##  
## $extensions  
## [1] "disqus"                      "dga_stats"  
## [3] "text_view"                     "webpage_view"  
## [5] "image_view"                    "recline_view"  
## [7] "datastore"                     "datapusher"  
## [9] "agsl"                          "datagovau"  
## [11] "datagovau_hierarchy"           "googleanalytics"  
## [13] "resource_proxy"                "spatial_metadata"  
## [15] "spatial_query"                 "harvest"  
## [17] "ckan_harvester"                "csw_harvester"  
## [19] "waf_harvester"                 "spatial_harvest_metadata_api"  
## [21] "ga-report"                    "sitemap"  
## [23] "sentry"                       "cesium_viewer"  
## [25] "wms_view"                     "kml_view"  
## [27] "geojson_view"                  "officedocs_view"  
## [29] "datajson_harvest"              "viewhelpers"  
## [31] "dashboard_preview"             "linechart"  
## [33] "barchart"                     "piechart"  
## [35] "basicgrid"                    "recline_grid_view"  
## [37] "recline_graph_view"            "recline_map_view"  
## [39] "pdf_view"                      "odata"  
## [41] "zip_view"
```

Draft Only

3 Organizations

Many organizations may contribute resources (such as datasets) to a repository. Resources are organized into packages and packages are provided by an organization.

To check how many organizations are represented on the server we might first query for the `ckanr::organization_list()`. Here we request it be provided in the raw JSON format direct from the API call (`as="json"`). We transform the JSON data structure into an R data structure (a list in this case) and then `magrittr::extract2()` the `result` of the CKAN query. We save the result locally to avoid having to query the server again if we wanted the same data (as we do below).

```
organization_list(as="json") %>%
  fromJSON(flatten=TRUE) %>%
  extract2('result') ->
  orgs
```

We can then report the number of organizations represented and the names of the columns contained in the organization table.

```
nrow(orgs)
## [1] 171
names(orgs)
##  [1] "image_display_url"   "display_name"      "description"
##  [4] "title"                "package_count"    "created"
##  [7] "approval_status"     "is_organization" "state"
## [10] "image_url"             "revision_id"      "packages"
## [13] "type"                  "id"                 "name"
```

Draft Only

4 Prettify Organizations

Using `jsonlite` (*Ooms et al., 2015*) we can display formatted JSON objects to make it considerably easier to review using `jsonlite::prettify()`.

```
orgs %>% toJSON() %>% pretty()  
## [  
##   {  
##     "image_display_url": "https://www.acnc.gov.au/images/ACNC_Logo.png",  
##     "display_name": "Australian Charities and Not-for-profits Commissi...",  
##     "description": "The independent national regulator of charities",  
##     "title": "Australian Charities and Not-for-profits Commission",  
##     "package_count": 5,  
##     "created": "2013-09-03T01:20:25.828188",  
##     "approval_status": "approved",  
##     "is_organization": true,  
##     "state": "active",  
##     "image_url": "https://www.acnc.gov.au/images/ACNC_Logo.png",  
##     "revision_id": "945f810c-e173-4a90-ba8f-70c9de6f69af",  
##     "packages": 5,  
##     "type": "organization",  
##     "id": "4d907503-b2aa-4c38-ac13-1273e791cb08",  
##     "name": "acnc"  
##   },  
##   {  
##     "image_display_url": "",  
##     "display_name": "ACT Government",  
##     "description": "",  
##     "title": "ACT Government",  
##     "package_count": 110,  
##     "created": "2015-09-16T03:52:10.428251",  
##     "approval_status": "approved",  
##     "is_organization": true,  
##     "state": "active",  
##     "image_url": "",  
##     "revision_id": "86c71d67-38e7-4569-b772-013b6f616d78",  
##     "packages": 110,  
##     "type": "organization",  
##     "id": "9ad7ef22-735a-45c1-9e83-0f702d56984b",  
##     "name": "act-government"  
##   },  
##   {  
##     "image_display_url": "https://data.gov.au/logos/aihw.gif",  
##     "display_name": "Australian Institute of Health and Welfare",  
##     "description": "The Australian Institute of Health and Welfare (AI...",  
##     "title": "Australian Institute of Health and Welfare",  
##     "package_count": 7,  
##     "created": "2013-05-12T09:15:15.624886",  
##   }]
```

Draft Only

5 Organizations and Packages

The dataset we are interested in has been provided by the Australian Taxation Office. Thus we want to find this organization in `org` dataset. A visual inspection of the dataset shows that the `title` column contains the appropriate strings for us to search.

```
orgs %>%
  extract("title") %>%
  ==('Australian Taxation Office') %>%
  which() %T>%
  print() ->
  ato

## [1] 29
```

There are 12 packages available from the ATO.

```
orgs[ato, 'packages']

## [1] 12
```

We will store the organization identification so as to investigate the packages further.

```
oid.ato <- orgs[ato, 'id'] %T>% print()

## [1] "90d1f157-c01f-4589-93bf-600dee01996e"
```

Out of interest we might select a few organizations before and after our organization of interest and list their number of packages.

```
orgs %>%
  '['(seq(ato-5, ato+5),) %>%
  select(name, packages)

##                                     name packages
## 24      australianinstituteofcriminology     1
## 25          australianmuseum            0
## 26 australianpesticidesandveterinarymedicinesauthority     1
## 27      australianprudentialregulationauthority    15
## 28      australianpublicservicecommission     16
## 29          australiantaxationoffice     12
## 30      bioregional-assessment-programme      0
## 31          brisbane-city-council      71
## 32 bureauofinfrastructuretransportandregionaleconomics     18
## 33          bureauofmeteorology        40
## 34      bureauofresourcesandenergyeconomics      0
```

Draft Only

6 Packages

Packages contain resources (often datasets). We can list the packages and note that by default `ckanr::package_list()` will limit the query to just 31 packages. We also note the number of packages available over the whole repository.

```
package_list(as="table")

## [1] "0-5m-contours"
## [2] "10m-contours"
## [3] "1-1-000-000-scale-australian-geoscience-map-sheet-index-national-geo..."
## [4] "1-100-000-scale-australian-geoscience-map-sheet-index-national-geosc..."
## [5] "1-250-000-scale-australian-geoscience-map-sheet-index-national-geosc..."
## [6] "19th-century-photographs-by-captain-samuel-sweet"
## [7] "19th-century-photographs-by-ernest-gall"
## [8] "19th-century-photographs-by-townsend-duryea"
## [9] "1m-contours"
## [10] "1-second-srtm-derived-hydrological-digital-elevation-model-dem-h-ver..."
## [11] "1-second-srtm-level-2-derived-digital-elevation-model-v1-063422"
## [12] "2001-02-to-2007-08-local-government-survey-victoria"
## [13] "2003-bushfire-affected-areas"
## [14] "2007-community-attitudes-to-privacy-survey-data"
## [15] "2008-assembly-election-summary-of-first-preference-votes-by-electora..."
## [16] "2008-assembly-election-summary-of-first-preference-votes-by-party-an..."
## [17] "2008-seismic-workstation-packages"
## [18] "2008-tree-canopy-urbanforest-7d1bf"
## [19] "2009-2059-act-population-projections"
## [20] "2009-2059-act-population-projections-females-by-single-year-of-age"
## [21] "2009-2059-act-population-projections-males-by-single-year-of-age"
## [22] "2009-2059-act-population-projections-persons-by-single-year-of-age"
## [23] "2009-green-light-report"
## [24] "2009-seismic-workstation-package"
## [25] "2011-seismic-workstation-packages"
## [26] "2011-tree-canopy-urbanforest-adec6"
## [27] "2012-offshore-petroleum-acreage-release-areas04479"
## [28] "2013-14-austender-ict-contract-statistics"
## [29] "2013-community-attitudes-to-privacy-survey-data"
## [30] "2013-offshore-petroleum-acreage-release-areasf2a22"
...
package_list(as="table", limit=NULL) %>% length()

## [1] 7118
```

Draft Only

7 Finding a Specific Package

To illustrate the use of the CKAN API we will access the controversial Corporate Tax Transparency dataset released 17 December 2015 by the Australian Taxation Office.

Then we search for a particular package hosted on the server. We are interested in one provided by the ATO.

```
package_search(q="australiantaxationoffice", as="json") %>%
  fromJSON(flatten=TRUE) %>%
  extract2('result') %>%
  extract2('results') ->
  pkgs.ato
names(pkgs.ato)

## [1] "license_title"                  "maintainer"
## [3] "relationships_as_object"       "jurisdiction"
## [5] "temporal_coverage_to"          "private"
## [7] "maintainer_email"               "num_tags"
## [9] "geospatial_topic"              "id"
## [11] "metadata_created"              "spatial_coverage"
...
nrow(pkgs.ato)
## [1] 10
```

Let's have a look at the titles to find the one we are interested in.

```
pkgs.ato$title

## [1] "Corporate Tax Transparency"
## [2] "Cumulative Total Tax Returns Received"
## [3] "Taxation Statistics 2010-11"
## [4] "Taxation Statistics 2009-10"
## [5] "Taxation statistics - individual sample files"
## [6] "Taxation Statistics 1994-95 to 1998-99"
## [7] "Ad-hoc requested data"
## [8] "Taxation Statistics 1999-00 to 2003-04"
## [9] "Taxation Statistics 2004-05 to 2008-09"
## [10] "ATO Web Analytics July 2013 to April 2014"

pkgs.ato$title %>%
  grep('Transparency', .) %>%
  extract(pkgs.ato$id, .) %T>%
  print() ->
  pid.ato.trans

## [1] "c2524c87-cea4-4636-acac-599a82048a26"
```

So that provides us with the package identifier for the Corporate Tax Transparency package from the ATO.

Draft Only

8 Resources

We can then list the contents of the package.

```
pkg.ato.trans <- package_show(pid.ato.trans, as="table") %>% print()  
## $license_title  
## [1] "Creative Commons Attribution 3.0 Australia"  
##  
## $maintainer  
## [1] "allanbarger"  
##  
....
```

This contains quite a bit of meta data.

```
names(pkg.ato.trans)  
## [1] "license_title"           "maintainer"  
## [3] "relationships_as_object" "jurisdiction"  
## [5] "temporal_coverage_to"     "private"  
## [7] "maintainer_email"         "num_tags"  
## [9] "geospatial_topic"         "id"  
## [11] "metadata_created"        "spatial_coverage"  
....  
pkg.ato.trans$name  
## [1] "corporate-transparency"  
pkg.ato.trans$metadata_created  
## [1] "2015-12-08T02:57:38.485976"  
pkg.ato.trans$license_title  
## [1] "Creative Commons Attribution 3.0 Australia"  
pkg.ato.trans$license_id  
## [1] "cc-by"  
pkg.ato.trans$type  
## [1] "dataset"  
pkg.ato.trans$num_resources  
## [1] 1
```

We see there is just 1 resource—this is a package containing a single dataset.

Draft Only

9 Resource Information

We can list some information about the dataset. The description reads:

This report contains the total income, taxable income and tax payable of over 1500 public and foreign private entities for the 2013-14 income year.

```
names(pkg.ato.trans$resources)
## [1] "cache_last_updated"      "package_id"
## [3] "webstore_last_updated"   "datastore_active"
## [5] "id"                      "size"
## [7] "wms_layer"               "state"
## [9] "hash"                     "description"
## [11] "format"                  "tracking_summary"
.....
pkg.ato.trans$resources[1,]$name
## [1] "2013-14 Report of Entity Tax Information"
pkg.ato.trans$resources[1,]$format
## [1] "XLSX"
pkg.ato.trans$resources[1,]$url
## [1] "http://data.gov.au/dataset/c2524c87-cea4-4636-acac-599a82048a26/resou..."
```

Draft Only

10 Download the Resource (Dataset)

From the supplied URL which points to an Excel spreadsheet we will want to extract the data of interest. Trial and error informs us that the second sheet of the spreadsheet contains the data of interest, though it contains three tables so we need to restrict the data to just the first table on that sheet.

In the following code block we record the url and the XLSX file name to create a temporary file into which we store the downloaded dataset. We then `utils::download.file()` and use `readxl::read_excel()` to save it into `transparency`. The `base::unlink()` does the house keeping to remove the downloaded file.

```
url <- pkg.ato.trans$resources[1,]$url
temp <- tempfile(fileext=".xlsx")
download.file(url, temp)
transparency <- read_excel(temp, sheet=2, skip=1)[1:1540, 2:6]
unlink(temp)
```

We now have a dataset supplied by a CKAN server available to us in R in a usable form as a data frame.

```
dim(transparency)
## [1] 1540      5
names(transparency)
## [1] "Name"           "ABN"             "Total Income $"
## [4] "Taxable Income $" "Tax Payable $"
head(transparency)
## Source: local data frame [6 x 5]
##
##           Name       ABN Total Income $
##           (chr)     (dbl)          (dbl)
## 1 3M AUSTRALIA PTY LTD 90000100096 452892659
## 2 A P EAGERS LIMITED 87009680013 2561960532
## 3 A2 AUSTRALIAN INVESTMENTS PTY LTD 93126014275 110173384
## 4 AAPC LIMITED 87009175820 570890761
## 5 ABACUS GROUP HOLDINGS LIMITED 31080604619 198340272
## 6 ABB GROUP INVESTMENT MANAGEMENT PTY LTD 47082803852 1173330692
....
```

We now have a dataset supplied by a CKAN server available to us in R in a usable form as a data frame.

Draft Only

11 Data Wrangling

```
ds <- transparency
names(ds) <- c("name", "abn", "total", "taxable", "payable")
ds %>% mutate(per=100*payable/total, rate=100*payable/taxable)
```

Draft Only

Data Science with R

Hands-On

Using CKAN Data Resources

12 Command Summary

This chapter has introduced, demonstrated and described the following R packages, functions, commands, operators, and datasets:

Draft Only

Data Science with R

Hands-On

Using CKAN Data Resources

13 Exercises

Draft Only

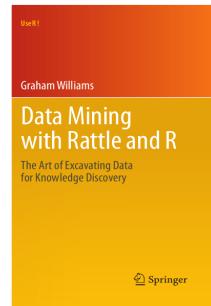
Data Science with R

Hands-On

Using CKAN Data Resources

14 Further Reading

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.

We identify below other resources that augment the material we have presented in this chapter.

•

Draft Only

Data Science with R

Hands-On

Using CKAN Data Resources

15 References

- Chamberlain S (2015). *ckanr: Client for the Comprehensive Knowledge Archive Network ('CKAN') 'API'*. R package version 0.1.0, URL <https://CRAN.R-project.org/package=ckanr>.
- Ooms J, Temple Lang D, Hilaiel L (2015). *jsonlite: A Robust, High Performance JSON Parser and Generator for R*. R package version 0.9.19, URL <https://CRAN.R-project.org/package=jsonlite>.
- R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- 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.

This document, sourced from CkanO.Rnw bitbucket revision 17, was processed by KnitR version 1.9 of 2015-01-20 and took 9.6 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04.3 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 8 cores and 12.3GB of RAM. It completed the processing 2015-12-19 14:22:24.

Draft Only

Generated 2015-12-19 14:22:30+11:00

Data Science with R

Summarising Data

Graham.Williams@togaware.com

9th June 2014

Visit <http://onepager.togaware.com/> for more OnePageR's.

The required packages for this module include:

```
library(rattle)      # The weatherAUS dataset.  
library(plyr)        # Group by operations.
```

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 Load the Data

We use the full **weatherAUS** dataset from **rattle** (Williams, 2014) to illustrate data summarisation over a more complex dataset.

```
ds <- weatherAUS
names(ds) <- normVarNames(names(ds)) # Lower case variable names.
names(ds)

## [1] "date"          "location"       "min_temp"
## [4] "max_temp"      "rainfall"        "evaporation"
## [7] "sunshine"       "wind_gust_dir"   "wind_gust_speed"
## [10] "wind_dir_9am"  "wind_dir_3pm"    "wind_speed_9am"
...
head(ds)

##           date location min_temp max_temp rainfall evaporation sunshine
## 1 2008-12-01  Albury     13.4     22.9      0.6        NA        NA
## 2 2008-12-02  Albury      7.4      25.1      0.0        NA        NA
## 3 2008-12-03  Albury     12.9      25.7      0.0        NA        NA
...
tail(ds)

##           date location min_temp max_temp rainfall evaporation sunshine
## 88763 2014-04-20  Uluru     10.3     29.6      0        NA        NA
## 88764 2014-04-21  Uluru     11.3     30.5      0        NA        NA
## 88765 2014-04-22  Uluru     10.1     31.6      0        NA        NA
...
ds[sample(nrow(ds), 6),]

##           date location min_temp max_temp rainfall evaporation sunshine
## 42691 2011-01-30 Melbourne    16.4     38.1      0.0       7.4
## 74988 2010-03-21    Perth     19.5     33.1      0.0       6.0
## 46982 2011-08-14  Portland    2.5      15.4      0.2       1.0
...
str(ds)

## 'data.frame': 88768 obs. of  24 variables:
## $ date        : Date, format: "2008-12-01" "2008-12-02" ...
## $ location    : Factor w/ 49 levels "Adelaide","Albany",...: 3 3 3 3 3 ...
## $ min_temp    : num  13.4 7.4 12.9 9.2 17.5 14.6 14.3 7.7 9.7 13.1 ...
...
summary(ds)

##           date          location      min_temp      max_temp
## Min.   :2007-11-01  Canberra: 2279  Min.   :-8.5  Min.   :-3.8
## 1st Qu.:2010-03-08  Sydney   : 2187  1st Qu.: 7.6  1st Qu.:18.0
## Median :2011-08-03  Adelaide: 2036  Median :12.0  Median :22.5
...

```

2 Dataset Indexing

Often we will be on the lookout for oddities or data typing that need fixing up. Once identified we will use the operations covered in a separate session on *Transforming* data.

We start by looking at some of the data. This introduces the concept of indexing our data frame.

```
ds[1,]                      # First observation.

##           date location min_temp max_temp rainfall evaporation sunshine
## 1 2008-12-01    Albury     13.4     22.9      0.6        NA        NA
##   wind_gust_dir wind_gust_speed wind_dir_9am wind_dir_3pm wind_speed_9am
## 1                  W                 44                  W                WNW                 20
## ...
## ...
ds[1,1]                      # First observation's first variable.

## [1] "2008-12-01"

ds[1:2,]                      # First two observations.

##           date location min_temp max_temp rainfall evaporation sunshine
## 1 2008-12-01    Albury     13.4     22.9      0.6        NA        NA
## 2 2008-12-02    Albury      7.4     25.1      0.0        NA        NA
##   wind_gust_dir wind_gust_speed wind_dir_9am wind_dir_3pm wind_speed_9am
## ...
## ...
ds[1:2, 3:4]                   # First two observations and variables 3 and 4.

##   min_temp max_temp
## 1     13.4     22.9
## 2      7.4     25.1

head(ds[3:4], 2)              # Single dimension treated as variable index.

##   min_temp max_temp
## 1     13.4     22.9
## 2      7.4     25.1

head(ds[,3:4], 2)             # Or we can leave the observation index empty.

##   min_temp max_temp
## 1     13.4     22.9
## 2      7.4     25.1
```

3 Textual Summaries

The `summary()` command provides a quick univariate overview of our dataset.

```
summary(ds, digits=6)

##      date          location      min_temp      max_temp
## Min. :2007-11-01  Canberra: 2279  Min.   :-8.5  Min.   :-3.8
## 1st Qu.:2010-03-08  Sydney   : 2187  1st Qu.: 7.6  1st Qu.:18.0
## Median :2011-08-03  Adelaide: 2036  Median  :12.0  Median  :22.5
## Mean   :2011-07-29  Brisbane: 2036  Mean    :12.2  Mean    :23.1
## 3rd Qu.:2012-11-27  Darwin   : 2036  3rd Qu.:16.8  3rd Qu.:28.0
## Max.   :2014-04-25  Hobart   : 2036  Max.    :33.9  Max.    :48.1
## (Other) :76158     NA's     :669   NA's    :493

##      rainfall      evaporation      sunshine      wind_gust_dir
## Min.   : 0.0   Min.   : 0       Min.   : 0       W      : 5850
## 1st Qu.: 0.0   1st Qu.: 3       1st Qu.: 5       SE     : 5796
## Median : 0.0   Median : 5       Median : 8       N      : 5739
## Mean   : 2.5   Mean   : 5       Mean   : 8       S      : 5669
## 3rd Qu.: 0.8   3rd Qu.: 7       3rd Qu.:11      SSE    : 5583
## Max.   :371.0  Max.   :82       Max.   :14      (Other):53353
## NA's   :1656   NA's   :32687   NA's   :35530   NA's   : 6778
## wind_gust_speed  wind_dir_9am  wind_dir_3pm  wind_speed_9am
## Min.   : 6       N      : 7200   SE     : 6928   Min.   : 0.0
## 1st Qu.: 31      SE     : 5668   W     : 6115   1st Qu.: 7.0
## Median : 39      E      : 5568   S     : 6071   Median :13.0
## Mean   : 40      SSE    : 5508   WSW   : 5846   Mean   :14.2
## 3rd Qu.: 48      S      : 5345   SSE   : 5804   3rd Qu.:20.0
## Max.   :135     (Other):52838  (Other):56048  Max.   :87.0
## NA's   :6738   NA's   : 6641   NA's   : 1956  NA's   :1159
## wind_speed_3pm  humidity_9am  humidity_3pm  pressure_9am
## Min.   : 0.0   Min.   : 0.0   Min.   : 0.0   Min.   : 980
## 1st Qu.:13.0  1st Qu.: 57.0  1st Qu.: 37.0  1st Qu.:1013
## Median :19.0  Median : 70.0  Median : 52.0  Median :1017
## Mean   :18.8  Mean   : 68.7  Mean   : 51.6  Mean   :1017
## 3rd Qu.:24.0  3rd Qu.: 83.0  3rd Qu.: 66.0  3rd Qu.:1022
## Max.   :87.0  Max.   :100.0  Max.   :100.0  Max.   :1041
## NA's   :1170  NA's   :1495   NA's   :1389   NA's   :8273
## pressure_3pm  cloud_9am   cloud_3pm   temp_9am
## Min.   : 979  Min.   :0       Min.   :0       Min.   :-5.9
## 1st Qu.:1010 1st Qu.:1       1st Qu.:2       1st Qu.:12.3
## Median :1015  Median :5       Median :5       Median :16.7
## Mean   :1015  Mean   :4       Mean   :4       Mean   :17.0
## 3rd Qu.:1020  3rd Qu.:7       3rd Qu.:7       3rd Qu.:21.5
## Max.   :1040  Max.   :9       Max.   :9       Max.   :40.2
## NA's   :8245  NA's   :32337  NA's   :33339  NA's   :1040
....
```

4 Textual Summaries—Warning

Do be weary of the results provided by `summary()`. The `summary()` command rounds the results to 4 digits by default. This can surprise us sometimes when we find `min()` and the reported minimum value from `summary()` disagree! Let's look at some random data and notice the reported minimum value.

```
eg <- sample(1e6:(1e7-1), 100)
max(eg)
## [1] 9882363

min(eg)
## [1] 1146522

summary(eg)

##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
## 1150000 3050000 5120000 5350000 8050000 9880000

summary(eg, digits=4)

##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
## 1147000 3051000 5123000 5348000 8051000 9882000

summary(eg, digits=5)

##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
## 1146500 3050700 5123000 5348100 8050900 9882400

summary(eg, digits=6)

##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
## 1146520 3050710 5123030 5348100 8050880 9882360

summary(eg, digits=7)

##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
## 1146522 3050710 5123028 5348103 8050881 9882363
```

5 PlyR: Summarise per Group to new Data Frame

The `plyr` (Wickham, 2014) package provides a clean and consistent approach to transforming data. We can easily, for example, transform a data frame into a new smaller data frame grouped by the location.

```
temp <- ddply(ds, "location", summarise,
               max=max(max_temp, na.rm=TRUE),
               min=min(min_temp, na.rm=TRUE))
temp
##          location  max  min
## 1      Adelaide 45.7  0.7
## 2      Albany   38.9  1.8
## 3     Albury   44.8 -2.5
....
```

The `plyr` package also provides the `.()` function as a convenient mechanism for listing variable names without the need to quote them. The function becomes more convenient when we have multiple variables to list.

```
temp <- ddply(ds, .(location), summarise,
               max=max(max_temp, na.rm=TRUE),
               min=min(min_temp, na.rm=TRUE))
temp
##          location  max  min
## 1      Adelaide 45.7  0.7
## 2      Albany   38.9  1.8
## 3     Albury   44.8 -2.5
....
```

We can review the resulting values, ordered by the maximum temperature.

```
temp[order(temp$max, decreasing=TRUE),]
##          location  max  min
## 49      Woomera 48.1  0.7
## 22      Moree   47.3 -3.3
## 20 MelbourneAirport 46.8 -0.4
....
```

Similarly, but ordered by the minimum temperature.

```
head(temp[order(temp$min),])
##          location  max  min
## 24 MountGinini 31.1 -8.5
## 41 Tuggeranong 40.1 -8.2
## 10 Canberra    42.0 -8.0
....
```

6 PlyR: Summarise per Group to Original Data Frame

Transform a data frame by adding the group summaries per original observation, simply by replacing `summarise` with `transform`

```
temps <- ddply(ds, .(location), transform,
               max=max(max_temp, na.rm=TRUE),
               min=min(min_temp, na.rm=TRUE))
```

Now notice that the top few values for `min` and `max` are constant, since they belong to the same group (`Adelaide`).

```
head(temps[c("date", "location", "min_temp", "min", "max_temp", "max")])
##           date location min_temp min max_temp max
## 1 2008-07-01 Adelaide     8.8 0.7    15.7 45.7
## 2 2008-07-02 Adelaide    12.7 0.7    15.8 45.7
## 3 2008-07-03 Adelaide     6.2 0.7    15.1 45.7
....
```

If we sample a few observations we see the various values of `min` and `max` across different `locations`.

```
temps[sample(nrow(temps), 10),
      c("date", "location", "min_temp", "min", "max_temp", "max")]
##           date location min_temp min max_temp max
## 88579 2013-10-18 Woomera     7.7 0.7    27.8 48.1
## 87470 2010-07-08 Woomera     0.7 0.7    15.3 48.1
## 77879 2009-09-08 Walpole     5.5 3.4    17.8 39.3
....
```

7 PlyR: Select One Observation Per Group

We can also select a single observation per group, using some criteria to decide which observation to pick. We replace the `summarise` or `transform` with a function to select the observation of interest.

```
temps <- ddply(ds, .(location),
                 function(x) x[x$max_temp == max(x$max_temp, na.rm=TRUE),])
head(temps[1:7])

##           date location min_temp max_temp rainfall evaporation sunshine
## 1       <NA>     <NA>      NA       NA       NA       NA       NA
## 2 2009-01-28 Adelaide    30.7    45.7       0     13.0    12.5
## 3 2010-01-18 Albany     17.8    38.9       0     11.8    12.8
....
```

Notice the unexpected rows of missing values. The vector comparison, using `==()`, will return `NA` whenever comparing `NA`'s and an index to `[]` of `NA` will return an `NA` row for each observation. We can get around this issue of missing values by testing whether we get `TRUE` from the comparison, rather than `FALSE` or `NA` by using `identical()`.

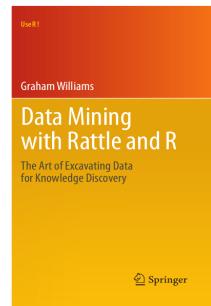
```
temps <- ddply(ds, .(location),
                 function(x) x[sapply(x$max_temp == max(x$max_temp, na.rm=TRUE),
                                       identical, TRUE),])
head(temps[1:7])

##           date location min_temp max_temp rainfall evaporation sunshine
## 1 2009-01-28 Adelaide    30.7    45.7       0     13.0    12.5
## 2 2010-01-18 Albany     17.8    38.9       0     11.8    12.8
## 3 2009-02-07 Albury     22.3    44.8       0       NA       NA
....
```

8 Further Reading

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 module is one of many OnePageR modules available from <http://onepager.togaware.com>. In particular follow the links on the website with a * which indicates the generally more developed OnePageR modules.



9 References

- 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/>.
- Wickham H (2014). *plyr: Tools for splitting, applying and combining data*. R package version 1.8.1, URL <http://CRAN.R-project.org/package=plyr>.
- 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 3.0.4, URL <http://rattle.togaware.com/>.

This document, sourced from SummaryO.Rnw revision 419, was processed by KnitR version 1.6 of 2014-05-24 and took 3.6 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-06-09 10:27:58.

One Page R Data Science Visual Discovery

Graham.Williams@togaware.com

3rd June 2018

Visit <https://essentials.togaware.com/onepagers> for more Essentials.

One of the most important tasks for any data scientist is to visualise data. Presenting data visually will often lead to new insights and discoveries, as well as providing glimpses in any issues with the data itself. Sometimes these will stand out in a visual presentation whilst scanning the data textually can often hide distributional wobble. A visual presentation is also an effective means for communicating insight to others.

20180603

R offers a comprehensive suite of tools to visualise data, with `ggplot2` (Wickham and Chang, 2016) being dominate amongst them. The `ggplot2` package implements a grammar for writing sentences describing the graphics. Using this package we construct a plot beginning with the dataset along with the aesthetics (e.g., the x-axis and y-axis) and then adding geometric elements, statistical operations, scales, facets, coordinates, and numerous other components to the plot.

In this guide we explore data using `ggplot2` and affiliated packages to gain insight into our data. The package provides an extensive collection of capabilities offering an infinite variety of visual possibilities. We will present some basics as a launching pad for plotting data but note that further opportunities abound and are well covered in many other resources many of which are available online.

Through this guide new R commands will be introduced. The reader is encouraged to review the command's documentation and understand what the command does. Help is obtained using the `? command as in:`

```
?read.csv
```

Documentation on a particular package can be obtained using the `help=` option of `library()`:

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

This chapter is intended to be hands on. To learn effectively the reader is encouraged to run R locally (e.g., RStudio or Emacs with ESS mode) and to replicate all commands as they appear here. Check that output is the same and it is clear how it is generated. Try some variations. Explore.

Copyright © 2000-2018 Graham Williams. This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#) allowing this work to be copied, distributed, or adapted, with attribution and provided under the same license.



1 Packages from the R Library

Packages used in this chapter include `dplyr` (Wickham *et al.*, 2018), `ggplot2`, `magrittr` (Bache and Wickham, 2014), `randomForest` (Breiman *et al.*, 2018), `rattle.data` (Williams, 2017c), `scales` (Wickham, 2017), `stringr` (Wickham, 2018), and, `rattle` (Williams, 2017b),

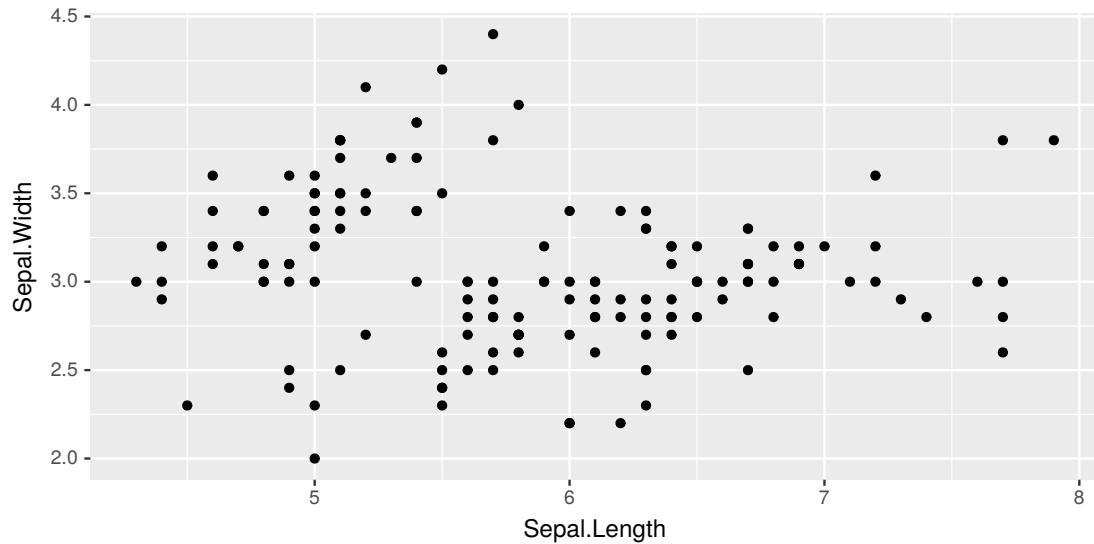
Packages are loaded here into the R session from the local library folders. Any packages that are missing can be installed using `utils::install.packages()`.

```
# Load required packages from local library into the R session.

library(dplyr)      # glimpse().
library(ggplot2)    # Visualise data.
library(magrittr)   # Data pipelines: %>% %<% %T>% equals().
library(randomForest) # na.roughfix() for missing data.
library(rattle)     # normVarNames().
library(rattle.data) # weatherAUS.
library(scales)     # commas(), percent().
library(stringr)   # str_replace_all().
```

2 Quickstart Scatter Plot

20180603



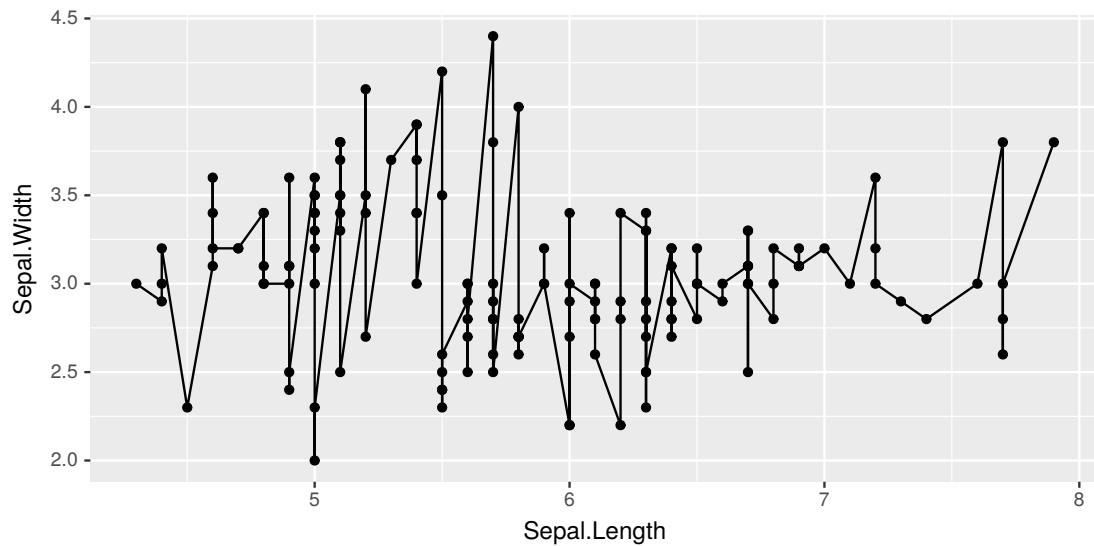
Here we illustrate the simplest of examples. The `datasets::iris` dataset is a widely used sample dataset in statistics and commonly used to illustrate concepts in statistical model building. Above we see a simple `scatter plot` which displays a two dimensional plot. By dimensions we refer to the x and y axes and associate one dimension (i.e., variable) to each axis. The observations from the dataset, which are the rows in the dataset, are then plotted, scattering the points over the plot.

To build this plot we reference the `datasets::iris` dataset in the computer's memory and pipe (`stringr::%>%`) it into `ggplot2::ggplot()`. The aesthetics `ggplot2::aes()` are set up with `Sepal.Length` set as the x= axis and `Sepal.Width` as the y= axis. To add the points (observations) to the plot we employ `ggplot2::geom_point()`.

```
iris %>%
  ggplot(aes(x=Sepal.Length, y=Sepal.Width)) +
  geom_point()
```

3 Adding a Line

20180603

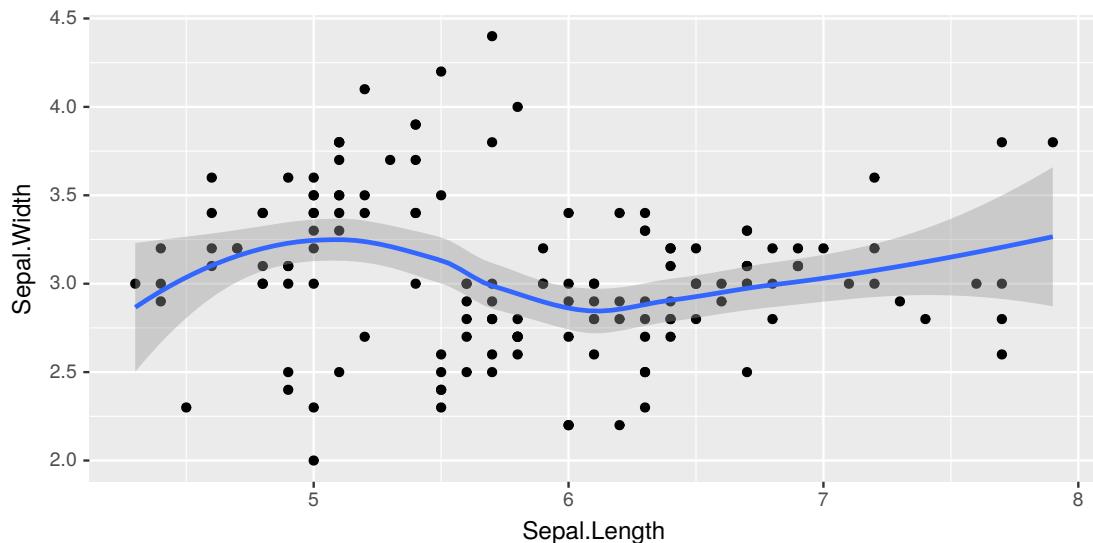


As an alternative to plotting the points we could join the dots and plot a line. One reason for doing so might be to attempt to observe more clearly any relationship between the two dimensions (i.e., the two variables we are plotting). Add a `ggplot2::geom_line()` to the canvas we can draw a line from left to right the follows the points exactly. The result is a rather jugged line and it is debatable whether it has added any further insight into our understanding of the data.

```
iris %>%
  ggplot(aes(x=Sepal.Length, y=Sepal.Width)) +
  geom_point() +
  geom_line()
```

4 Adding a Smooth Line

20180603



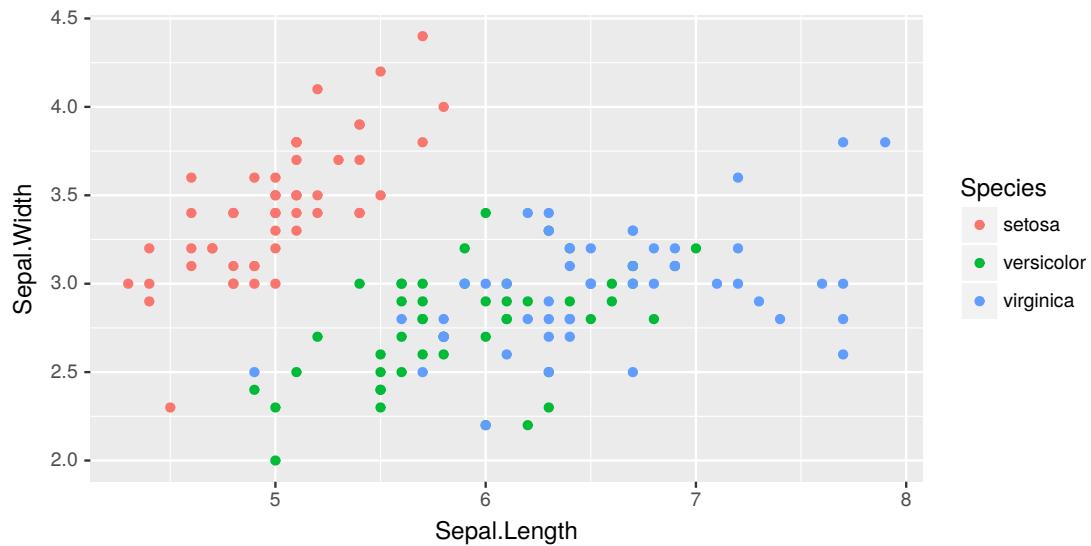
The rather jagged line from point to point does not give a very convincing picture of any relationship between the two variables. A common approach is to introduce a statistical function to “fit” a single smoother line to the points. We aim to draw a smoother line from left to right that follows the points but without the requirement of going through every point.

The resulting plot here uses `ggplot2::geom_smooth()` with a so called [locally weighted scatterplot smoothing](#) method or LOESS to produce the smoothed line. As a result we can observe statistically something of a relationship between the two variables although it is not a simple relationship.

```
iris %>%
  ggplot(aes(x=Sepal.Length, y=Sepal.Width)) +
  geom_point() +
  stat_smooth(method="loess")
```

5 Colour the Points

20180603



A key variable in the `datasets::iris` dataset is the `Species`. By colouring the dots according to the `Species` we suddenly identify a quite significant relationship in the data. Indeed we might call this knowledge discovery. It is very clear that all of the observations in the top left belong to the `setosa` species whilst the next clump is primarily `versicolor` and the right hand observations are `virginica`. It is clear that there is some so-called *structure* in this dataset.

```
iris %>%
  ggplot(aes(x=Sepal.Length, y=Sepal.Width, colour=Species)) +
  geom_point()
```

The colour is added simply by specifying a further aesthetic, `colour=Species`. This results in the different values of the variable `Species` being coloured differently.

6 Initialise the Weather Dataset

The modestly large **weatherAUS** dataset from **rattle.data** is used to further illustrate the capabilities of **ggplot2**. For plots which generate large images a random subset of the same dataset is deployed to allow replication in a timely manner. The dataset is loaded and processed following the template introduced in Williams (2017a).

```
# Initialise the dataset as per the template.

library(rattle)

dsname <- "weatherAUS"
ds     <- get(dsname)
```

The dataset can be summarised using `tibble::glimpse()`.

```
glimpse(ds)

## Observations: 145,460
## Variables: 24
## $ Date      <date> 2008-12-01, 2008-12-02, 2008-12-03, ...
## $ Location   <fct> Albury, Albury, Albury, Albu...
## $ MinTemp    <dbl> 13.4, 7.4, 12.9, 9.2, 17.5, 14.6, 14...
## $ MaxTemp    <dbl> 22.9, 25.1, 25.7, 28.0, 32.3, 29.7, ...
## $ Rainfall   <dbl> 0.6, 0.0, 0.0, 0.0, 1.0, 0.2, 0.0, 0...
## $ Evaporation <dbl> NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ Sunshine   <dbl> NA, NA, NA, NA, NA, NA, NA, NA, ...
## $ WindGustDir <ord> W, WNW, WSW, NE, W, WNW, W, W, NNW, ...
## $ WindGustSpeed <dbl> 44, 44, 46, 24, 41, 56, 50, 35, 80, ...
## $ WindDir9am  <ord> W, NNW, W, SE, ENE, W, SW, SSE, SE, ...
## $ WindDir3pm  <ord> WNW, WSW, WSW, E, NW, W, W, NW, S...
## $ WindSpeed9am <dbl> 20, 4, 19, 11, 7, 19, 20, 6, 7, 15, ...
## $ WindSpeed3pm <dbl> 24, 22, 26, 9, 20, 24, 24, 17, 28, 1...
## $ Humidity9am <int> 71, 44, 38, 45, 82, 55, 49, 48, 42, ...
## $ Humidity3pm <int> 22, 25, 30, 16, 33, 23, 19, 19, 9, 2...
## $ Pressure9am <dbl> 1007.7, 1010.6, 1007.6, 1017.6, 1010...
## $ Pressure3pm <dbl> 1007.1, 1007.8, 1008.7, 1012.8, 1006...
## $ Cloud9am    <int> 8, NA, NA, NA, 7, NA, 1, NA, NA, ...
## $ Cloud3pm    <int> NA, NA, 2, NA, 8, NA, NA, NA, NA...
## $ Temp9am     <dbl> 16.9, 17.2, 21.0, 18.1, 17.8, 20.6, ...
## $ Temp3pm     <dbl> 21.8, 24.3, 23.2, 26.5, 29.7, 28.9, ...
## $ RainToday   <fct> No, No, No, No, No, No, No, No, ...
## $ RISK_MM     <dbl> 0.0, 0.0, 0.0, 1.0, 0.2, 0.0, 0.0, 0...
## $ RainTomorrow <fct> No, No, No, No, No, No, No, Yes, ...
```

7 Normalise Variable Names

Variable names are normalised so as to have some certainty in interacting with the data. The convenience function `rattle::normVarNames()` can do this.

```
# Review the variables before normalising their names.

names(ds)

## [1] "Date"          "Location"       "MinTemp"
## [4] "MaxTemp"       "Rainfall"        "Evaporation"
## [7] "Sunshine"      "WindGustDir"     "WindGustSpeed"
## [10] "WindDir9am"    "WindDir3pm"      "WindSpeed9am"
.....
# Capture the original variable names for use in plots.

vnames <- names(ds)

# Normalise the variable names.

names(ds) %>% normVarNames()

# Confirm the results are as expected.

names(ds)

## [1] "date"          "location"       "min_temp"
## [4] "max_temp"      "rainfall"        "evaporation"
## [7] "sunshine"      "wind_gust_dir"  "wind_gust_speed"
## [10] "wind_dir_9am" "wind_dir_3pm"   "wind_speed_9am"
.....
# Index the original variable names by the new names.

names(vnames) <- names(ds)

vnames

##           date      location      min_temp
## "Date"      "Location"    "MinTemp"
## "max_temp"  "rainfall"    "evaporation"
## "MaxTemp"   "Rainfall"    "Evaporation"
.....
```

The variable names now conform to our expectations of them and in accordance to our chosen style as documented in

8 Variables and Model Target

Reviewing the variables we will note the different roles each of the variables play. Again we make use of the template in identifying variable roles.

```
# Note the available variables.

vars <- names(ds) %T>% print()

## [1] "date"          "location"       "min_temp"
## [4] "max_temp"      "rainfall"        "evaporation"
## [7] "sunshine"       "wind_gust_dir"   "wind_gust_speed"
## [10] "wind_dir_9am"  "wind_dir_3pm"    "wind_speed_9am"
## [13] "wind_speed_3pm" "humidity_9am"   "humidity_3pm"
## [16] "pressure_9am"  "pressure_3pm"   "cloud_9am"
## [19] "cloud_3pm"     "temp_9am"       "temp_3pm"
## [22] "rain_today"    "risk_mm"        "rain_tomorrow"

# Note the target variable.

target <- "rain_tomorrow"

# Place the target variable at the beginning of the vars.

vars <- c(target, vars) %>% unique() %T>% print()

## [1] "rain_tomorrow"  "date"          "location"
## [4] "min_temp"       "max_temp"      "rainfall"
## [7] "evaporation"   "sunshine"      "wind_gust_dir"
## [10] "wind_gust_speed" "wind_dir_9am"  "wind_dir_3pm"
## [13] "wind_speed_9am"  "wind_speed_3pm" "humidity_9am"
## [16] "humidity_3pm"   "pressure_9am"  "pressure_3pm"
## [19] "cloud_9am"      "cloud_3pm"     "temp_9am"
## [22] "temp_3pm"       "rain_today"    "risk_mm"
```

9 Modelling Roles

```

# Note the risk variable which measures the severity of the outcome.

risk <- "risk_mm"

# Note the identifiers.

id <- c("date", "location")

# Initialise ignored variables: identifiers.

ignore <- c(risk, id)

# Remove the variables to ignore.

vars <- setdiff(vars, ignore)

# Identify the input variables for modelling.

inputs <- setdiff(vars, target) %T>% print()

## [1] "min_temp"          "max_temp"          "rainfall"
## [4] "evaporation"       "sunshine"          "wind_gust_dir"
## [7] "wind_gust_speed"   "wind_dir_9am"     "wind_dir_3pm"
## [10] "wind_speed_9am"    "wind_speed_3pm"   "humidity_9am"
## [13] "humidity_3pm"      "pressure_9am"    "pressure_3pm"
## [16] "cloud_9am"         "cloud_3pm"        "temp_9am"
## [19] "temp_3pm"          "rain_today"

# Also record them by indicies.

inputi <-
  inputs %>%
  sapply(function(x) which(x == names(ds)), USE.NAMES=FALSE) %T>%
  print()

## [1] 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21
## [20] 22

```

10 Variable Types

```
# Identify the numeric variables by index.

numi <-
  ds %>%
  sapply(is.numeric) %>%
  which() %>%
  intersect(inputi) %T>%
  print()

## [1] 3 4 5 6 7 9 12 13 14 15 16 17 18 19 20 21

# Identify the numeric variables by name.

numc <-
  ds %>%
  names() %>%
  extract(numi) %T>%
  print()

## [1] "min_temp"          "max_temp"          "rainfall"
## [4] "evaporation"       "sunshine"          "wind_gust_speed"
.....

# Identify the categoric variables by index.

cati <-
  ds %>%
  sapply(is.factor) %>%
  which() %>%
  intersect(inputi) %T>%
  print()

## [1] 8 10 11 22

# Identify the categoric variables by name.

catc <-
  ds %>%
  names() %>%
  extract(cati) %T>%
  print()

## [1] "wind_gust_dir" "wind_dir_9am"  "wind_dir_3pm"
## [4] "rain_today"
.....

# Normalise the levels of all categoric variables.

for (v in catc)
  levels(ds[[v]]) %>>% normVarNames()
```

11 Missing Value Imputation

We can also perform missing value imputation but note that this is not something we should be doing lightly (inventing new data). We do so here simply to avoid warnings that would otherwise advise us of missing data when using `ggplot2`. Note the use of `randomForest::na.roughfix()` to perform missing value imputation for us.

```
# Count the number of missing values.

ds[vars] %>% is.na() %>% sum()
## [1] 343248

# Impute missing values.

ds[vars] %<>% na.roughfix()

# Confirm that no missing values remain.

ds[vars] %>% is.na() %>% sum()
## [1] 0
```

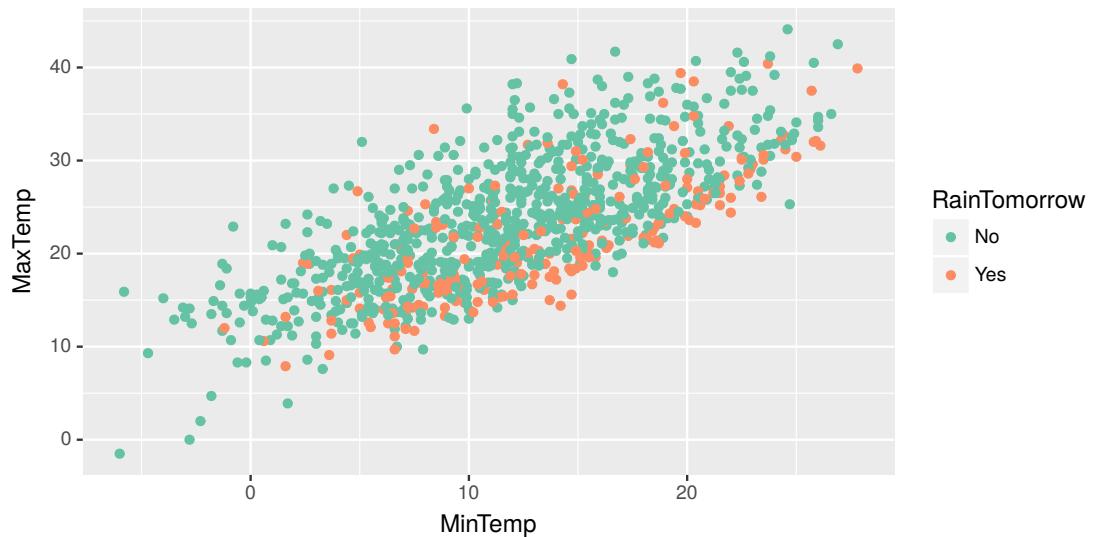
12 Review the Dataset

The dataset is now ready to be visually explored but before doing so it is useful to have another `tibble::glimpse()`.

```
glimpse(ds)

## Observations: 145,460
## Variables: 24
## $ date <date> 2008-12-01, 2008-12-02, 2008-12-0...
## $ location <fct> Albury, Albury, Albury, Albury, Al...
## $ min_temp <dbl> 13.4, 7.4, 12.9, 9.2, 17.5, 14.6, ...
## $ max_temp <dbl> 22.9, 25.1, 25.7, 28.0, 32.3, 29.7...
## $ rainfall <dbl> 0.6, 0.0, 0.0, 0.0, 1.0, 0.2, 0.0, ...
## $ evaporation <dbl> 4.8, 4.8, 4.8, 4.8, 4.8, 4.8, 4.8, ...
## $ sunshine <dbl> 8.4, 8.4, 8.4, 8.4, 8.4, 8.4, 8.4, ...
## $ wind_gust_dir <ord> w, wnw, wsw, ne, w, wnw, w, w, nnw...
## $ wind_gust_speed <dbl> 44, 44, 46, 24, 41, 56, 50, 35, 80...
## $ wind_dir_9am <ord> w, nnw, w, se, ene, w, sw, sse, se...
## $ wind_dir_3pm <ord> wnw, wsw, wsw, e, nw, w, w, nw, ...
## $ wind_speed_9am <dbl> 20, 4, 19, 11, 7, 19, 20, 6, 7, 15...
## $ wind_speed_3pm <dbl> 24, 22, 26, 9, 20, 24, 24, 17, 28, ...
## $ humidity_9am <dbl> 71, 44, 38, 45, 82, 55, 49, 48, 42...
## $ humidity_3pm <int> 22, 25, 30, 16, 33, 23, 19, 19, 9, ...
## $ pressure_9am <dbl> 1007.7, 1010.6, 1007.6, 1017.6, 10...
## $ pressure_3pm <dbl> 1007.1, 1007.8, 1008.7, 1012.8, 10...
## $ cloud_9am <dbl> 8, 5, 5, 5, 7, 5, 1, 5, 5, 5, 5, 8...
## $ cloud_3pm <dbl> 5, 5, 2, 5, 8, 5, 5, 5, 5, 5, 5, 8...
## $ temp_9am <dbl> 16.9, 17.2, 21.0, 18.1, 17.8, 20.6...
## $ temp_3pm <dbl> 21.8, 24.3, 23.2, 26.5, 29.7, 28.9...
## $ rain_today <fct> no, no, no, no, no, no, no, no, ...
## $ risk_mm <dbl> 0.0, 0.0, 0.0, 1.0, 0.2, 0.0, 0.0, ...
## $ rain_tomorrow <fct> No, No, No, No, No, No, No, Ye...
```

13 Scatter Plot



The simplest plot is a [scatter plot](#) which displays points scattered over a plot. If the dataset is large the resulting plot will be rather dense. For illustrative purposes a random subset of just 1,000 observations is used. A linear relationship between the two variables can be seen.

```
ds %>%
  sample_n(1000) %>%
  ggplot(aes(x=min_temp, y=max_temp, colour=rain_tomorrow)) +
  geom_point() +
  scale_colour_brewer(palette="Set2") +
  labs(x      = vnames["min_temp"],
       y      = vnames["max_temp"],
       colour = vnames["rain_tomorrow"])
```

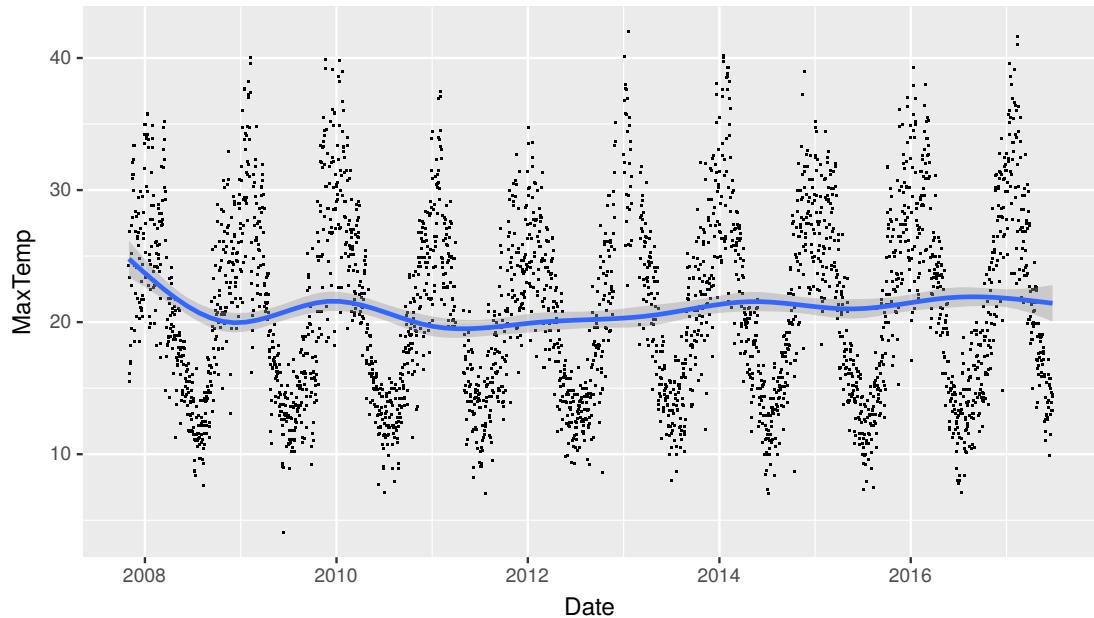
The random sample of 1,000 rows is generated using `dplyr::sample_n()` and is then piped through to `ggplot2::ggplot()`. The function argument identifies the aesthetics of the plot so that `x=` associates the variable `min_temp` with the x-axis and `y=` associates the variable `max_temp` with the y-axis.

In addition the `colour=` option provides a mechanism to distinguish between days where the observation `rain_tomorrow` is Yes and where it is No. A colour palette can be chosen using `ggplot2::scale_colour_brewer()`.

A graphical layer is added to the plot consisting of (x, y) points coloured appropriately. The function `ggplot2::geom_point()` achieves this.

The original variable names stored as `vnames` are used to label the plot using `ggplot2::labs()`. The original names will make more sense to the reader than our chosen normalised names.

14 Scatter Plot, Smooth Fitted Curve



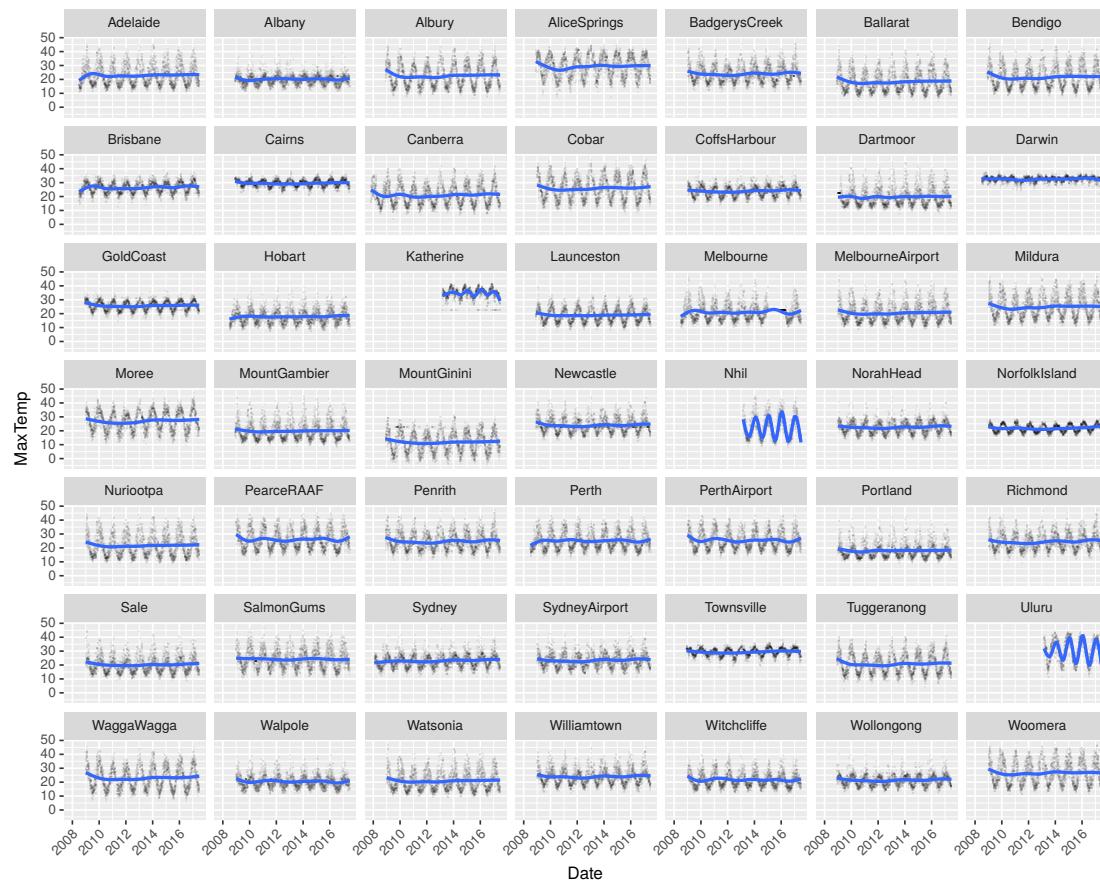
This scatter plot of `x=date` against `y=max_temp` shows a pattern of seasonality over the dataset and a trend line over the period of the dataset.

```
ds %>%
  filter(location=="Canberra") %>%
  ggplot(aes(x=date, y=max_temp)) +
  geom_point(shape=".") +
  geom_smooth(method="gam", formula=y~s(x, bs="cs")) +
  labs(x=vnames["date"], y=vnames["max_temp"])
```

The scatter plot is again created using `ggplot2::geom_point()`. Typical of scatter plots of big data there will be many overlaid points. To reduce the impact the points are reduced to a small dot using `shape=".."`.

An additional layer is added consisting of a smooth fitted curve using `ggplot2::geom_smooth()`. The dataset has many points and so a smoothing method recommended is `method="gam"` which will automatically be chosen if not specified but with a message to that effect. The formula specified using `formula=` is also the default for `method="gam"`.

15 Scatter Plot, Faceted Locations

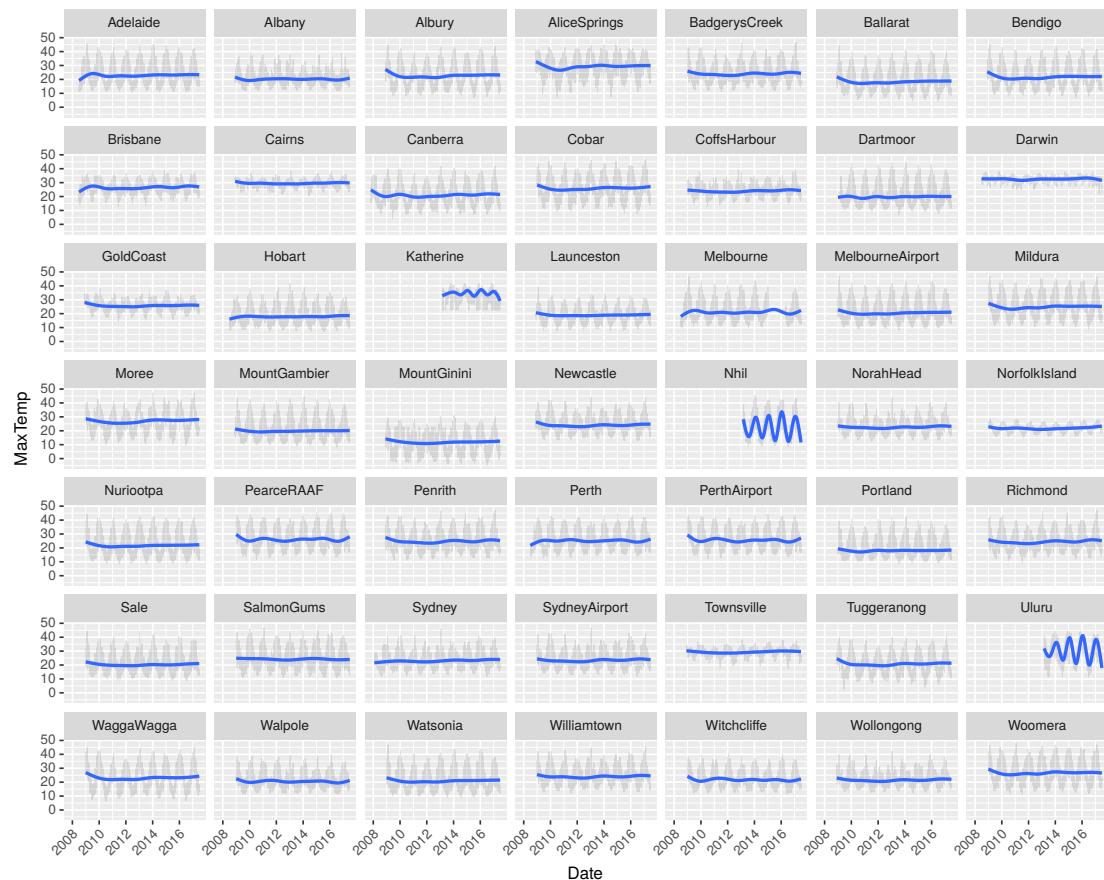


Partitioning the dataset by a categoric variable will reduce the blob effect for big data. The plot here uses `location` as the faceted variable to separately plot each location's maximum temperature over time. Notice the seasonal effect across all plots, some with quite different patterns.

```
ds %>%
  ggplot(aes(x=date, y=max_temp)) +
  geom_point(alpha=0.05, shape=".") +
  geom_smooth(method="gam", formula=y~s(x, bs="cs")) +
  facet_wrap(~location) +
  theme(axis.text.x=element_text(angle=45, hjust=1)) +
  labs(x=vnames["date"], y=vnames["max_temp"])
```

The plot uses `facet_wrap()` to separately plot each location. For the points `ggplot2::geom_point()` is provided an `alpha=` to reduce the effect of overlaid points. Using smaller dots on the plots by way of `shape=` also de-clutters the plot significantly and improves the presentation and emphasises the patterns. The x axis tick labels are rotated 45° using `angle=45` within `ggplot2::element_text()` to avoid the labels overlapping. The `hjust=1` forces the labels to be right justified.

16 Line Plot, Faceted Locations, Thin Lines

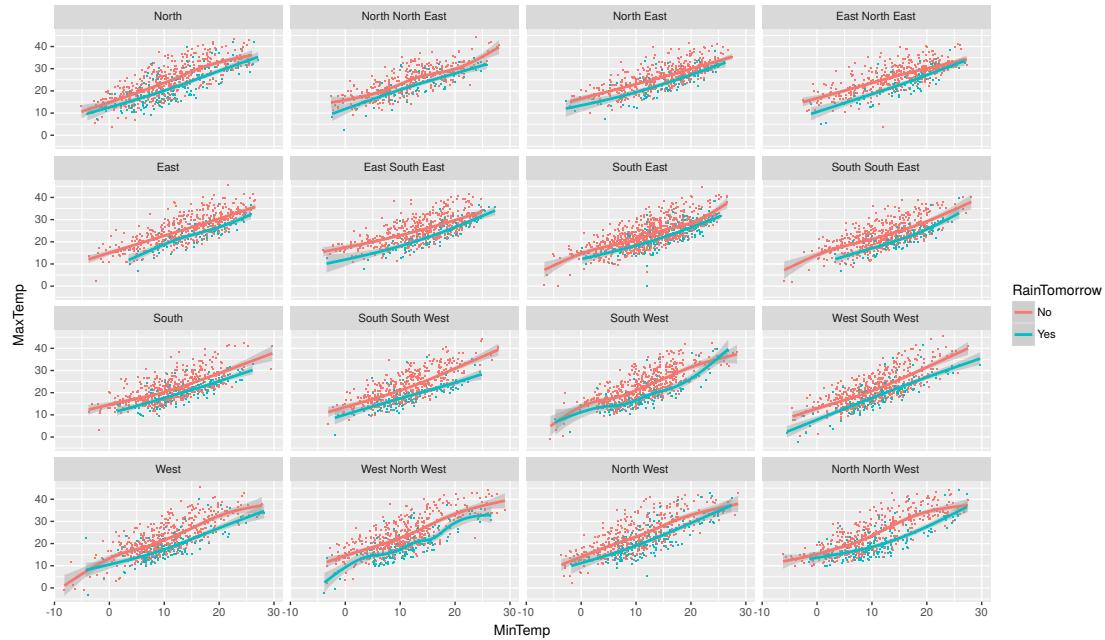


An alternative is to present the plot as a line chart rather than a scatter plot. It does make more sense for a time series plot such as this, though the effect is little changed due to the amount of data being displayed.

```
ds %>%
  ggplot(aes(x=date, y=max_temp)) +
  geom_line(alpha=0.1, size=0.05) +
  geom_smooth(method="gam", formula=y~s(x, bs="cs")) +
  facet_wrap(~location) +
  theme(axis.text.x=element_text(angle=45, hjust=1)) +
  labs(x=vnames["date"], y=vnames["max_temp"])
```

Changing to lines simply uses `ggplot2::geom_line()` instead of `ggplot2::geom_point()`. Very thin lines are used as specified through the `size=` option. Nonetheless, the data remains quite dense.

17 Faceted Wind Directions



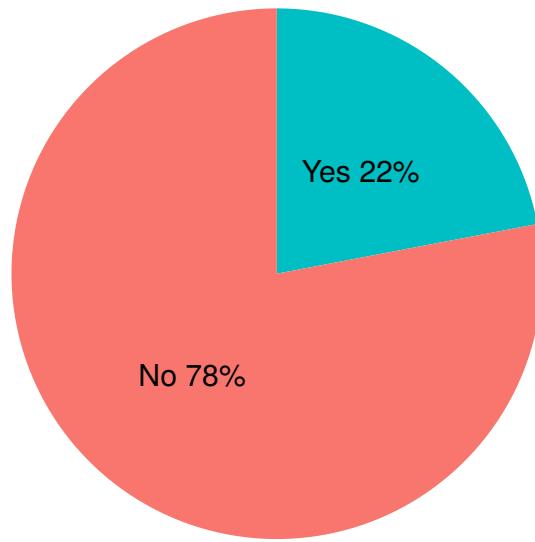
Labels of a faceted plot can be modified as here expanding n to North, s to South, etc. Observe that the linear relationship for rainy days is below that for dry days. The maximum temperature is generally closer to the minimum temperature on days where it rains the following day.

```
lblr <- function(x)
{
  x %>%
    str_replace_all("n", "North ") %>%
    str_replace_all("s", "South ") %>%
    str_replace_all("e", "East ") %>%
    str_replace_all("w", "West ")
    str_replace(" $", "")
}

ds %>%
  sample_n(10000) %>%
  ggplot(aes(x=min_temp, y=max_temp, colour=rain_tomorrow)) +
  geom_point(shape=".") +
  geom_smooth(method="gam", formula=y~s(x, bs="cs")) +
  facet_wrap(~wind_dir_3pm, labeller=labeler(wind_dir_3pm=lblr)) +
  labs(x      = vnames["min_temp"],
       y      = vnames["max_temp"],
       colour = vnames["rain_tomorrow"])
```

The function to remap the directions uses `stringr::str_replace_all()` to do the work. It is then transformed into a `ggplot2::labeler()` for `wind_dir_3pm=`.

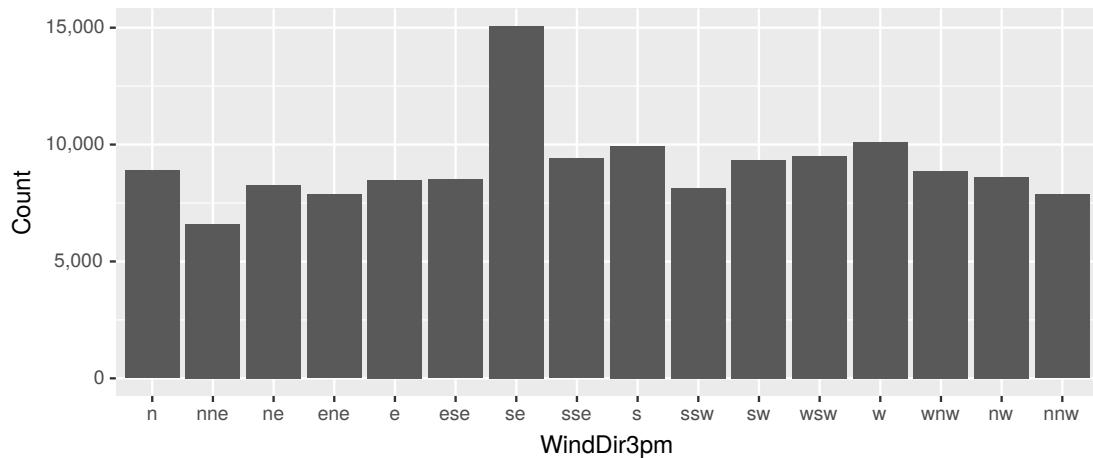
18 Pie Chart



A [pie chart](#) is a popular circular plot showing the relative proportions through angular slices. Generally, pie charts are not recommended, particularly for multiple wedges, because humans generally have difficulty perceiving the relative angular differences between slices. For two or three slices it may be argued that the pie chart is just fine, and if further information is provided, such as labelling the slices with their sizes.

```
ds %>%
  group_by(rain_tomorrow) %>%
  count() %>%
  ungroup() %>%
  mutate(per=round(`n`/sum(`n`), 2)) %>%
  mutate(label=paste(rain_tomorrow, percent(per))) %>%
  arrange(per) %>%
  ggplot(aes(x=1, y=per, fill=rain_tomorrow)) +
  geom_bar(stat="identity") +
  coord_polar(theta='y') +
  theme_void() +
  theme(legend.position="none") +
  geom_text(aes(x=1, y=cumsum(per)-per/2, label=label), size=8)
```

19 Histogram



Another common plot is the [histogram](#) or bar chart which displays the count of observations using bars. The bars present the frequency of the levels of the categoric variable `wind_dir_3pm` from the dataset.

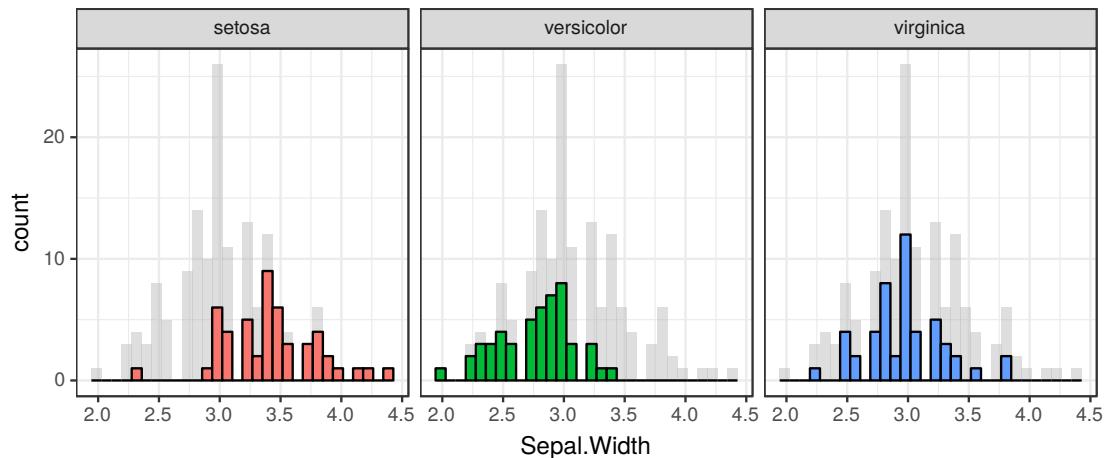
```
ds %>%
  ggplot(aes(x=wind_dir_3pm)) +
  geom_bar() +
  scale_y_continuous(labels=comma) +
  labs(x=vnames["wind_dir_3pm"], y="Count")
```

A histogram is generated using `ggplot2::geom_bar()`. Only an x-axis is required as the aesthetic and `wind_dir_3pm` is chosen.

20 Histogram with Background

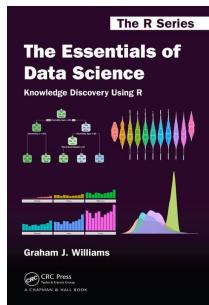
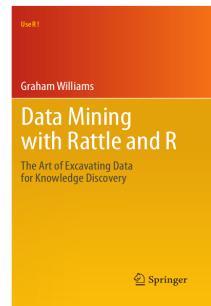
Originally from <https://drsimonj.svbtle.com/plotting-background-data-for-groups-with-ggplot2> who used the iris dataset. Convert to use the weather dataset.

```
## 'stat_bin()' using 'bins = 30'. Pick better value with
## 'binwidth'.
## 'stat_bin()' using 'bins = 30'. Pick better value with
## 'binwidth'.
```



21 Further Reading and Acknowledgements

The [Rattle](#) book ([Williams, 2011](#)), published by Springer, provides a comprehensive introduction to data mining and analytics using Rattle and R. It is available from [Amazon](#). Rattle provides a graphical user interface through which the user is able to load, explore, visualise, and transform data, and to build, evaluate, and export models. Through its Log tab it specifically aims to provide an R template which can be exported and serve as the starting point for further programming with data in R.



The [Essentials of Data Science](#) book ([Williams, 2017a](#)), published by CRC Press, provides a comprehensive introduction to data science through programming with data using R. It is available from [Amazon](#). The book provides a template based approach to doing data science and knowledge discovery. Templates are provided for data wrangling and model building. These serve as generic starting points for programming with data, and are designed to require minimal effort to get started. Visit <https://essentials.togaware.com> for further guides and templates.

Other resources include:

- The [GGPlot2 documentation](#) is quite extensive and useful
- The [R Cookbook](#) is a great resource explaining how to do many types of plots using ggplot2.

22 References

- Bache SM, Wickham H (2014). *magrittr: A Forward-Pipe Operator for R*. R package version 1.5, URL <https://CRAN.R-project.org/package=magrittr>.
- Breiman L, Cutler A, Liaw A, Wiener M (2018). *randomForest: Breiman and Cutler's Random Forests for Classification and Regression*. R package version 4.6-14, URL <https://CRAN.R-project.org/package=randomForest>.
- R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- Wickham H (2017). *scales: Scale Functions for Visualization*. R package version 0.5.0, URL <https://CRAN.R-project.org/package=scales>.
- Wickham H (2018). *stringr: Simple, Consistent Wrappers for Common String Operations*. R package version 1.3.1, URL <https://CRAN.R-project.org/package=stringr>.
- Wickham H, Chang W (2016). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 2.2.1, URL <https://CRAN.R-project.org/package=ggplot2>.
- Wickham H, François R, Henry L, Müller K (2018). *dplyr: A Grammar of Data Manipulation*. R package version 0.7.5, URL <https://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.
- Williams GJ (2017a). *The Essentials of Data Science: Knowledge discovery using R*. The R Series. CRC Press.
- Williams GJ (2017b). *rattle: Graphical User Interface for Data Science in R*. R package version 5.1.0, URL <https://CRAN.R-project.org/package=rattle>.
- Williams GJ (2017c). *rattle.data: Rattle Datasets*. R package version 1.0.2, URL <https://CRAN.R-project.org/package=rattle.data>.

This document, sourced from VisualiseO.Rnw bitbucket revision 241, was processed by KnitR version 1.20 of 2018-02-20 10:11:46 UTC and took 21.9 seconds to process. It was generated by gjw on Ubuntu 18.04 LTS.

Data Science with R

Transform and Manipulate Data

Graham.Williams@togaware.com

9th July 2014

Visit <http://onepager.togaware.com/> for more OnePageR's.

In this module we introduce approaches to manipulate and transform our data.

The required packages for this module include:

```
library(rattle)      # The weatherAUS datasets and normVarNames()
library(ggplot2)     # Visualise the transforms.
library(plyr)        # Transform using ddplyr()
library(dplyr)        # Transform using ddplyr()
library(reshape2)     # melt() and dcast()
```

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 Data

```
library(rattle)
ds           <- weatherAUS
names(ds)   <- normVarNames(names(ds))    # Lower case variable names.
str(ds)

## 'data.frame': 88768 obs. of  24 variables:
## $ date          : Date, format: "2008-12-01" "2008-12-02" ...
## $ location      : Factor w/ 49 levels "Adelaide","Albany",...: 3 3 3 3 3 ...
## $ min_temp       : num  13.4 7.4 12.9 9.2 17.5 14.6 14.3 7.7 9.7 13.1 ...
....
```

2 Factors

3 Factors: Drop Unused Levels

The complete list of levels of a factor are maintained even when we take a subset of a dataset which contains only a subset of the original levels.

For example, suppose we subset the weather dataset on location:

```

cities <- c("Adelaide", "Brisbane", "Canberra", "Darwin")
levels(ds$location)

## [1] "Adelaide"      "Albany"        "Albury"
## [4] "AliceSprings"  "BadgerysCreek"  "Ballarat"
## [7] "Bendigo"       "Brisbane"       "Cairns"
## [10] "Canberra"      "Cobar"         "CoffsHarbour"
.....
summary(ds$location)

##          Adelaide           Albany          Albury          AliceSprings
##             2036                  1883                 1883                  1883
##          BadgerysCreek        Ballarat        Bendigo        Brisbane
##             1852                  1883                 1883                  2036
.....
dss     <- subset(ds, location %in% cities)
levels(dss$location)

## [1] "Adelaide"      "Albany"        "Albury"
## [4] "AliceSprings"  "BadgerysCreek"  "Ballarat"
## [7] "Bendigo"       "Brisbane"       "Cairns"
## [10] "Canberra"      "Cobar"         "CoffsHarbour"
.....
summary(dss$location)

##          Adelaide           Albany          Albury          AliceSprings
##             2036                  0                   0                   0
##          BadgerysCreek        Ballarat        Bendigo        Brisbane
##             0                      0                   0                  2036
.....

```

Notice that the levels remain unchanged even though there are no observations of the other locations. We can re-factor the levels using `factor()`:

```

dss$location <- factor(dss$location)
levels(dss$location)

## [1] "Adelaide" "Brisbane" "Canberra" "Darwin"
summary(dss$location)

## Adelaide Brisbane Canberra Darwin
##      2036      2036     2279      2036

```

4 Factors: Reorder Levels

By default the levels of a factor are ordered alphabetically. We can change the order simply by providing `levels=` to `factor()`.

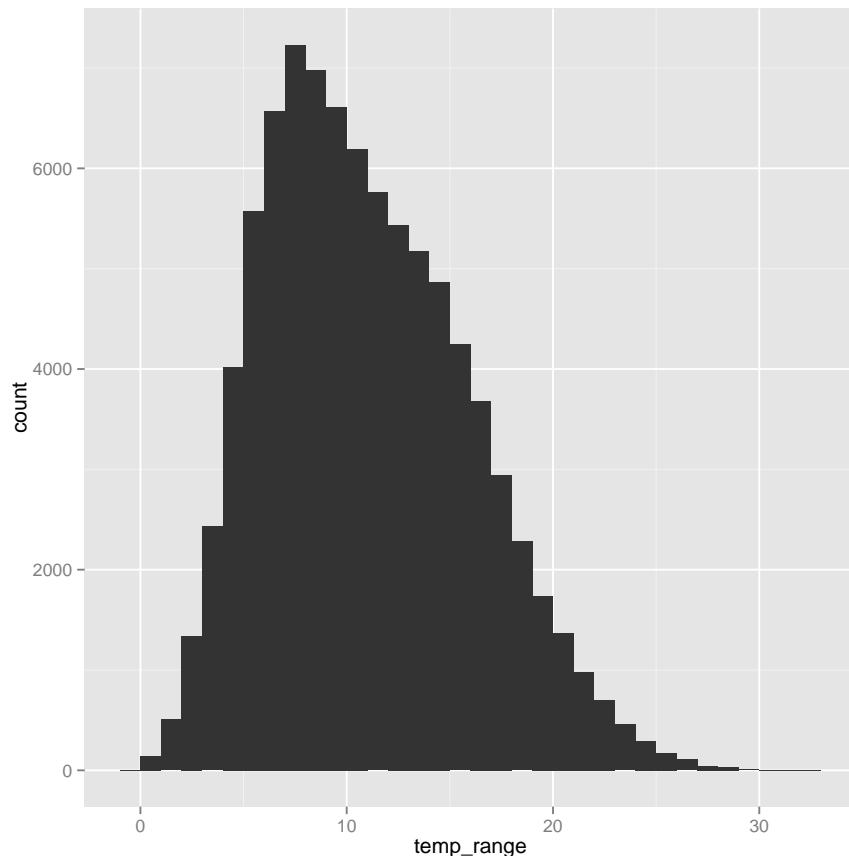
```
levels(dss$location)
## [1] "Adelaide" "Brisbane" "Canberra" "Darwin"
summary(dss$location)
## Adelaide Brisbane Canberra Darwin
##      2036      2036     2279      2036
dss$location <- factor(dss$location, levels=rev(levels(dss$location)))

levels(dss$location)
## [1] "Darwin"    "Canberra"   "Brisbane"   "Adelaide"
summary(dss$location)
##    Darwin Canberra Brisbane Adelaide
##      2036      2279      2036      2036
```

5 Data Frame: Add a Column

Here we simply name the column as part of the data frame and it gets added to it.

```
ds$temp_range <- ds$max_temp - ds$min_temp  
str(ds)  
  
## 'data.frame': 88768 obs. of 25 variables:  
## $ date : Date, format: "2008-12-01" "2008-12-02" ...  
## $ location : Factor w/ 49 levels "Adelaide","Albany",...: 3 3 3 3 3 ...  
## $ min_temp : num 13.4 7.4 12.9 9.2 17.5 14.6 14.3 7.7 9.7 13.1 ...  
....  
p <- ggplot(ds, aes(x=temp_range))  
p <- p + geom_bar(binwidth=1)  
p
```



6 Transform: Add a Column

An alternative is to use `transform()` which can be neater when adding several columns, avoiding the use of the `$` nomenclature.

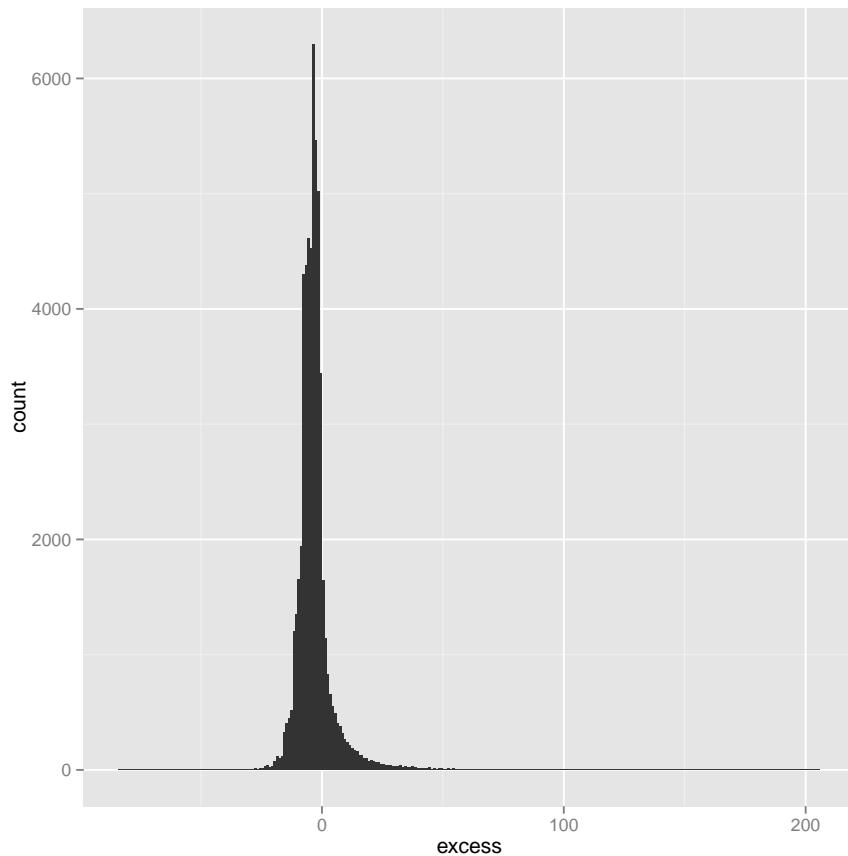
```
ds <- transform(ds,
                 temp_range=max_temp-min_temp,
                 excess=rainfall-evaporation)
sum(ds$excess, na.rm=TRUE)

## [1] -176068

str(ds)

## 'data.frame': 88768 obs. of  26 variables:
## $ date          : Date, format: "2008-12-01" "2008-12-02" ...
## $ location      : Factor w/ 49 levels "Adelaide","Albany",...: 3 3 3 3 3 ...
## $ min_temp       : num  13.4 7.4 12.9 9.2 17.5 14.6 14.3 7.7 9.7 13.1 ...
## ...
## $ excess         : num  -176068 -176068 -176068 -176068 -176068 ...

ggplot(ds, aes(x=excess)) + geom_bar(binwidth=1)
```



7 Subset Data

Exercise: Research the subset() function and illustrate its usage.

8 Transform Using DPLYR

The `plyr` (Wickham, 2014a) package provides a collection of the most useful functions for manipulating data. Its concepts, once understood, are very powerful and allow us to express numerous tasks simply and efficiently.

Like `apply()`, the `plyr` functions operate on data frames, matrices, lists, vectors or arrays. An operation is applied to some collection of items (e.g., each group of observations or group of list elements) in the input data structure, and the results are packaged into a new data structure.

Generally, the pattern is like `ddply(data, variables, function, ...)` where in this case (as indicated by the first d) the input data is a data frame and the result (the second d) is also a data frame. The rows of the data frame will be grouped by the variables identified, and for each group the function is applied to obtain the resulting data. The remaining arguments are treated as arguments to the function.

Exercise: Explore and provide examples.

9 Summarise Data Using dplyr()

dplyr (Wickham and Francois, 2014) introduces a grammar of data manipulation and processes data much more efficiently than plyr (Wickham, 2014a) (anywhere from 20 times to 1000 times faster) and other R packages through parallel processing using Rcpp (Eddelbuettel and Francois, 2014).

```
weatherAUS %>%
  group_by(Location) %>%
  summarise(total = sum(Rainfall)) %>%
  arrange(desc(total)) %>%
  head(5)

## Source: local data frame [5 x 2]
##
##       Location total
## 1        Darwin 11092
## 2 SydneyAirport  5295
## 3 MountGambier  4050
## 4       Perth  3723
## 5     Bendigo  3286
```

10 Removing Columns

```
tail(ds$excess)
## [1] NA NA NA NA NA NA

names(ds)

## [1] "date"          "location"       "min_temp"
## [4] "max_temp"      "rainfall"        "evaporation"
## [7] "sunshine"       "wind_gust_dir"  "wind_gust_speed"
## [10] "wind_dir_9am"  "wind_dir_3pm"   "wind_speed_9am"
## [13] "wind_speed_3pm" "humidity_9am"   "humidity_3pm"
## [16] "pressure_9am"   "pressure_3pm"   "cloud_9am"
## [19] "cloud_3pm"      "temp_9am"        "temp_3pm"
## [22] "rain_today"     "risk_mm"        "rain_tomorrow"
## [25] "temp_range"    "excess"

ds$excess <- NULL
tail(ds$excess)

## NULL

names(ds)

## [1] "date"          "location"       "min_temp"
## [4] "max_temp"      "rainfall"        "evaporation"
## [7] "sunshine"       "wind_gust_dir"  "wind_gust_speed"
## [10] "wind_dir_9am"  "wind_dir_3pm"   "wind_speed_9am"
## [13] "wind_speed_3pm" "humidity_9am"   "humidity_3pm"
## [16] "pressure_9am"   "pressure_3pm"   "cloud_9am"
## [19] "cloud_3pm"      "temp_9am"        "temp_3pm"
## [22] "rain_today"     "risk_mm"        "rain_tomorrow"
## [25] "temp_range"
```

11 Subset Data

Exercise: Discuss the subset function.

12 Wide to Long Data

Let's take a sample dataset to illustrate the concepts of wide and long data.

```
dss <- subset(ds, date==max(date))
dim(dss)

## [1] 49 25

head(dss)

##           date      location min_temp max_temp rainfall evaporation
## 1883 2014-04-25      Albury     5.3     22.5     0.0        NA
## 3735 2014-04-25 BadgerysCreek   16.5     21.2     1.8        NA
## 5587 2014-04-25       Cobar    12.9     30.5     0.0      7.4
....
```

This data is in wide format. We can convert it to long format, which is sometimes useful when using, for example, `ggplot2` (Wickham and Chang, 2014). We use `reshape2` (Wickham, 2014b) to do this. In long format we essentially maintain a single measurement per observation. The measurement for our data are all those columns recording some measure of the weather—that is, all variables except for `date` and `location`.

```
library(reshape2)
dssm <- melt(dss, c("date", "location"))

## Warning: attributes are not identical across measure variables; they will be
## dropped

dim(dssm)

## [1] 1127     4

head(dssm)

##           date      location variable value
## 1 2014-04-25      Albury min_temp   5.3
## 2 2014-04-25 BadgerysCreek min_temp  16.5
## 3 2014-04-25       Cobar min_temp  12.9
.....

tail(dssm)

##           date      location variable value
## 1122 2014-04-25      Hobart temp_range  9.1
## 1123 2014-04-25   Launceston temp_range 17.8
## 1124 2014-04-25 AliceSprings temp_range 19.3
.....

dssm[sample(nrow(dssm), 6),]

##           date      location      variable value
## 415 2014-04-25 Melbourne wind_dir_3pm      N
## 672 2014-04-25 Nuriootpa pressure_9am 1017.7
## 166 2014-04-25 Ballarat  evaporation <NA>
....
```

This is now clearly long data.

13 Long to Wide Data

```
dssmc <- dcast(dssm, date + location ~ variable)
dim(dss)

## [1] 49 25

dim(dssmc)

## [1] 49 25

head(dss)

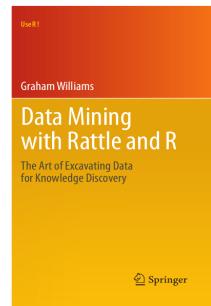
##           date      location min_temp max_temp rainfall evaporation
## 1883 2014-04-25      Albury     5.3    22.5     0.0        NA
## 3735 2014-04-25 BadgerysCreek  16.5    21.2     1.8        NA
## 5587 2014-04-25      Cobar    12.9    30.5     0.0       7.4
## 7439 2014-04-25 CoffsHarbour  14.4    25.0     0.0       3.0
## 9291 2014-04-25      Moree    17.7    30.0     0.4        NA
## 11174 2014-04-25 Newcastle   11.0    21.2    11.8        NA
##           sunshine wind_gust_dir wind_gust_speed wind_dir_9am wind_dir_3pm
## 1883          NA             NNW            22                 S            NNE
## 3735          NA             NNE            20                 S            ESE
.....
head(dssmc)

##           date      location min_temp max_temp rainfall evaporation sunshine
## 1 2014-04-25      Adelaide      9    23.3     0.2      <NA>      <NA>
## 2 2014-04-25      Albany     <NA>     21      <NA>      2.2      4.4
## 3 2014-04-25      Albury     5.3    22.5     0      <NA>      <NA>
## 4 2014-04-25 AliceSprings  10.5    29.8     0      6.6     11.1
## 5 2014-04-25 BadgerysCreek  16.5    21.2     1.8      <NA>      <NA>
## 6 2014-04-25 Ballarat     1.9    15.9     0      <NA>      <NA>
##           wind_gust_dir wind_gust_speed wind_dir_9am wind_dir_3pm wind_speed_9am
## 1             NNW            39              NE            WNW            13
## 2             <NA>          <NA>          <NA>          SSW          <NA>
....
```

14 Further Reading

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 module is one of many OnePageR modules available from <http://onepager.togaware.com>. In particular follow the links on the website with a * which indicates the generally more developed OnePageR modules.



15 References

- Eddelbuettel D, Francois R (2014). *Rcpp: Seamless R and C++ Integration*. R package version 0.11.1, URL <http://www.rcpp.org>, <http://dirk.eddelbuettel.com/code/rcpp.html>, <https://github.com/RcppCore/Rcpp>.
- 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/>.
- Wickham H (2014a). *plyr: Tools for splitting, applying and combining data*. R package version 1.8.1, URL <http://CRAN.R-project.org/package=plyr>.
- Wickham H (2014b). *reshape2: Flexibly reshape data: a reboot of the reshape package*. R package version 1.4, URL <http://CRAN.R-project.org/package=reshape2>.
- Wickham H, Chang W (2014). *ggplot2: An implementation of the Grammar of Graphics*. R package version 1.0.0, URL <http://ggplot2.org>, <https://github.com/hadley/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 3.0.4, URL <http://rattle.togaware.com/>.

This document, sourced from TransformO.Rnw revision 456, was processed by KnitR version 1.6 of 2014-05-24 and took 3 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-07-09 21:46:26.

Data Science with R

Case Studies in Data Science

Graham.Williams@togaware.com

9th August 2014

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

In this chapter we work through numerous case studies using publicly available data, primarily from the Australian Government's open data web site, <http://data.gov.au>. We explain each of the steps and the R commands that achieve the required analyses.

The required packages for this chapter include:

```
library(xlsx)      # Read Excel spreadsheets.  
library(rattle)    # normVarNames().  
library(stringr)   # String manipulation.  
library(tidyr)     # Tidy the dataset.  
library(dplyr)     # Data manipulation.  
library(ggplot2)   # Visualise data.  
library(scales)    # Include commas in numbers.  
library(directlabels) # Dodging labels for ggplot2.
```

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 ATO Web Analytics

The ATO provided some of its web log data for [GovHack 2014](#). It is available on the Australian Government's data sharing web site data.gov.au and specifically within the [ATO Web Analytics](#) section. The CSV data files are provided within zip containers.

We record the locations here so that we can refer to them in the R code that follows.

```
datagovau <- "http://data.gov.au/dataset/"
atogovau <- file.path(datagovau, "9b57d00a-da75-4db9-8899-6537dd60eeba/resource")
```

All the datasets are for the period from July 2013 to April 2014 broken down by month and by traffic source (internal or external). For convenience, since the period is used within the filenames, we also record the period as a string here, together with the actual months recorded.

```
period <- " - July 2013 to April 2014.csv"
months <- c("Jul-13", "Aug-13", "Sep-13", "Oct-13", "Nov-13",
          "Dec-13", "Jan-14", "Feb-14", "Mar-14", "Apr-14")
```

Each dataset is a [comma-separated value](#) (CSV) file. The actual filenames are a string of words separated by space. Each CSV file is contained within a separate ZIP archive, containing just that single CSV file. The ZIP files are named the same as the CSV file but all lowercase, no spaces, and replacing the .csv extension with .zip. Thus, for a ZIP file named something like:

`asamplefileofdataascsvjuly2013toapril2014.zip`

it will contain a CSV file named:

`A Sample File Of Data As CSV - July 2013 to April 2014.csv`

A support function will be useful to convert the CSV filenames into ZIP filenames (easier than the other way round). This involves string operations using `stringr` ([Wickham, 2012](#)), and we deploy the pipe operator (%>%) from `magrittr` ([Bache and Wickham, 2014](#)) for a simple and clearly expressed function.

```
csv2zip <- function(fname)
{
  fname %>% tolower() %>% str_replace_all(" |-|csv$", "") %>% str_c("zip")
}
```

To illustrate with the sample filenames as above:

```
tefname <- str_c("A Sample File Of Data As CSV", period)
tefname
## [1] "A Sample File Of Data As CSV - July 2013 to April 2014.csv"
```

That is converted into the following filename:

```
tezname <- csv2zip(tefname)
tezname
## [1] "asamplefileofdataascsvjuly2013toapril2014.zip"
```

This is how the providers of this particular dataset have chosen to package their data.

1.1 The ATO Web Analytics Datasets

There are seven datasets provided through [data.gov.au](#). Each of the CSV filenames is noted and recorded here. We can then automatically download the ZIP file into a temporary location, extract the CSV, and read it into R.

The text descriptions are those provided on the web site.

Browser Access Browsers used to access the ATO website.

```
brfname <- str_c("Browser by month and traffic source", period)
brzname <- csv2zip(brfname)
```

Entry Pages Starting (entry) pages for the ATO website.

```
enfname <- str_c("Entry pages by month and traffic source", period)
enzname <- csv2zip(enfname)
```

Site Referrers Websites linking to [ato.gov.au](#) (entry referrers) that were used to access the ATO website content.

```
refname <- str_c("Entry referrers by month and traffic source", period)
rezname <- csv2zip(refname)
```

Exit Pages Pages on which users left the ATO website (exit pages).

```
exfname <- str_c("Exit pages by month and traffic source", period)
exzname <- csv2zip(exfname)
```

Local Keyword Searches Keywords and phrases used in the ATO website search engine.

```
kwfname <- str_c("Local keywords (top 100) by month and traffic source", period)
kzwname <- csv2zip(kwfname)
```

Operating System Access Operating system used to access the ATO website.

```
osfname <- str_c("Operating System (platform) by month and traffic", period)
oszname <- csv2zip(osfname)
```

Pages Viewed Pages viewed on the ATO website.

```
vifname <- str_c("Pages by month and traffic source", period)
vizname <- csv2zip(vifname)
```

In the following single page sections we analyse the various datasets in a variety of ways.

2 ATO Entry Pages

The path to the file containing the dataset to download is constructed as a URL. We begin with the string we created earlier, `atogovau`, and append an internal identifier from the web site that locates the dataset web page. We then point to the download area and the ZIP filename to download. The `file.path()` command adds the path separator “/” between its arguments:

```
fname <- file.path(atogovau,
                     "121dfe58-044b-4f21-b4ee-6f9335a44413",
                     "download",
                     enzname)
fname
## [1] "http://data.gov.au/dataset//9b57d00a-da75-4db9-8899-6537dd60eeba/reso..."
```

We next create a new temporary local file using `tempfile()` and then cause the dataset to be downloaded from the Internet into this temporary file using `download.file()`:

```
temp <- tempfile(fileext=".zip")
download.file(fname, temp)
```

The progress of the download will be displayed interactively.

The content of the ZIP file is then extracted using `unz()`, which reads the ZIP file and extracts the specified file. It actually opens a so-called connection and the output of that connection is piped here into `read.csv()`. We also remove the temporary file using `unlink()`.

```
entry <- temp %>% unz(enfname) %>% read.csv()
unlink(temp)
```

We will load this dataset into our generic variable `ds` after piping it into `tbl_df()` from `dplyr` ([Wickham and Francois, 2014](#)) as a convenience for printing the data frame:

```
dsname <- "entry"
ds      <- dsname %>% get() %>% tbl_df()
```

As we often do, the variable names are “simplified,” or at least normalised, using `normVarNames()` from `rattle` ([Williams, 2014](#)). We record the list of available variables in `vars`.

```
names(ds) <- normVarNames(names(ds))
names(ds)

## [1] "entry_page"    "month"        "source"       "views"        "visits"
vars <- names(ds)
```

Often we will want to experiment with our datasets and then perhaps revert to a clean original dataset. It is a good idea to take a copy of the dataset, as we do below, calling it `dso`. Note that this is the source dataset, `entry`, with the normalised variable names. We can remove (using `rm()`) the actual source dataset so as to reduce our memory footprint. This is particularly important for large datasets.

```
dso <- ds
rm(entry)
```

2.1 Explore

We can now explore some of the characteristics of the dataset. Our aim, as always, is to get familiar with it—part of *living and breathing the data*.

```
ds

## Source: local data frame [207,119 x 5]
##
##                                     entry_...
## 1                               http://www.ato.gov...
## 2                               http://www.ato.gov...
## 3 http://www.ato.gov.au/individuals/lodging-your-tax-return/e-...
## 4 http://www.ato.gov.au/individuals/lodging-your-tax-return/e-...
....
```

Out of the 207,119, there are 33,262 different entry points.

```
length(levels(ds$entry_page))

## [1] 33262
```

We summarise the remaining columns to get a feel for what theses columns might be recording.

```
summary(ds[-1])

##      month           source          views        visits
##  Apr-14 :23324    4 :     1   Min.   :    1   Min.   :    1
##  Jul-13 :23067  External:144400  1st Qu.:     4  1st Qu.:     1
##  Aug-13 :22849   Internal:62718 Median :    18 Median :     4
##  Oct-13 :22736                    Mean   : 1019  Mean   : 169
##  Nov-13 :22478                    3rd Qu.:    88  3rd Qu.:    20
##  Sep-13 :21084                    Max.   :9106745 Max.   :1349083
##  (Other):71581                   NA's   :1
```

The variable *month* appears to report the month and year. The *source* looks to record, presumably, whether the person browsing is external to the ATO or internal to the ATO. For an entry point the *views* looks to report the number of views for that month (and broken down between internal and external views). Similarly for *visits*.

There appears to be an odd value for source (the 4) and we might wonder whether there is a data quality issue here. Let's have a look at the full frequency table for *month*:

```
table(ds$month)

##
##      Apr-14     Aug-13     Dec-13 External   Feb-14    Jan-14    Jul-13    Mar-14
##      23324     22849     19090       1     17222    17266    23067    18002
##      Nov-13     Oct-13     Sep-13
....
```

Something odd happening there. **External** should not be a value for *month*—it belongs to the next column (*source*). There is apparently some misalignment for this one observarion.

2.2 Data Quality Issue

Having identified a data quality issue we need to have a closer look to determine how widespread the issue is. Of course, initial indications are that there is a single bad observation in the dataset.

The first task is to identify where the errant observation is:

```
which(ds$source=="4")
## [1] 198499
```

We see that this is observation number 198,499.

Going back to the original source CSV file we need to find that row in the file itself. There are many ways to do this, including loading the file into a spreadsheet application or even just a text editor. A simple Linux command line approach will pipe the results of `tail` into `head` as in the following command line (replacing the `<filename>` with the actual filename). Even for extremely large files, this will take almost no time at all as both commands are very efficient, and is likely quicker than loading the data into Libre Office (or Microsoft/Excel for that matter):

```
$ tail -n+198495 <filename>.csv | head -n10
```

We discover that row 198,500 (given that the first row of the CSV file is the header row) has the literal value:

```
"http://www.ato.gov.au/content/00268103.htm","",March 2014""",External,4,3,
```

Compare that to another row that is valid:

```
http://www.ato.gov.au/content/00171495.htm,Mar-14,External,7,2
```

Something would appear to have gone wrong in the extraction of the dataset at source—the CSV file is the original from the web site so perhaps the extraction at the ATO was less than perfect.

We will simply remove that errant row:

```
dim(ds)
## [1] 207119      5

issue <- which(ds$source=="4")
issue

## [1] 198499

dso <- ds <- ds[-issue,]
dim(ds)

## [1] 207118      5
```

Notice the use of `which()` to identify the observation number that contains the error, and then `-issue` as the index of the data frame, which removes that row.

There is no further evidence that there is any other similar issues in the dataset for now, so we can proceed with our analysis.

2.3 Clean And Enhance

We might notice that the levels for the `month` are in alphabetic order whilst we would normally want these to be ordered chronologically. We can fix that:

```
levels(ds$month)
## [1] "Apr-14"   "Aug-13"   "Dec-13"   "External" "Feb-14"   "Jan-14"
## [7] "Jul-13"   "Mar-14"   "Nov-13"   "Oct-13"   "Sep-13"

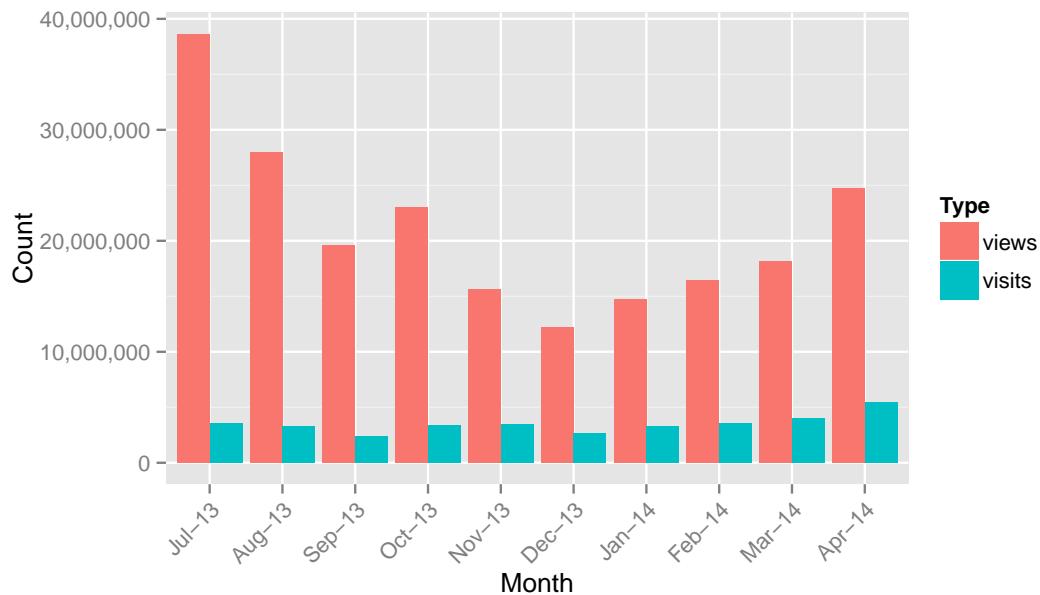
ds$month <- factor(ds$month, levels=months)
levels(ds$month)
## [1] "Jul-13"  "Aug-13"  "Sep-13"  "Oct-13"  "Nov-13"  "Dec-13"  "Jan-14"
## [8] "Feb-14"  "Mar-14"  "Apr-14"
```

2.4 Explore Some More

The total number of views/visits to the ATO website may be interest.

```
format(sum(ds$views), big.mark=",")
## [1] "211,110,271"
format(sum(ds$visits), big.mark=",")
## [1] "35,050,249"
```

We can then explore the views/visits per month.



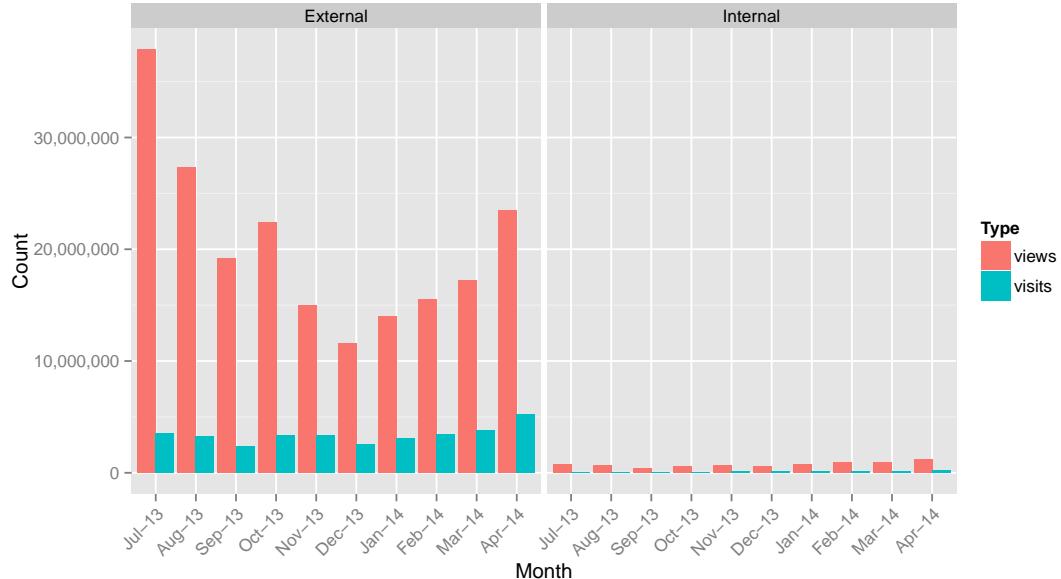
```
ds %>%
  group_by(month) %>%
  summarise(views = sum(views), visits = sum(visits)) %>%
  gather(type, count, -month) %>%
  ggplot(aes(x = month, y = count, fill = type)) +
  geom_bar(stat = "identity", position = "dodge") +
  scale_y_continuous(labels = comma) +
  labs(fill = "Type", x = "Month", y = "Count") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

We can see an interesting pattern of views versus visits in that there's a reasonably flat number of visits over the period, however the number of views (and also we would suggest views per visit) is dramatically increased for July. We would really need to analyse the relative change in views/visit over time to confirm that observation, but we'll stay with the visual for now.

The July spike may well correspond to the Australian financial year ending in June and starting in July. We might also observe the holiday season around December when there must be less interest in taxation topics.

2.5 External versus Internal Views/Visits per Month

The breakdown between **External** and **Internal** may be of interest.



```
ds %>%
  group_by(month, source) %>%
  summarise(views = sum(views), visits = sum(visits)) %>%
  gather(type, count, -c(month, source)) %>%
  ggplot(aes(x=month, y=count, fill=type)) +
  geom_bar(stat="identity", position="dodge") +
  scale_y_continuous(labels=comma) +
  labs(fill="Type", x="Month", y="Count") +
  theme(axis.text.x=element_text(angle=45, hjust=1)) +
  facet_wrap(~source)
```

We immediately see that relatively speaking there are very few internal views/visits. This should not be surprising, as the ATO has only about 20,000 staff, compared to the population of Australia at over 23 million. Of course, access to the ATO web site is not limited to Australia. This distribution between the external and internal sources helps to garner our confidence in the quality of the data and in our understanding of the actual variables (i.e., we have perhaps a valid interpretation of what *source* records).

Given the clear differentiation between the external and internal populations, we might think to partition our analysis into the two cohorts. At a guess, we might expect the behaviours exhibited internally to be quite different to those exhibited through external accesses.

2.6 External Visits

We might want to examine just the external (i.e., the public, so we assume) browsing of the ATO web site.

```
ds <- ds %>% subset(source=="External")
dim(ds)

## [1] 144400      5

ds

## Source: local data frame [144,400 x 5]
##
##                                         entry_...
##                                         http://www.ato.gov...
## 1
....
```

We now want to review some of the entry pages. Let's pick a random sample:

```
ds$entry_page[sample(nrow(ds), 10)]

## [1] http://www.ato.gov.au/content/16689.htm ...
## [2] http://www.ato.gov.au/tax-professionals/consultation--tax-practitione...
## [3] http://www.ato.gov.au/content/00210181.htm ...
## [4] http://www.ato.gov.au/general/new-legislation/in-detail/a-z-index/ ...
....
```



```
ds$entry_page %>% unique() %>% length()

## [1] 31363
```

We can list the most frequent ones:

```
tbl <- ds$entry_page %>% table()
summary(tbl)

## Number of cases in table: 144400
## Number of factors: 1
```

We should carefully take a look at this table as on first impressions it might not be what we expected. Why are there many zeros, and why does it only go up to 11?

This brings us back to ensuring we understand what the dataset records. We may have lost sight by now of the fact that this is not a unit level dataset—i.e., it is not a record of individual visits. Rather, the dataset consists of *views* and *visits* aggregated monthly. So to get a real indication of entry page's popularity over the whole period we should aggregate the overall data.

```
dsa <- ds %>% group_by(entry_page) %>% summarise(total=sum(visits))
head(dsa$total)

## [1] 197   3    1    2    2    1
```

Exercise: Split the URLs into components and aggregate and analyse.

3 ATO Browser Data

The ATO browser dataset is extracted in the same way. The specific project code is again obtained from the URL from [data.gov.au](#). We download the ZIP file into a temporary location, extract the CSV, and read it into R.

```

fname <- file.path(atogovau,
                     "08e33518-dd6f-42d8-a035-f4c0fc7f18f3",
                     "download",
                     brzname)
temp <- tempfile(fileext=".zip")
download.file(fname, temp)
browser <- read.csv(unz(temp, brfname))
unlink(temp)

dsname      <- "browser"
ds          <- dsname %>% get() %>%tbl_df()
names(ds)    <- normVarNames(names(ds))
names(ds)[3] <- "source"
names(ds)

## [1] "browser" "month"   "source"   "views"   "visits"

vars        <- names(ds)
dso         <- ds
ds

## Source: local data frame [1,357 x 5]
##
##           browser month   source   views visits
## 1           Chrome Jul-13 External 7765921 691120
## 2           Chrome Jul-13 Internal  454    110
## 3 Microsoft Internet Explorer 10.x Jul-13 External 6773492 557509
## 4           Mobile Safari Jul-13 External 6067298 519093
## 5 Microsoft Internet Explorer 8.x Jul-13 External 5078805 455815
## 6 Microsoft Internet Explorer 8.x Jul-13 Internal 666710 69007
## 7           Firefox Jul-13 External 4440016 436873
## 8           Firefox Jul-13 Internal 19699 1387
## 9 Microsoft Internet Explorer 9.x Jul-13 External 3923983 386112
## 10          Safari Jul-13 External 2168245 245044
## ...
...     ...     ...     ...     ...

```

3.1 Explore

Now we continue *living and breathing the data*:

```
ds[sample(nrow(ds), 6),]  
## Source: local data frame [6 x 5]  
##  
##          browser month   source views visits  
## 1233  Netscape Navigator 4.x Apr-14 External  4204   3698  
....  
  
summary(ds)  
##  
##          browser      month      source  
## Chrome           : 20  Mar-14 :163  External:1304  
## Firefox          : 20  Oct-13 :143  Internal:  53  
## Microsoft Internet Explorer 6.x: 20  Apr-14 :141  
## Microsoft Internet Explorer 7.x: 20  Nov-13 :137  
## Microsoft Internet Explorer 8.x: 20  Feb-14 :136  
## External          : 11  Jan-14 :132  
## (Other)           :1246 (Other):505  
....
```

We can see that the data records aggregated monthly observations of the browsers connecting to the ATO web site. The connection may be internal to the organisation or external. It would appear most connections are external. Given a browser, month, and source, we have provided for us the aggregated number of views and visits.

3.2 Clean And Enhance

We might notice that the levels for the `month` are in alphabetic order whilst we would normally want these to be ordered chronologically. We can fix that:

```
levels(ds$month)
## [1] "Apr-14" "Aug-13" "Dec-13" "Feb-14" "Jan-14" "Jul-13" "Mar-14"
## [8] "Nov-13" "Oct-13" "Sep-13"

ds$month <- factor(ds$month, levels=months)
levels(ds$month)
## [1] "Jul-13" "Aug-13" "Sep-13" "Oct-13" "Nov-13" "Dec-13" "Jan-14"
## [8] "Feb-14" "Mar-14" "Apr-14"
```

We might be interested in the average number of views per visit, We can simply add this as an extra variable of the dataset.

```
ds$ratio <- ds$views/ds$visits
summary(ds$ratio)

##      Min. 1st Qu. Median      Mean 3rd Qu.      Max.
##      1.00    1.00    2.00    3.33    4.00   151.00
```

3.3 Explore Internal Usage

Let's get some understanding of the internal versus external profiles of browser usage. Firstly, how many internal versus external visits? The `dplyr` package is the way to go here. We pass the dataset on to the `group_by()` function, grouping the dataset by the values of the `source` variable, and then on to `summarise()` to construct a new data frame tabling the results.

```
freq <- ds %>%
  group_by(source) %>%
  summarise(total=sum(visits))
freq
## Source: local data frame [2 x 2]
##
##      source     total
## 1 External 33946553
## 2 Internal 1103712
```

We see that internal visits account for just 3% of all visits.

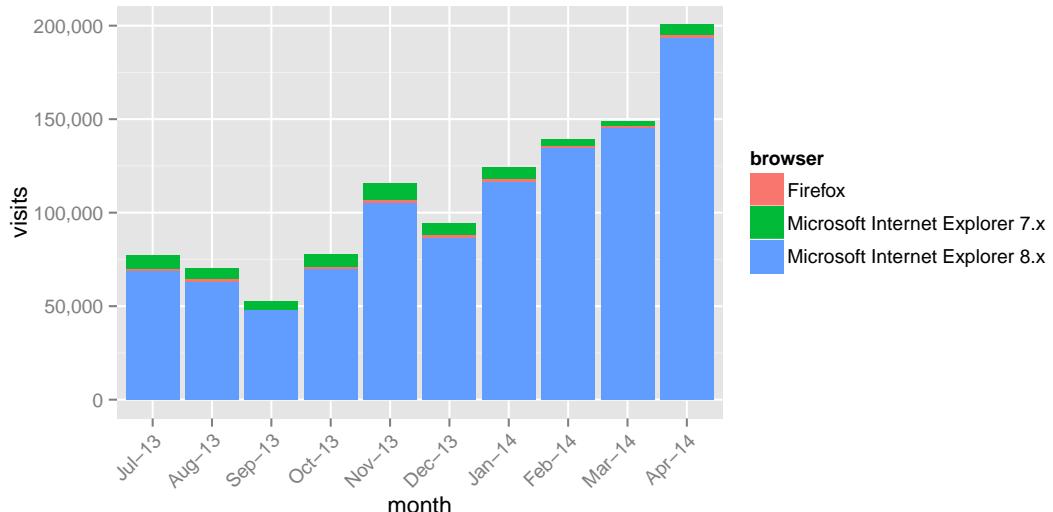
For the Internal users we now check which browsers are being used. Once again `dplyr` comes in handy.

```
ib <- ds %>%
  filter(source == "Internal") %>%
  group_by(browser) %>%
  summarise(total=sum(visits)) %>%
  arrange(desc(total))
ib
## Source: local data frame [8 x 2]
##
##      browser     total
## 1 Microsoft Internet Explorer 8.x 1032002
## 2 Microsoft Internet Explorer 7.x  56681
## 3 Firefox        12793
## 4 Microsoft Internet Explorer 6.x  1440
## 5 Chrome          792
## 6 External         2
## 7 Microsoft Internet Explorer 9.x  1
## 8 Safari           1
```

The ATO apparently deploys Microsoft Internet Explorer 8 as part of its standard operating environment. There's a few other browsers, but they are relatively rarely used.

3.4 Internal Usage Over Time

It could be interesting to explore the browser usage profile over time. For this we will plot using `ggplot2` (Wickham and Chang, 2014). A simple bar chart might be informative. Here we include all observations where the source is Internal and we have a decent number of visits (smaller number of visits will not show up the in the plot even if they are included in the data because there are relatively far to few).



```
ds %>%
  filter(source=="Internal", visits > 1000) %>%
  ggplot(aes(month, visits, fill=browser)) +
  geom_bar(stat="identity") +
  scale_y_continuous(labels=comma) +
  theme(axis.text.x=element_text(angle=45, hjust=1))
```

It is interesting, if also puzzling, to see quite an increase in visits over the 10 months. We might ask if the data is actually complete as a 4-fold increase in internal visits between September 2013 and April 2014 sounds rather sudden.

Exercise: Ensure the legend is the same order as the plot.

3.5 External Visits

We might compare the internal browser usage to the external browser usage

```
eb <- ds %>%
  filter(source == "External") %>%
  group_by(browser) %>%
  summarise(total=sum(visits)) %>%
  arrange(desc(total))
head(eb, 10)

## Source: local data frame [10 x 2]
##
##          browser   total
## 1           Chrome 7508320
## 2 Microsoft Internet Explorer 10.x 5258583
## 3        Mobile Safari 5016803
## 4          Firefox 4296359
## 5 Microsoft Internet Explorer 8.x 3736043
## 6 Microsoft Internet Explorer 9.x 3458002
## 7          Safari 2282737
## 8 Microsoft Internet Explorer 7.x 1498382
## 9         Mozilla 634484
## 10 Microsoft Internet Explorer 6.x 151233
```

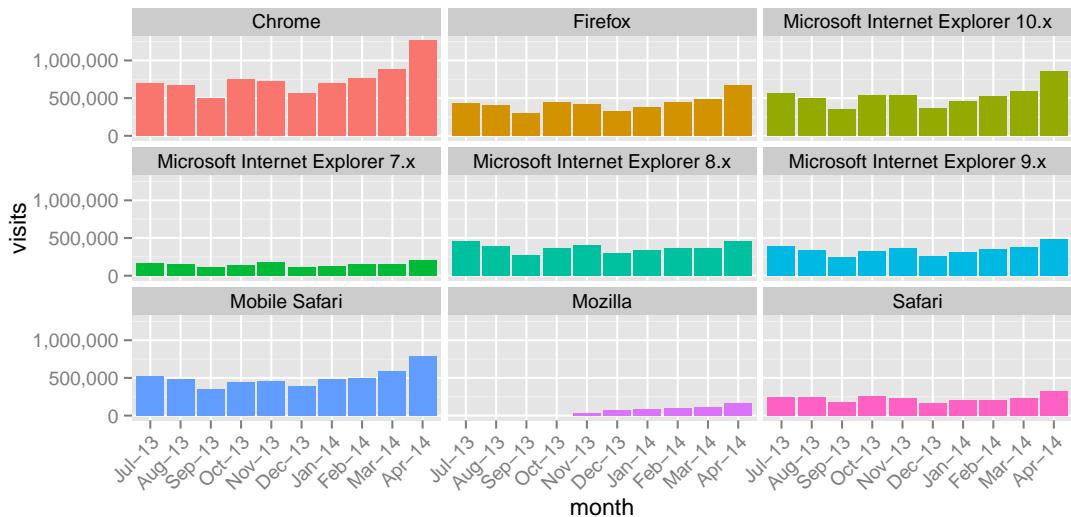
The ATO probably has careful control of the use of browsers within the organisation. Outside of the organisation, the choice of free and open source browsers is clearly more dominant.

```
tail(eb, 15)

## Source: local data frame [15 x 2]
##
##          browser   total
## 392      steller     1
## 393 Swingtel SX3 Linux     1
## 394    travelviajes     1
## 395       TTECHNO     1
## 396    TUUIUNMQAA     1
## 397 TyroTerminalAdapter     1
## 398 undefined GoogleToolbarBB     1
## 399          w3m     1
## 400 windows internet explorer 9     1
## 401          Windows NT     1
## 402        X9 NOTE Linux     1
## 403        Xiaomi 2013061 TD     1
## 404 xmlset roodkcableoj28840ybtide     1
## 405        ZTEU795 TD     1
## 406        ZTEU880F1 TD     1
```

Looking at the other end of the scale, we see quite a spread of single hit browsers. In fact, out of the 406 there are 142 browsers with a single visit, 297 with less than 10 visits, 372 with less than 100 visits, 391 with less than 1000 visits, and 394 with less than 20,000 visits.

3.6 External Visits



```
ds
  filter(source == "External", visits > 20000) %>%
  ggplot(aes(month, visits, fill= browser)) %>%
  geom_bar(stat="identity") +
  facet_wrap(~browser) +
  scale_y_continuous(labels=comma) +
  theme(axis.text.x=element_text(angle=45, hjust=1)) +
  theme(legend.position="none")
```

4 ATO Top 100 Keywords

We now load the ATO top 100 keywords for analysis. Once again, the specific project ID is obtained from the project URL at data.gov.au. We download the ZIP file into a temporary location, extract the CSV, and read it into R as the *keywords* dataset.

```
fname    <- file.path(atogovau,
                      "c6b745ec-439c-4dc4-872a-f48f5368d58e",
                      "download",
                      kwzname)
temp     <- tempfile(fileext=".zip")
download.file(fname, temp)
keywords <- read.csv(unz(temp, kwfname))
unlink(temp)
```

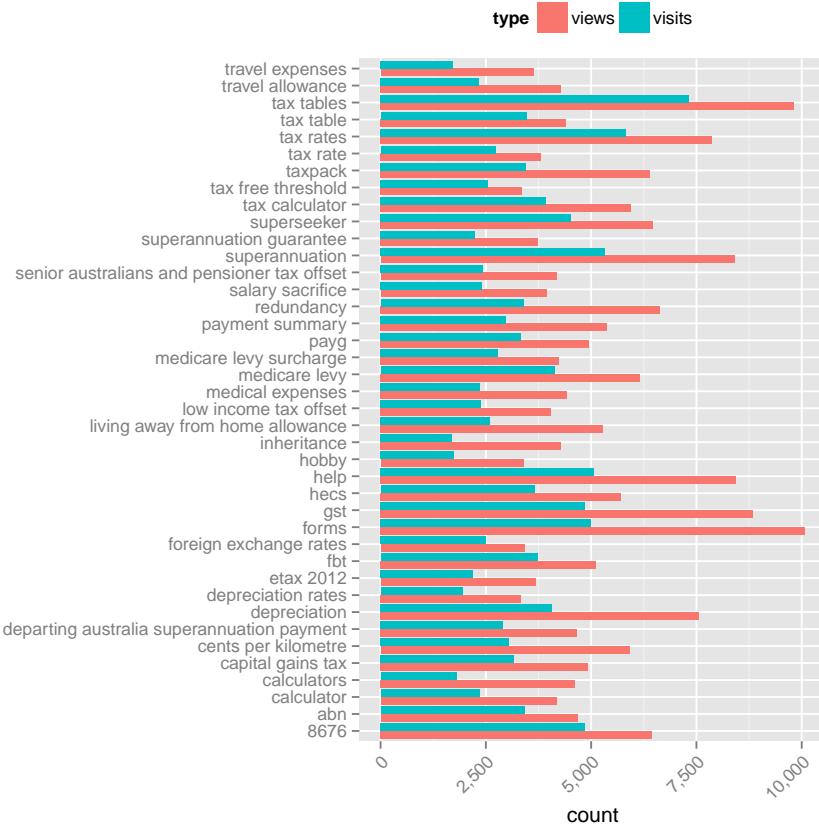
We prepare the dataset for analysis, and as usual we copy it to the generic *ds*, normalise and simplify the column names, order the months chronologically, and save a copy as *dso*:

```
dsname   <- "keywords"
ds       <- dsname %>% get() %>%tbl_df()
names(ds) <- normVarNames(names(ds))
names(ds)[1] <- "keyword"
names(ds)[3] <- "source"
ds$month <- factor(ds$month, levels=months)
vars     <- names(ds)
dso      <- ds

ds
## Source: local data frame [1,951 x 5]
##
##           keyword month source views visits
## 1         taxpack Jul-13 External  2262    942
## 2         taxpack Jul-13 Internal   43     24
## 3        payment summary Jul-13 External  2467   1050
## 4        payment summary Jul-13 Internal   33     21
.....
ds
group_by(keyword) %>%
summarise/views=sum(views), visits=sum(visits)) %>%
arrange(desc(views)) %>%
head(40)

## Source: local data frame [40 x 3]
##
##           keyword views visits
## 1         forms 10078  5004
## 2        tax tables 9820  7325
## 3          gst  8837  4857
## 4         help  8439  5075
....
```

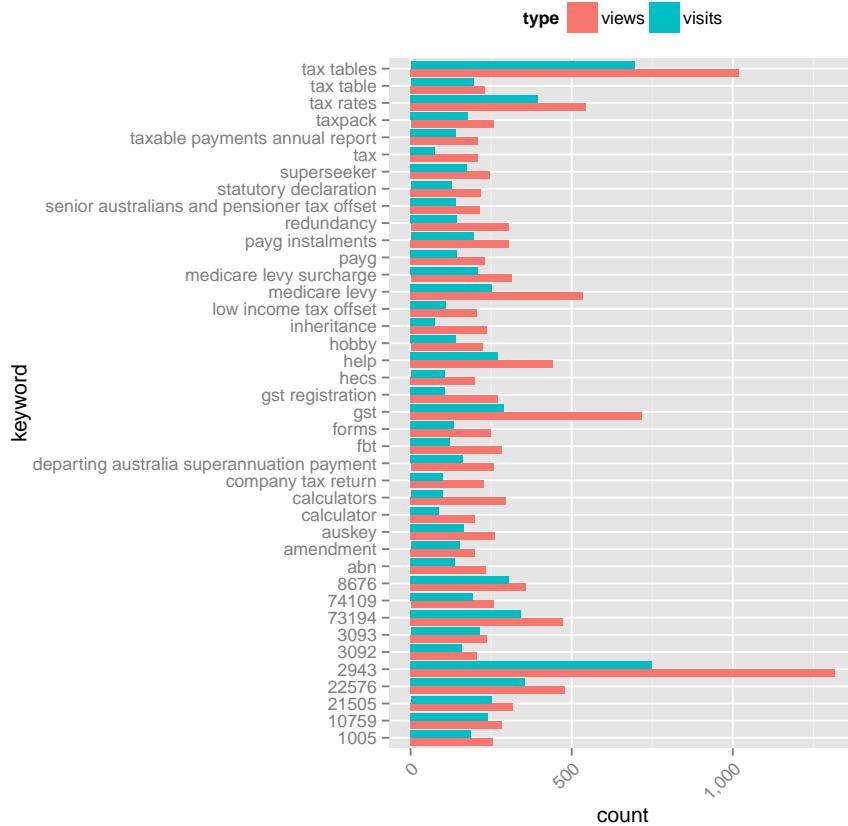
4.1 Plot Top 40



```
ds %>%
  group_by(keyword) %>%
  summarise(views = sum(views), visits = sum(visits)) %>%
  arrange(desc(views)) %>%
  head(40) %>%
  gather(type, count, -keyword) %>%
  ggplot(aes(x=keyword, y=count, fill=type)) +
  geom_bar(stat="identity", position="dodge") +
  scale_y_continuous(labels=comma) +
  theme(axis.text.x=element_text(angle=45, hjust=1)) +
  theme(legend.position="top") +
  labs(x="") +
  coord_flip()
```

Exercise: Order by the count of views with highest at the top.

4.2 Internal Only



```
ds %>%
  subset(source=="Internal") %>%
  group_by(keyword) %>%
  summarise(views=sum(views), visits=sum(visits)) %>%
  arrange(desc(views)) %>%
  head(40) %>%
  gather(type, count, -keyword) %>%
  ggplot(aes(x=keyword, y=count, fill=type)) +
  geom_bar(stat="identity", position="dodge") +
  scale_y_continuous(labels=comma) +
  theme(axis.text.x=element_text(angle=45, hjust=1)) +
  theme(legend.position="top") +
  coord_flip()
```

5 ATO OS Data

The ATO operating system dataset is loaded and analysed. Once again, the specific project ID is obtained from the URL from data.gov.au. We download the ZIP file into a temporary location, extract the CSV, and read it into R.

6 Risk Scoring Dashboard

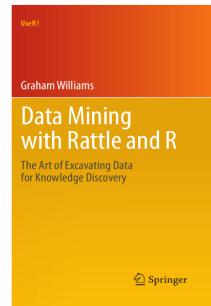
UNDER DEVELOPMENT

```
library(gdata)  
  
ds <- read.xls("results.xlsx")  
dim(ds)
```

7 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.



8 References

- Bache SM, Wickham H (2014). *magrittr: magrittr - a forward-pipe operator for R*. R package version 1.0.1, URL <http://CRAN.R-project.org/package=magrittr>.
- 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/>.
- Wickham H (2012). *stringr: Make it easier to work with strings*. R package version 0.6.2, URL <http://CRAN.R-project.org/package=stringr>.
- 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/>.

This document, sourced from CaseStudiesO.Rnw revision 479, was processed by KnitR version 1.6 of 2014-05-24 and took 8 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 2014-08-09 20:39:02.

13. Case Study: Web Log Analysis

Chapter pending / not available

Data Science with R

Building Models—A Template

Graham.Williams@togaware.com

20th August 2014

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

In this module we introduce a generic template for building models using R. The intention is that this document and the included scripts serve as a template for building a model and evaluating the model. This module builds on the Data module where we developed a template for preparing a dataset for analysing. The **weather** dataset from rattle (Williams, 2014) is used and for modeller we use **rpart** (Therneau and Atkinson, 2014).

The required packages for this module include:

```
library(rpart)          # Build decision tree model with rpart().  
library(randomForest)    # Build model with randomForest().  
library(ada)              # Build boosted trees model with ada().  
library(rattle)           # Display tree model with fancyRpartPlot().  
library(ROCR)             # Use prediction() for evaluation.  
library(party)            # Build conditional tree models with ctree() and cforest().  
library(ggplot2)           # Display evaluations.
```

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 Getting Started—Load the Dataset

In the Data module we loaded the **weather** dataset, processed it, and saved it to file. Here we load the dataset and review its contents.

```
(load("weather_130704.RData"))

## [1] "ds"      "dsname"  "dspath"   "dsdate"  "target"  "risk"    "id"
## [8] "ignore"  "vars"    "nobs"    "omit"    "inputi"  "inputc"  "numi"
## [15] "numc"   "cati"    "catc"

dsname
## [1] "weather"

dspath
## [1] "/home/gjw/R/x86_64-pc-linux-gnu-library/3.1/rattle/csv/weather.csv"

dsdate
## [1] "_130704"

dim(ds)
## [1] 366  24

id
## [1] "date"     "location"

target
## [1] "rain_tomorrow"

risk
## [1] "risk_mm"

ignore
## [1] "date"           "location"        "risk_mm"         "temp_3pm"
## [5] "pressure_9am"   "temp_9am"

vars
## [1] "min_temp"       "max_temp"       "rainfall"
## [4] "evaporation"    "sunshine"       "wind_gust_dir"
## [7] "wind_gust_speed" "wind_dir_9am"   "wind_dir_3pm"
## [10] "wind_speed_9am"  "wind_speed_3pm" "humidity_9am"
## [13] "humidity_3pm"    "pressure_3pm"   "cloud_9am"
## [16] "cloud_3pm"       "rain_today"     "rain_tomorrow"
```

2 Step 4: Prepare—Formula to Describe the Goal

We continue on from the Data module where we had Steps 1, 2, and 3 and the beginnings of Step 4 of a data mining process.

The next step is to describe the model to be built by way of writing a formula to capture our intent. The formula describes the model to be built as being constructed to predict the target variable based on the other (suitable) variables available in the dataset. The notation used to express this is to name the target (*rain_tomorrow*), followed by a tilde (~) followed by a period (.) to represent all other variables (these variables will be listed in *vars* in our case).

```
(form <- formula(paste(target, "~ .")))
## rain_tomorrow ~ .
```

The formula indicates that we will fit a model to predict *rain_tomorrow* from all of the other variables.

3 Step 4: Prepare—Training and Testing Datasets

A common methodology for model building is to randomly partition the available data into a **training** dataset and **testing** dataset. We sometimes also introduce a third dataset called the **validation** dataset, used during the building of the model, but for now we will use just the two.

First we (optionally) initiate the random number sequence with a randomly selected seed, and report what the seed is so that we could repeat the experiments presented here if required. For consistency in this module we use a particular seed of 123.

```
(seed <- sample(1:1000000, 1))
## [1] 534896
seed <- 123
set.seed(seed)
```

Next we partition the dataset into two subsets. The first is a 70% random sample for building the model (the training dataset) and the second is the remainder, used to evaluate the performance of the model (the testing dataset).

```
length(train <- sample(nobs, 0.7*nobs))
## [1] 256
length(test  <- setdiff(seq_len(nobs), train))
## [1] 110
```

Any modelling we do will thus build the model on the 70% training dataset. Our model will then be surely quite good at predicting these observations. But how will the model perform in general when we use it to predict other, yet unseen, observations? The model's performance on the data on which it was trained will be a very optimistic (or biased) estimate of the true performance of the model on other datasets.

The testing dataset is a hold-out dataset in that it has not been used at all for building the model. So when we apply the model to this dataset we would expect it to have a lesser performance (e.g., a higher error rate). This is what we will generally observe, and we will see this in the following sections.

The overall error rate measured on the training dataset will be shown to be less than the error rate calculated on the testing dataset.

The error rate (or the performance measure in general) calculated on the testing dataset is closer to what we will obtain in general when we begin to use the model. It is an unbiased estimate of the true performance of the model.

We also record the actual outcomes and the risks.

```
actual.train <- ds[train, target]
actual      <- ds[test, target]
risks       <- ds[test, risk]
```

4 Step5: Build—Decision Tree

Now we build an `rpart()` decision tree, as an example model builder.

```
ctrl <- rpart.control(maxdepth=3)
system.time(model <- m.rp <- rpart(form, ds[train, vars], control=ctrl))

##      user    system elapsed
## 0.017    0.000    0.017

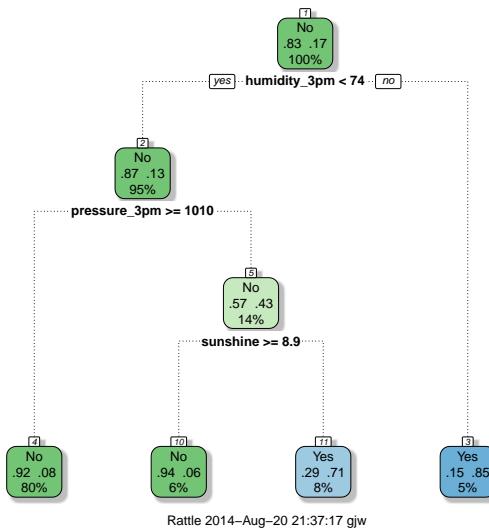
model

## n= 256
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 256 43 No (0.83203 0.16797)
##   2) humidity_3pm< 73.5 243 32 No (0.86831 0.13169)
##     4) pressure_3pm>=1010 206 16 No (0.92233 0.07767) *
##     5) pressure_3pm< 1010 37 16 No (0.56757 0.43243)
##       10) sunshine>=8.85 16  1 No (0.93750 0.06250) *
##       11) sunshine< 8.85 21  6 Yes (0.28571 0.71429) *
##   3) humidity_3pm>=73.5 13  2 Yes (0.15385 0.84615) *

mtype <- "rpart" # Record the type of the model for later use.
```

We can also draw the model.

```
fancyRpartPlot(model)
```



See the Decision Tree module for an explanation of decision tree models.

5 Step 6: Evaluate—Training Accuracy and AUC

We could now evaluate the model performance on the training dataset. As we noted above, this will give us a biased (optimistic) estimate of the true performance of the model.

First we use the model to predict the class of the observations of the training dataset.

```
head(cl <- predict(model, ds[train, vars], type="class"))

## 106 288 149 321 341 17
## No No No Yes Yes
## Levels: No Yes
```

We are going to compare this to the actual class for these observations from the training dataset.

```
head(actual.train)

## [1] No No No No Yes Yes
## Levels: No Yes
```

We can calculate the overall accuracy over the training dataset, as simply the sum of the number of times the prediction agrees with the actual class, divided by the size of the training dataset (which is the same as the number of actual values of the class).

```
(acc <- sum(cl == actual.train, na.rm=TRUE)/length(actual.train))

## [1] 0.9023
```

The model has an overall accuracy of 90%.

We could alternatively calculate the overall error rate in a similar fashion:

```
(err <- sum(cl != actual.train, na.rm=TRUE)/length(actual.train))

## [1] 0.09766
```

The model has an overall error rate of 10%.

The area under the so-called ROC curve can also be calculated using ROCR (Sing *et al.*, 2013).

```
pr <- predict(model, ds[train, vars], type="prob") [,2]
pred <- prediction(pr, ds[train, target])
(atr <- attr(performance(pred, "auc"), "y.values") [[1]])

## [1] 0.7882
```

The area under the curve (AUC) is 79% of the total error.

6 Step 6: Evaluate—Training Accuracy and AUC

As we have noted though, performing any evaluation on the training dataset provides a biased estimate of the actual performance. We must instead evaluate the performance of our models on a previously unseen dataset (at least unseen by the algorithm building the model).

So we now evaluate the model performance on the testing dataset.

```
cl <- predict(model, ds[test, vars], type="class")
(acc <- sum(cl == actual, na.rm=TRUE)/length(actual))
## [1] 0.8364
```

The overall accuracy is 84%.

```
(err <- sum(cl != actual, na.rm=TRUE)/length(actual))
## [1] 0.1636
```

The overall error rate is 16%.

```
pr <- predict(model, ds[test, vars], type="prob") [,2]
pred <- prediction(pr, ds[test, target])
(ate <- attr(performance(pred, "auc"), "y.values") [[1]])
## [1] 0.6827
```

The AUC is 68%.

All of these performance measures are less than what we found on the trianing dataset, as we should expect.

7 Step 6: Evaluate—Confusion Matrix

Exersice: In one or two paragraphs, explain the concept of the confusion matrix and define true/false positive/negative.

Firstly for the training dataset.

```
cl <- predict(model, ds[train, vars], type="class")
round(100*table(actual.train, cl, dnn=c("Actual", "Predicted"))/length(actual.train))

##      Predicted
## Actual No Yes
##   No    80   3
##   Yes   7   10
....
```

Now for the testing dataset.

```
cl <- predict(model, ds[test, vars], type="class")
round(100*table(actual, cl, dnn=c("Actual", "Predicted"))/length(actual))

##      Predicted
## Actual No Yes
##   No    74   5
##   Yes   11   10
....
```

Once again notice that the performance on the testing dataset is lesser than the performance on the training dataset.

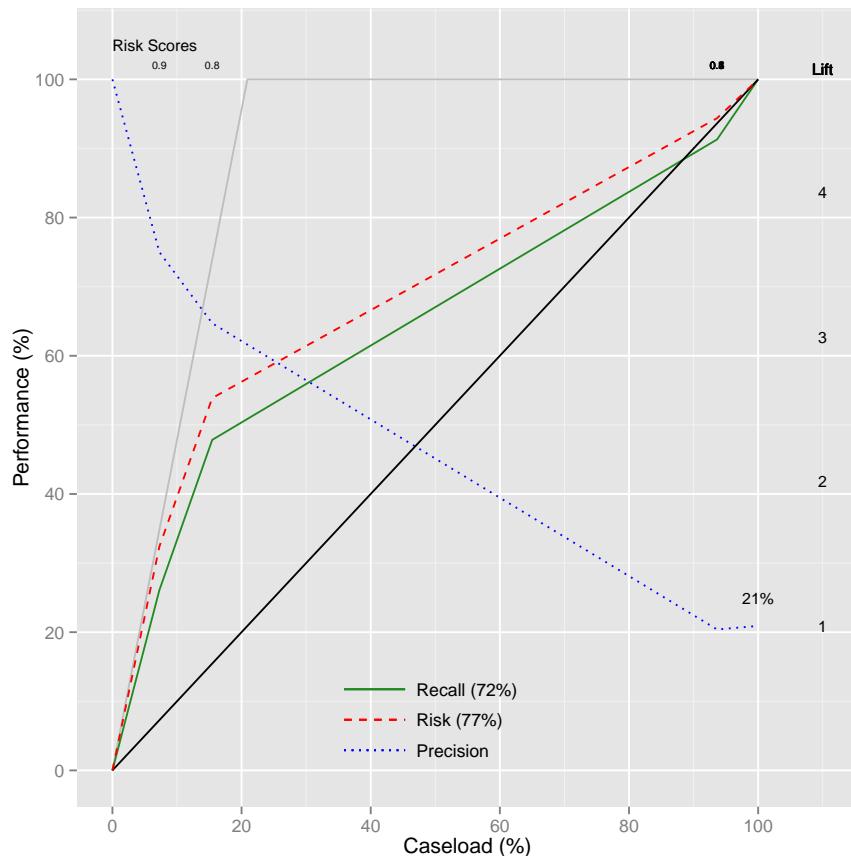
A **confusion matrix** is also called an **error matrix** or **contingency table**. The problem with calling them an error matrix is that not every cell in the table reports an error. The diagonals record the accuracy. Thus a quick look by a new data scientist might lead them to think the errors are high, based on the diagonals. The potential for this confusion is a good reason to call them something other than an error matrix.

Exercise: In the context of predicting whether it will rain tomorrow, and my decision to take an umbrella, explain the different consequences of a false positive and a false negative.

8 Step 6: Evaluate—Risk Chart

A risk chart is also known as an accumulative performance plot.

```
riskchart(pr, ds[test, target], ds[test, risk])
```



9 Step 7: Experiment—Framework

We can repeat the modelling multiple times, randomly selecting different datasets for training, to get an estimate of the actual expected performance and variation we see in the performance. The helper function `experi()` can be used to assist us here. It is available as <http://onepager.togaware.com/experi.R> and we show some of the coding of `experi()` below.

```
experi <- function(form, ds, dsname, target, modeller, details="",
                     n=100, control=NULL,
                     keep=FALSE, # Keep the last model built.
                     prob="prob",
                     class="class",
                     log="experi.log")
{
  suppressPackageStartupMessages(require(pROC))

  user <- Sys.getenv("LOGNAME")
  node <- Sys.info()[[ "nodename" ]]

  wsrpart.model <- modeller=="wsrpart"

  numclass <- length(levels(ds[,target]))

  start.time <- proc.time()

  seeds <- cors <- strs <- aucs <- accs <- NULL
  for (i in seq_len(n))
  {
    loop.time <- proc.time()

    seeds <- c(seeds, seed <- sample(1:1000000, 1))
    set.seed(seed)

    . . .

    result[-c(1:7)] <- round(result[-c(1:7)], 2)

    row.names(result) <- NULL

    if (keep)
    {
      if (numclass==2)
      {
        attr(result, "pr") <- pr
        attr(result, "test") <- test
      }
      attr(result, "model") <- model
    }
  }
}
```

```
    return(result)  
}
```

10 Step 7: Experiment—Results

Run the experiments using the algorihtms `rpart` (Therneau and Atkinson, 2014), `randomForest` (Breiman *et al.*, 2012), `ada` (Culp *et al.*, 2012), `ctree()` and `cforest()` from `party` (Hothorn *et al.*, 2013).

```
source("http://onepager.togaware.com/experi.R")

n <- 10

ex.rp <- experi(form, ds[vars], dsname, target, "rpart", "1", n=n, keep=TRUE)

ex.rf <- experi(form, ds[vars], dsname, target, "randomForest", "500", n=n, keep=TRUE,
                 control=list(na.action=na.omit))

ex.ad <- experi(form, ds[vars], dsname, target, "ada", "50", n=n, keep=TRUE)

ex.ct <- experi(form, ds[vars], dsname, target, "ctree", "1", n=n, keep=TRUE)

# Generates: error code 1 from Lapack routine 'dgesdd'
ex.cf <- experi(form, ds[vars], dsname, target, "cforest", "500", n=n, keep=TRUE)

results <- rbind(ex.rp, ex.rf, ex.ad, ex.ct)
rownames(results) <- results$modeller
results$modeller <- NULL
results

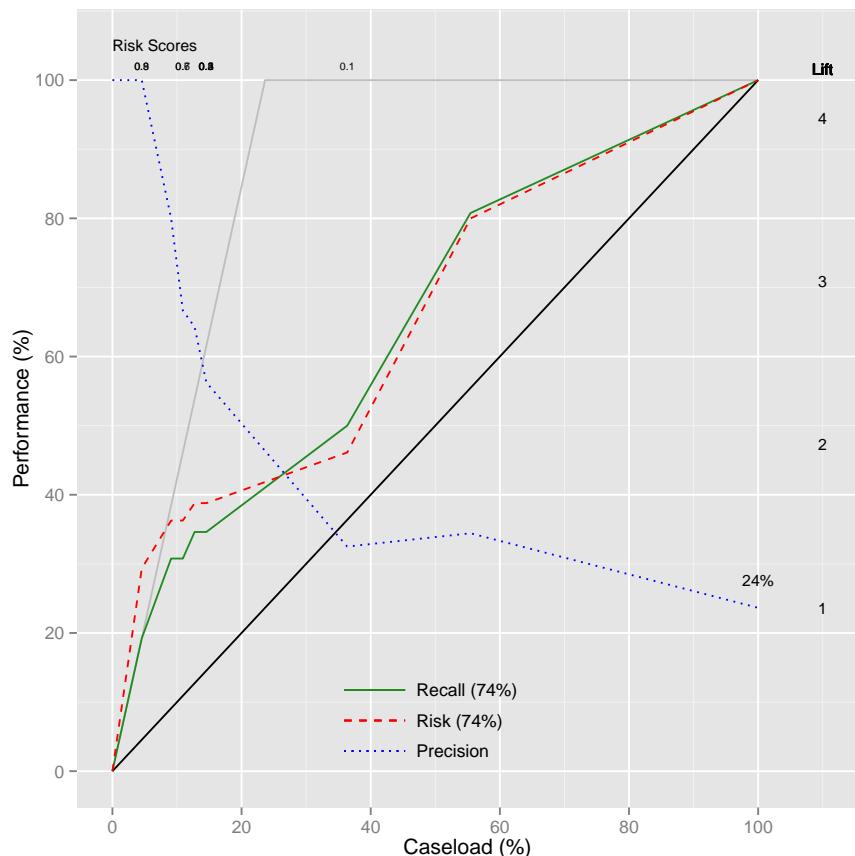
##               auc auc.sd cor cor.sd str str.sd acc acc.sd n user
## rpart_1        0.72  0.08 NA     NA   NA     NA 0.81  0.02 10 0.94
## randomForest_500 0.84  0.08 NA     NA   NA     NA 0.77  0.04 10 2.38
## ada_50         0.82  0.02 NA     NA   NA     NA 0.84  0.03 10 14.07
## ctree_1         0.73  0.04 NA     NA   NA     NA 0.82  0.03 10 0.91
##               elapsed
## rpart_1          0.95
## randomForest_500 2.39
## ada_50           14.11
## ctree_1          0.92
```

Exercise: Repeat these experiments using other model builders including C5.0, J48, lm, svm, nnet.

11 Step 7: Experiment—Riskchart Decision Tree

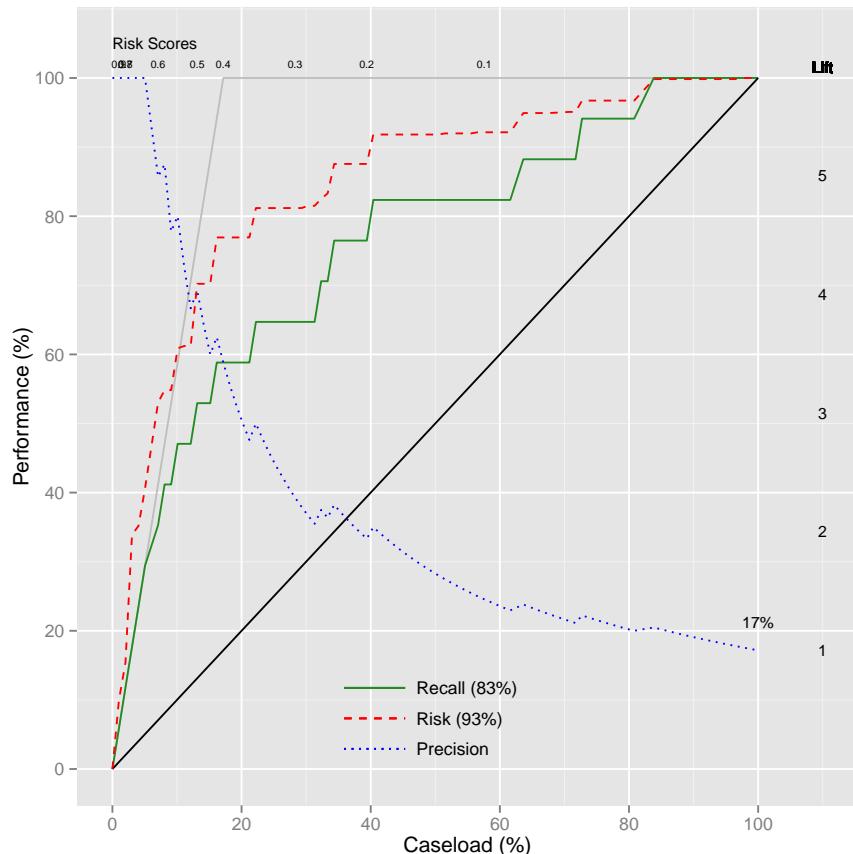
The result of the call to `experi()` includes the attributes `pr` and `test` if the number of classes is two and `keep=TRUE`. We can thus use this to generate a risk chart for the last of the models built in the experiment.

```
ex <- ex.rp
pr <- attr(ex, "pr")
test <- attr(ex, "test")
riskchart(pr, ds[test, target], ds[test, risk])
```



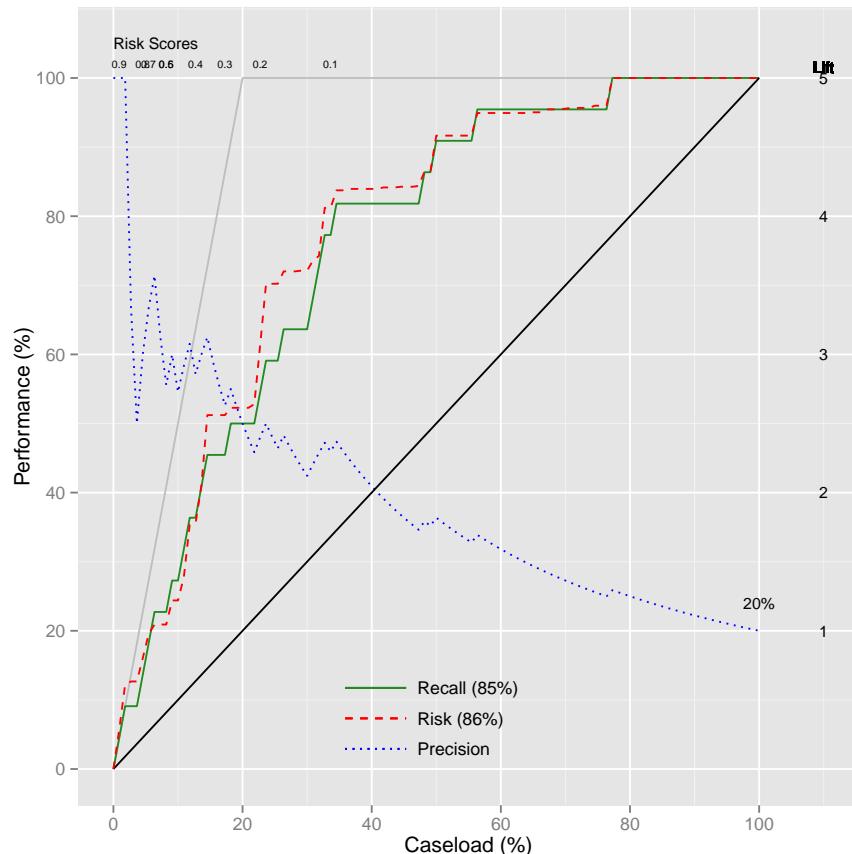
12 Step 7: Experiment—Riskchart Random Forest

```
ex <- ex.rf
pr <- attr(ex, "pr")
test <- attr(ex, "test")
riskchart(pr, ds[test, target], ds[test, risk])
```



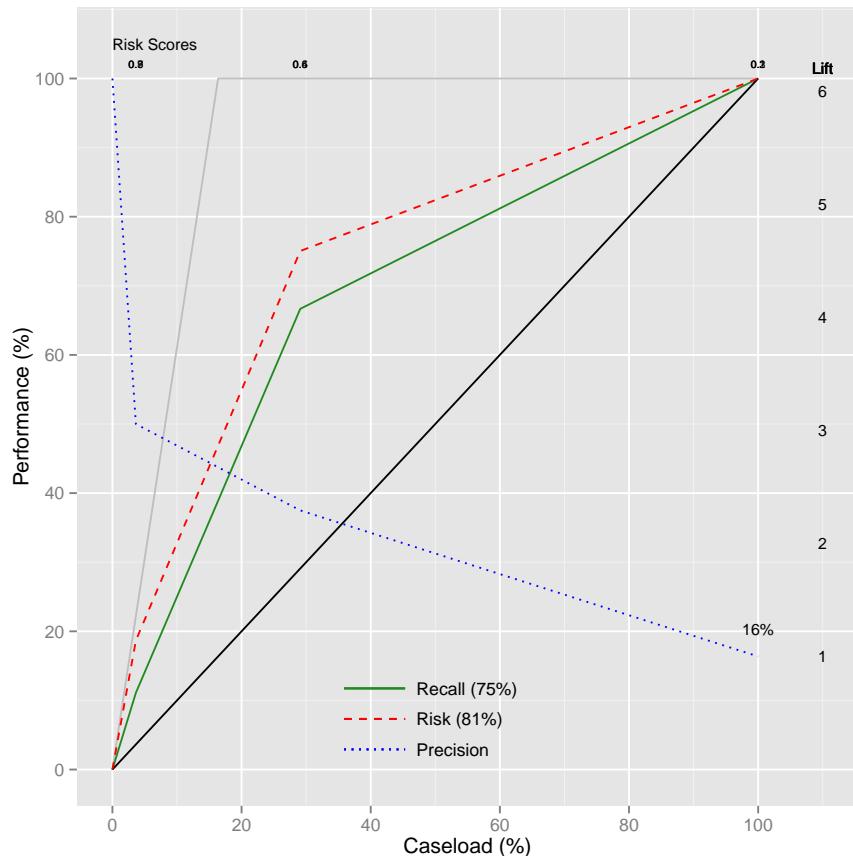
13 Step 7: Experiment—Riskchart Ada Boost

```
ex <- ex.ad
pr <- attr(ex, "pr")
test <- attr(ex, "test")
riskchart(pr, ds[test, target], ds[test, risk])
```



14 Step 7: Experiment—Riskchart Conditional Tree

```
ex <- ex.ct
pr <- attr(ex, "pr")
test <- attr(ex, "test")
riskchart(pr, ds[test, target], ds[test, risk])
```



15 Step 8: Finish Up—Save Model

We save the model, together with the dataset and other variables, into a binary R file.

```
fname <- "models"
if (! file.exists(dname)) dir.create(dname)
time.stamp <- format(Sys.time(), "%Y%m%d_%H%M%S")
fstem <- paste(dsname, mtype, time.stamp, sep="_")
(fname <- file.path(dname, sprintf("%s.RData", fstem)))
## [1] "models/weather_rpart_20140820_213740.RData"

save(ds, dsname, vars, target, risk, ignore,
      form, nobs, seed, train, test, model, mtype, pr,
      file= fname)
```

We can then load this later and replicate the process.

```
(load(fname))
## [1] "ds"      "dsname"   "vars"     "target"   "risk"    "ignore"   "form"
## [8] "nobs"    "seed"     "train"    "test"     "model"   "mtype"   "pr"
```

Note that by using generic variable names we can load different model files and perform common operations on them without changing the names within a script. However, do note that each time we load such a saved model file we overwrite any other variables of the same name.

16 Other Models—Random Forest

```
ctrl <- rpart.control(maxdepth=3)
system.time(model <- m.rf <- rpart(form, ds[train, vars], control=ctrl))

##      user    system elapsed
## 0.014    0.000    0.014

model

## n= 256
##
## node), split, n, loss, yval, (yprob)
##           * denotes terminal node
##
## 1) root 256 43 No (0.83203 0.16797)
## 2) humidity_3pm< 73.5 243 32 No (0.86831 0.13169)
## 4) pressure_3pm>=1010 206 16 No (0.92233 0.07767) *
## 5) pressure_3pm< 1010 37 16 No (0.56757 0.43243)
## 10) sunshine>=8.85 16 1 No (0.93750 0.06250) *
## 11) sunshine< 8.85 21 6 Yes (0.28571 0.71429) *
## 3) humidity_3pm>=73.5 13 2 Yes (0.15385 0.84615) *
```

17 Review—Model Process

Here in one block is the code to perform model building.

```
# Required packages
library(rpart)    # Model builder
library(rattle)   # riskchart()
library(ROCR)     # prediction()

# Load dataset.
load("weather_130704.RData")

# Formula for modelling.
form      <- formula(paste(target, " ~ ."))

# Training and test datasets.
seed      <- sample(1:1000000, 1)
set.seed(seed)
train     <- sample(nobs, 0.7*nobs)
test      <- setdiff(seq_len(nobs), train)
actual    <- ds[test, target]
risks     <- ds[test, risk]

# Build model.
ctrl      <- rpart.control(maxdepth=3)
m.rp      <- rpart(form, data=ds[train, vars], control=ctrl)
mtype    <- "rpart"
model    <- m.rp

# Review model.
fancyRpartPlot(model)

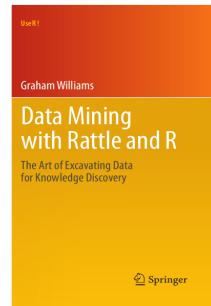
# Evaluate the model.
classes   <- predict(model, ds[test, vars], type="class")
acc       <- sum(classes == actual, na.rm=TRUE)/length(actual)
err       <- sum(classes != actual, na.rm=TRUE)/length(actual)
predicted <- predict(model, ds[test, vars], type="prob")[,2]
pred      <- prediction(predicted, ds[test, target])
ate       <- attr(performance(pred, "auc"), "y.values")[[1]]
riskchart(predicted, actual, risks)
psfchart(predicted, actual)
round(table(actual, classes, dnn=c("Actual", "Predicted"))/length(actual), 2)
```

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.

The process we have presented here for modelling from data has been tuned over many years of delivering analytics and data mining projects. An early influence was [CRISP-DM](#) the CRoss Industry Standard Process for Data Mining.



19 References

- Breiman L, Cutler A, Liaw A, Wiener M (2012). *randomForest: Breiman and Cutler's random forests for classification and regression*. R package version 4.6-7, URL <http://CRAN.R-project.org/package=randomForest>.
- Culp M, Johnson K, Michailidis G (2012). *ada: ada: an R package for stochastic boosting*. R package version 2.0-3, URL <http://CRAN.R-project.org/package=ada>.
- Hothorn T, Hornik K, Strobl C, Zeileis A (2013). *party: A Laboratory for Recursive Partitioning*. R package version 1.0-9, URL <http://CRAN.R-project.org/package=party>.
- 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/>.
- Sing T, Sander O, Beerenwinkel N, Lengauer T (2013). *ROCR: Visualizing the performance of scoring classifiers*. R package version 1.0-5, URL <http://CRAN.R-project.org/package=ROCR>.
- Therneau TM, Atkinson B (2014). *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-8, URL <http://CRAN.R-project.org/package=rpart>.
- 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/>.

This document, sourced from ModelsO.Rnw revision 502, was processed by KnitR version 1.6 of 2014-05-24 and took 24.3 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 2014-08-20 21:37:41.

Data Science with R

Cluster Analysis

Graham.Williams@togaware.com

22nd June 2014

Visit <http://onepager.togaware.com/> for more OnePageR's.

We focus on the unsupervised method of [cluster analysis](#) in this chapter. Cluster analysis is a topic that has been much studied by statisticians for decades and widely used in data mining.

The required packages for this module include:

```
library(rattle)      # The weather dataset and normVarNames().  
library(randomForest) # Impute missing values using na.roughfix().  
library(ggplot2)      # Visualise the data through plots.  
library(animation)    # Demonstrate kmeans.  
library(reshape2)     # Reshape data for plotting.  
library(fpc)          # Tuning clustering with kmeansruns() and clusterboot().  
library(clusterCrit)  # Clustering criteria.  
library(wskm)         # Weighted subspace clustering.  
library(amap)         # hclusterpar  
library(cba)          # Dendrogram plot  
library(dendroextras) # To colour clusters  
library(kohonen)      # Self organising maps.
```

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 Load Weather Dataset for Modelling

We use the **weather** dataset from **rattle** (Williams, 2014) and normalise the variable names. Missing values are imputed using `na.roughfix()` from **randomForest** (Breiman *et al.*, 2012), particularly because `kmeans()` does not handle missing values itself. Here we set up the dataset for modelling. Notice in particular we identify the numeric input variables (`numi` is an integer vector containing the column index for the numeric variables and `numc` is a character vector containing the column names). Many clustering algorithms only handle numeric variables.

```
# Required packages
library(rattle)           # Load weather dataset. Normalise names normVarNames().
library(randomForest)      # Impute missing using na.roughfix().

# Identify the dataset.
dsname    <- "weather"
ds        <- get(dsname)
names(ds) <- normVarNames(names(ds))
vars      <- names(ds)
target    <- "rain_tomorrow"
risk      <- "risk_mm"
id        <- c("date", "location")

# Ignore the IDs and the risk variable.
ignore    <- union(id, if (exists("risk")) risk)

# Ignore variables which are completely missing.
mvc       <- sapply(ds[vars], function(x) sum(is.na(x))) # Missing value count.
mvn       <- names(ds)[(which(mvc == nrow(ds)))]           # Missing var names.
ignore    <- union(ignore, mvn)

# Initialise the variables
vars      <- setdiff(vars, ignore)

# Variable roles.
inputc    <- setdiff(vars, target)
inputi    <- sapply(inputc, function(x) which(x == names(ds)), USE.NAMES=FALSE)
numi     <- intersect(inputi, which(sapply(ds, is.numeric)))
numc     <- names(ds)[numi]
cati     <- intersect(inputi, which(sapply(ds, is.factor)))
catc     <- names(ds)[cati]

# Impute missing values, but do this wisely - understand why missing.
if (sum(is.na(ds[vars]))) ds[vars] <- na.roughfix(ds[vars])

# Number of observations.
nobs     <- nrow(ds)
```

2 Introducing Cluster Analysis

The aim of cluster analysis is to identify groups of observations so that within a group the observations are most similar to each other, whilst between groups the observations are most dissimilar to each other. Cluster analysis is essentially an unsupervised method.

Our human society has been “clustering” for a long time to help us understand the environment we live in. We have clustered the animal and plant kingdoms into a hierarchy of similarities. We cluster chemical structures. Day-by-day we see grocery items clustered into similar groups. We cluster student populations into similar groups of students from similar backgrounds or studying similar combinations of subjects.

The concept of similarity is often captured through the measurement of distance. Thus we often describe cluster analysis as identifying groups of observations so that the distance between the observations within a group is minimised and between the groups the distance is maximised. Thus a distance measure is fundamental to calculating clusters.

There are some caveats to performing automated cluster analysis using distance measures. We often observe, particularly with large datasets, that a number of interesting clusters will be generated, and then one or two clusters will account for the majority of the observations. It is as if these larger clusters simply lump together those observations that don’t fit elsewhere.

3 Distance Calculation: Euclidean Distance

Suppose we pick the first two observations from our dataset and the first 5 numeric variables:

```
ds[1:2, numi[1:5]]
##   min_temp max_temp rainfall evaporation sunshine
## 1      8     24.3     0.0       3.4      6.3
## 2     14     26.9     3.6       4.4      9.7
x <- ds[1, numi[1:5]]
y <- ds[2, numi[1:5]]
```

Then $x - y$ is simply:

```
x-y
##   min_temp max_temp rainfall evaporation sunshine
## 1      -6     -2.6    -3.6       -1     -3.4
```

Then the square of each difference is:

```
sapply(x-y, '^', 2)
##   min_temp     max_temp     rainfall evaporation     sunshine
##      36.00      6.76      12.96      1.00      11.56
```

The sum of the squares of the differences:

```
sum(sapply(x-y, '^', 2))
## [1] 68.28
```

Finally the square root of the sum of the squares of the differences (also known as the [Euclidean distance](#)) is:

```
sqrt(sum(sapply(x-y, '^', 2)))
## [1] 8.263
```

Of course we don't need to calculate this so manually ourselves. R provides `dist()` to calculate the distance (Euclidean distance by default):

```
dist(ds[1:2, numi[1:5]])
##      1
## 2 8.263
```

We can also calculate the [Manhattan distance](#):

```
sum(abs(x-y))
## [1] 16.6
dist(ds[1:2, numi[1:5]], method="manhattan")
##      1
## 2 16.6
```

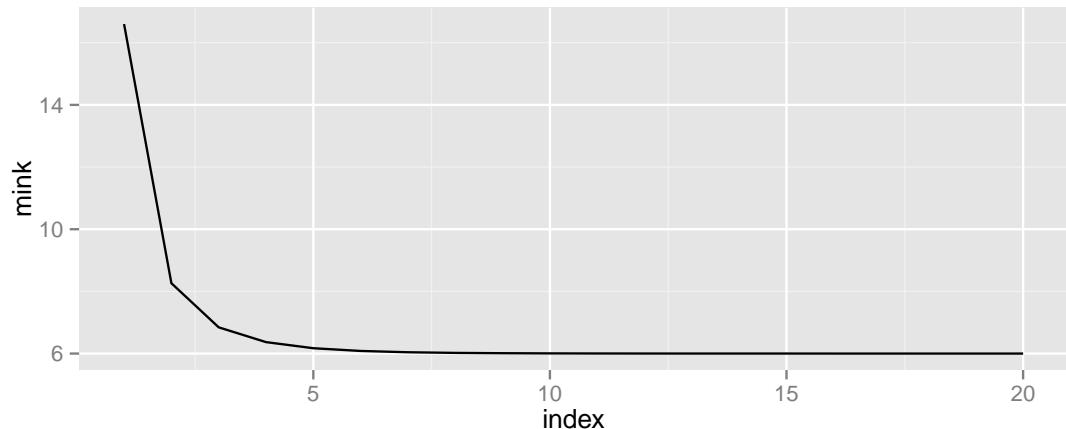
4 Minkowski Distance

```
dist(ds[1:2, numi[1:5]], method="minkowski", p=1)
##      1
## 2 16.6

dist(ds[1:2, numi[1:5]], method="minkowski", p=2)
##      1
## 2 8.263

dist(ds[1:2, numi[1:5]], method="minkowski", p=3)
##      1
## 2 6.844

dist(ds[1:2, numi[1:5]], method="minkowski", p=4)
##      1
## 2 6.368
```



5 General Distance

```
dist(ds[1:5, numi[1:5]])  
##      1     2     3     4  
## 2 8.263  
## 3 7.812 7.434  
## 4 41.375 38.067 37.531  
....  
  
daisy(ds[1:5, numi[1:5]])  
## Dissimilarities :  
##      1     2     3     4  
## 2 8.263  
## 3 7.812 7.434  
....  
  
daisy(ds[1:5, cati])  
## Dissimilarities :  
##      1     2     3     4  
## 2 0.6538  
## 3 0.6923 0.5385  
....
```

6 K-Means Basics: Iterative Cluster Search

The k-means algorithm is a traditional and widely used clustering algorithm.

The algorithm begins by specifying the number of clusters we are interested in—this is the k . Each of the k clusters is identified as the vector of the average (i.e., the mean) value of each of the variables for observations within a cluster. A random clustering is first constructed, the k means calculated, and then using the distance measure we gravitate each observation to its nearest mean. The means are then recalculated and the points re-gravitate. And so on until there is no change to the means.

7 K-Means: Using kmeans()

Here is our first attempt to cluster our dataset.

```
model <- m.km <- kmeans(ds, 10)
## Warning: NAs introduced by coercion
## Error: NA/NaN/Inf in foreign function call (arg 1)
```

The error is because there are non-numeric variables that we are attempting to cluster on.

```
set.seed(42)
model <- m.km <- kmeans(ds[numi], 10)
```

So that appears to have succeeded to build 10 clusters. The sizes of the clusters can readily be listed:

```
model$size
## [1] 29 47 24 55 21 33 35 50 41 31
```

The cluster centers (i.e., the *means*) can also be listed:

```
model$centers
##   min_temp max_temp rainfall evaporation sunshine wind_gust_speed
## 1    5.8448     20.38   0.75862       6.000   10.524        52.66
## 2   13.0340     31.42   0.09362       7.677   10.849        43.32
## 3   13.9833     21.02   7.14167       4.892    2.917        39.54
....
```

The component `m.km$cluster` reports which of the 10 clusters each of the original observations belongs:

```
head(model$cluster)
## [1] 4 8 6 6 6 10

model$iter
## [1] 6

model$ifault
## [1] 0
```

8 Scaling Datasets

We noted earlier that a unit of distance is different for differently measure variables. For example, one year of difference in age seems like it should be a larger difference than \$1 difference in our income. A common approach is to rescale our data by subtracting the mean and dividing by the standard deviation. This is often referred to as a z-score. The result is that the mean for all variables is 0 and a unit of difference is one standard deviation.

The R function `scale()` can perform this transformation on our numeric data. We can see the effect in the following:

```
summary(ds[numi[1:5]])  
  
##      min_temp          max_temp        rainfall      evaporation  
##  Min.   :-5.30    Min.   : 7.6    Min.   : 0.00    Min.   : 0.20  
##  1st Qu.: 2.30    1st Qu.:15.0    1st Qu.: 0.00    1st Qu.: 2.20  
##  Median : 7.45    Median :19.6    Median : 0.00    Median : 4.20  
....  
  
summary(scale(ds[numi[1:5]]))  
  
##      min_temp          max_temp        rainfall      evaporation  
##  Min.   :-2.0853   Min.   :-1.936   Min.   :-0.338   Min.   :-1.619  
##  1st Qu.:-0.8241   1st Qu.:-0.826   1st Qu.:-0.338   1st Qu.:-0.870  
##  Median : 0.0306   Median :-0.135   Median :-0.338   Median :-0.121  
....
```

The `scale()` function also provides some extra information, recording the actual original means and the standard deviations:

```
dsc <- scale(ds[numi[1:5]])  
attr(dsc, "scaled:center")  
  
##      min_temp          max_temp        rainfall      evaporation      sunshine  
##       7.266         20.550        1.428         4.522        7.915  
  
attr(dsc, "scaled:scale")  
  
##      min_temp          max_temp        rainfall      evaporation      sunshine  
##       6.026         6.691        4.226         2.669        3.468
```

Compare that information with the output from `mean()` and `sd()`:

```
sapply(ds[numi[1:5]], mean)  
  
##      min_temp          max_temp        rainfall      evaporation      sunshine  
##       7.266         20.550        1.428         4.522        7.915  
  
sapply(ds[numi[1:5]], sd)  
  
##      min_temp          max_temp        rainfall      evaporation      sunshine  
##       6.026         6.691        4.226         2.669        3.468
```

9 K-Means: Scaled Dataset

```
set.seed(42)
model <- m.kms <- kmeans(scale(ds[numi]), 10)
model$size
## [1] 34 54 15 70 24 32 30 44 43 20
model$centers
##   min_temp max_temp rainfall evaporation sunshine wind_gust_speed
## 1    1.0786    1.6740 -0.31018     1.43079    1.0397      0.6088
## 2    0.5325    0.9939 -0.24074     0.56206    0.8068     -0.2149
## 3    0.8808   -0.2307   3.77323     0.01928   -0.7599      0.4886
....
model$totss
## [1] 5840
model$withinss
## [1] 249.4 272.4 211.2 328.0 149.2 287.7 156.8 366.2 262.0 137.0
model$tot.withinss
## [1] 2420
model$betweenss
## [1] 3420
model$iter
## [1] 8
model$ifault
## [1] 0
```

10 Animate Cluster Building

Using `kmeans.ani()` from `animation` (Xie, 2013), we can produce an animation that illustrates the kmeans algorithm.

```
library(animation)
```

We generate some random data for two variables over 100 observations.

```
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")

dim(x)

## [1] 100    2

head(x)

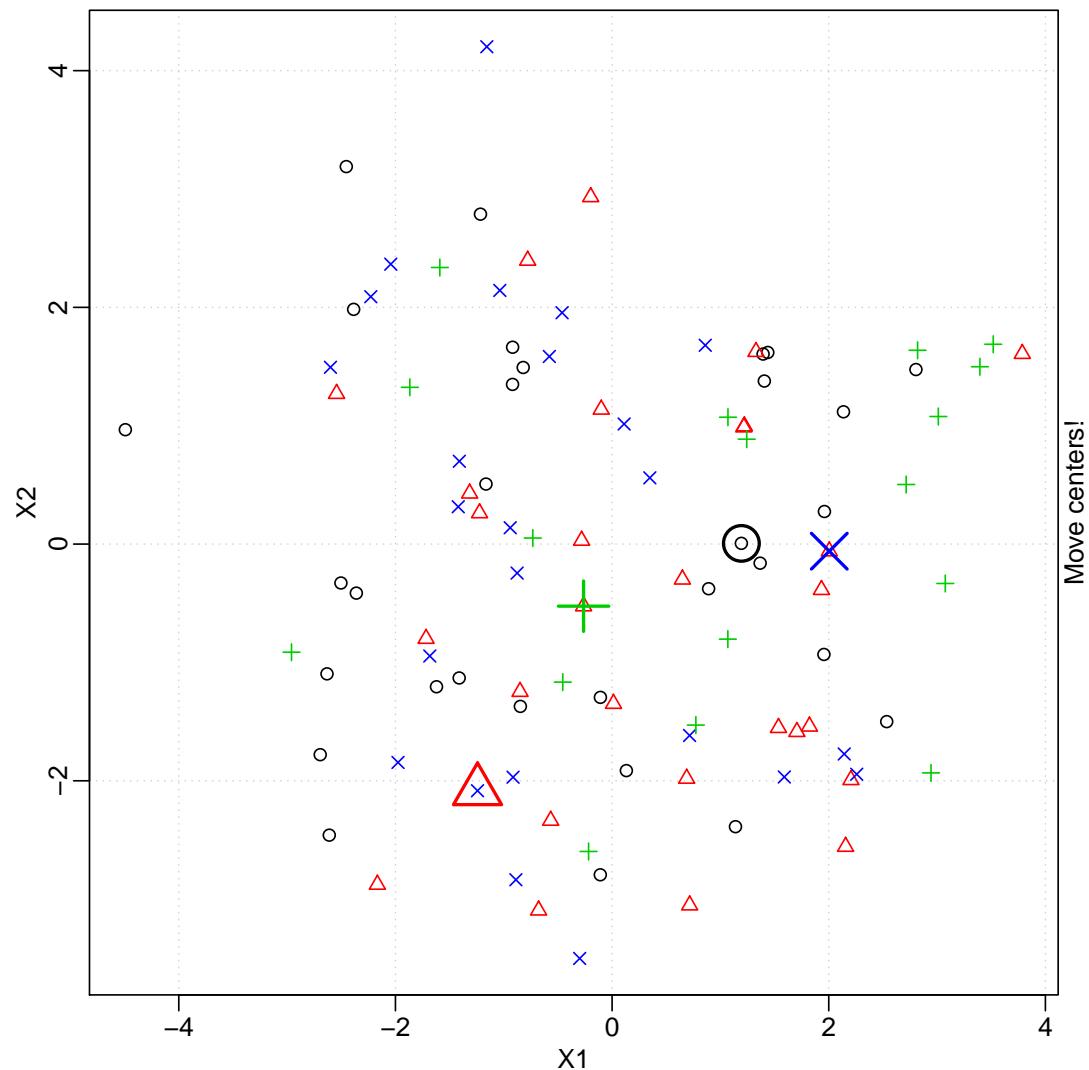
##      X1      X2
## [1,] 1.394 1.606
## [2,] 3.012 1.078
## [3,] 1.405 1.378
## [4,] -0.750 0.850
## [5,] 0.500 1.500
## [6,] 2.250 0.250

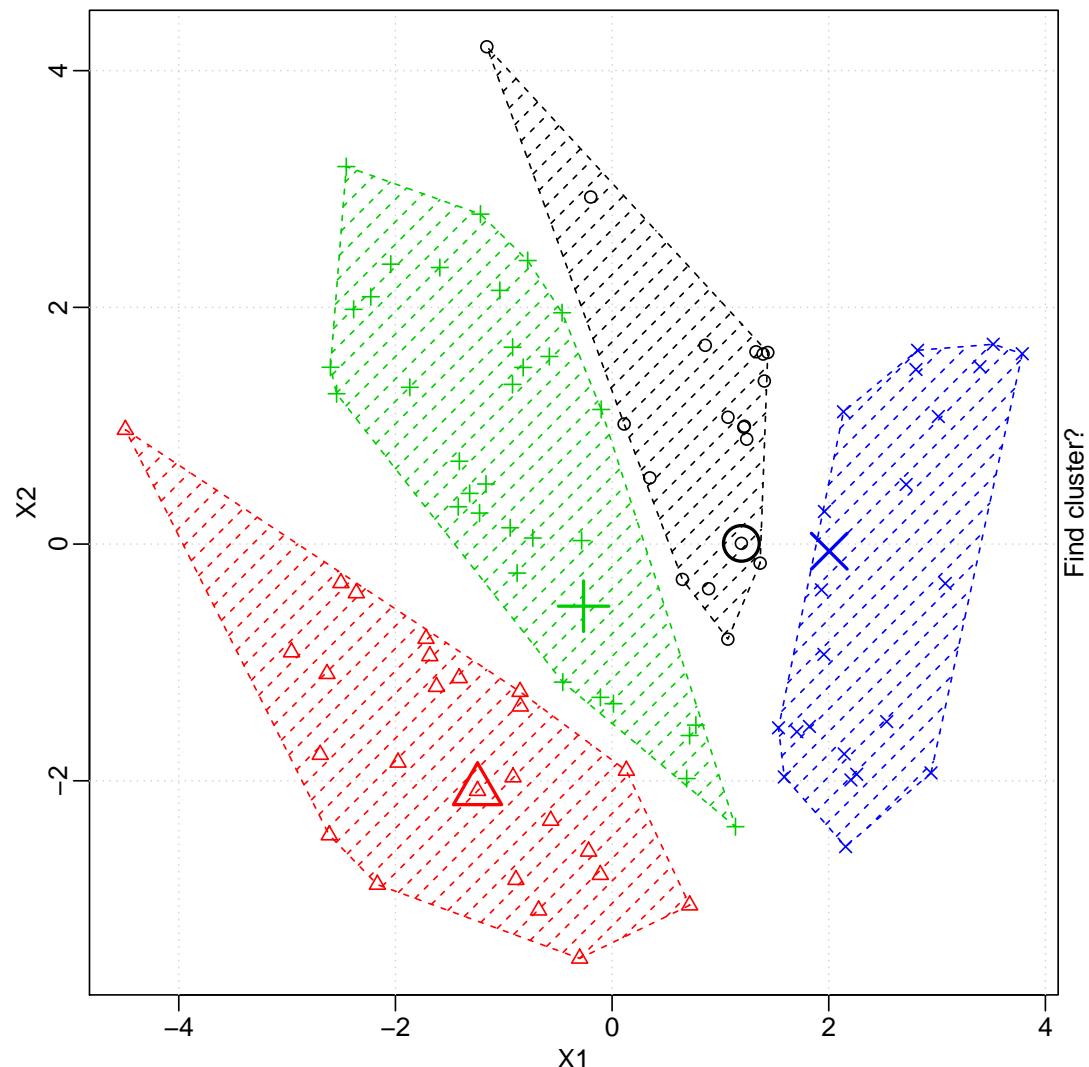
....
```

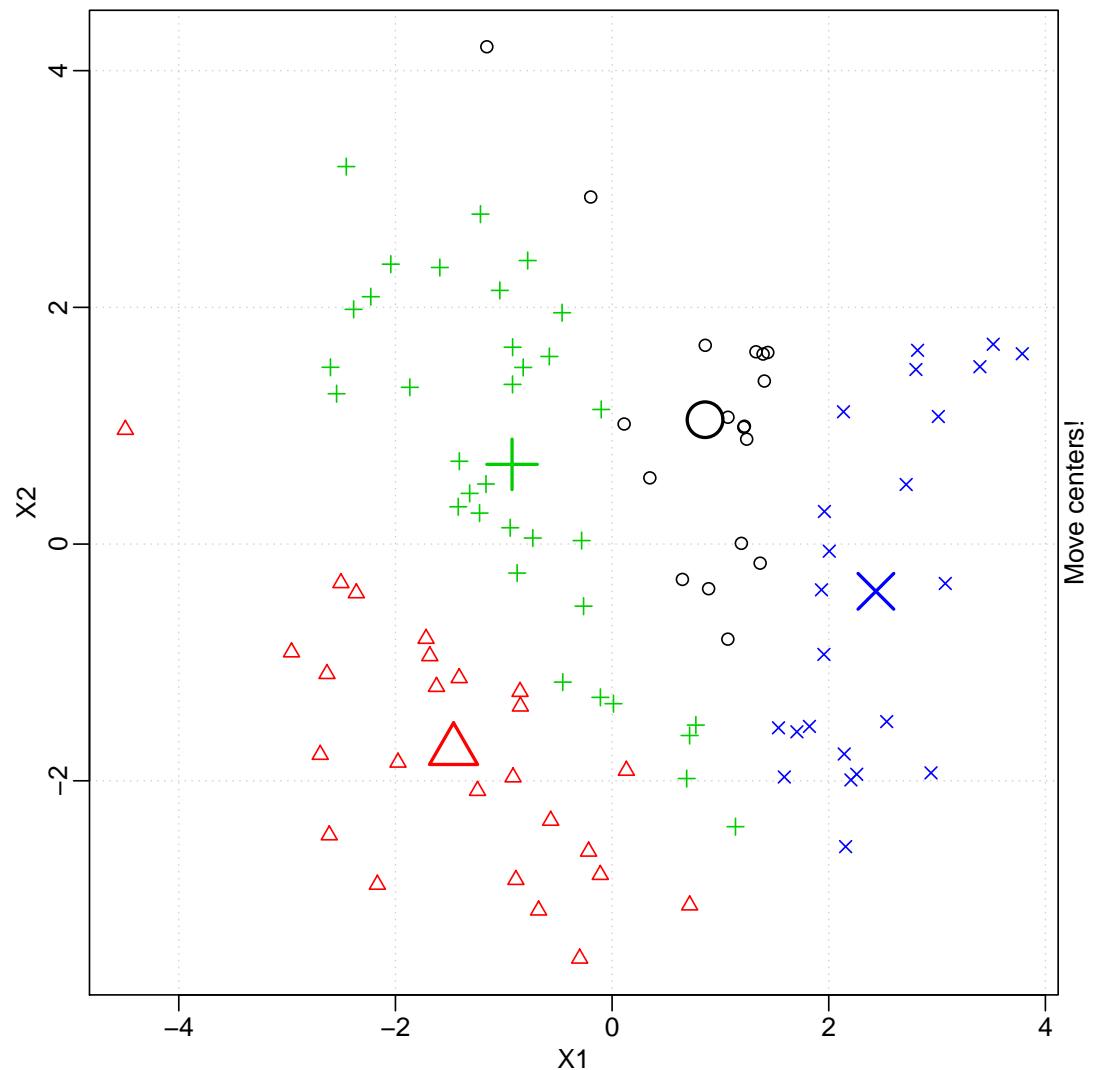
The series of plots over the following pages show the convergence of the kmeans algorithm to identify 4 clusters.

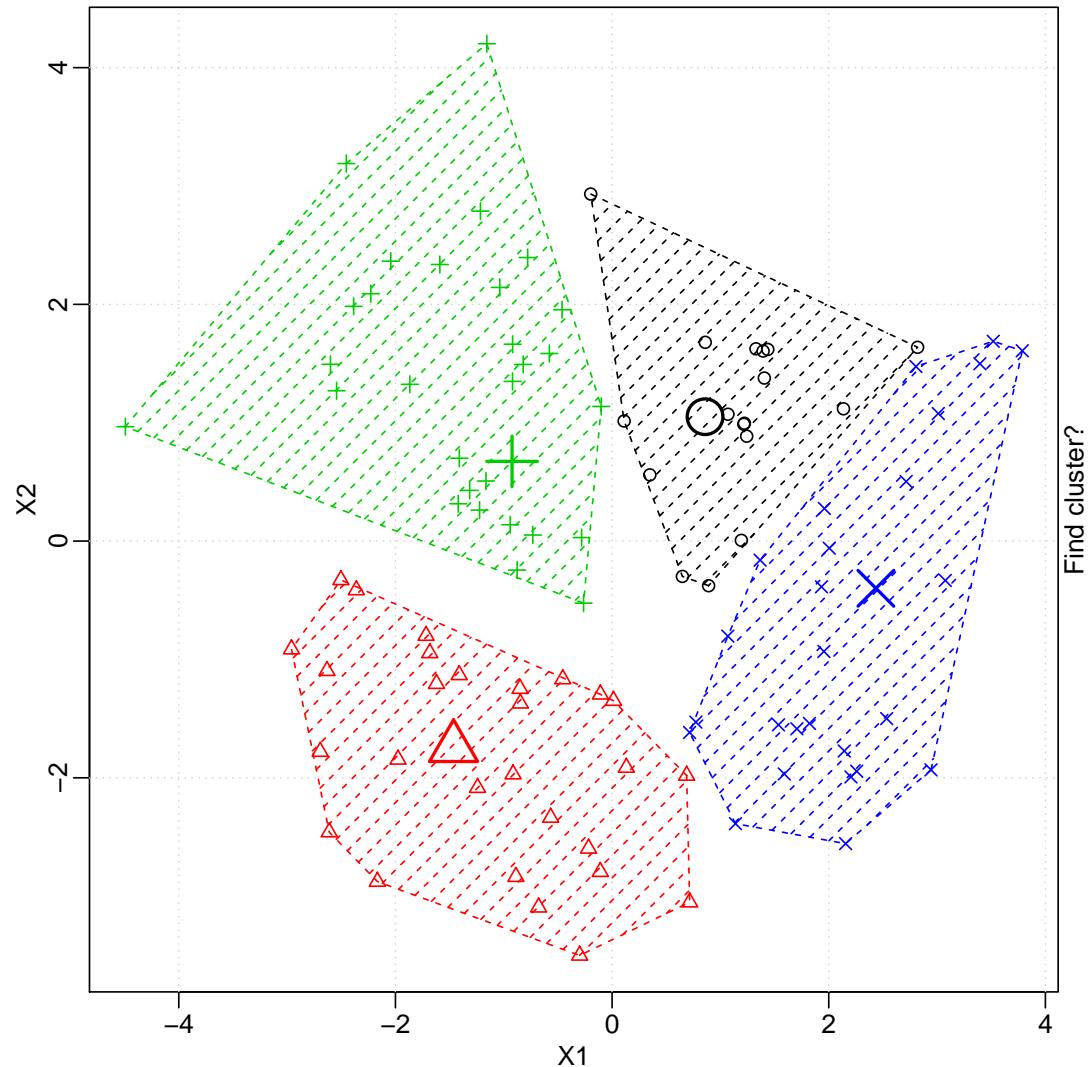
```
par(mar=c(3, 3, 1, 1.5), mgp=c(1.5, 0.5, 0), bg="white")
kmeans.ani(x, centers=4, pch=1:4, col=1:4)
```

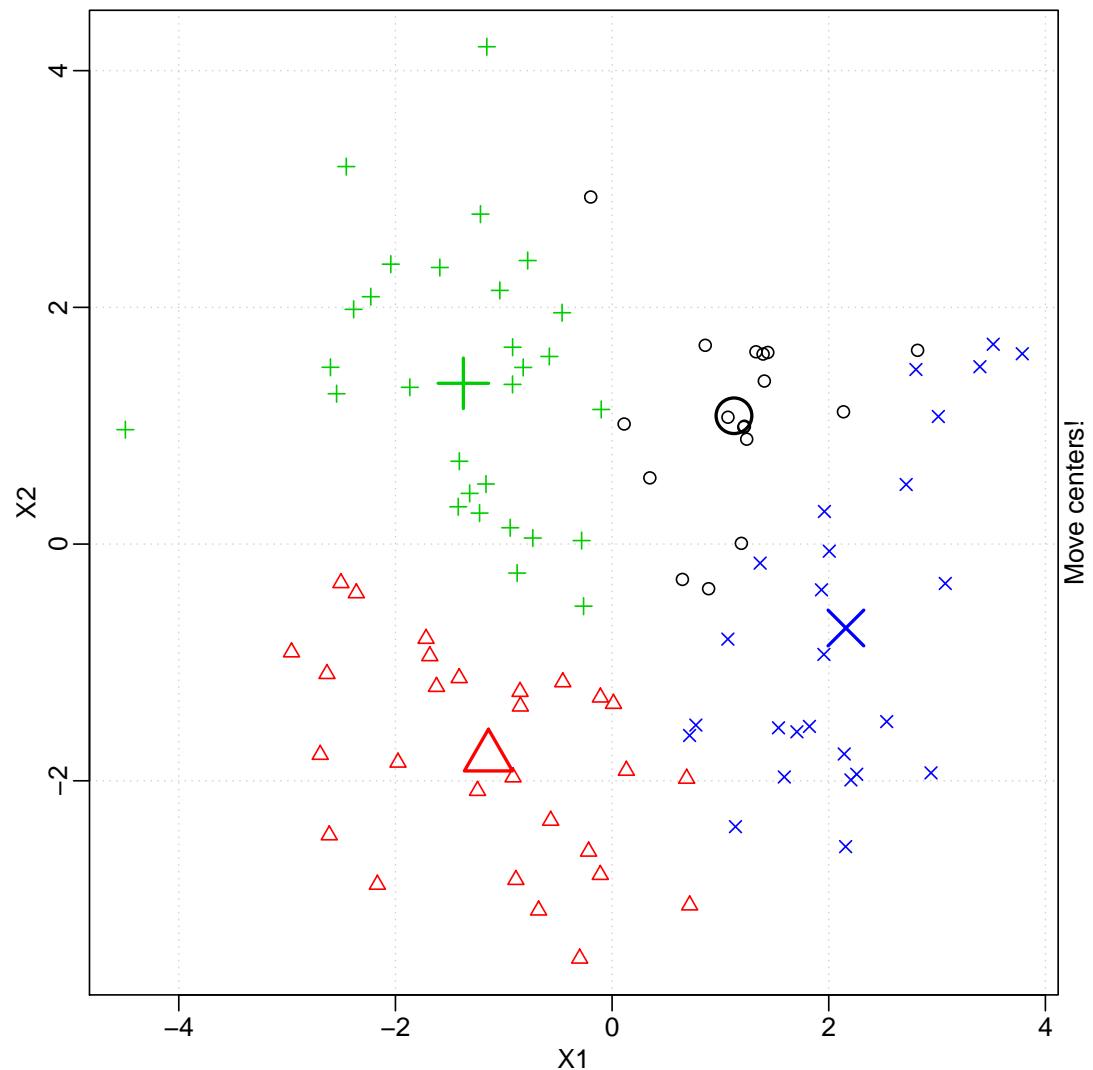
The first plot, on the next page, shows a random allocation of points to one of the four clusters, together with 4 random means. The points are then mapped to their closest means, to define the four clusters we see in the second plot. The means are then recalculated for each of the clusters, as seen in the third plot. The following plots then iterate between showing the means nearest each of the points, then re-calculating the means. Eventually, the means do not change location, and the algorithm converges.

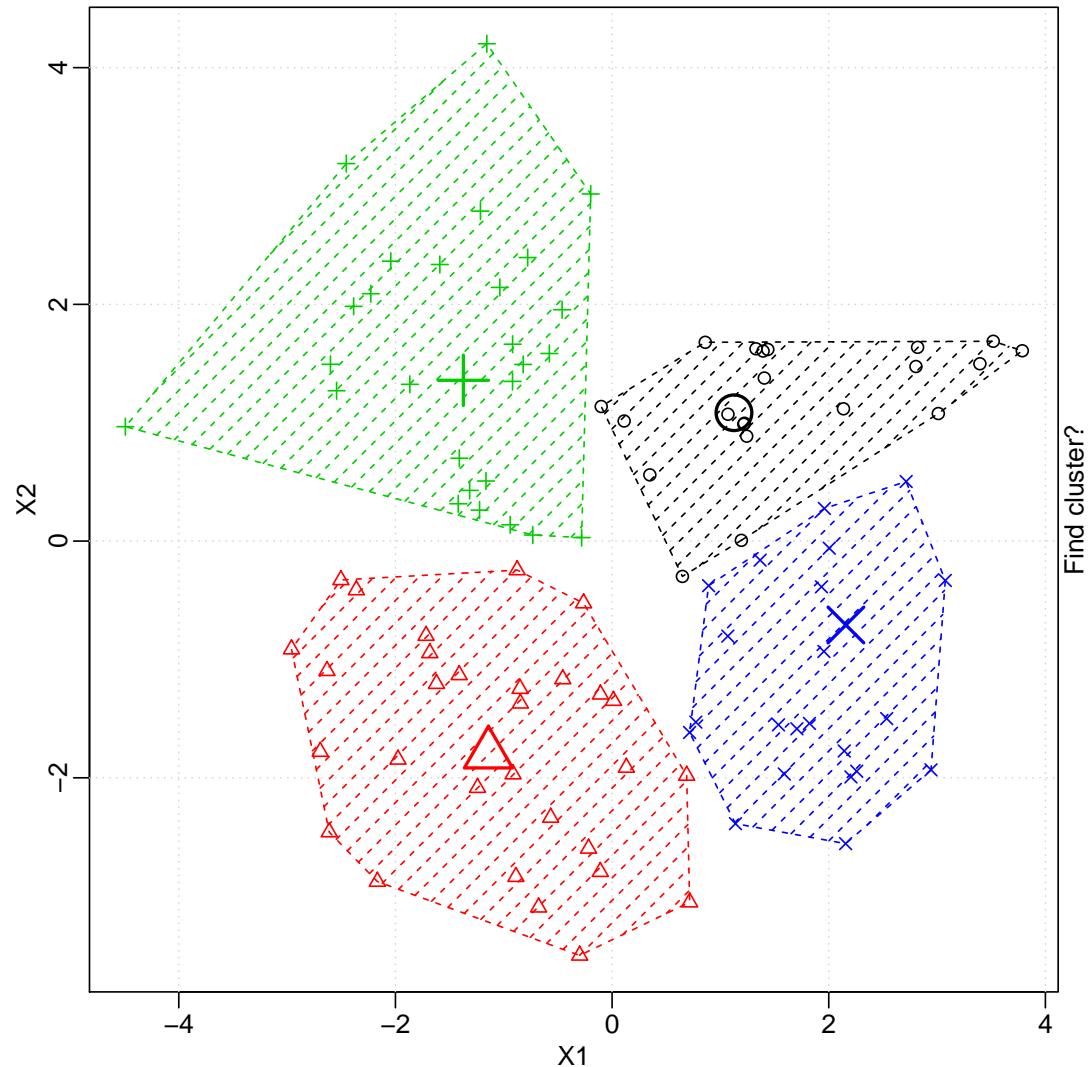


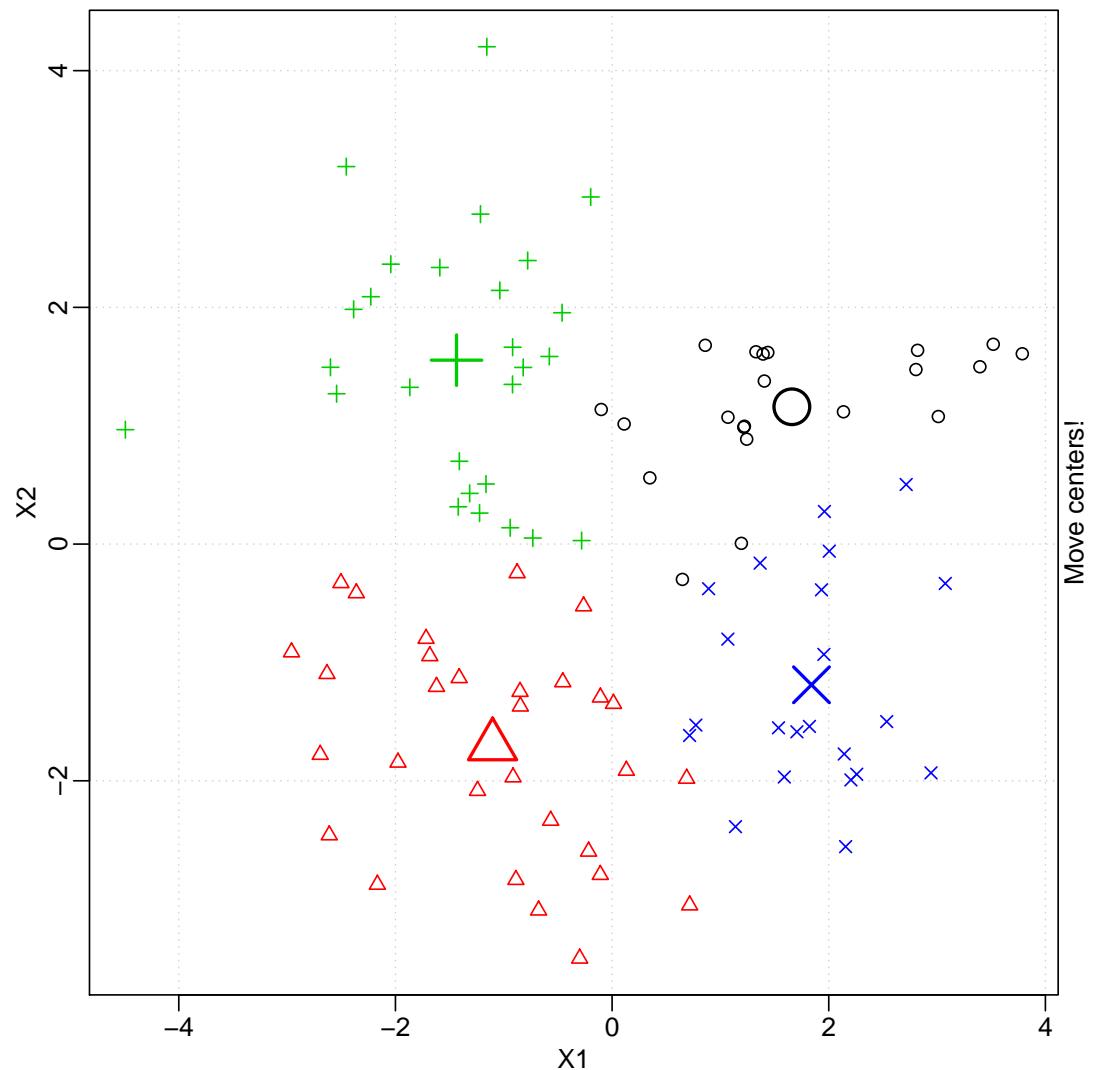


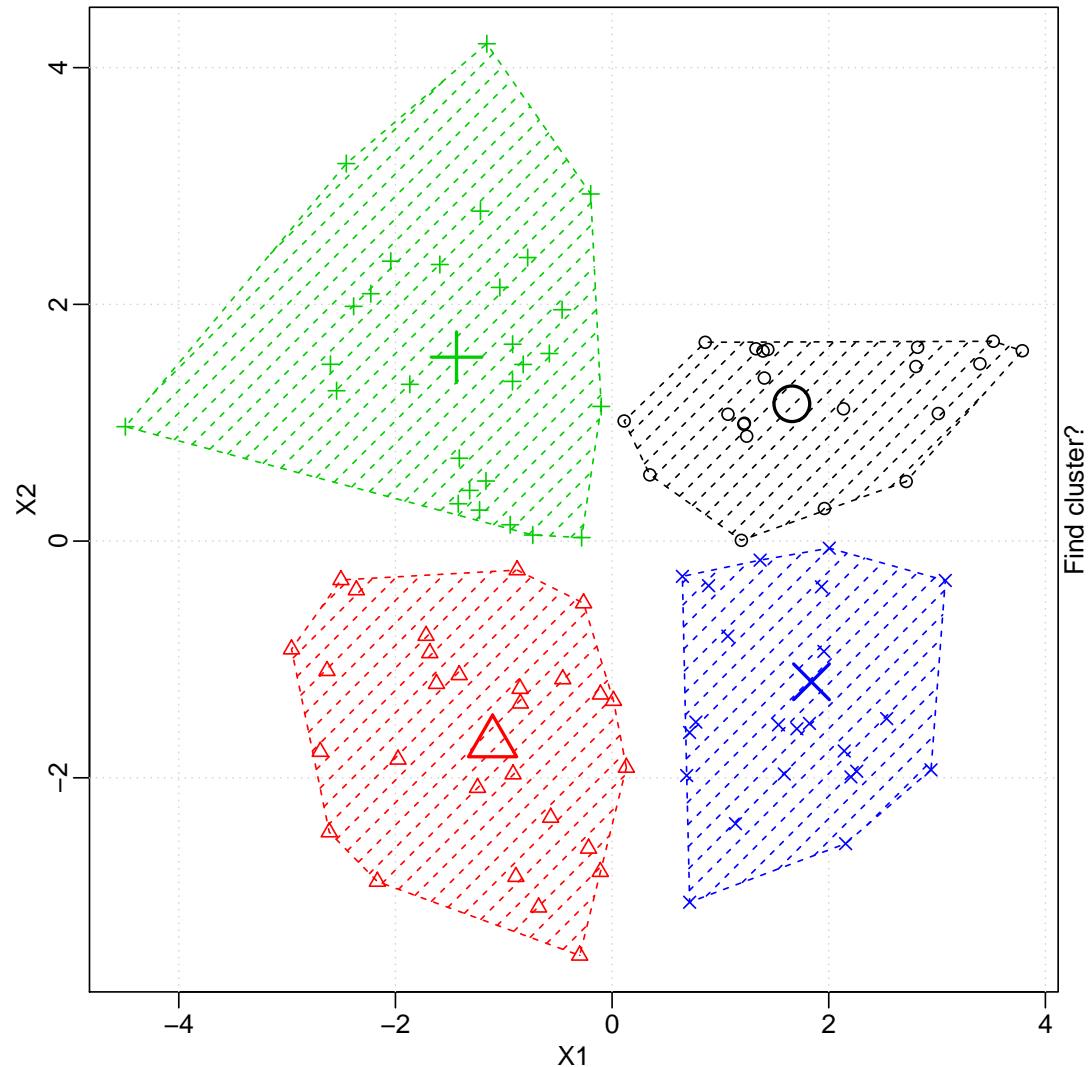


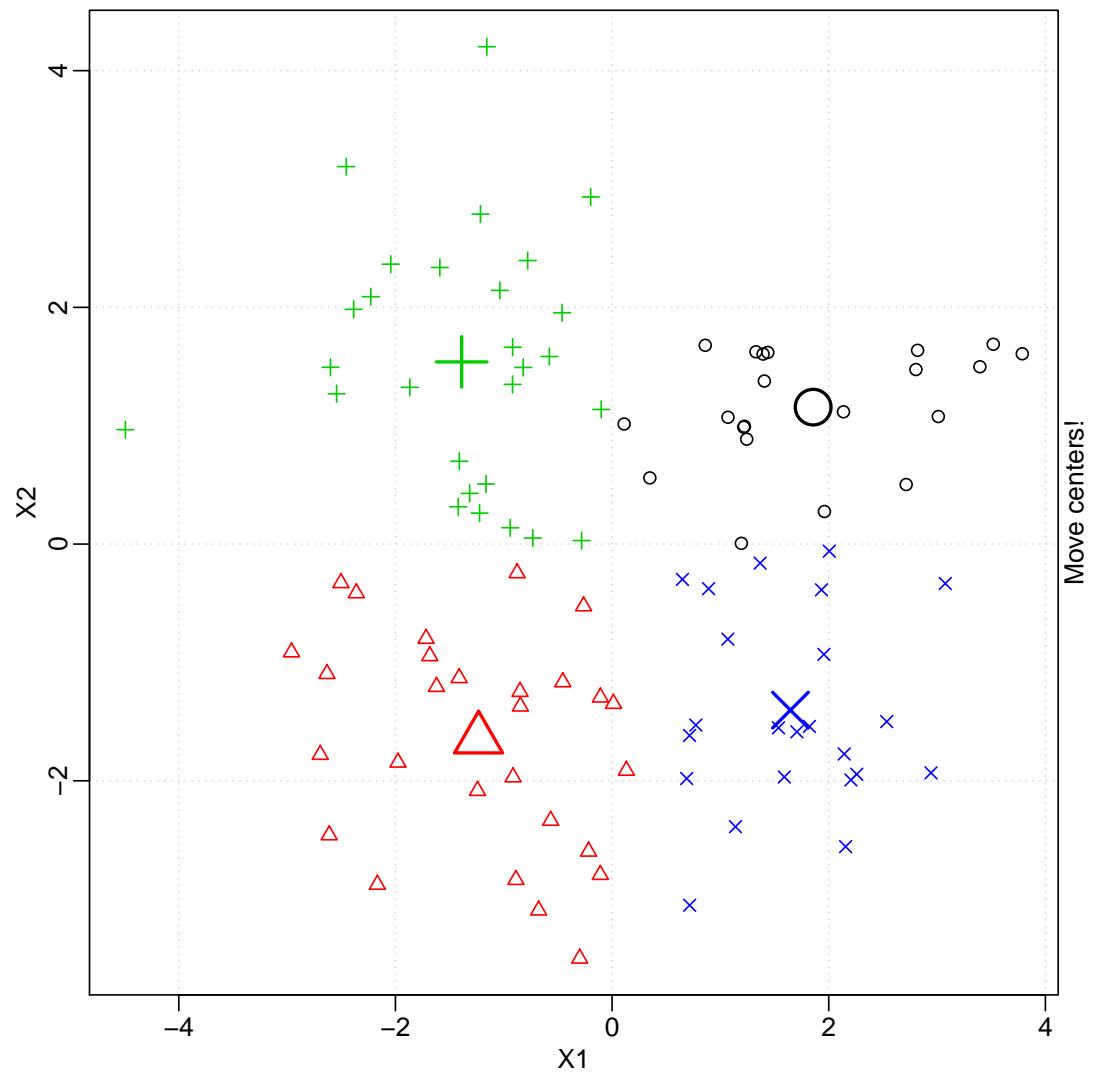


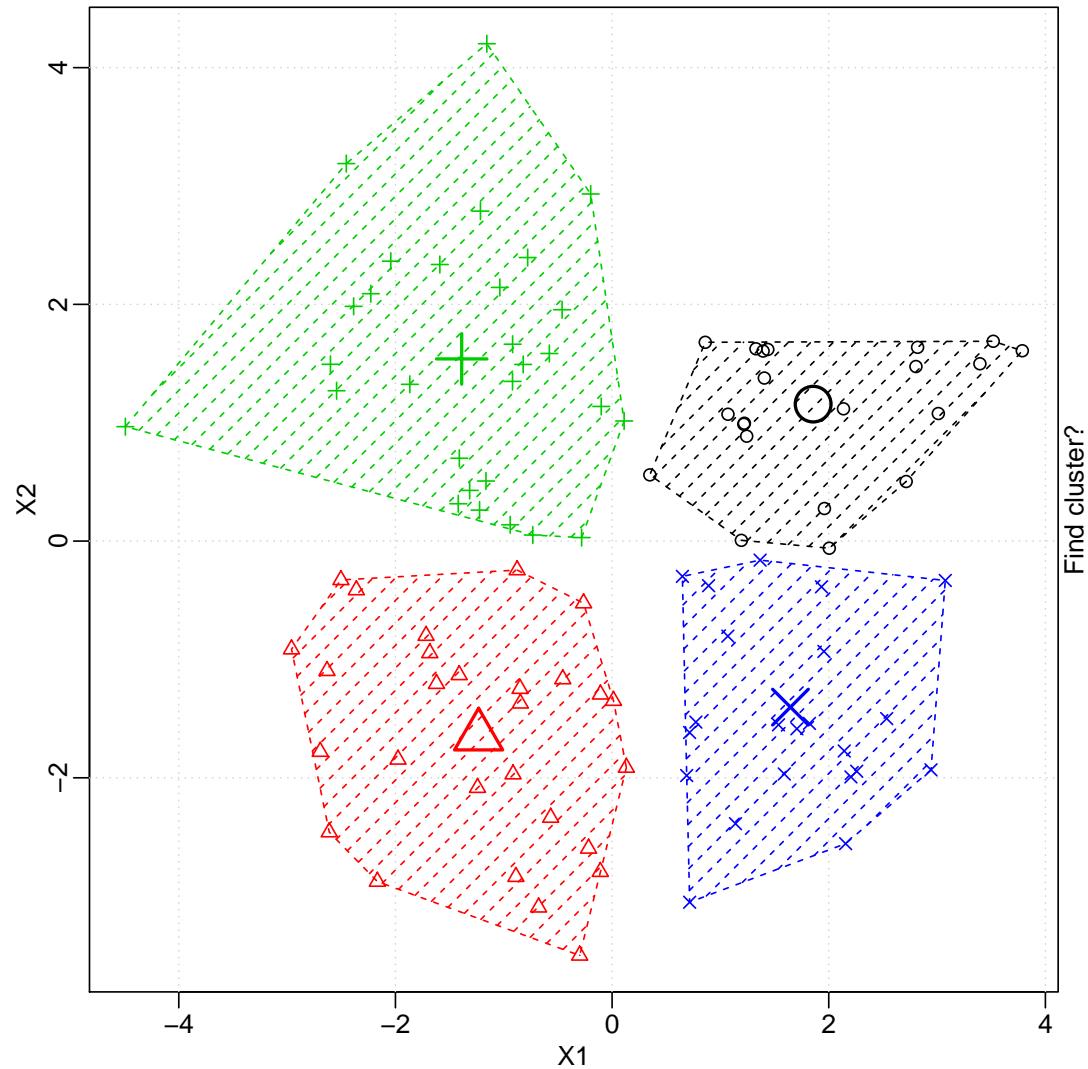


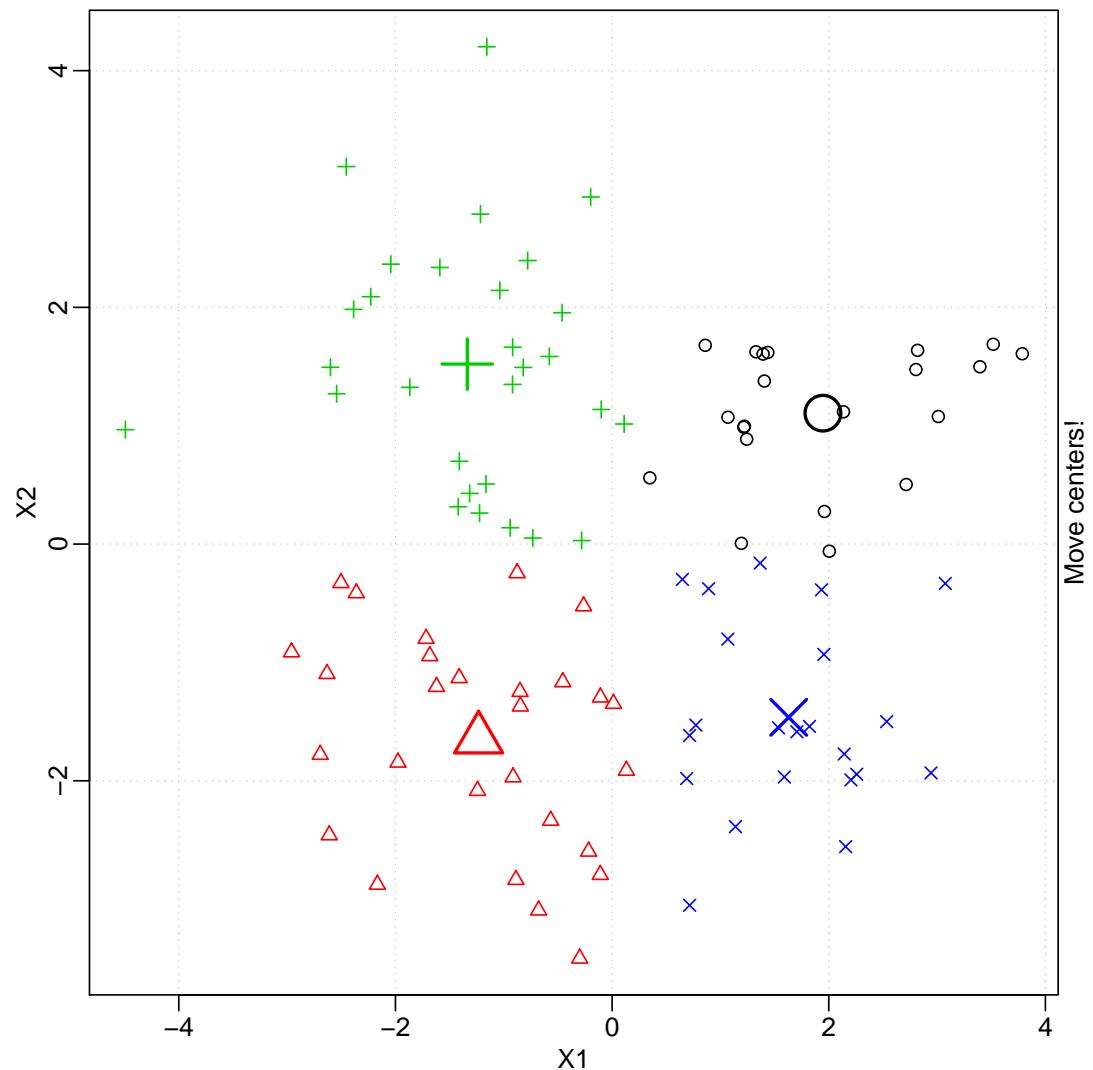


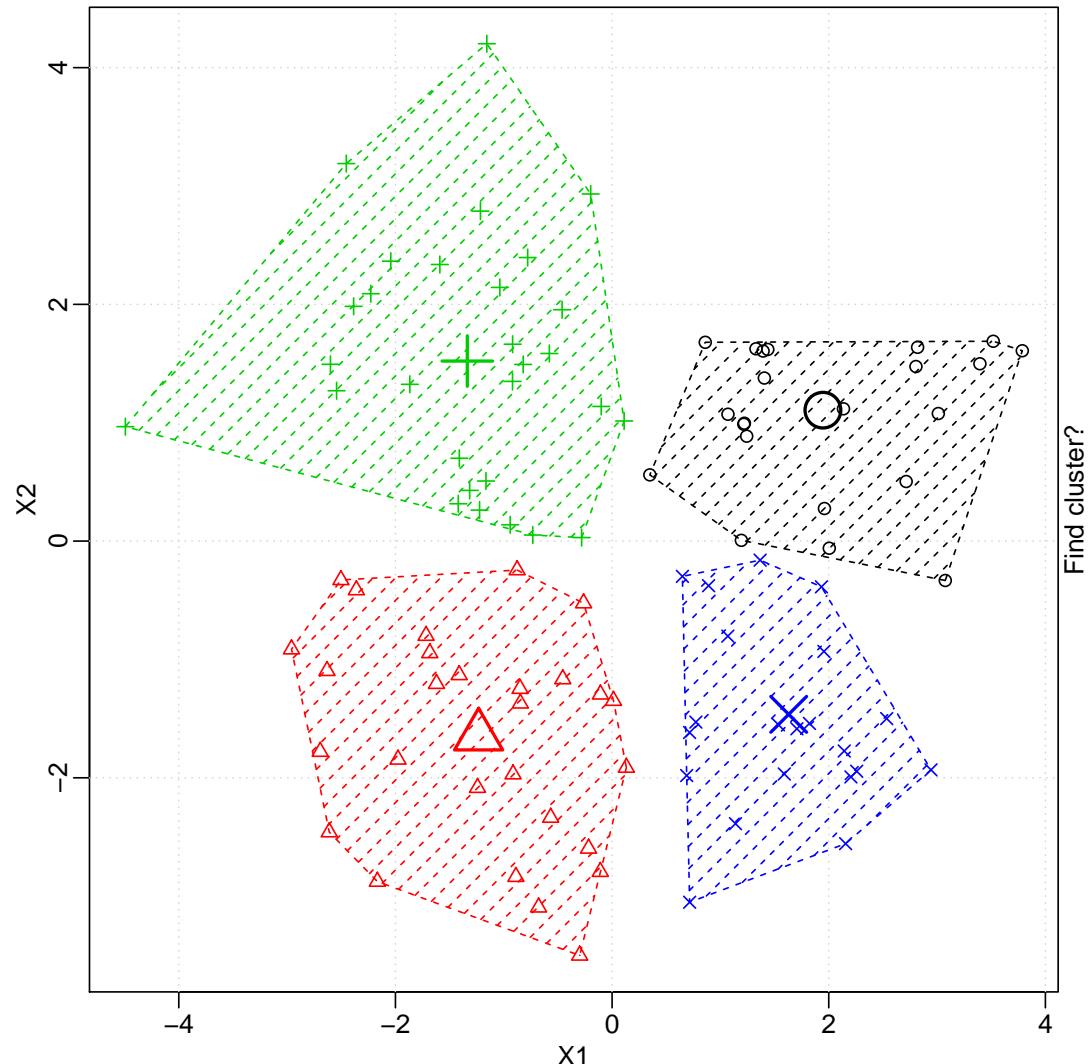


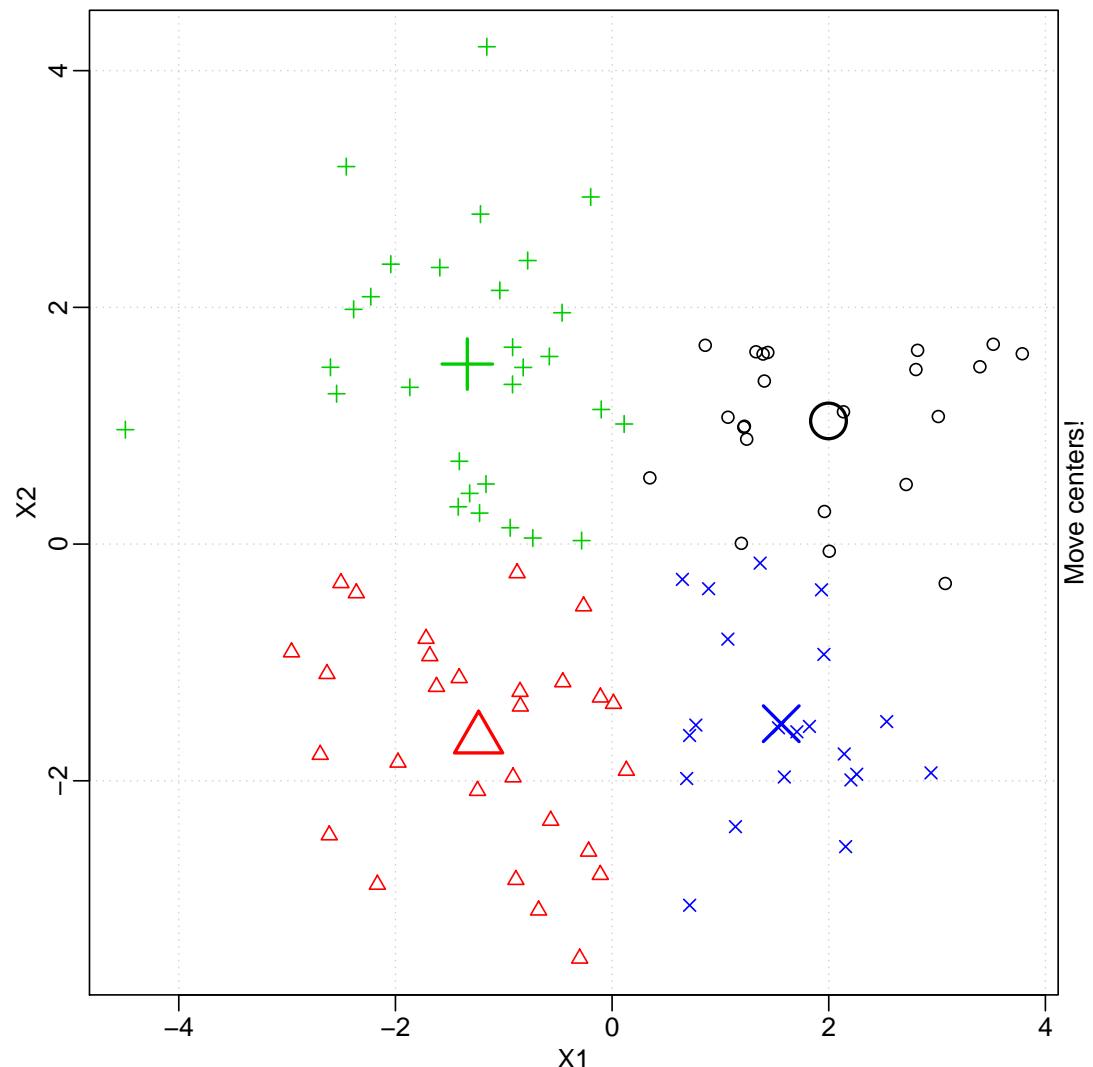


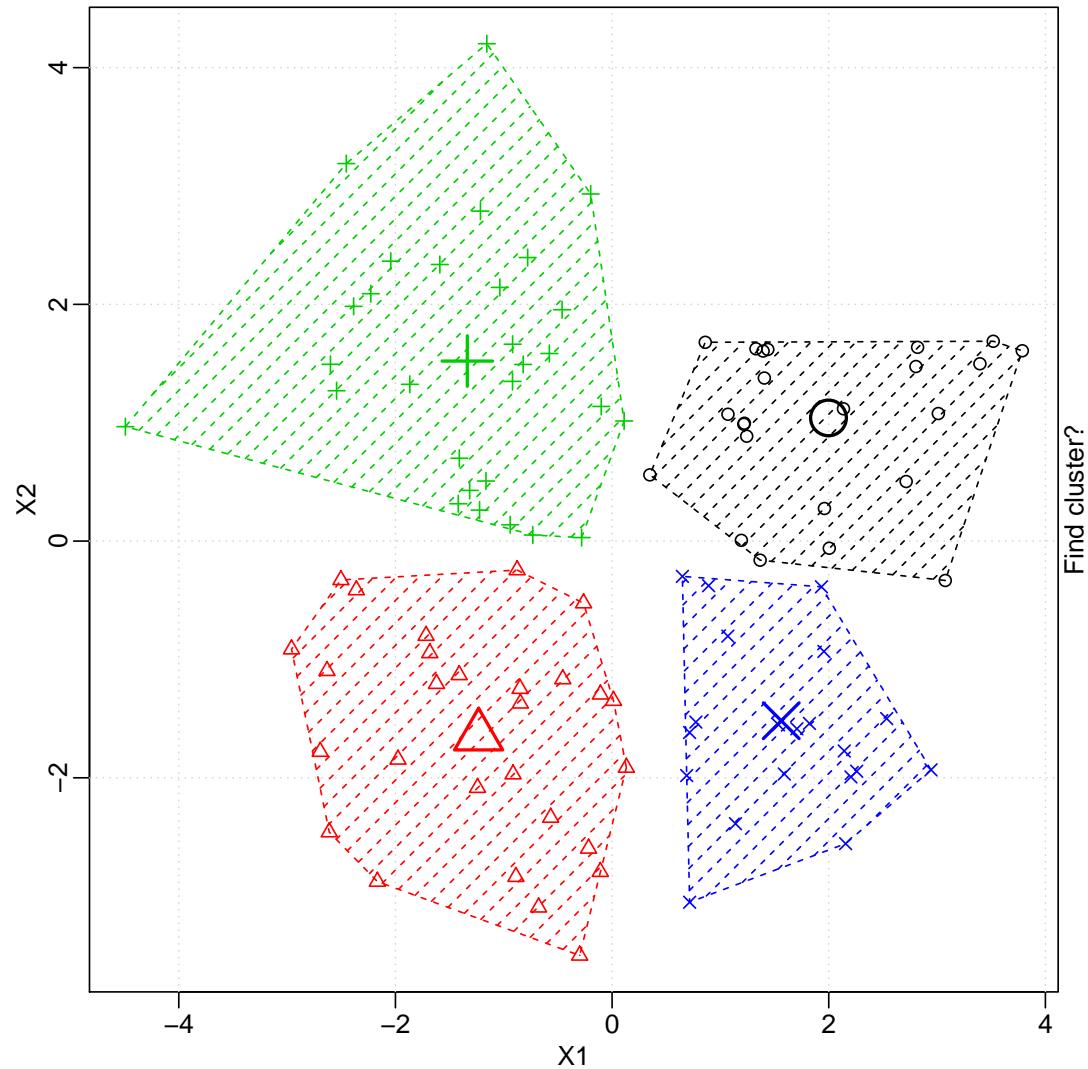


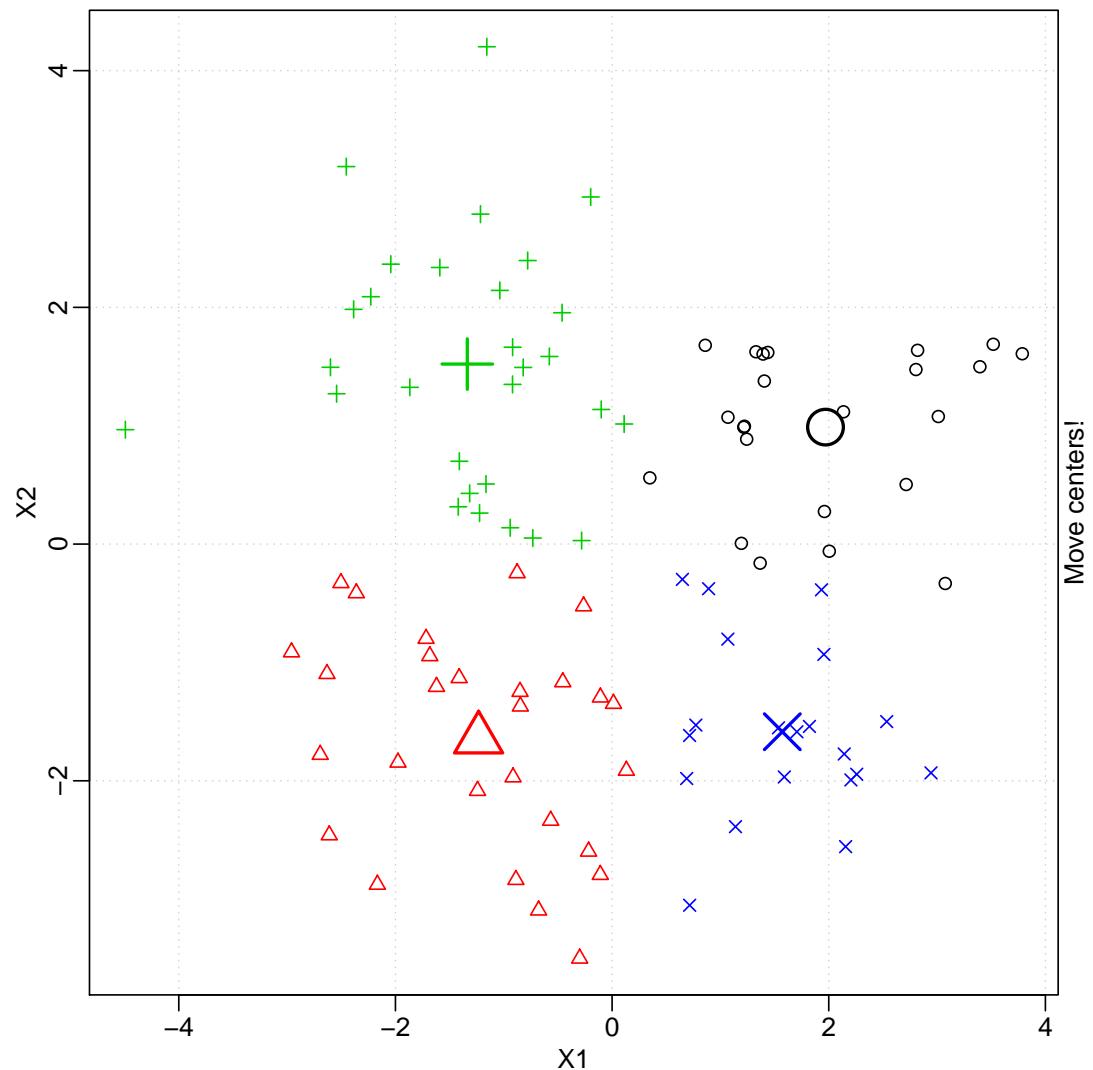


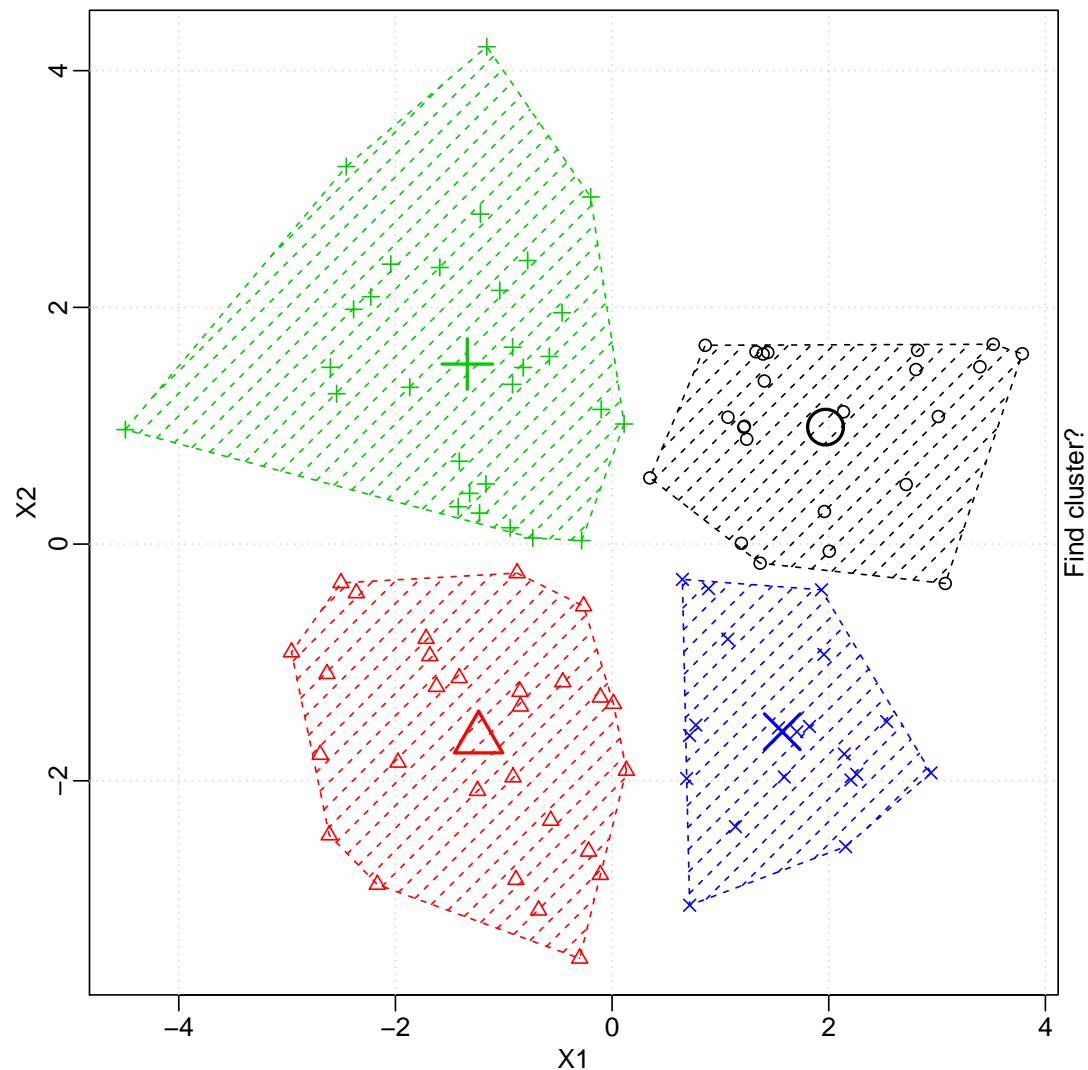




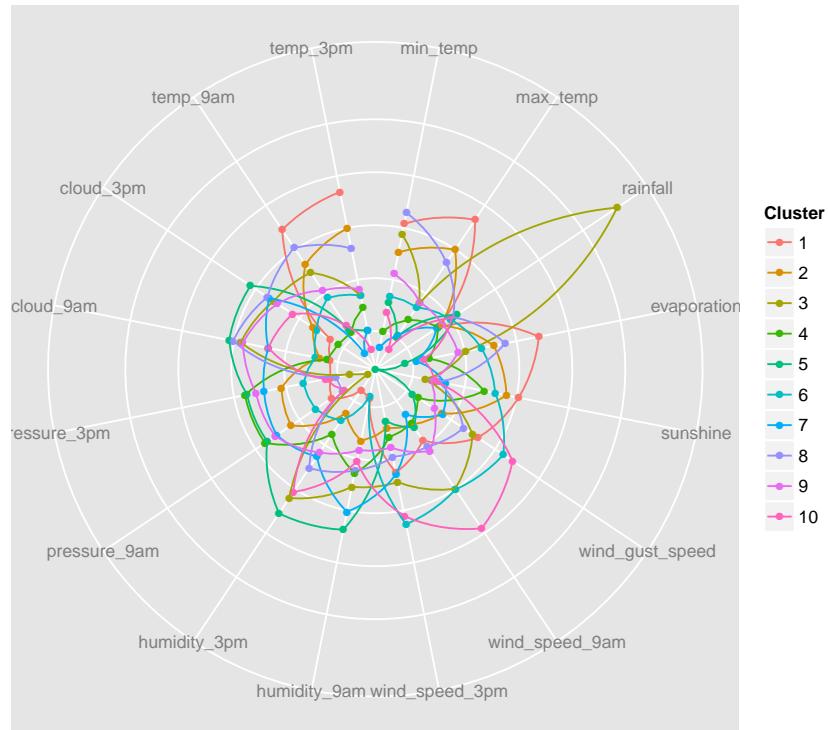






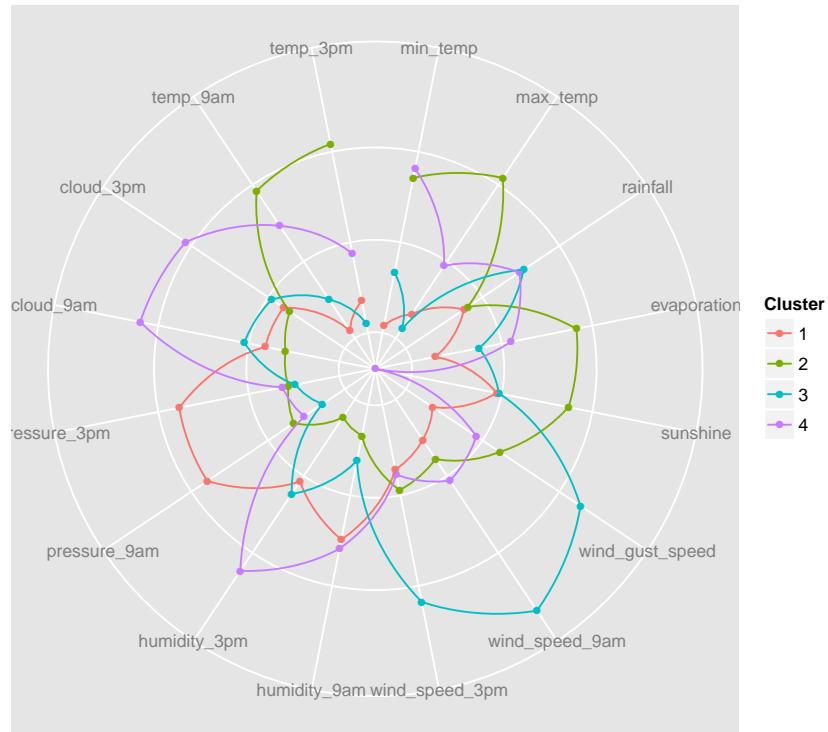


11 Visualise the Cluster: Radial Plot Using GGPlot2



```
dscm <- melt(model$centers)
names(dscm) <- c("Cluster", "Variable", "Value")
dscm$Cluster <- factor(dscm$Cluster)
dscm$Order <- as.vector(sapply(1:length(numi), rep, 10))
p <- ggplot(subset(dscm, Cluster %in% 1:10),
            aes(x=reorder(Variable, Order),
                y=Value, group=Cluster, colour=Cluster))
p <- p + coord_polar()
p <- p + geom_point()
p <- p + geom_path()
p <- p + labs(x=NULL, y=NULL)
p <- p + theme(axis.ticks.y=element_blank(), axis.text.y = element_blank())
p
```

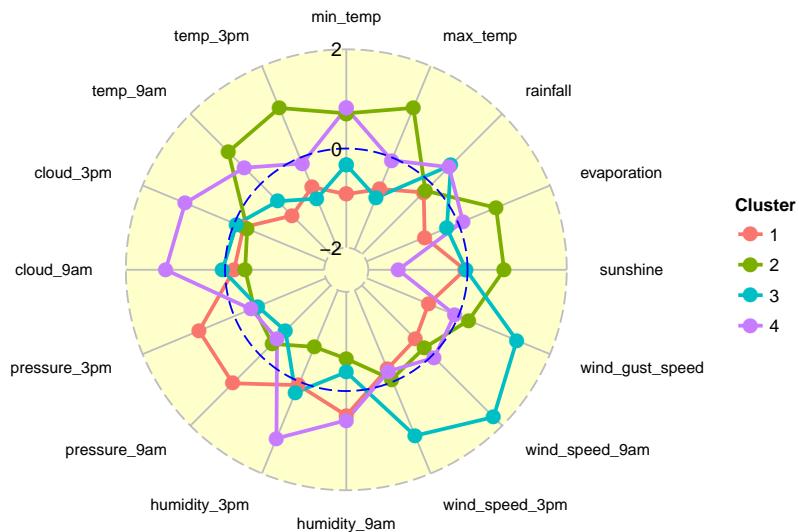
12 Visualize the Cluster: Radial Plot with K=4



```

nclust <- 4
model <- m.kms <- kmeans(scale(ds[numi]), nclust)
dscm <- melt(model$centers)
names(dscm) <- c("Cluster", "Variable", "Value")
dscm$Cluster <- factor(dscm$Cluster)
dscm$Order <- as.vector(sapply(1:length(numi), rep, nclust))
p <- ggplot(dscm,
             aes(x=reorder(Variable, Order),
                  y=Value, group=Cluster, colour=Cluster))
p <- p + coord_polar()
p <- p + geom_point()
p <- p + geom_path()
p <- p + labs(x=NULL, y=NULL)
p <- p + theme(axis.ticks.y=element_blank(), axis.text.y = element_blank())
p
  
```

13 Visualise the Cluster: Cluster Profiles with Radial Plot

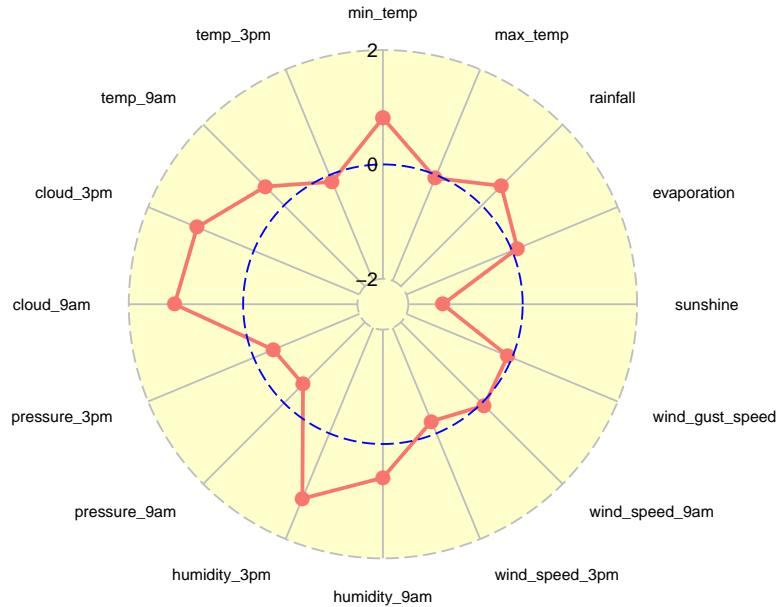


The radial plot here is carefully engineered to most effectively present the cluster profiles. The R code to generate the plot is defined as `CreateRadialPlot()` and was originally available from Paul Williamson's [web site](#) (Department of Geography, University of Liverpool):

```
source("http://onepager.togaware.com/CreateRadialPlot.R")
dsc <- data.frame(group=factor(1:4), model$centers)
CreateRadialPlot(dsc, grid.min=-2, grid.max=2, plot.extent.x=1.5)
```

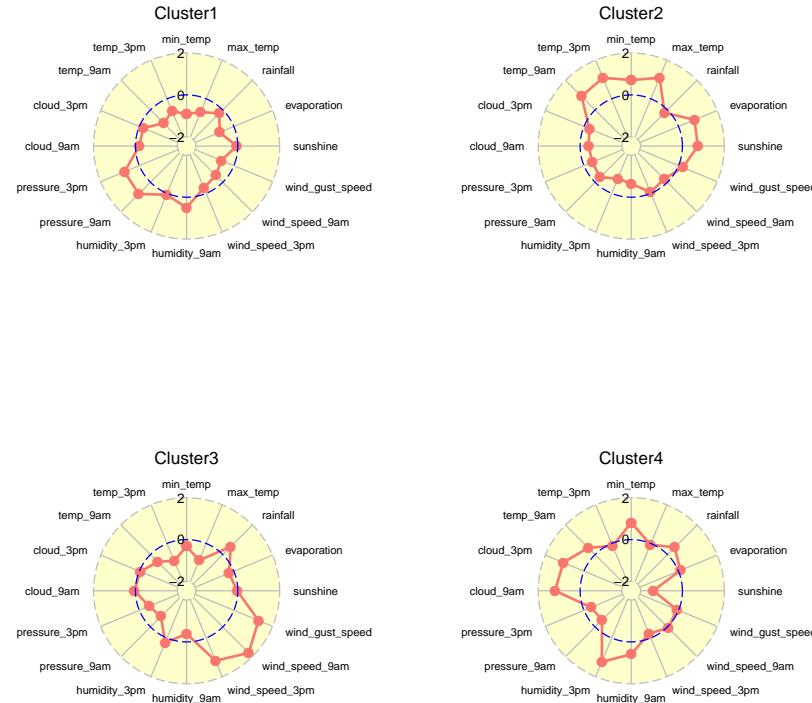
We can quickly read the profiles and gain insights into the 4 clusters. Having re-scaled all of the data we know that the “0” circle is the mean for each variable, and the range goes up to 2 standard deviations from the mean, in either direction. We observe that cluster 1 has a center with higher pressures, whilst the cluster 2 center has higher humidity and cloud cover and low sunshine, cluster 3 has high wind speeds and cluster 4 has higher temperatures, evaporation and sunshine.

14 Visualise the Cluster: Single Cluster Radial Plot



```
CreateRadialPlot(subset(dsc, group==4), grid.min=-2, grid.max=2, plot.extent.x=1.5)
```

15 Visualise the Cluster: Grid of Radial Plots



```

p1 <- CreateRadialPlot(subset(dsc, group==1),
                       grid.min=-2, grid.max=2, plot.extent.x=2)
p2 <- CreateRadialPlot(subset(dsc, group==2),
                       grid.min=-2, grid.max=2, plot.extent.x=2)
p3 <- CreateRadialPlot(subset(dsc, group==3),
                       grid.min=-2, grid.max=2, plot.extent.x=2)
p4 <- CreateRadialPlot(subset(dsc, group==4),
                       grid.min=-2, grid.max=2, plot.extent.x=2)
library(gridExtra)
grid.arrange(p1+ggtitle("Cluster1"), p2+ggtitle("Cluster2"),
            p3+ggtitle("Cluster3"), p4+ggtitle("Cluster4"))

```

16 K-Means: Base Case Cluster

```
model <- m.kms <- kmeans(scale(ds[numi]), 1)
model$size
## [1] 366
model$centers
##   min_temp max_temp rainfall evaporation sunshine wind_gust_speed
## 1 9.98e-17 1.274e-16 -2.545e-16 -1.629e-16 -5.836e-16      1.99e-16
##   wind_speed_9am wind_speed_3pm humidity_9am humidity_3pm pressure_9am
## 1      -1.323e-16      -2.87e-16     -4.162e-16     -1.11e-16    -4.321e-15
.....
model$totss
## [1] 5840
model$withinss
## [1] 5840
model$tot.withinss
## [1] 5840
model$betweenss
## [1] -1.819e-11
model$iter
## [1] 1
model$ifault
## NULL
```

Notice that this base case provides the centers of the original data, and the starting measure of the within sum of squares.

17 K-Means: Multiple Starts

18 K-Means: Cluster Stability

Rebuilding multiple clusterings using different random starting points will lead to different clusters being identified. We might expect that clusters that are regularly being identified with different starting points, might be more robust as actual clusters representing some cohesion among the observations belonging to that cluster.

The function `clusterboot()` from `fpc` (Hennig, 2014) provides a convenient tool to identify robust clusters.

```
library(fpc)
model <- m.kmcb <- clusterboot(scale(ds[numi]),
                                    clustermethod=kmeansCBI,
                                    runs=10,
                                    krange=10,
                                    seed=42)

## boot 1
## boot 2
## boot 3
## boot 4
....
model

## * Cluster stability assessment *
## Cluster method: kmeans
## Full clustering results are given as parameter result
## of the clusterboot object, which also provides further statistics
....
str(model)

## List of 31
## $ result      :List of 6
##   ..$ result      :List of 11
##     ...$ cluster    : int [1:366] 1 10 5 7 3 1 1 1 1 ...
....
```

19 Evaluation of Clustering Quality

Numerous measures are available for evaluating a clustering. Many are stored within the data structure returned by `kmeans()`

The total sum of squares.

```
model <- kmeans(scale(ds[numi]), 10)
model$totss
## [1] 5840
model$withinss
## [1] 172.1 219.2 237.6 217.2 336.6 228.4 254.2 291.9 310.5 126.0
model$tot.withinss
## [1] 2394
model$betweenss
## [1] 3446
```

The basic concept is the sum of squares. This is typically a sum of the square of the distances between observations.

20 Evaluation: Within Sum of Squares

The within sum of squares is a measure of how close the observations are within the clusters. For a single cluster this is calculated as the average squared distance of each observation within the cluster from the cluster mean. Then the total within sum of squares is the sum of the within sum of squares over all clusters.

The total within sum of squares generally decreases as the number of clusters increases. As we increase the number of clusters they individually tend to become smaller and the observations closer together within the clusters. As k increases, the changes in the total within sum of squares would be expected to reduce, and so it flattens out. A good value of k might be where the reduction in the total weighted sum of squares begins to flatten.

```
model$withinss  
## [1] 172.1 219.2 237.6 217.2 336.6 228.4 254.2 291.9 310.5 126.0  
model$tot.withinss  
## [1] 2394
```

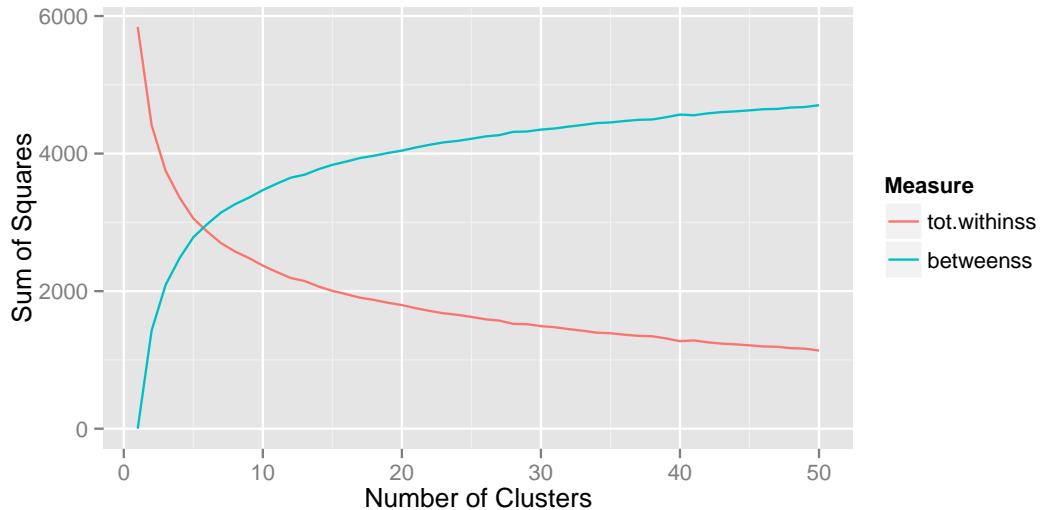
The total within sum of squares is a common measure that we aim to minimise in building a clustering.

21 Evaluation: Between Sum of Squares

The between sum of squares is a measure of how far the clusters are from each other.

```
model$betweenss  
## [1] 3446
```

A good clustering will have a small within sum of squares and a large between sum of squares. Here we see the relationship between these two measures:



22 K-Means: Selecting k Using Scree Plot

```

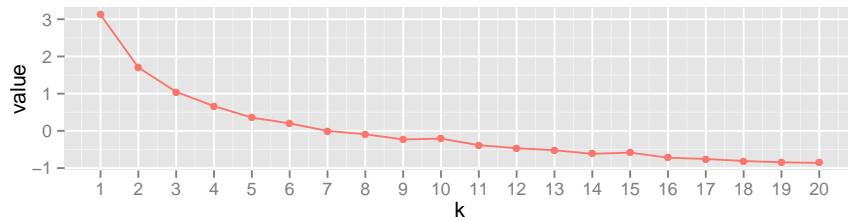
crit <- vector()
nk <- 1:20
for (k in nk)
{
  m <- kmeans(scale(ds[numi]), k)
  crit <- c(crit, sum(m$withinss))
}
crit

## [1] 5840 4414 3753 3368 3057 2900 2697 2606 2465 2487 2310 2228 2173 2075
## [15] 2108 1970 1937 1882 1846 1830

dsc <- data.frame(k=nk, crit=scale(crit))
dscm <- melt(dsc, id.vars="k", variable.name="Measure")

p <- ggplot(dscm, aes(x=k, y=value, colour=Measure))
p <- p + geom_point(aes(shape=Measure))
p <- p + geom_line(aes(linetype=Measure))
p <- p + scale_x_continuous(breaks=nk, labels=nk)
p <- p + theme(legend.position="none")
p

```



23 K-Means: Selecting k Using Calinski-Harabasz

The Calinski-Harabasz criteria, also known as the variance ratio criteria, is the ratio of the *between sum of squares* (divided by $k - 1$) to the *within sum of squares* (divided by $n - k$)—the sum of squares is a measure of the variance. The relative values can be used to compare clusterings of a single dataset, with higher values being better clusterings. The criteria is said to work best for spherical clusters with compact centres (as with normally distributed data) using k-means with Euclidean distance.

```
library(fpc)
nk <- 1:20
model <- km.c <- kmeansruns(scale(ds[numi]), krangle=nk, criterion="ch")
class(model)

## [1] "kmeans"

model

## K-means clustering with 2 clusters of sizes 192, 174
##
## Cluster means:
##   min_temp max_temp rainfall evaporation sunshine wind_gust_speed
## ....
model$crit

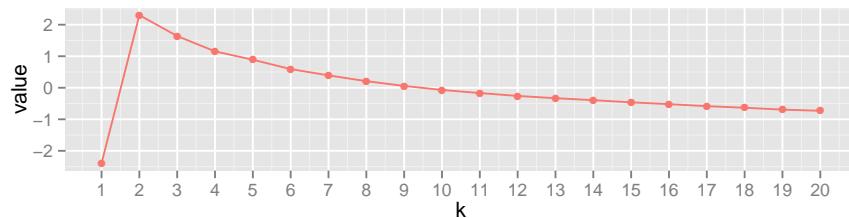
## [1] 0.00 117.55 100.97 88.81 82.16 74.75 69.75 65.18 61.38 58.18
## [11] 55.71 53.44 51.63 50.07 48.34 46.90 45.32 44.07 42.57 41.65

model$bestk

## [1] 2

dsc <- data.frame(k=nk, crit=scale(km.c$crit))
dscm <- melt(dsc, id.vars="k", variable.name="Measure")

p <- ggplot(dscm, aes(x=k, y=value, colour=Measure))
p <- p + geom_point(aes(shape=Measure))
p <- p + geom_line(aes(linetype=Measure))
p <- p + scale_x_continuous(breaks=nk, labels=nk)
p <- p + theme(legend.position="none")
p
```



24 K-Means: Selecting k Using Average Silhouette Width

The average silhouette width criteria is more computationally expensive than the Calinski-Harabasz criteria which is an issue for larger datasets. A dataset of 50,000 observations and 15 scaled variables, testing from 10 to 40 clusters, 10 runs, took 30 minutes for the Calinski-Harabasz criteria compared to minutes using the average silhouette width criteria.

check timing

```
library(fpc)
nk <- 1:20
model <- km.a <- kmeansruns(scale(ds[numi]), krangle=nk, criterion="asw")
class(model)

## [1] "kmeans"

model

## K-means clustering with 2 clusters of sizes 174, 192
##
## Cluster means:
##   min_temp max_temp rainfall evaporation sunshine wind_gust_speed
## ...
## 
model$crit

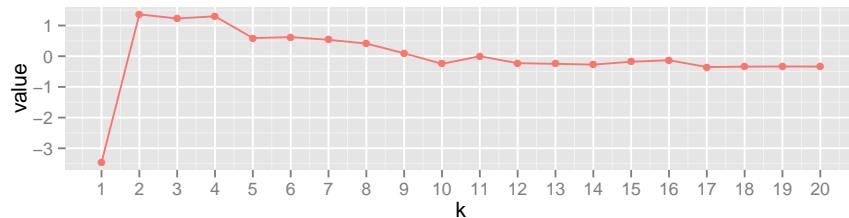
## [1] 0.0000 0.2255 0.2192 0.2225 0.1894 0.1908 0.1867 0.1811 0.1662 0.1502
## [11] 0.1617 0.1512 0.1502 0.1490 0.1533 0.1557 0.1453 0.1459 0.1462 0.1460

model$bestk

## [1] 2

dsc <- data.frame(k=nk, crit=scale(km.a$crit))
dscm <- melt(dsc, id.vars="k", variable.name="Measure")

p <- ggplot(dscm, aes(x=k, y=value, colour=Measure))
p <- p + geom_point(aes(shape=Measure))
p <- p + geom_line(aes(linetype=Measure))
p <- p + scale_x_continuous(breaks=nk, labels=nk)
p <- p + theme(legend.position="none")
p
```



25 K-Means: Using clusterCrit Calinski_Harabasz

The `clusterCrit` (Desgraupes, 2013) package provides a comprehensive collection of clustering criteria. Here we illustrate its usage with the Calinski_Harabasz criteria. Do note that we obtain a different model here to that above, hence different calculations of the criteria.

```
library(clusterCrit)
crit <- vector()
for (k in 1:20)
{
  m <- kmeans(scale(ds[numi]), k)
  crit <- c(crit, as.numeric(intCriteria(as.matrix(ds[numi]), m$cluster,
                                             "Calinski_Harabasz")))
}
crit[is.nan(crit)] <- 0 #
crit

## [1] 0.00 81.20 87.88 86.71 75.78 64.34 64.48 49.87 51.83 48.81 42.78
## [12] 45.04 43.03 44.53 40.12 38.76 39.26 38.67 35.19 32.33

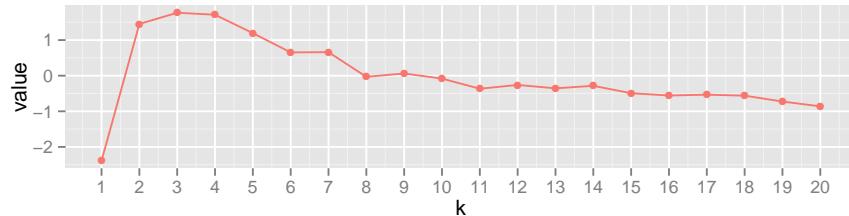
bestCriterion(crit, "Calinski_Harabasz")

## [1] 3
```

In this case $k = 3$ is the optimum choice.

```
dsc <- data.frame(k=nk, crit=scale(crit))
dscm <- melt(dsc, id.vars="k", variable.name="Measure")

p <- ggplot(dscm, aes(x=k, y=value, colour=Measure))
p <- p + geom_point(aes(shape=Measure))
p <- p + geom_line(aes(linetype=Measure))
p <- p + scale_x_continuous(breaks=nk, labels=nk)
p <- p + theme(legend.position="none")
p
```



26 K-Means: Compare All Criteria

We can generate all criteria and then plot them. There are over 40 criteria, and the are noted on the help page for `intCriteria()`. We generate all the criteria here and then plot the first 6 below, with the remainder in the following section.

```
m <- kmeans(scale(ds[numi]), 5)
ic <- intCriteria(as.matrix(ds[numi]), m$cluster, "all")
names(ic)

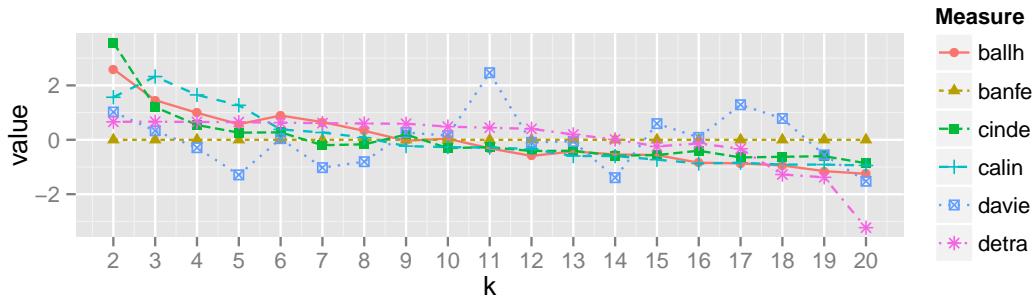
## [1] "ball_hall"          "banfeld_raftery"    "c_index"
## [4] "calinski_harabasz" "davies_bouldin"   "det_ratio"
## [7] "dunn"               "gamma"              "g_plus"
## [10] "gdi11"              "gdi12"              "gdi13"
.....
crit <- data.frame()
for (k in 2:20)
{
  m <- kmeans(scale(ds[numi]), k)
  crit <- rbind(crit, as.numeric(intCriteria(as.matrix(ds[numi]), m$cluster,
                                              "all")))
}
names(crit) <- substr(sub("_", "", names(ic)), 1, 5) # Shorten for plots.
crit <- data.frame(sapply(crit, function(x) {x[is.nan(x)] <- 0; x}))

dsc <- cbind(k=2:20, data.frame(sapply(crit, scale)))

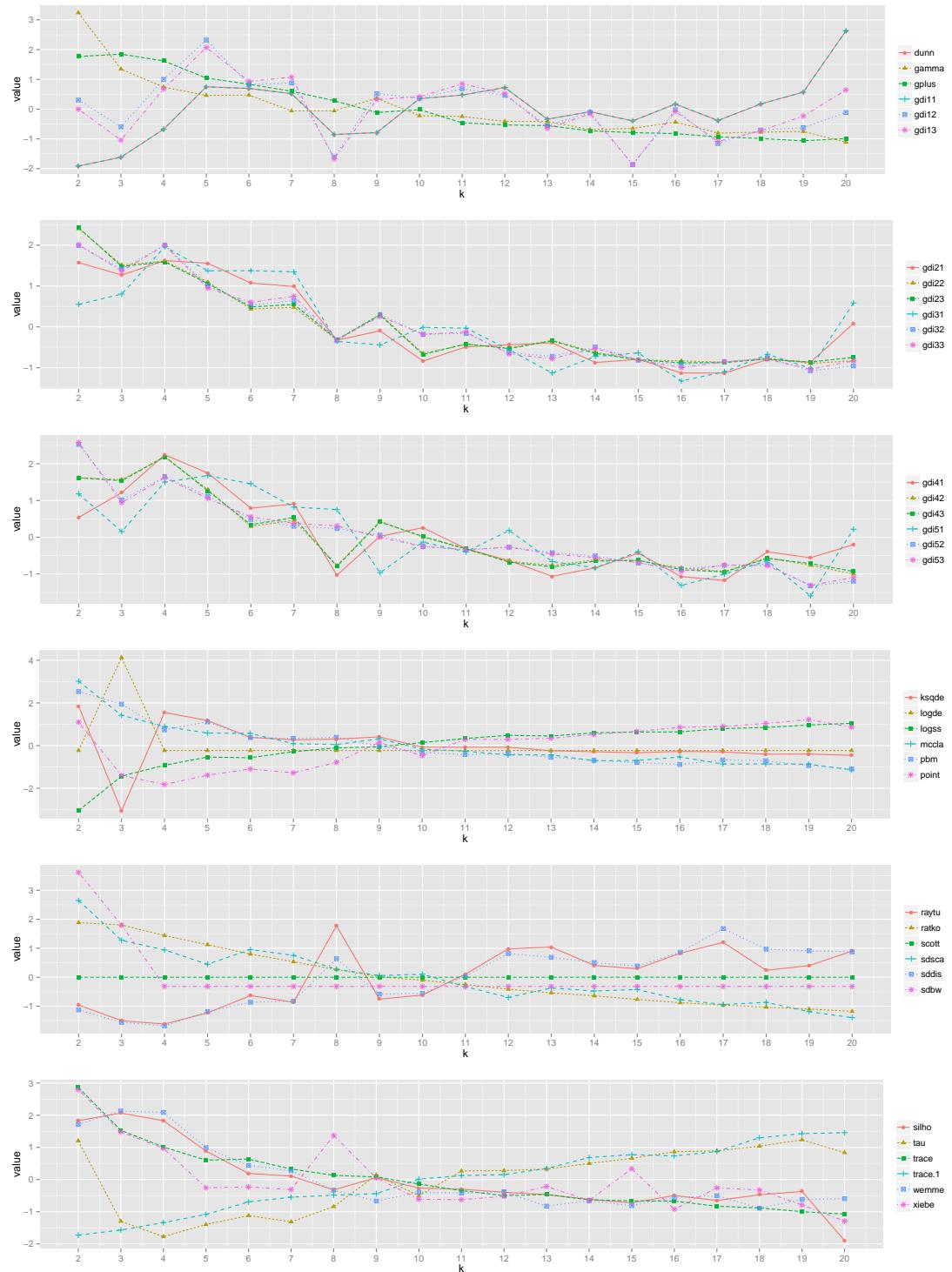
dscm <- melt(dsc, id.vars="k", variable.name="Measure")
dscm$value[is.nan(dscm$value)] <- 0

ms <- as.character(unique(dscm$Measure))

p <- ggplot(subset(dscm, Measure %in% ms[1:6]), aes(x=k, y=value, colour=Measure))
p <- p + geom_point(aes(shape=Measure)) + geom_line(aes(linetype=Measure))
p <- p + scale_x_continuous(breaks=nk, labels=nk)
p
```



27 K-Means: Plot All Criteria



28 K-Means: predict()

rattle (Williams, 2014) provides a `predict.kmeans()` to assign new observations to their nearest means.

```
set.seed(42)
train <- sample(nobs, 0.7*nobs)
test  <- setdiff(seq_len(nobs), train)

model <- kmeans(ds[train, numi], 2)
predict(model, ds[test, numi])

##   4   5   6   8  11  14  15  16  17  21  28  30  32  36  44  47  50  55
## "2" "2" "2" "2" "1" "1" "1" "1" "1" "1" "1" "2" "1" "2" "1" "1" "2" "2"
## 57  61  62  63  67  74  75  76  77  80  84  92  94  95  97  98  99 100
## "2" "1" "1" "1" "1" "1" "2" "1" "1" "2" "1" "1" "1" "1" "1" "2" "1" "2" "2"
....
```

29 Entropy Weighted K-Means

Sometimes it is better to build clusters based on subsets of variables, particularly if there are many variables. Subspace clustering and bicluster analysis are approaches to doing this.

We illustrate the concept here using `wskm` (Williams *et al.*, 2012), for weighted subspace k-means. We use the `ewkm()` (entropy weighted k-means).

```
set.seed(42)
library(wskm)
m.ewkm <- ewkm(ds, 10)

## Warning:  NAs introduced by coercion
## Error: NA/Nan/Inf in foreign function call (arg 1)
```

The error is expected and once again only numeric variables can be clustered.

```
m.ewkm <- ewkm(ds[numi], 10)

## ****Clustered converged. Terminate!

round(100*m.ewkm$weights)

##   min_temp max_temp rainfall evaporation sunshine wind_gust_speed
## 1       0       0     100          0       0           0
## 2       0       0       0      100       0           0
## 3       0       0     100          0       0           0
## 4       0       0       0          0       0           0
## 5       6       6       6          6       6           6
## 6       0       0       0      100       0           0
## 7       0       0       0      100       0           0
## 8       0       0       0          0       0           0
## 9       6       6       6          6       6           6
## 10      0       0     100          0       0           0
....
```

Exercise: Plot the clusters.

Exercise: Rescale the data so all variables have the same range and then rebuild the cluster, and comment on the differences.

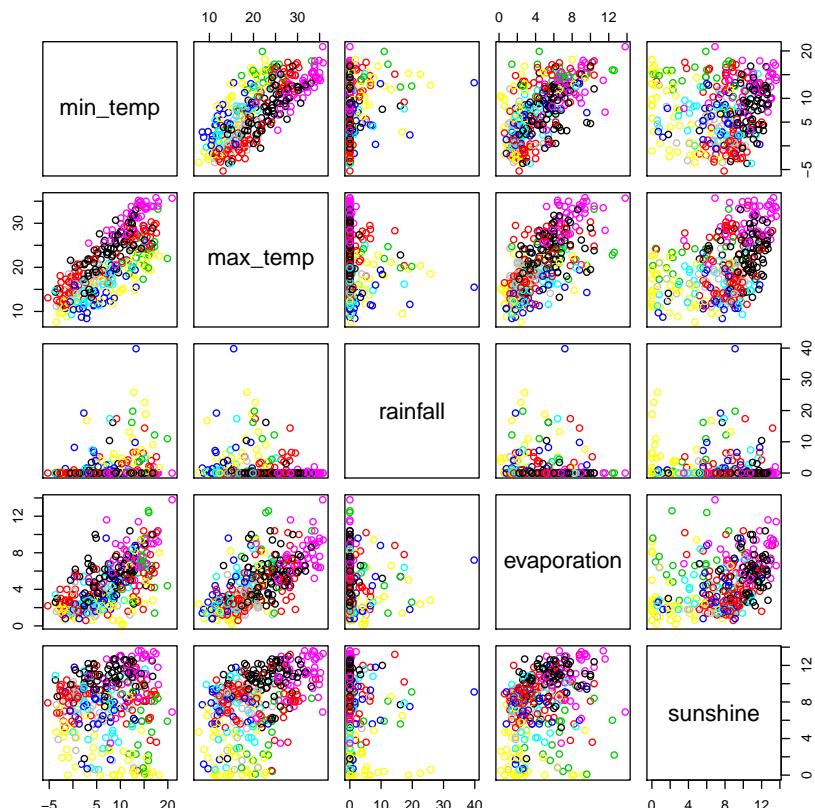
Exercise: Discuss why `ewkm` might be better than k-means. Consider the number of variables as an advantage, particularly in the context of the curse of dimensionality.

30 Partitioning Around Medoids: PAM

```
model <- pam(ds[numi], 10, FALSE, "euclidean")

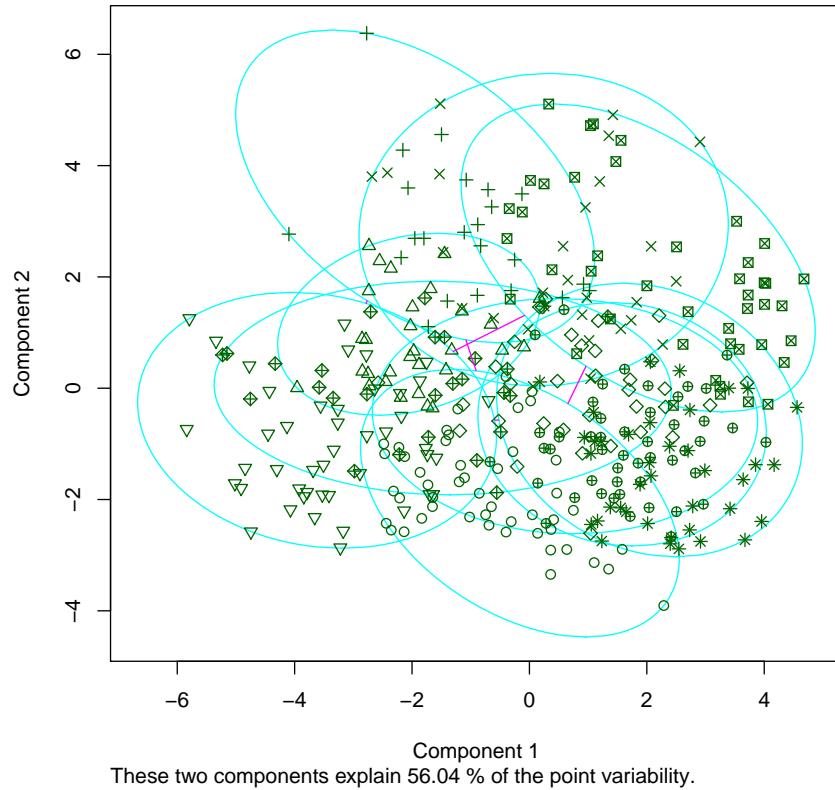
summary(model)

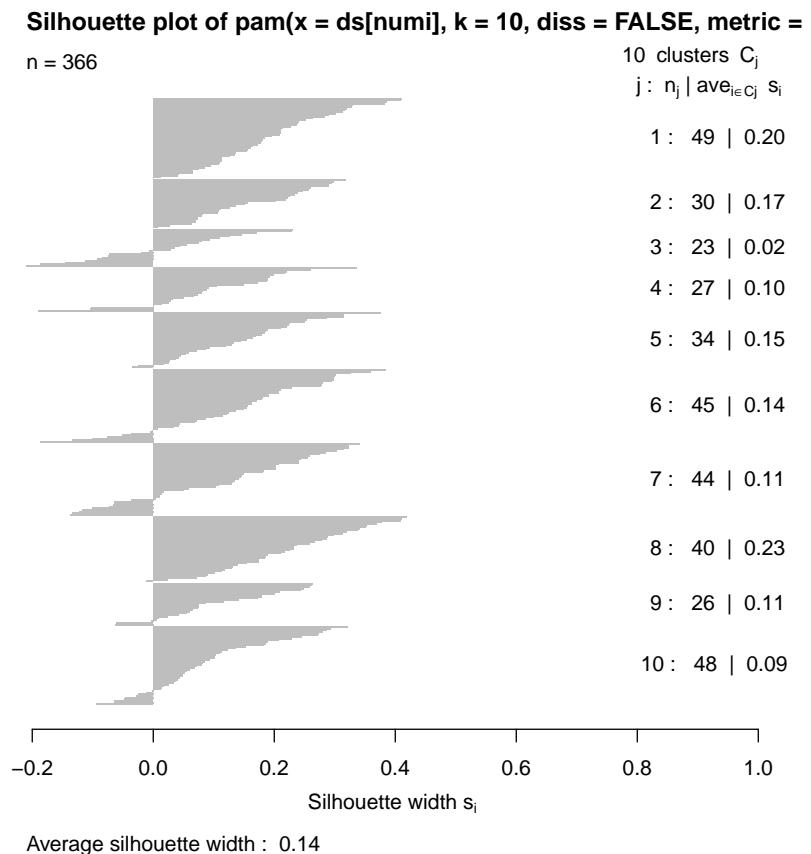
## Medoids:
##      ID min_temp max_temp rainfall evaporation sunshine wind_gust_speed
## [1,] 11     9.1    25.2     0.0      4.2     11.9          30
## [2,] 38    16.5    28.2     4.0      4.2      8.8          39
## ...
## 
plot(ds[numi[1:5]], col=model$clustering)
points(model$medoids, col=1:10, pch=4)
```



```
plot(model)
```

```
clusplot(pam(x = ds[numi], k = 10, diss = FALSE, metric = "euclidean"))
```





31 Clara

32 Hierarchical Cluster in Parallel

Use `hclusterpar()` from `amap` (Lucas, 2011).

```
library(amap)
model <- hclusterpar(na.omit(ds[numi]),
                      method="euclidean",
                      link="ward",
                      nbproc=1)
```

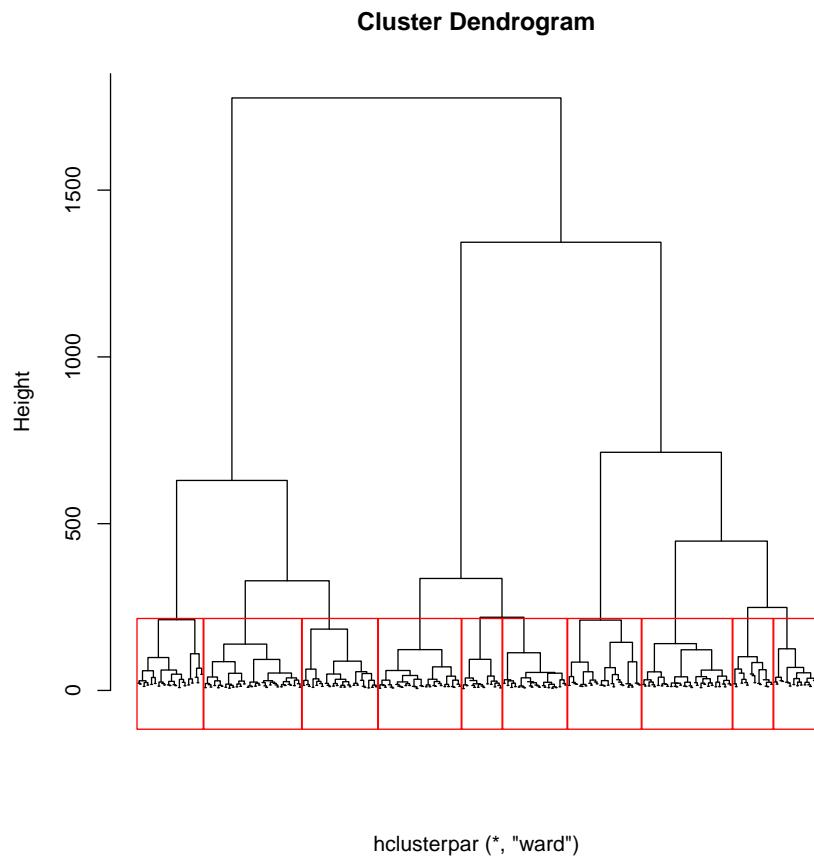
33 Plotting Hierarchical Cluster

Plot from cba (Buchta and Hahsler, 2014).

```
plot(model, main="Cluster Dendrogram", xlab="", labels=FALSE, hang=0)

#Add in rectangles to show the clusters.

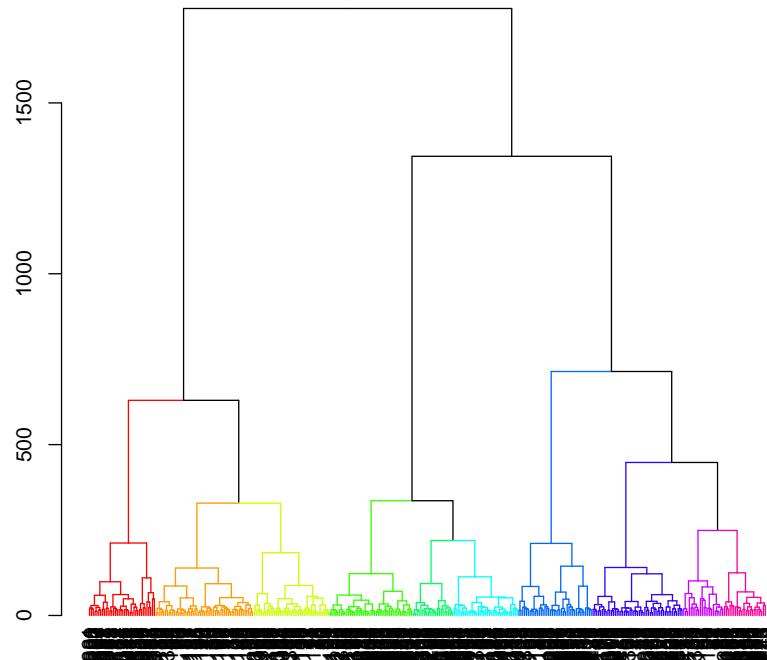
rect.hclust(model, k=10)
```



34 Add Colour to the Hierarchical Cluster

Using the `dendroextras` (Jefferis, 2014) package to add colour to the dendrogram:

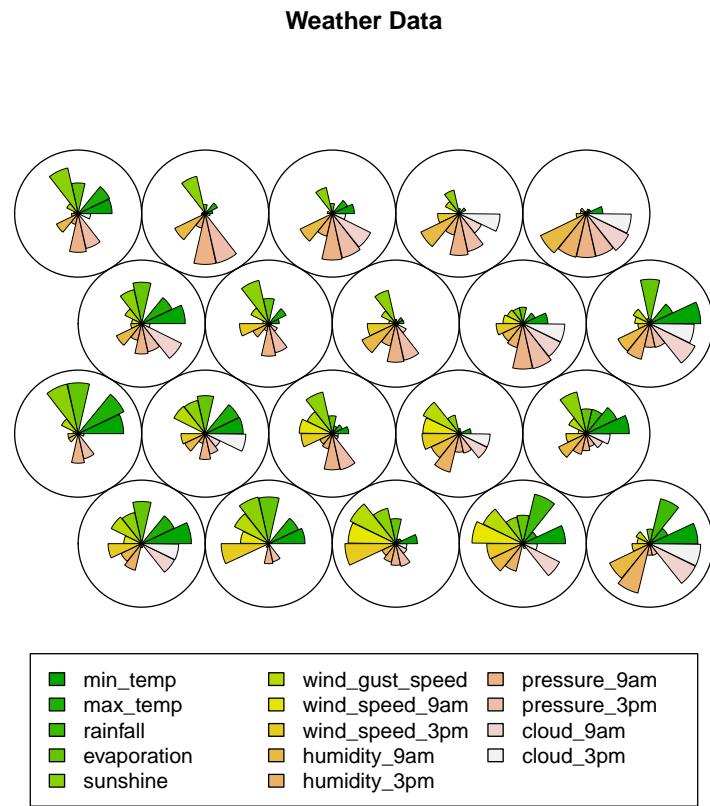
```
library(dendroextras)  
plot(colour_clusters(model, k=10), xlab="")
```



35 Hierarchical Cluster Binary Variables

Exercise: Clustering a large population based on the patterns of missing data within the population is a technique for grouping observations exhibiting similar patterns of behaviour assuming missing by pattern.... We can convert each variable to a binary 1/0 indicating present/missing and then use mona() for a hierarchical clustering. Demonstrate this. Include a levelplot.

36 Self Organising Maps: SOM

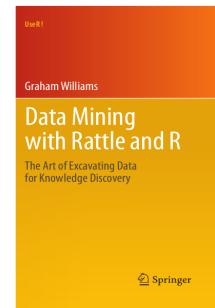


```
library("kohonen")
set.seed(42)
model <- som(scale(ds[numi[1:14]]), grid = somgrid(5, 4, "hexagonal"))
plot(model, main="Weather Data")
```

37 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 module is one of many OnePageR modules available from <http://onepager.togaware.com>. In particular follow the links on the website with a * which indicates the generally more developed OnePageR modules.



Other resources include:

- [Practical Data Science with R](#) by Nina Zumel and John Mount, March 2014, has a good chapter on Cluster Analysis with some depth of explanation of the sum of squares measures, and good examples of R code for performing cluster analysis. It also covers `clusterboot()` and `kmeansruns()` from `fpc`.
- The radar or radial plot code originated from an [RStudio Blog Posting](#).
- The definition of all criteria used to measure the goodness of a clustering can be found in a Vinette of the `clusterCrit` ([Desgraupes, 2013](#)) package. Also available on [CRAN](#).

38 References

- Breiman L, Cutler A, Liaw A, Wiener M (2012). *randomForest: Breiman and Cutler's random forests for classification and regression*. R package version 4.6-7, URL <http://CRAN.R-project.org/package=randomForest>.
- Buchta C, Hahsler M (2014). *cba: Clustering for Business Analytics*. R package version 0.2-14, URL <http://CRAN.R-project.org/package=cba>.
- Desgranges B (2013). *clusterCrit: Clustering Indices*. R package version 1.2.3, URL <http://CRAN.R-project.org/package=clusterCrit>.
- Hennig C (2014). *fpc: Flexible procedures for clustering*. R package version 2.1-7, URL <http://CRAN.R-project.org/package=fpc>.
- Jefferis G (2014). *dendroextras: Extra functions to cut, label and colour dendrogram clusters*. R package version 0.1-4, URL <http://CRAN.R-project.org/package=dendroextras>.
- Lucas A (2011). *amap: Another Multidimensional Analysis Package*. R package version 0.8-7, URL <http://CRAN.R-project.org/package=amap>.
- 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/>.
- 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 3.0.4, URL <http://rattle.togaware.com/>.
- Williams GJ, Huang JZ, Chen X, Wang Q, Xiao L (2012). *wskm: Weighted k-means Clustering*. R package version 1.4.0, URL <http://CRAN.R-project.org/package=wskm>.
- Xie Y (2013). *animation: A gallery of animations in statistics and utilities to create animations*. R package version 2.2, URL <http://CRAN.R-project.org/package=animation>.

This document, sourced from ClustersO.Rnw revision 440, was processed by KnitR version 1.6 of 2014-05-24 and took 30.6 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-06-22 12:38:37.

Data Science with R

Association Rules

Graham.Williams@togaware.com

3rd August 2014

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

Association analysis defined Data Mining at its roots in 1989 and during the 1990s. It remains one of the preeminent techniques for modelling big data and so remains a core tool for the data scientist's toolbox.

As an unsupervised learning technique it has delivered considerable benefit in areas ranging from the traditional shopping basket analysis to the analysis of who bought what other books or who watched what other videos, and in areas including health care, telecommunications, and so on. Often for any data mining project we might usually begin with association analysis to identify issues with our data and then to build multiple local models. The analysis aims to identify patterns that are linked by some commonality (such as by a common person).

In this chapter we review association analysis and will discover new insights into our data through the building of association rule models.

The required packages for this module include:

```
library(arules)      # Association rules.  
library(dplyr)       # Data munging:tbl_df(), %>%.
```

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 The last.fm Dataset

We begin with a simple but real dataset example. The dataset to illustrate the actual application of association rules is the [last.fm](#) dataset. This can be downloaded from [Johannes Ledolter's](#) web site.

```
fname      <- "http://www.biz.uiowa.edu/faculty/jledolter/DataMining/lastfm.csv"
lastfm     <- read.csv(fname, stringsAsFactors=FALSE)
```

Here we process the dataset as usual, following the steps presented in Chapter [Data](#).

```
dsname    <- "lastfm"
ds        <- get(dsname) %>%tbl_df()
ds

## Source: local data frame [289,955 x 4]
##
##   user           artist sex country
## 1 1 red hot chili peppers f Germany
## 2 1 the black dahlia murder f Germany
## 3 1 goldfrapp f Germany
## 4 1 dropkick murphys f Germany
## 5 1 le tigre f Germany
## 6 1 schandmaul f Germany
## 7 1 edguy f Germany
## 8 1 jack johnson f Germany
## 9 1 eluveitie f Germany
## 10 1 the killers f Germany
## ... ... ... ...
```

1.1 Preparing the last.fm Dataset

As usual we begin by cleaning the dataset. In this case there is very little that is required, simply selecting out the identifier (the user) and the items (the artist in this case). We use `select()` from `dplyr` (Wickham and Francois, 2014) to do so. Then our baskets for association analysis will be the artists that each individual user listens to. We also remove any duplicated user/artist entries using `unique()`. The transformations are performed through a pipe as implemented in `magrittr` (Bache and Wickham, 2014) and popularized by `dplyr`.

```
ds <- ds %>% select(user, artist) %>% unique()
ds

## Source: local data frame [289,953 x 2]
##
##   user           artist
## 1    1 red hot chili peppers
## 2    1 the black dahlia murder
## 3    1          goldfrapp
## 4    1      dropkick murphys
## 5    1          le tigre
## 6    1        schandmaul
## 7    1          edguy
## 8    1      jack johnson
## 9    1          eluveitie
## 10   1       the killers
## ... ...     ...
```

The dataset is now a simple data frame of two columns.

1.2 Initial Exploration of last.fm

We have a dataset now that is ready for association rule analysis and we can transform it into a transactions data object. To do so we use `as()` from `arules` (Hahsler *et al.*, 2014).

```
library(arules)

trans <- as(split(ds$artist, ds$user), "transactions")
```

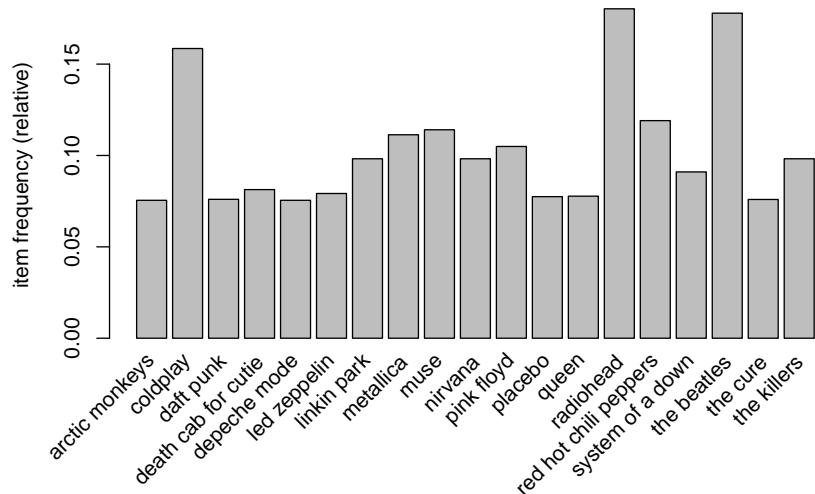
The baskets can be listed using `inspect()` and here we limit the list to the first 5 transactions.

```
inspect(trans[1:5])

##   items                      transactionID
## 1 {dropkick murphys,
##     edguy,
##     eluveitie,
##     goldfrapp,
##     guano apes,
##     jack johnson,
##     john mayer,
##     ...
## }
```

A plot of the more frequent items is produced by `itemFrequencyPlot()`. We can tune how many items are displayed using the `support=` threshold (how frequently an item appears, proportionately).

```
itemFrequencyPlot(trans, support=0.075)
```



1.3 Association Rule Model for last.fm

To build the model we simply call `apriori()`, an implementation of the apriori algorithm for association rule discovery. The two common parameters are `support=` and `confidence=`.

```
model <- apriori(trans, parameter=list(support=0.01, confidence=0.5))

##
## parameter specification:
##   confidence minval smax arem  aval originalSupport support minlen maxlen
##           0.5     0.1     1 none FALSE             TRUE     0.01      1      10
##   target    ext
##   rules FALSE
##
## algorithmic control:
##   filter tree heap memopt load sort verbose
##   0.1 TRUE TRUE FALSE TRUE     2     TRUE
##
## apriori - find association rules with the apriori algorithm
## version 4.21 (2004.05.09)          (c) 1996-2004 Christian Borgelt
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[1004 item(s), 15000 transaction(s)] done [0.03s].
## sorting and recoding items ... [655 item(s)] done [0.01s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 4 done [0.03s].
## writing ... [50 rule(s)] done [0.00s].
## creating S4 object ... done [0.00s].
```

1.4 Inspecting the last.fm Model

We again use `inspect()` to display the model. This will list all association rules that satisfy the criteria specified in building the model (the `support=` and `confidence=`).

```
inspect(model)

##   lhs                  rhs      support confidence lift
## 1 {t.i.}              => {kanye west} 0.01040    0.5673  8.854
## 2 {the pussycat dolls} => {rihanna}    0.01040    0.5778 13.416
## 3 {the fray}           => {coldplay}   0.01127    0.5168  3.260
## 4 {sonata arctica}   => {nightwish}  0.01347    0.5101  8.236
## 5 {judas priest}     => {iron maiden} 0.01353    0.5075  8.563
## 6 {the kinks}          => {the beatles} 0.01360    0.5299  2.979
## 7 {travis}             => {coldplay}   0.01373    0.5628  3.550
## 8 {the flaming lips}  => {radiohead}  0.01307    0.5297  2.939
## 9 {megadeth}           => {metallica}  0.01627    0.5281  4.744
....
```

```
inspect(subset(model, subset=lift>8))

##   lhs                  rhs      support confidence lift
## 1 {t.i.}              => {kanye west} 0.01040    0.5673  8.854
## 2 {the pussycat dolls} => {rihanna}    0.01040    0.5778 13.416
## 3 {sonata arctica}   => {nightwish}  0.01347    0.5101  8.236
## 4 {judas priest}     => {iron maiden} 0.01353    0.5075  8.563

inspect(sort(subset(model, subset=lift>8), by="confidence"))

##   lhs                  rhs      support confidence lift
## 1 {the pussycat dolls} => {rihanna}    0.01040    0.5778 13.416
## 2 {t.i.}                => {kanye west} 0.01040    0.5673  8.854
## 3 {sonata arctica}   => {nightwish}  0.01347    0.5101  8.236
## 4 {judas priest}     => {iron maiden} 0.01353    0.5075  8.563
```

2 Understanding the Algorithm: Sample Dataset

We now use a smaller artificial dataset to illustrate the model building algorithm.

To illustrate the concepts of association analysis we will generate a random dataset of some 5 items and 10 baskets. Before doing so we use `set.seed()` to ensure we get the same “random” dataset each time and we set up some constants.

```
set.seed(42)
nb <- 10      # Number of baskets.
ni <- 5       # Number of items.
nc <- 40      # Number of combinations.
```

A simple use of `sample()` combined with `sprintf()` generates the baskets identified with an initial “b” and items with an initial “i”. The use of `sort()` ensures a nicely sorted order to the baskets. With `unique()` we remove duplicate items from a basket. We revert the `rownames()` to a complete sequence for convenience.

```
ds <- data.frame(id=sort(sprintf("b%02d", sample(1:nb, nc, replace=TRUE))),
                  item=sprintf("i%1d", sample(1:ni, nc, replace=TRUE)))
ds <- unique(ds)
rownames(ds) <- NULL
```

The different baskets contain differing number of items as we can easily see with a simple `dplyr` sequence.

```
ds %>% group_by(id) %>% tally()
## Source: local data frame [10 x 2]
##
##      id     n
## 1   b01     3
## 2   b02     2
## 3   b03     2
## ...
## ...
```

We can also compactly list the contents of the baskets, again using `dplyr`.

```
ds %>% group_by(id) %>% summarise(items=paste(sort(item), collapse=", "))
## Source: local data frame [10 x 2]
##
##      id     items
## 1   b01    i1, i2, i3
## 2   b02    i3, i5
## 3   b03    i4, i5
## 4   b04    i2, i4
## 5   b05    i1, i2, i4
## 6   b06    i1, i4
## 7   b07 i2, i3, i4, i5
## 8   b08 i1, i3, i4, i5
## 9   b09    i2, i4, i5
## 10  b10   i1, i2, i3, i4
```

2.1 1-Itemsets and Support

The basic concept of association analysis is that of **items** in baskets. Our aim is usually to identify sets of items that commonly occur together in a basket. Thus we talk about **itemsets**.

The simplest itemset is a set containing a single item. Thus $\{i_1\}$ is an itemset containing a single item, i_1 . We say that this itemset has a frequency of 5. That is, it appears in 5 of the 10 baskets:

```
ds %>% group_by(id) %>% summarise(i1="i1" %in% item) %>% filter(i1)

## Source: local data frame [5 x 2]
##
##   id   i1
## 1 b01 TRUE
## 2 b05 TRUE
## 3 b06 TRUE
## 4 b08 TRUE
## 5 b10 TRUE
```

Because this itemset has a single item (i_1) we refer to it as a **1-itemset**.

The frequencies of the other 1-itemsets can be easily calculated:

```
ds %>% group_by(item) %>% tally()

## Source: local data frame [5 x 2]
##
##   item n
## 1 i1 5
## 2 i2 6
## 3 i3 5
## 4 i4 8
## 5 i5 5
```

This leads us to the concept of **support**. We define support as the proportion of all baskets that contain the itemset. There are 10 baskets, so the support for the 1-itemset $\{i_1\}$ is 5/10 or 0.5. Similarly we can calculate the support for each of our 1-itemsets:

```
ds %>% group_by(item) %>% tally() %>% mutate(s=n/nb)

## Source: local data frame [5 x 3]
##
##   item n   s
## 1 i1 5 0.5
## 2 i2 6 0.6
## 3 i3 5 0.5
## 4 i4 8 0.8
## 5 i5 5 0.5
```

This might suggest that if a 1-itemset has a support of less than 0.6 then it might not be so interesting (since all items occur with at least that frequency). We use support to tune the number of “interesting” itemsets we extract from our dataset.

2.2 1-Itemsets—Using ARules

The `arules` package provides support for association analysis in R. Once loaded we can transform our dataset into a *transactions* data structure. To do so we use `split()` to transform our data frame into a list of lists, each list being a basket of items, represented as a factor. The levels of the factors are the basket items.

```
library(arules)
dst <- as(split(ds$item, ds$id), "transactions")
dst

## transactions in sparse format with
## 10 transactions (rows) and
## 5 items (columns)
```

The item frequencies can be calculated simply using `itemFrequency()`:

```
itemFrequency(dst)

## i1 i2 i3 i4 i5
## 0.5 0.6 0.5 0.8 0.5
```

Notice these are the same frequencies we manually calculated earlier.

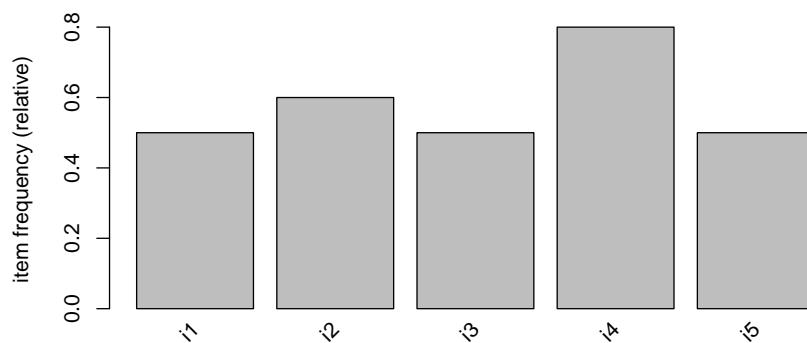
We can obtain the actual frequency count by using `type="absolute"` where the default is "relative". Using "absolute" we see the actual counts of the items. These again agree with our earlier calculations.

```
itemFrequency(dst, type="absolute")

## i1 i2 i3 i4 i5
## 5 6 5 8 5
```

We can also obtain a frequency plot for the 1-itemsets:

```
itemFrequencyPlot(dst)
```



2.3 k-Itemsets—Using ARules

We extend the concept of a 1-itemset to the general concept of a k-itemset. For example, the 2-itemset i_1, i_2 has a frequency of 3 and so a support of 0.3.

We can calculate the 2-itemsets and their frequencies:

```
merge(ds, ds, by="id") %>%
  subset(as.character(item.x) < as.character(item.y)) %>%
  mutate(itemset=paste(item.x, item.y)) %>%
  group_by(itemset) %>%
  tally()

## Source: local data frame [10 x 2]
##
##   itemset n
## 1     i1 i2 3
## 2     i1 i3 3
## 3     i1 i4 4
## 4     i1 i5 1
## 5     i2 i3 3
## 6     i2 i4 5
## 7     i2 i5 2
## 8     i3 i4 3
## 9     i3 i5 3
## 10    i4 i5 4
```

Similarly, the 3-itemset $\{i_1, i_2, i_3\}$ has a frequency of 2 and so a support of 0.2.

```
merge(ds, ds, by="id") %>%
  merge(ds, by="id") %>%
  subset(as.character(item.x) < as.character(item.y) &
         as.character(item.y) < as.character(item)) %>%
  mutate(itemset=paste(item.x, item.y, item)) %>%
  group_by(itemset) %>%
  tally()

## Source: local data frame [9 x 2]
##
##   itemset n
## 1     i1 i2 i3 2
## 2     i1 i2 i4 2
## 3     i1 i3 i4 2
## 4     i1 i3 i5 1
## 5     i1 i4 i5 1
## 6     i2 i3 i4 2
## 7     i2 i3 i5 1
## 8     i2 i4 i5 2
## 9     i3 i4 i5 2
```

3 APriori Principle—ARules

We noted above that we are interested in finding the frequent itemsets—that is, the itemsets that have support that is at least some user specified threshold. A basic observation is that if a 2-itemset is frequent, then both of the items, by themselves as 1-itemsets, must also be frequent. Similarly, for a 3-itemset to be frequent, each of the possible 2-itemsets that only contain two items from the 3-itemset, must also be frequent. And so on.

This observation is the basis for an algorithm to identify all frequent itemsets. We begin with identifying the 1-frequent itemsets and then only consider the 2-itemsets that can be constructed from the items in the frequent 1-itemsets. And so on.

The `arules` package implements the apriori algorithm for association rules.

4 Confidence

Once we have generated candidate (i.e., frequent enough) itemsets, we next generate the candidate association rules and retain those that have enough confidence. The confidence is a measure of the strength of an association rule. It is the frequency of occurrence of the right-hand items in the rule from among those baskets that contain the items on the left-hand side of the rule.

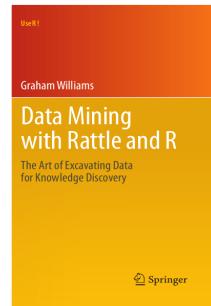
5 Further Reading

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:

- Christian Borgelt is owed a much gratitude for the early work he did on implementing association rules in C.



6 References

- Bache SM, Wickham H (2014). *magrittr: magrittr - a forward-pipe operator for R*. R package version 1.0.1, URL <http://CRAN.R-project.org/package=magrittr>.
- Hahsler M, Buchta C, Gruen B, Hornik K (2014). *arules: Mining Association Rules and Frequent Itemsets*. R package version 1.1-3, URL <http://CRAN.R-project.org/package=arules>.
- 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/>.
- 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.

This document, sourced from ARulesO.Rnw revision 470, was processed by KnitR version 1.6 of 2014-05-24 and took 5.4 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 2014-08-03 17:34:03.

Data Science with R

Decision Trees

Graham.Williams@togaware.com

3rd August 2014

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

Decision trees are widely used in data mining and well supported in R (R Core Team, 2014). Decision tree learning deploys a divide and conquer approach, known as recursive partitioning. It is usually implemented as a greedy search using information gain or the Gini index to select the best input variable on which to partition our dataset at each step.

This Module introduces rattle (Williams, 2014) and rpart (Therneau and Atkinson, 2014) for building decision trees. We begin with a step-by-step example of building a decision tree using Rattle, and then illustrate the process using R beginning with Section 14. We cover both classification trees and regression trees.

The required packages for this module include:

```
library(rattle)      # GUI for building trees and fancy tree plot
library(rpart)        # Popular decision tree algorithm
library(rpart.plot)   # Enhanced tree plots
library(party)         # Alternative decision tree algorithm
library(partykit)     # Convert rpart object to BinaryTree
library(RWeka)        # Weka decision tree J48.
library(C50)          # Original C5.0 implementation.
```

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.

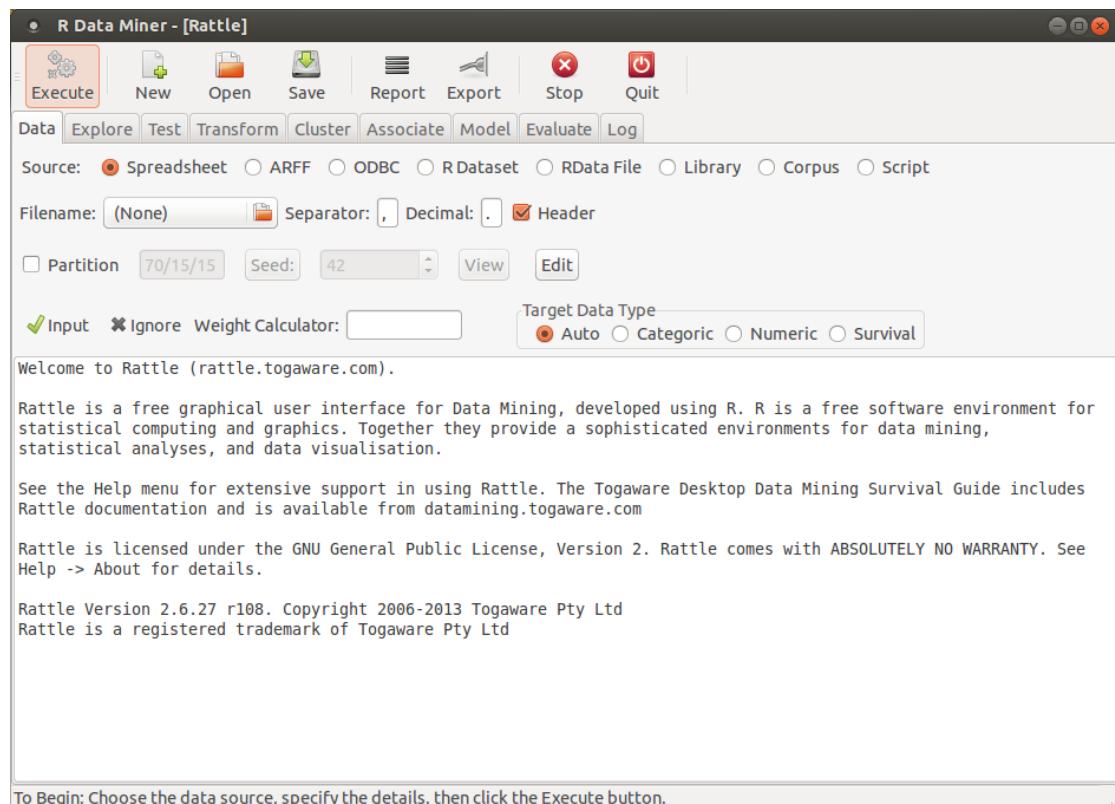


1 Start Rattle

After starting R (perhaps via RStudio) we can start up `rattle` (Williams, 2014) from the R Console prompt. Simply load the `rattle` package then invoke the `rattle()`, as in:

```
library(rattle)
rattle()
```

We will see the following Window. Notice the row of buttons, below which we see a series of tabs that we will work through. Remember, in Rattle, that after we set up a particular tab we must press the Execute button to have the tab take effect.

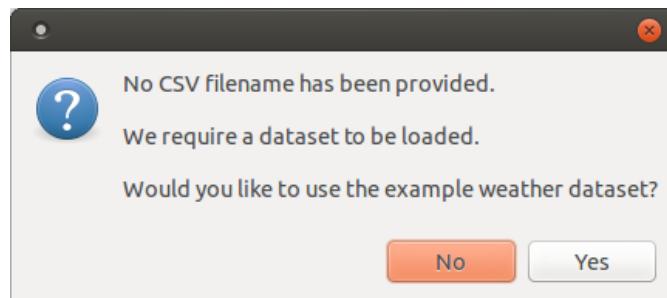


2 Load Example Weather Dataset

Rattle provides a number of sample datasets. We can easily load them into Rattle. By default, Rattle will load the weather dataset.

We load the dataset in two simple steps

1. Click on the Execute button and an example dataset is offered.
2. Click on Yes to load the weather dataset.



We can use this dataset for [predictive modelling](#) to predict if it might rain tomorrow (aka [statistical classification](#) and [supervised learning](#)), or to predict how much rain we might get tomorrow (aka [regression analysis](#)).

3 Summary of the Weather Dataset

The **weather** dataset from **rattle** consists of daily observations of various weather related data over one year at one location (Canberra Airport). Each observation has a date and location. These are the *id* variables for this dataset.

The observations include the temperature during the day, humidity, the number of hours of sunshine, wind speed and direction, and the amount of evaporation. These are the *input* variables for this dataset.

Together with each day's observations we record whether it rains the following day and how much rain was received. These will be the *target* variables for this dataset.

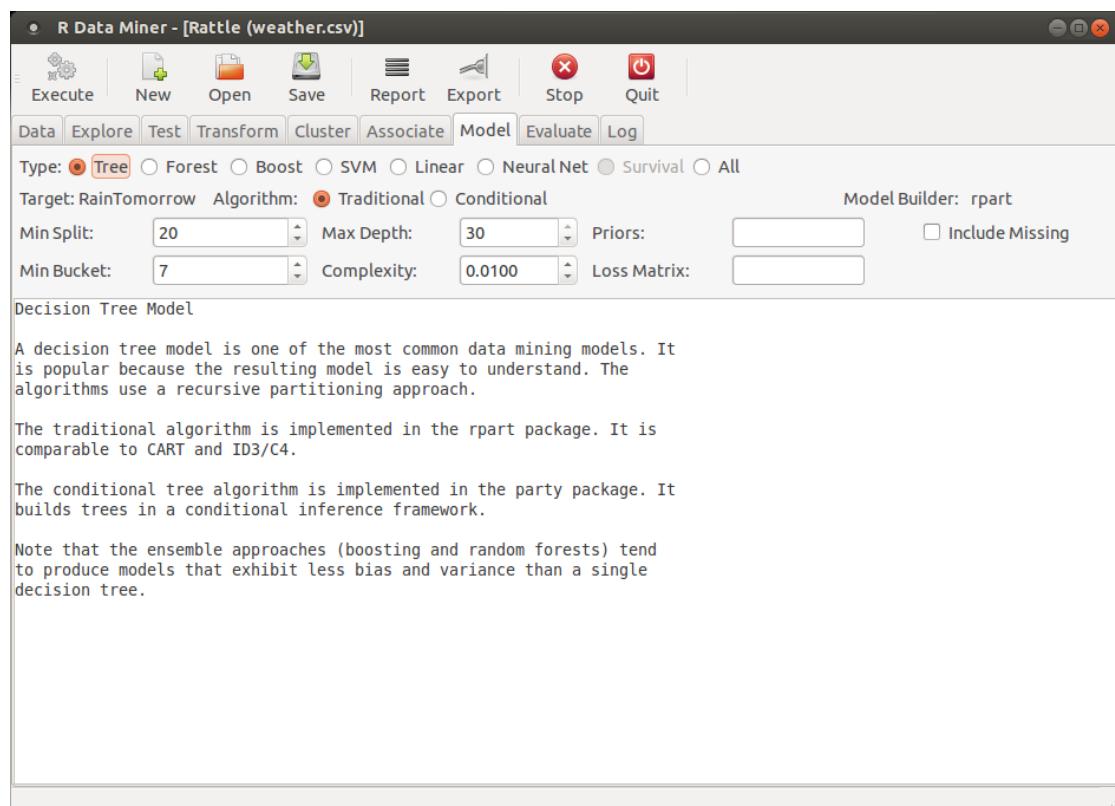
Scroll through the list of variables to notice that default roles have been assigned the variables.

No.	Variable	Data Type	Input	Target	Risk	Ident	Ignore	Weight	Comment
16	Pressure9am	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 190
17	Pressure3pm	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 193
18	Cloud9am	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 9
19	Cloud3pm	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 9
20	Temp9am	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 178
21	Temp3pm	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 200
22	RainToday	Categoric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 2
23	RISK_MM	Numeric	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 47
24	RainTomorrow	Categoric	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 2

Roles noted. 366 observations and 20 input variables. The target is RainTomorrow. Categoric 2. Classification models enabled.

4 Model Tab — Decision Tree

We can now click on the Model tab to display the modelling options. The default modelling option is to build a decision tree. Various options to tune the building of a decision tree are provided. Underneath `rpart` (Therneau and Atkinson, 2014) is used to build the tree, and many more options are available through using `rpart()` directly, as we will see later in this Module.



5 Build Tree to Predict RainTomorrow

We can simply click the Execute button to build our first decision tree. Notice the time taken to build the tree, as reported in the status bar at the bottom of the window. A summary of the tree is presented in the text view panel. We note that a classification model is built using `rpart()`.

```

R Data Miner - [Rattle (weather.csv)]
Execute New Open Save Report Export Stop Quit
Data Explore Test Transform Cluster Associate Model Evaluate Log
Type: Tree Forest Boost SVM Linear Neural Net Survival All
Target: RainTomorrow Algorithm: Traditional Conditional Model Builder: rpart
Min Split: 20 Max Depth: 30 Priors: Include Missing
Min Bucket: 7 Complexity: 0.0100 Loss Matrix: Rules Draw
Summary of the Decision Tree model for Classification (built using 'rpart'):

n= 256

node), split, n, loss, yval, (yprob)
  * denotes terminal node

1) root 256 41 No (0.83984375 0.16015625)
  2) Pressure3pm>=1011.9 204 16 No (0.92156863 0.07843137)
    4) Cloud3pm< 7.5 195 10 No (0.94871795 0.05128205) *
    5) Cloud3pm>=7.5 9 3 Yes (0.33333333 0.66666667) *
  3) Pressure3pm< 1011.9 52 25 No (0.51923077 0.48076923)
    6) Sunshine>=8.85 25 5 No (0.80000000 0.20000000) *
    7) Sunshine< 8.85 27 7 Yes (0.25925926 0.74074074) *

Classification tree:
rpart(formula = RainTomorrow ~ ., data = crs$dataset[crs$train,
  c(crs$input, crs$target)], method = "class", parms = list(split = "information"),
  control = rpart.control(usesurrogate = 0, maxsurrogate = 0))

Variables actually used in tree construction:
[1] Cloud3pm    Pressure3pm Sunshine

Root node error: 41/256 = 0.16016
The Decision Tree model has been built. Time taken: 0.09 secs

```

The number of observations from which the model was built is reported. This is 70% of the observations available in the **weather** dataset. The weather dataset is quite a tiny dataset in the context of data mining, but suitable for our purposes here.

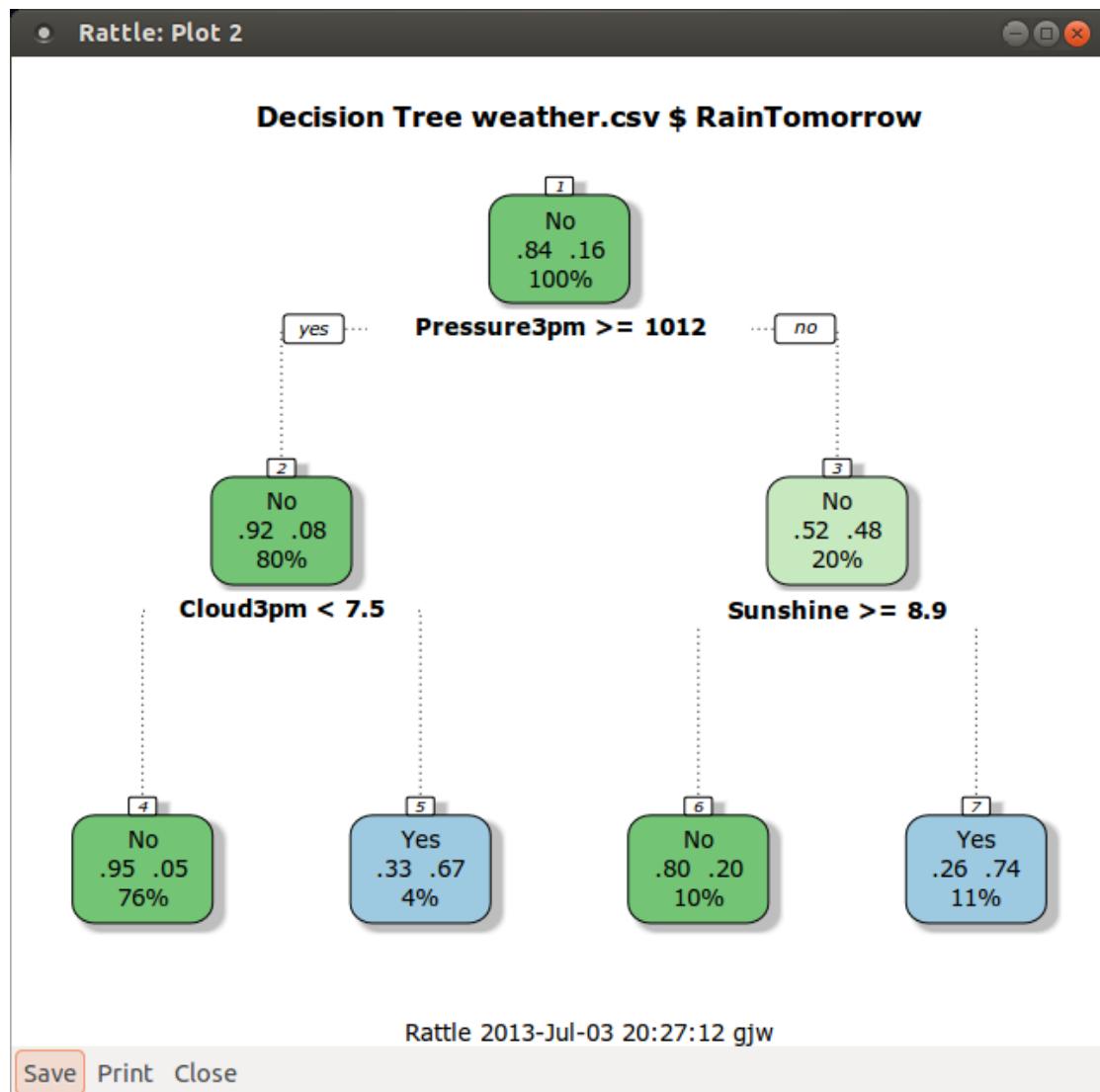
A legend for reading the information in the textual representation of the decision tree is then presented.

The legend indicates that each node is identified (numbered), followed by a split (which will usually be in the form of a test on the value of a variable), the number of entities *n* at that node, the number of entities that are incorrectly classified (the *loss*), the default classification for the node (the *yval*), and then the distribution of classes in that node (the *yprobs*). The next line indicates that a “*” denotes a terminal node of the tree (i.e., a leaf node—the tree is not split any further at that node).

The distribution is ordered by levels of the class and the order is the same for all nodes. The order here is: No, Yes.

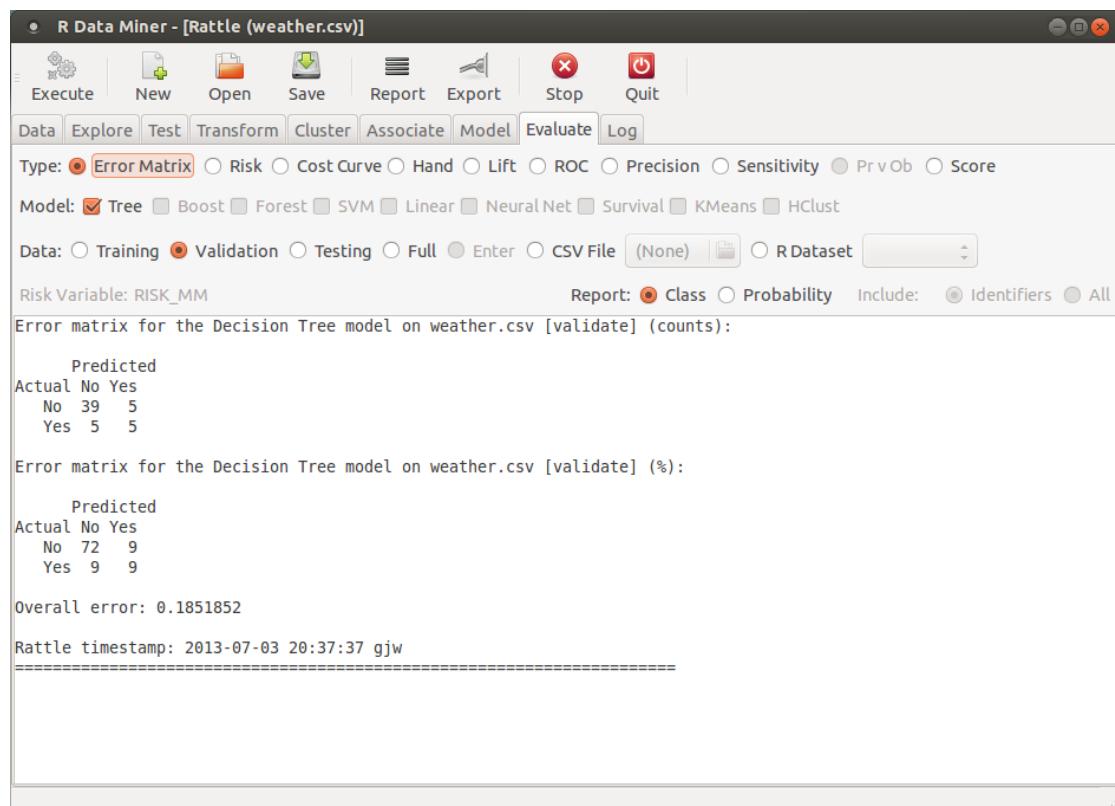
6 Decision Tree Predicting RainTomorrow

Click the Draw button to display a tree (Settings → Advanced Graphics).



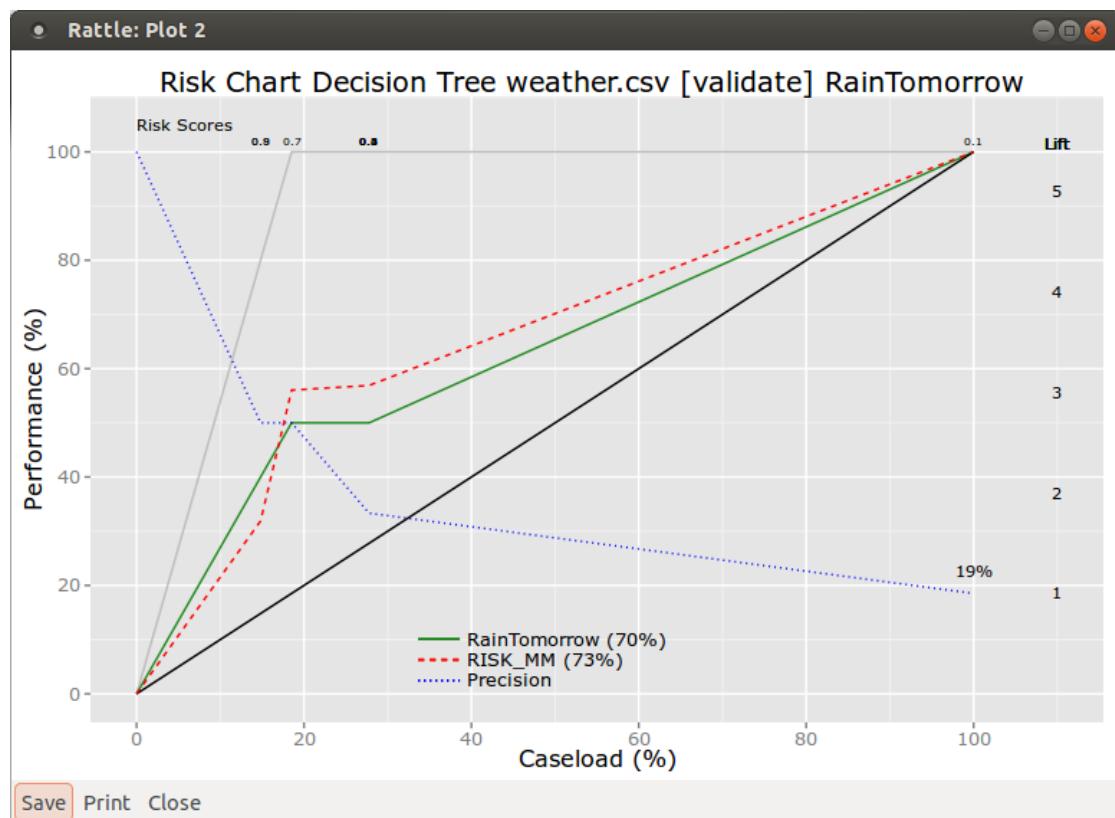
7 Evaluate Decision Tree—Error Matrix

Click Evaluate tab—options to evaluate model performance. Click Execute to display simple error matrix. Identify the True/False Positives/Negatives.



8 Decision Tree Risk Chart

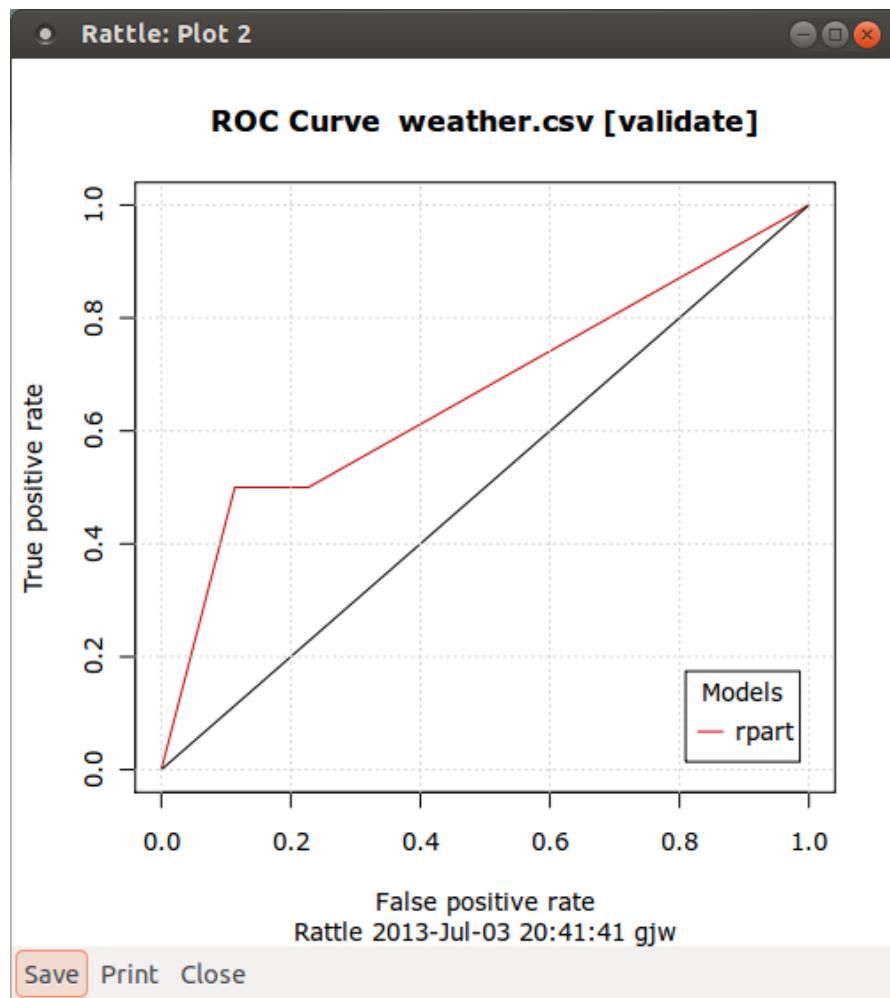
Click the Risk type and then Execute.



Exercise: Research how to interpret a risk chart. Explain the risk chart in one or two paragraphs.

9 Decision Tree ROC Curve

Click the ROC type and then Execute.



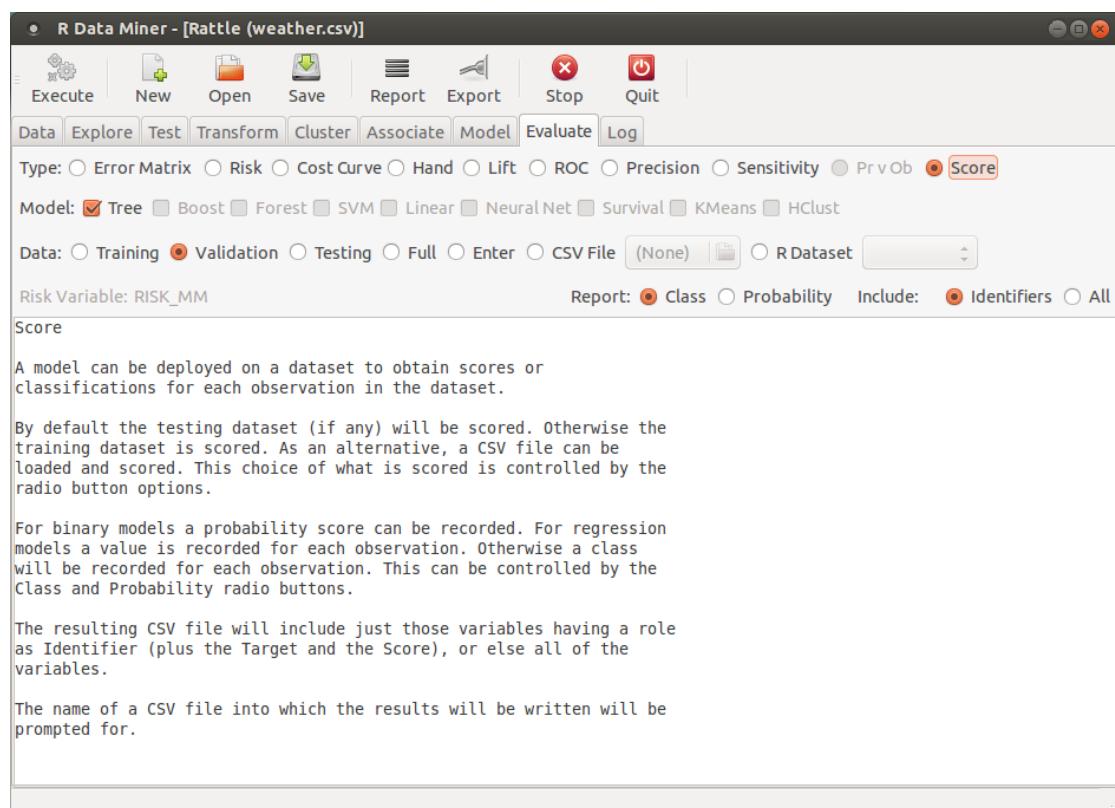
Exercise: Research how to interpret an ROC curve. Explain the ROC curve in one or two paragraphs.

10 Other Evaluation Plots

Exercise: Research the cost curve, the Hand plots, the Lift chart and the Precision and Sensitivity plots. Produce an example of each and explain each one of them in one or two paragraphs.

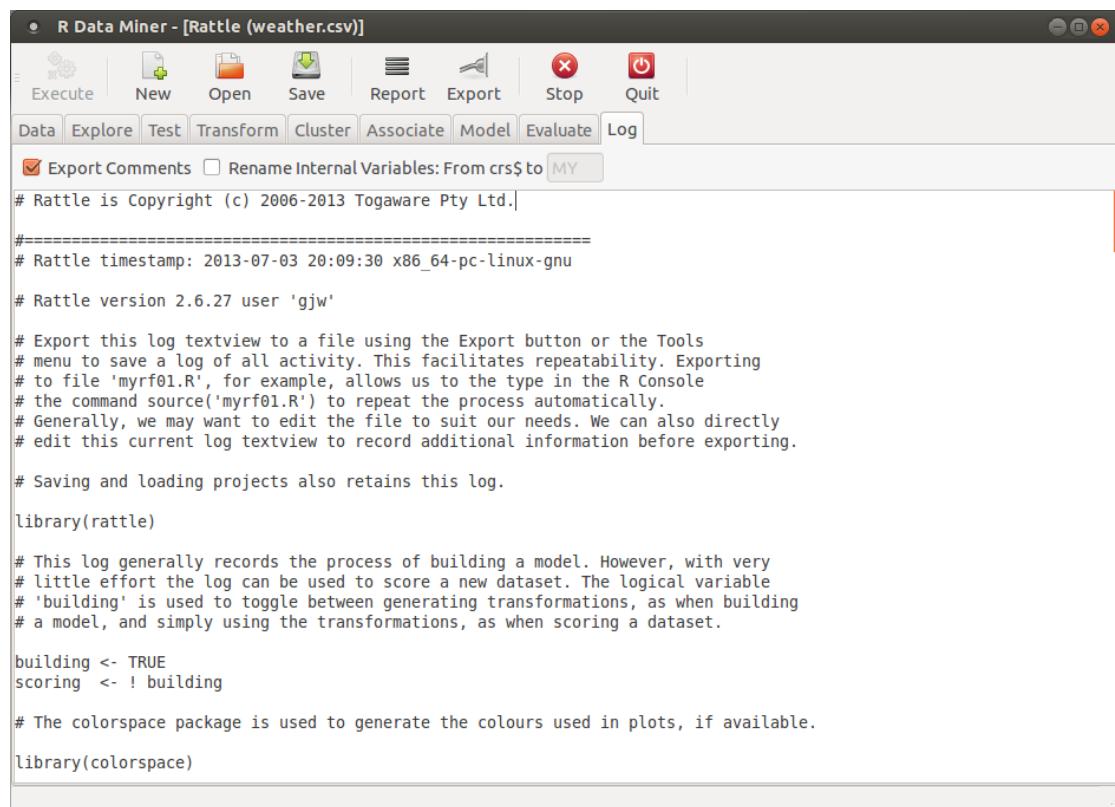
11 Score a Dataset

Click the Score type to score a new dataset using model.



12 Log of R Commands

Click the Log tab for a history of all your interactions. Save the log contents as a script to repeat what we did.



```
R Data Miner - [Rattle (weather.csv)]
Execute New Open Save Report Export Stop Quit
Data Explore Test Transform Cluster Associate Model Evaluate Log
 Export Comments  Rename Internal Variables: From crs$ to MY
# Rattle is Copyright (c) 2006-2013 Togaware Pty Ltd.
=====
# Rattle timestamp: 2013-07-03 20:09:30 x86_64-pc-linux-gnu
# Rattle version 2.6.27 user 'gjw'
# Export this log textview to a file using the Export button or the Tools
# menu to save a log of all activity. This facilitates repeatability. Exporting
# to file 'myrf01.R', for example, allows us to type in the R Console
# the command source('myrf01.R') to repeat the process automatically.
# Generally, we may want to edit the file to suit our needs. We can also directly
# edit this current log textview to record additional information before exporting.
# Saving and loading projects also retains this log.
library(rattle)
# This log generally records the process of building a model. However, with very
# little effort the log can be used to score a new dataset. The logical variable
# 'building' is used to toggle between generating transformations, as when building
# a model, and simply using the transformations, as when scoring a dataset.
building <- TRUE
scoring <- ! building
# The colorspace package is used to generate the colours used in plots, if available.
library(colorspace)
```

13 From GUI to R—rpart()

The Log tab shows the call to rpart() to build the model. We can click on the Export button to save the script to file and that script can then be used to rerun this model building process, automatically within R.

The command to build the model is presented in the Log tab exactly as it is passed on to R to invoke the model building. It takes a little time to understand it, and the remainder of this module covers interacting directly with R to achieve the same results.

The screenshot shows the R Data Miner interface with the title bar "R Data Miner - [Rattle (weather.csv)]". The menu bar includes "Execute", "New", "Open", "Save", "Report", "Export", "Stop", and "Quit". Below the menu is a tab bar with "Data", "Explore", "Test", "Transform", "Cluster", "Associate", "Model", "Evaluate", and "Log", where "Log" is selected. A checkbox "Export Comments" is checked. The main window displays the R code used to build a decision tree:

```

#=====
# Rattle timestamp: 2013-07-03 20:25:41 x86_64-pc-linux-gnu
# Decision Tree
# The 'rpart' package provides the 'rpart' function.
require(rpart, quietly=TRUE)
# Reset the random number seed to obtain the same results each time.
set.seed(crv$seed)
# Build the Decision Tree model.
crs$rpart <- rpart(RainTomorrow ~ .,
  data=crs$dataset[crs$train, c(crs$input, crs$target)],
  method="class",
  parms=list(split="information"),
  control=rpart.control(usesurrogate=0,
    maxsurrogate=0))
# Generate a textual view of the Decision Tree model.
print(crs$rpart)
printcp(crs$rpart)
cat("\n")
# Time taken: 0.09 secs

```

As we will soon learn we would write this sequence of command ourselves as:

```

set.seed(42)
library(rattle)
library(rpart)
ds      <- weather
target <- "RainTomorrow"
nobs   <- nrow(ds)
form   <- formula(paste(target, "~ ."))
train  <- sample(nobs, 0.70 * nobs)
vars   <- -c(1,2,23)
model  <- rpart(form, ds[train, vars], parms=list(split="information"))

```

14 Prepare Weather Data for Modelling

See the separate Data and Model modules for template for preparing data and building models. In brief, we set ourselves up for modelling the **weather** dataset with the following commands, extending the simpler example we have just seen.

```
set.seed(1426)
library(rattle)
data(weather)
dsname      <- "weather"
ds          <- get(dsname)
id          <- c("Date", "Location")
target      <- "RainTomorrow"
risk         <- "RISK_MM"
ignore       <- c(id, if (exists("risk")) risk)
(vars        <- setdiff(names(ds), ignore))

## [1] "MinTemp"      "MaxTemp"      "Rainfall"      "Evaporation"
## [5] "Sunshine"      "WindGustDir"   "WindGustSpeed" "WindDir9am"
## [9] "WindDir3pm"    "WindSpeed9am"  "WindSpeed3pm"  "Humidity9am"
## [13] "Humidity3pm"   "Pressure9am"   "Pressure3pm"   "Cloud9am"
....
inputs      <- setdiff(vars, target)
(nobs       <- nrow(ds))

## [1] 366

(numerics   <- intersect(inputs, names(ds)[which(sapply(ds[vars], is.numeric))]))

## [1] "MinTemp"      "MaxTemp"      "Rainfall"      "Sunshine"
## [5] "WindDir9am"   "WindDir3pm"   "WindSpeed9am" "WindSpeed3pm"
## [9] "Humidity9am"  "Humidity3pm"   "Pressure9am"   "Pressure3pm"
## [13] "Cloud9am"     "Cloud3pm"
....
(categorics <- intersect(inputs, names(ds)[which(sapply(ds[vars], is.factor))]))

## [1] "Evaporation"  "WindGustDir"   "WindGustSpeed" "Temp9am"
## [5] "Temp3pm"

(form       <- formula(paste(target, "~ .")))

## RainTomorrow ~ .

length(train <- sample(nobs, 0.7*nobs))
## [1] 256

length(test  <- setdiff(seq_len(nobs), train))
## [1] 110

actual      <- ds[test, target]
risks       <- ds[test, risk]
```

15 Review the Dataset

It is always a good idea to review the data.

```
dim(ds)
## [1] 366 24

names(ds)

## [1] "Date"          "Location"       "MinTemp"        "MaxTemp"
## [5] "Rainfall"       "Evaporation"    "Sunshine"       "WindGustDir"
## [9] "WindGustSpeed" "WindDir9am"     "WindDir3pm"     "WindSpeed9am"
## [13] "WindSpeed3pm"  "Humidity9am"   "Humidity3pm"   "Pressure9am"
## [17] "Pressure3pm"   "Cloud9am"      "Cloud3pm"      "Temp9am"
.....
head(ds)

##           Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 1 2007-11-01 Canberra     8.0    24.3     0.0      3.4      6.3
## 2 2007-11-02 Canberra    14.0    26.9     3.6      4.4     9.7
## 3 2007-11-03 Canberra   13.7    23.4     3.6      5.8     3.3
## 4 2007-11-04 Canberra   13.3    15.5    39.8      7.2     9.1
.....
tail(ds)

##           Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 361 2008-10-26 Canberra     7.9    26.1     0.0      6.8      3.5
## 362 2008-10-27 Canberra    9.0    30.7     0.0      7.6     12.1
## 363 2008-10-28 Canberra    7.1    28.4     0.0     11.6     12.7
## 364 2008-10-29 Canberra   12.5    19.9     0.0      8.4      5.3
.....
str(ds)

## 'data.frame': 366 obs. of  24 variables:
## $ Date        : Date, format: "2007-11-01" "2007-11-02" ...
## $ Location    : Factor w/ 49 levels "Adelaide","Albany",...: 10 10 10 10 ...
## $ MinTemp     : num  8 14 13.7 13.3 7.6 6.2 6.1 8.3 8.8 8.4 ...
## $ MaxTemp     : num  24.3 26.9 23.4 15.5 16.1 16.9 18.2 17 19.5 22.8 ...
.....
summary(ds)

##           Date                  Location        MinTemp        MaxTemp
##  Min.   :2007-11-01   Canberra   :366   Min.   :-5.30   Min.   : 7.6
##  1st Qu.:2008-01-31   Adelaide   : 0    1st Qu.: 2.30   1st Qu.:15.0
##  Median :2008-05-01   Albany     : 0    Median : 7.45   Median :19.6
##  Mean   :2008-05-01   Albury     : 0    Mean   : 7.27   Mean   :20.6
.....
```

16 Build Decision Tree Model

Buld a decision tree using `rpart()`. Once the different variables have been defined (form, ds, train, and vars) this some command can be re-used.

```
model <- rpart(formula=form, data=ds[train, vars])
```

Notice in the above command we have named each of the arguments. If we have a look at the structure of `rpart` we see that the arguments are in their expected order, and hence the use of the argument names, `formula=` and `data=` is optional.

```
str(rpart)
## function (formula, data, weights, subset, na.action=na.rpart, method,
##           model=FALSE, x=FALSE, y=TRUE, parms, control, cost, ...)
model <- rpart(form, ds[train, vars])
```

Whilst they are optional, they can assist in reading the code, and so it is recommended that we use the argument names in function calls.

A textual presentation of the model is concise but informative, once we learn how to read it. Note this tree is different to the previous one, since we have randomly selected a different dataset to train the model.

```
model
## n= 256
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
##  1) root 256 38 No (0.85156 0.14844)
##    2) Humidity3pm< 71 238 25 No (0.89496 0.10504)
##      4) Pressure3pm>=1010 208 13 No (0.93750 0.06250) *
##      5) Pressure3pm< 1010 30 12 No (0.60000 0.40000)
##        10) Sunshine>=9.95 14  1 No (0.92857 0.07143) *
##        11) Sunshine< 9.95 16  5 Yes (0.31250 0.68750) *
##    3) Humidity3pm>=71 18  5 Yes (0.27778 0.72222) *
```

Refer to Section 5 for an explanation of the format of the textual presentation of the decision tree. The first few lines indicate the number of observation from which the tree was built (`n =`) and then a legend for reading the information in the textual representation of the tree.

The legend indicates that a node number will be provided, followed by a split (which will usually be in the form of a variable operation and value), the number of entities `n` at that node, the number of entities that are incorrectly classified (the `loss`), the default classification for the node (the `yval`), and then the distribution of classes in that node (the `yprobs`). The distribution is ordered by class and the order is the same for all nodes. The next line indicates that a “`*`” denotes a terminal node of the tree (i.e., a leaf node—the tree is not split any further at that node).

17 Summary of the Model

```

summary(model)

## Call:
## rpart(formula=form, data=ds[train, vars])
## n= 256
##
##      CP nsplit rel error xerror   xstd
## 1 0.21053     0     1.0000 1.0000 0.1497
## 2 0.07895     1     0.7895 0.9474 0.1464
## 3 0.01000     3     0.6316 1.0263 0.1513
##
## Variable importance
## Humidity3pm    Sunshine Pressure3pm      Temp9am Pressure9am      Temp3pm
##          25           17           14           9           8           8
## Cloud3pm      MaxTemp      MinTemp
##          7            6            5
##
## Node number 1: 256 observations,    complexity param=0.2105
## predicted class=No    expected loss=0.1484  P(node) =1
##   class counts:  218    38
##   probabilities: 0.852 0.148
## left son=2 (238 obs) right son=3 (18 obs)
## Primary splits:
##   Humidity3pm < 71    to the left,  improve=12.750, (0 missing)
##   Pressure3pm < 1011   to the right, improve=11.240, (0 missing)
##   Cloud3pm    < 6.5    to the left,  improve=11.010, (0 missing)
##   Sunshine     < 6.45   to the right, improve= 9.975, (2 missing)
##   Pressure9am < 1018   to the right, improve= 8.381, (0 missing)
## Surrogate splits:
##   Sunshine     < 0.75   to the right, agree=0.949, adj=0.278, (0 split)
##   Pressure3pm < 1002   to the right, agree=0.938, adj=0.111, (0 split)
##   Temp3pm     < 7.6    to the right, agree=0.938, adj=0.111, (0 split)
##   Pressure9am < 1005   to the right, agree=0.934, adj=0.056, (0 split)
##
## Node number 2: 238 observations,    complexity param=0.07895
## predicted class=No    expected loss=0.105  P(node) =0.9297
##   class counts:  213    25
##   probabilities: 0.895 0.105
## left son=4 (208 obs) right son=5 (30 obs)
## Primary splits:
##   Pressure3pm < 1010   to the right, improve=5.973, (0 missing)
##   Cloud3pm    < 6.5    to the left,  improve=4.475, (0 missing)
...

```

In the following pages we dissect the various components of this summary.

18 Complexity Parameter

We can print a table of optimal prunings based on a complexity parameter using `printcp()`. The data is actually stored as `model$cptable`.

```
printcp(model)

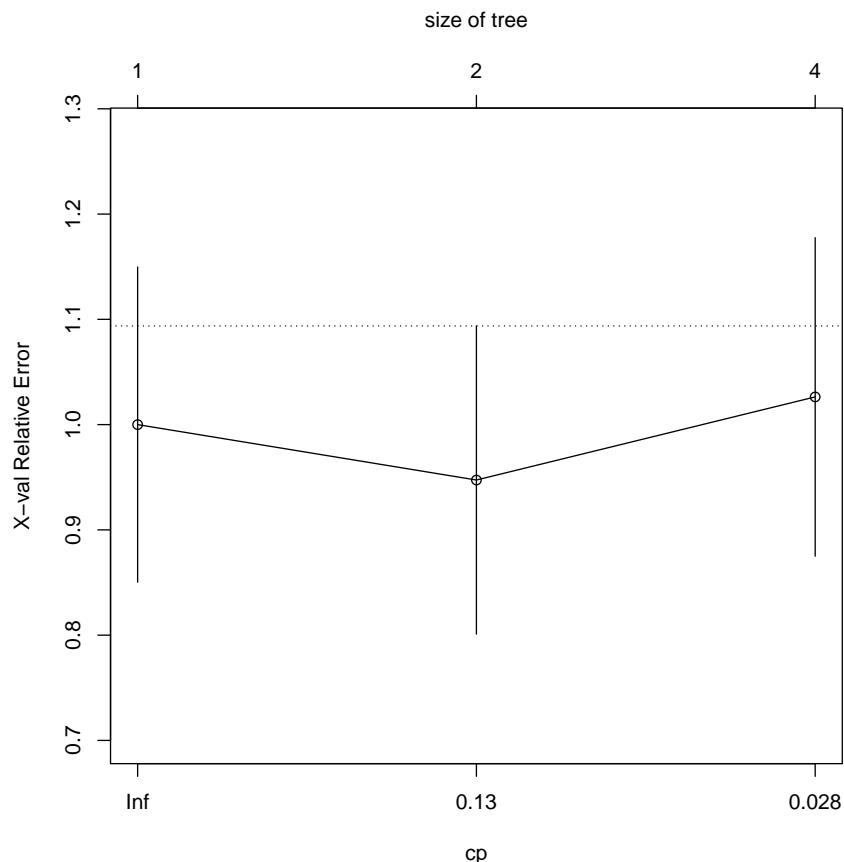
##
## Classification tree:
## rpart(formula=form, data=ds[train, vars])
##
## Variables actually used in tree construction:
## [1] Humidity3pm Pressure3pm Sunshine
##
## Root node error: 38/256=0.15
##
## n= 256
##
##      CP nsplit rel error xerror xstd
## 1 0.211      0     1.00   1.00 0.15
## 2 0.079      1     0.79   0.95 0.15
## 3 0.010      3     0.63   1.03 0.15
```

Exercise: Research what the complexity parameter does. Explain/illustrate it in one or two paragraphs.

19 Complexity Parameter Plot

The `plotcp()` plots the cross-validation results. Here we see a set of possible cost-complexity prunings of the tree. We might choose to prune using the leftmost complexity parameter which has a mean below the horizontal line.

```
plotcp(model)
```

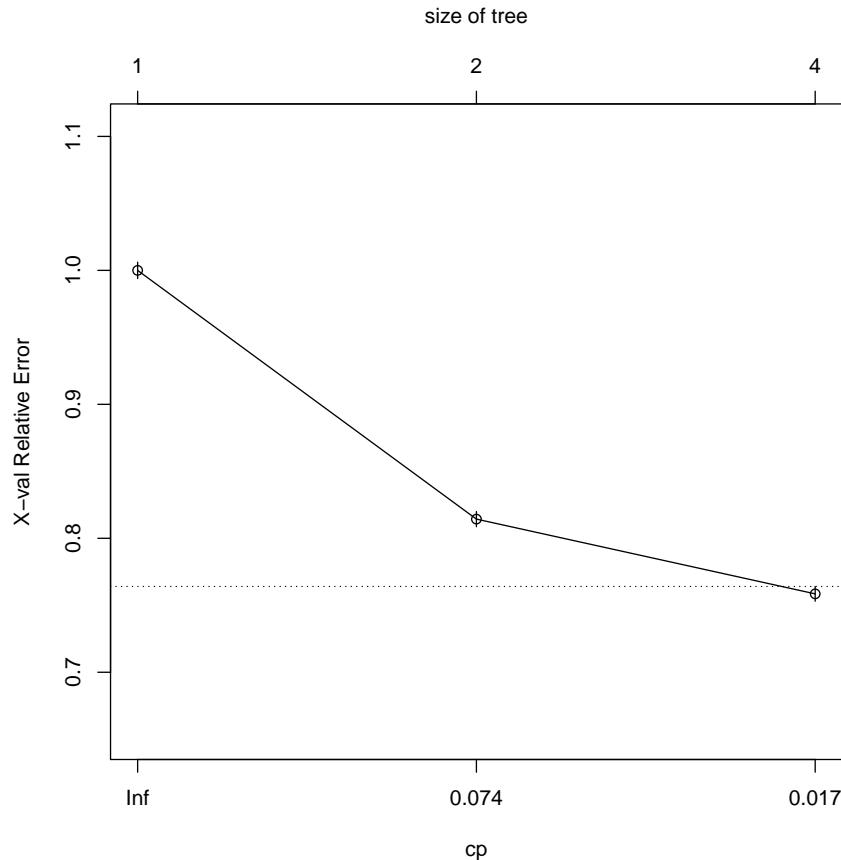


20 More Interesting Complexity Behaviour

We don't see much of the true behaviour of the complexity parameter with our small dataset. We can build a more interesting model based on the larger **weatherAUS** dataset from **rattle**.

If we build a default model then we can plot the complexity parameter as before.

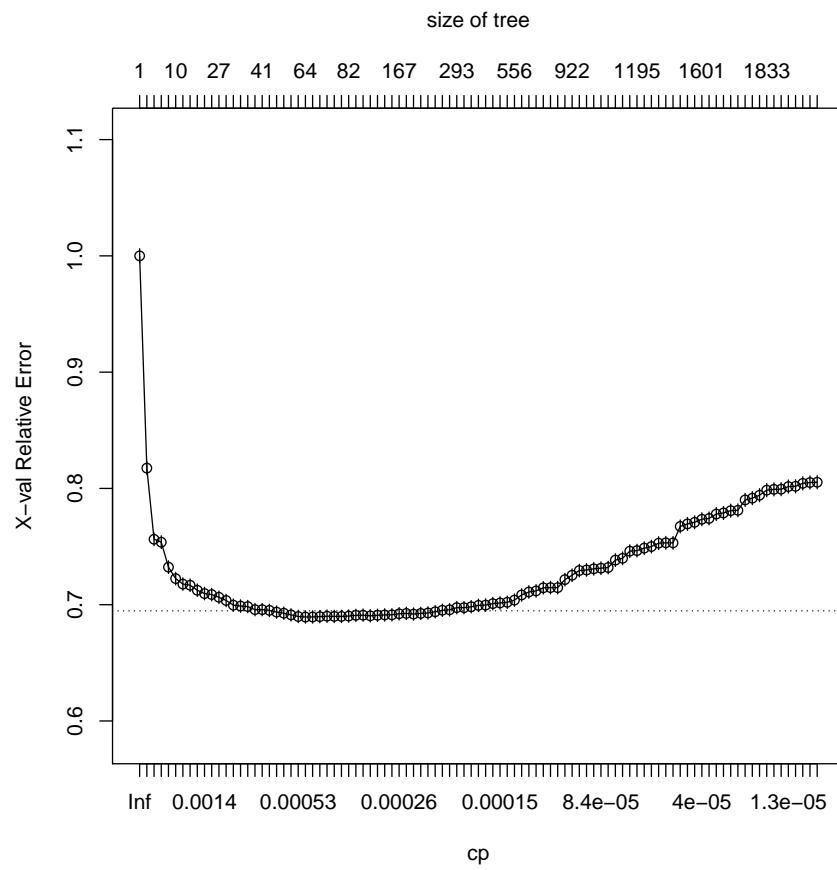
```
tmodel <- rpart(form, weatherAUS[vars])
plotcp(tmodel)
```



21 More Interesting Complexity Behaviour—cp=0

We can set the `cp=` argument to be 0, so that no pruning of the tree is performed.

```
tmodel <- rpart(form, weatherAUS[vars], control=rpart.control(cp=0))
plotcp(tmodel)
```



Notice that as we continue to build the model, by recursive partitioning, the model gets more complex but the performance does not improve, and in fact over time the model performance starts to deteriorate because of [overfitting](#).

22 More Interesting Complexity Behaviour—Numeric View

We can look at the raw data to have a more precise and detailed view of the data. Here we only list specific rows from the complexity parameter table.

```
tmodel$cptable[c(1:5, 22:29, 80:83),]

##      CP nsplit rel error xerror      xstd
## 1 1.890e-01     0 1.0000 1.0000 0.006125
## 2 2.909e-02     1 0.8110 0.8175 0.005685
## 3 7.486e-03     3 0.7528 0.7563 0.005515
## 4 6.440e-03     4 0.7453 0.7538 0.005508
## 5 4.277e-03     8 0.7196 0.7323 0.005445
## 22 5.347e-04    57 0.6611 0.6912 0.005320
## 23 5.225e-04    59 0.6600 0.6898 0.005315
## 24 4.861e-04    63 0.6579 0.6894 0.005314
## 25 4.375e-04    67 0.6560 0.6894 0.005314
## 26 3.889e-04    69 0.6551 0.6897 0.005315
## 27 3.727e-04    72 0.6540 0.6901 0.005316
## 28 3.646e-04    75 0.6528 0.6899 0.005316
## 29 3.403e-04    77 0.6521 0.6899 0.005316
## 80 3.646e-05   1607 0.4535 0.7742 0.005566
## 81 3.472e-05   1640 0.4522 0.7780 0.005577
## 82 3.240e-05   1658 0.4516 0.7789 0.005580
## 83 3.093e-05   1679 0.4509 0.7809 0.005585
```

See how the relative error continues to decrease as the tree becomes more complex, but the cross validated error decreases and then starts to increase! We might choose a sensible value of `cp=` from this table.

Exercise: Choose some different values of `cp=` based on the table above and explore the effect. Explain what initially appears to be an oddity about the different looking trees we get.

23 Variable Importance

Exercise: Research how the variable importance is calculated. Explain it in one or two paragraphs.

```
model$variable.importance  
## Humidity3pm      Sunshine Pressure3pm      Temp9am Pressure9am      Temp3pm  
##    13.1468        9.2091     7.3894      4.6458      4.2920      4.2504  
## Cloud3pm        MaxTemp     MinTemp      Rainfall  
##    3.6436        3.0330     2.8339      0.1991  
....
```

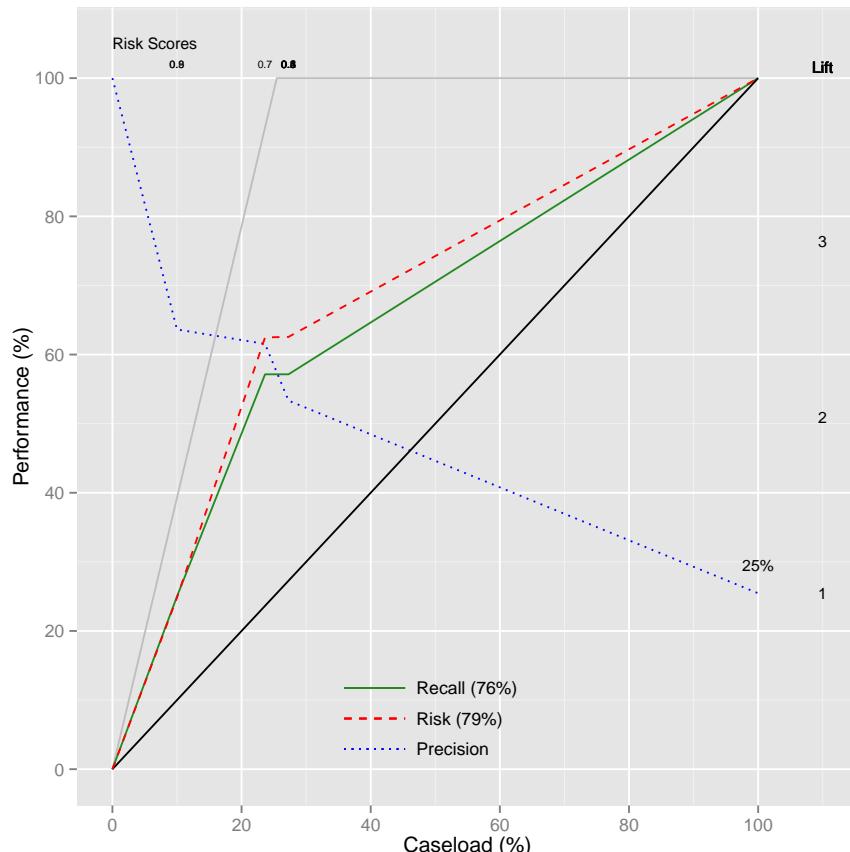
24 Node Details and Surrogates

Exercise: In your own words explain how to read the node information. Research the concept of surrogates to handle missing values ad in one or two paragraphs, explain it.

25 Decision Tree Performance

Here we plot the performance of the decision tree, showing a risk chart. The areas under the recall and risk curves are also reported.

```
predicted <- predict(model, ds[test, vars], type="prob") [,2]
riskchart(predicted, actual, risks)
```



An error matrix shows, clockwise from the top left, the percentages of true negatives, false positives, true positives, and false negatives.

```
predicted <- predict(model, ds[test, vars], type="class")
sum(actual != predicted)/length(predicted) # Overall error rate
## [1] 0.2

round(100*table(actual, predicted, dnn=c("Actual", "Predicted"))/length(predicted))
##      Predicted
## Actual No Yes
##   No  65   9
##   Yes 11  15
....
```

26 Visualise Decision Tree as Rules

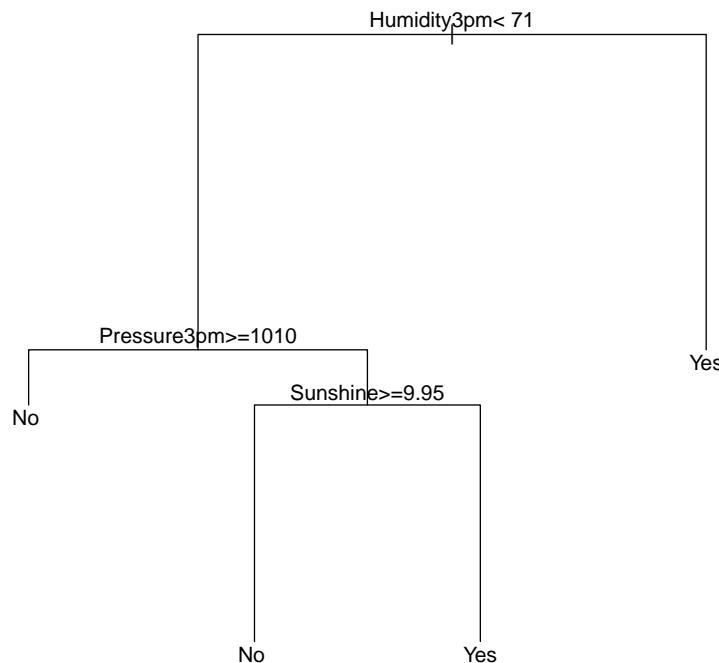
We can use the following function to print the paths through the decision tree as rules.

```
asRules.rpart <- function(model)
{
  if (!inherits(model, "rpart")) stop("Not a legitimate rpart tree")
  #
  # Get some information.
  #
  frm      <- model$frame
  names    <- row.names(frm)
  ylevels <- attr(model, "ylevels")
  ds.size <- model$frame[1,]$n
  #
  # Print each leaf node as a rule.
  #
  for (i in 1:nrow(frm))
  {
    if (frm[i,1] == "<leaf>")
    {
      # The following [,5] is hardwired - needs work!
      cat("\n")
      cat(sprintf(" Rule number: %s ", names[i]))
      cat(sprintf("[yval=%s cover=%d (%.0f%%) prob=%0.2f]\n",
                  ylevels[frm[i,]$yval], frm[i,]$n,
                  round(100*frm[i,]$n/ds.size), frm[i,]$yval2[,5]))
      pth <- path.rpart(model, nodes=as.numeric(names[i]), print.it=FALSE)
      cat(sprintf("  %s\n", unlist(pth)[-1]), sep="")
    }
  }
}

asRules(model)

##
##  Rule number: 4 [yval=No cover=208 (81%) prob=0.06]
##    Humidity3pm< 71
##    Pressure3pm>=1010
##
##  Rule number: 10 [yval=No cover=14 (5%) prob=0.07]
##    Humidity3pm< 71
##    Pressure3pm< 1010
##    Sunshine>=9.95
##
##  Rule number: 11 [yval=Yes cover=16 (6%) prob=0.69]
##    Humidity3pm< 71
##    Pressure3pm< 1010
##    Sunshine< 9.95
....
```

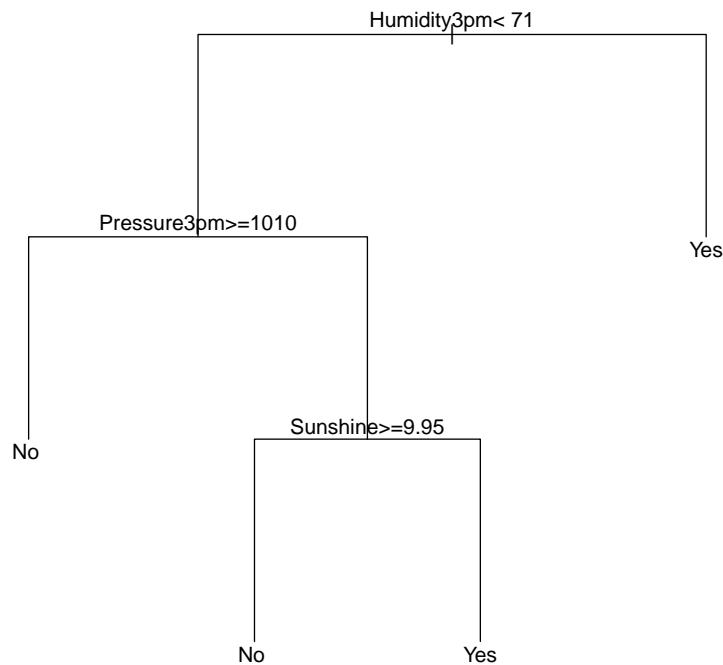
27 Visualise Decision Trees



```
plot(model)
text(model)
```

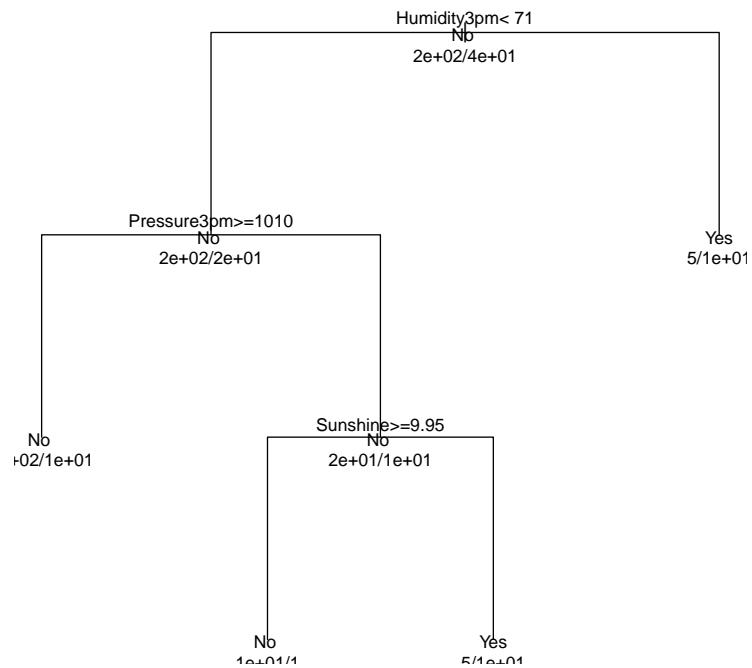
The default plot of the model is quite basic. In this plot we move to the left in the binary tree if the condition is true.

28 Visualise Decision Trees: Uniform



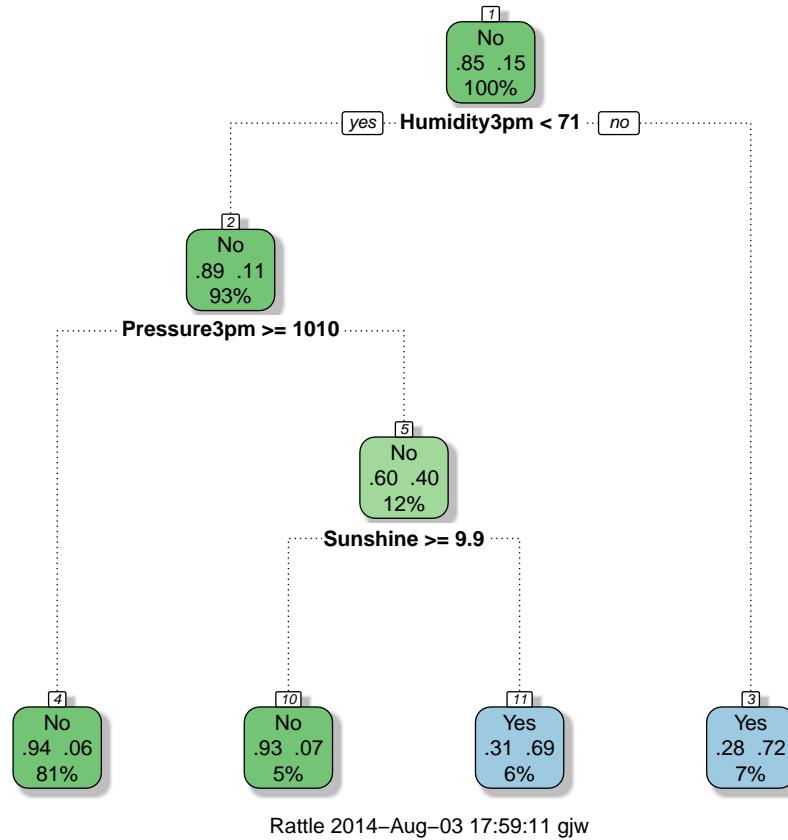
```
plot(model, uniform=TRUE)
text(model)
```

29 Visualise Decision Trees: Extra Information



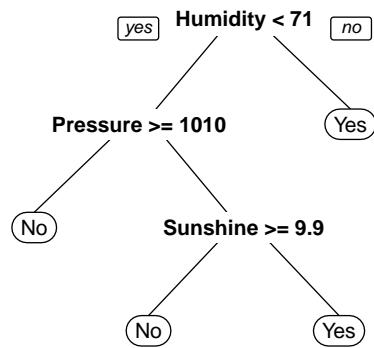
```
plot(model, uniform=TRUE)
text(model, use.n=TRUE, all=TRUE, cex=.8)
```

30 Fancy Plot



The `rattle` package provides a fancy plot based on the functionality provided by `rpart.plot` (Milborrow, 2014) and using colours from `RColorBrewer` (Neuwirth, 2011), tuned for use in `rattle`. The same options can be passed directly to `prp()` to achieve the same plot and colours, as we see in the following pages. The colours are specially constructed in `rattle`.

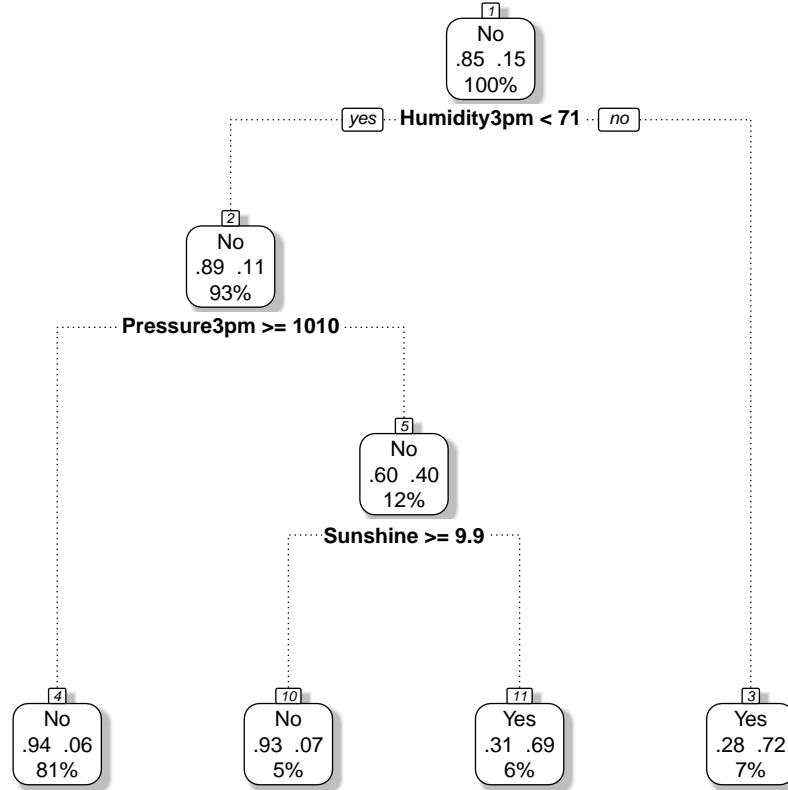
31 Enhanced Plots: Default



```
prp(model)
```

Stephen Milborrow's `rpart.plot` provides a suite of enhancements to the basic `rpart plot()` command. The following pages exhibit the various (and quite extensive) options provided by `rpart.plot` and specifically `prp()`.

32 Enhanced Plots: Favourite



```
prp(model, type=2, extra=104, nn=TRUE, fallen.leaves=TRUE,
  faclen=0, varlen=0, shadow.col="grey", branch.lty=3)
```

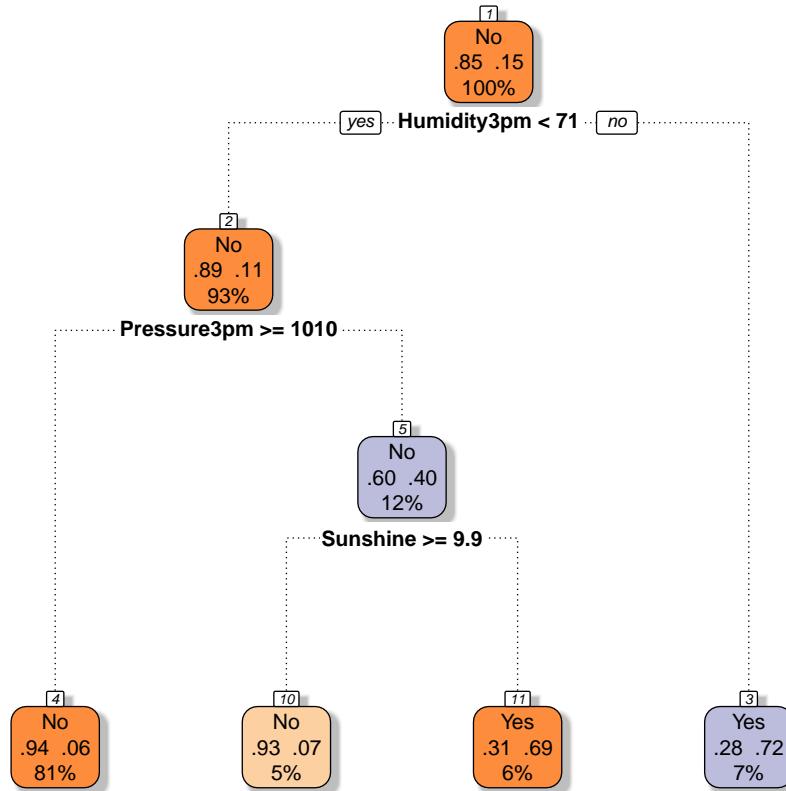
This is a plot that I find particularly useful, neat, and informative, particularly for classification models.

The leaf nodes are each labelled with the predicted class. They are neatly lined up at the bottom of the figure (`fallen.leaves=TRUE`), to visually reinforce the structure. We can see the straight lines from the top to the bottom which lead to decisions quickly, whilst the more complex paths need quite a bit more information in order to make a decision.

Each node includes the probability for each class, and the percentage of observations associated with the node (`extra=104`). The node numbers are included (`nn=TRUE`) so we can cross reference each node to the text decision tree, or other decision tree plots, or a rule set generated from the decision tree.

Using a dotted line type (`branch.lty=3`) removes some of the focus from the heavy lines and back to the nodes, whilst still clearly identifying the links. The grey shadow is an optional nicety.

33 Enhanced Plot: With Colour

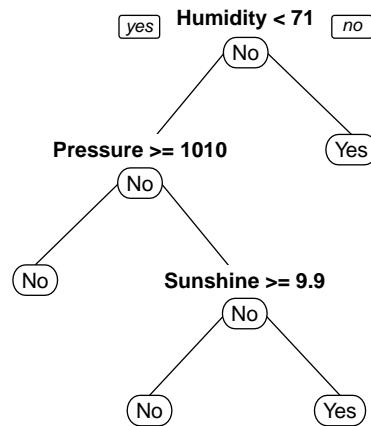


```

col <- c("#FD8D3C", "#FD8D3C", "#FD8D3C", "#BCBDDC",
        "#FDD0A2", "#FD8D3C", "#BCBDDC")
prp(model, type=2, extra=104, nn=TRUE, fallen.leaves=TRUE,
     faclen=0, varlen=0, shadow.col="grey", branch.lty=3, box.col=col)
  
```

The `fancyRpartPlot()` function from `rattle` (Williams, 2014) generates scaled colour for colouring the boxes depending on the decision and the strength. The hard work is in generating the scaled colours for the nodes. Here we use other palettes for the colours rather than those used by `fancyRpartPlot()`.

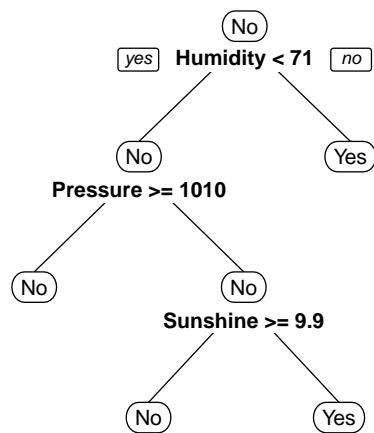
34 Enhanced Plots: Label all Nodes



```
prp(model, type=1)
```

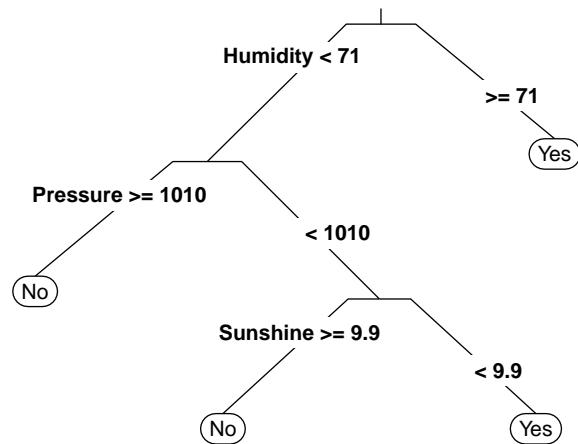
Here all nodes are labelled with the majority class.

35 Enhanced Plots: Labels Below Nodes



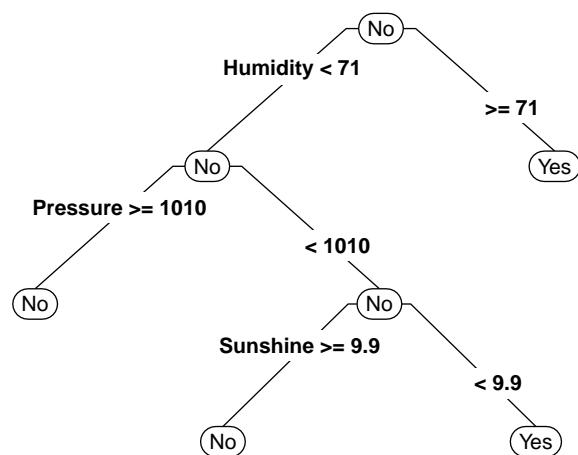
```
prp(model, type=2)
```

36 Enhanced Plots: Split Labels



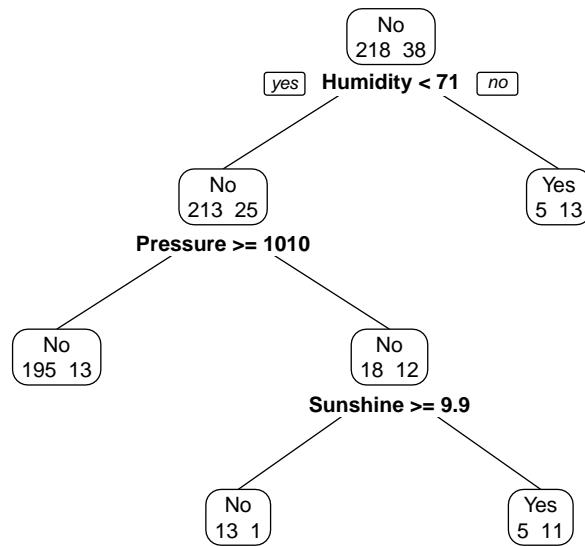
```
prp(model, type=3)
```

37 Enhanced Plots: Interior Labels



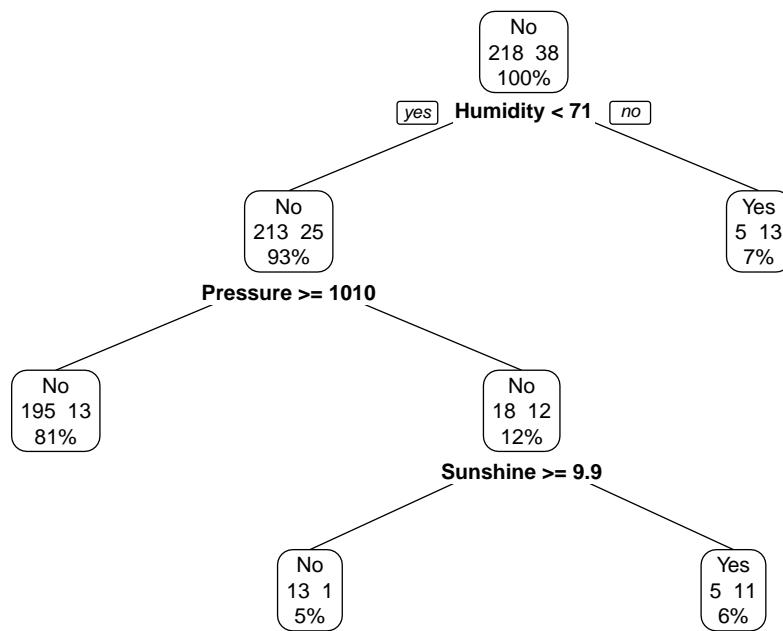
```
prp(model, type=4)
```

38 Enhanced Plots: Number of Observations



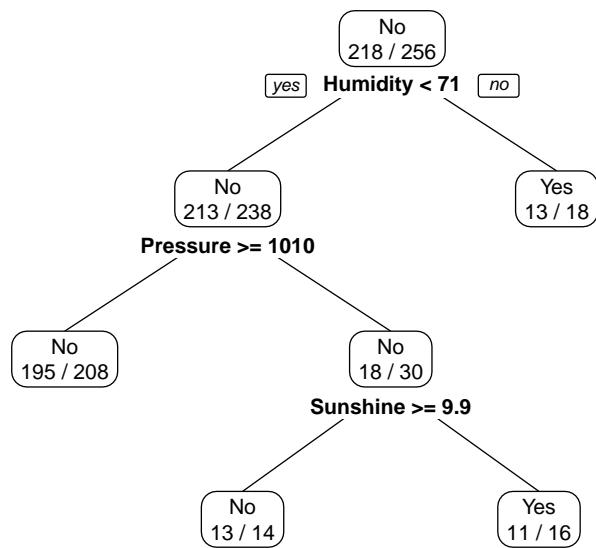
```
prp(model, type=2, extra=1)
```

39 Enhanced Plots: Add Percentage of Observations



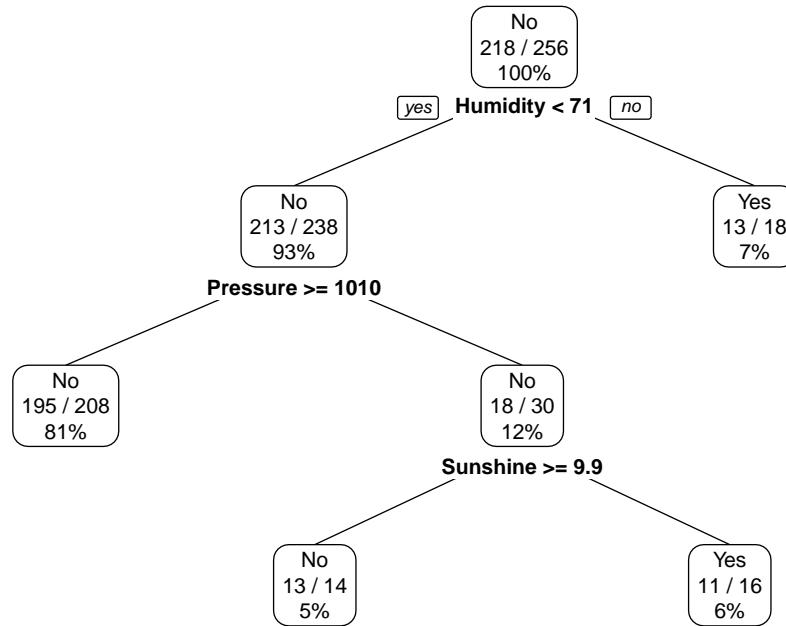
```
prp(model, type=2, extra=101)
```

40 Enhanced Plots: Classification Rate



```
prp(model, type=2, extra=2)
```

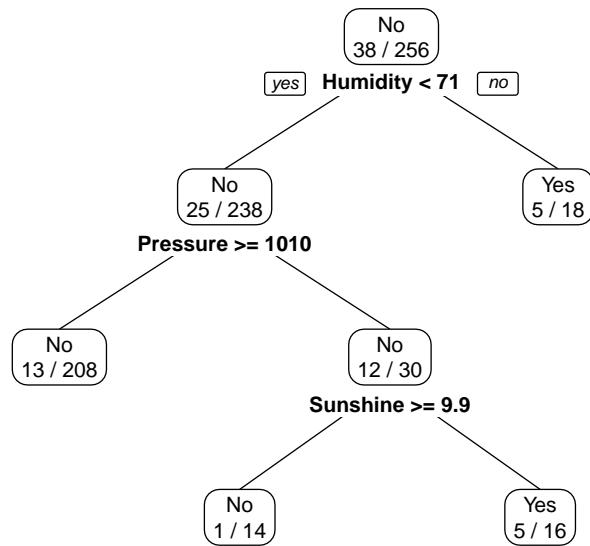
41 Enhanced Plots: Add Percentage of Observations



```
prp(model, type=2, extra=102)
```

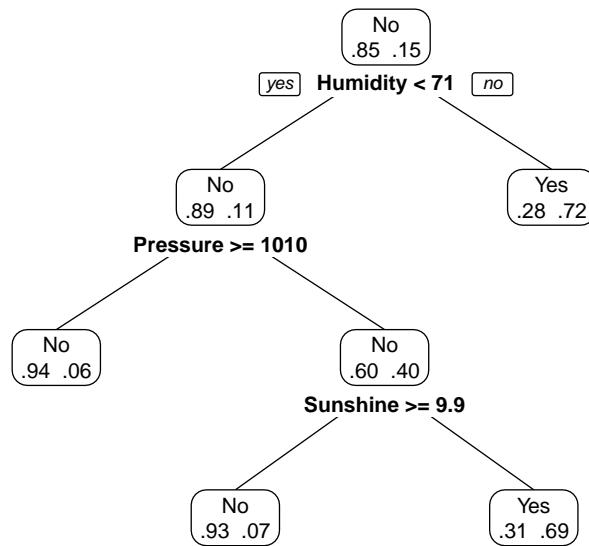
Notice the pattern? When we add 100 to the `extra=` option then the percentage of observations located with each node is then included in the plot.

42 Enhanced Plots: Misclassification Rate



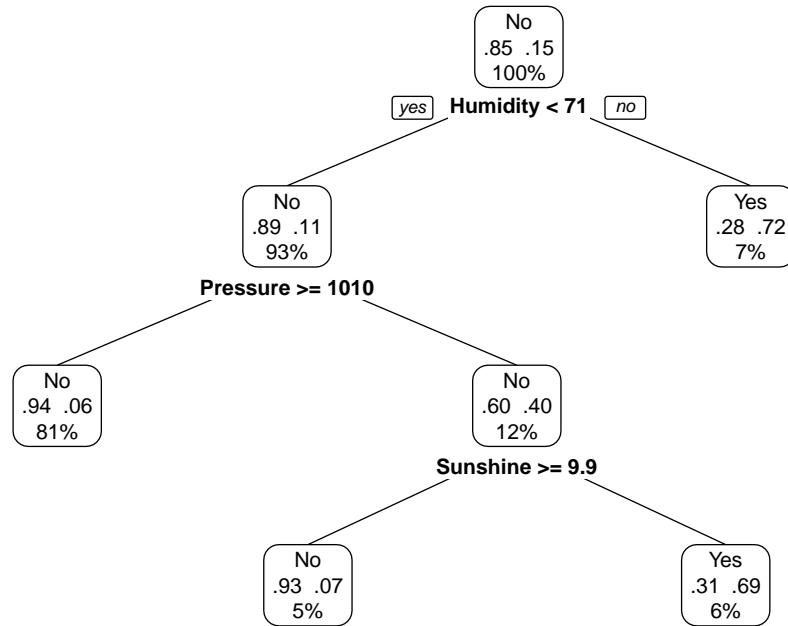
```
prp(model, type=2, extra=3)
```

43 Enhanced Plots: Probability per Class



```
prp(model, type=2, extra=4)
```

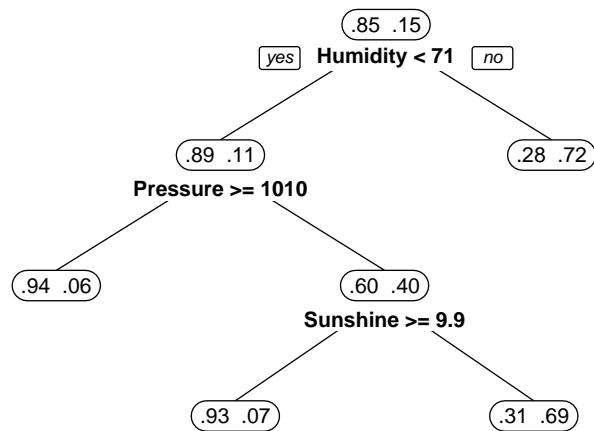
44 Enhanced Plots: Add Percentage Observations



```
prp(model, type=2, extra=104)
```

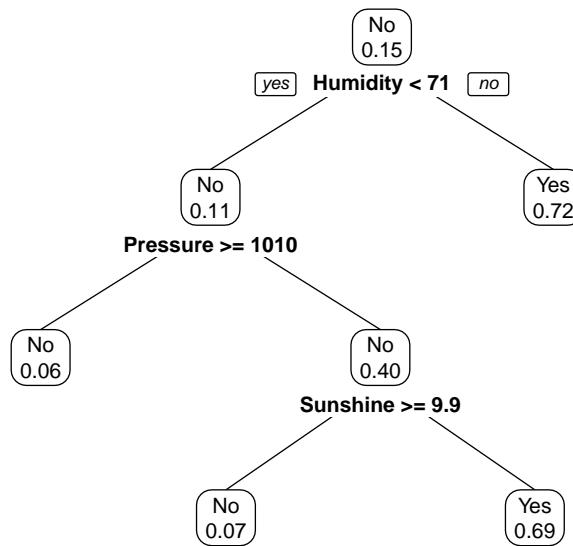
This is a particularly informative plot for classification models. Each node includes the probability of each class. Each node also includes the percentage of the training dataset that corresponds to that node.

45 Enhanced Plots: Only Probability Per Class



```
prp(model, type=2, extra=5)
```

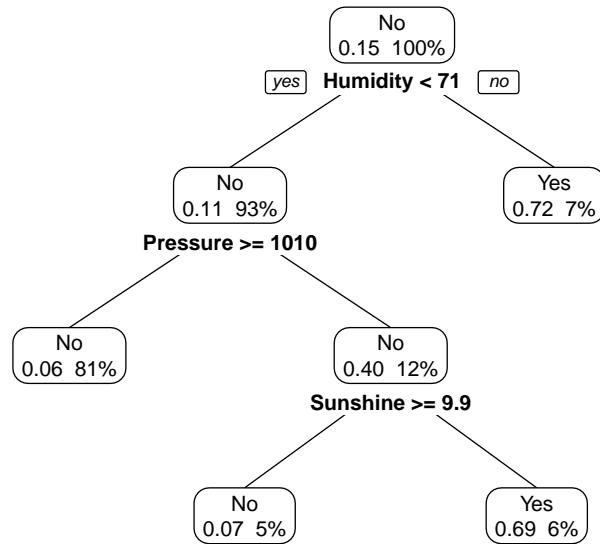
46 Enhanced Plots: Probability of Second Class



```
prp(model, type=2, extra=6)
```

This is particularly useful for binary classification, as here, where the second class is usually the positive response.

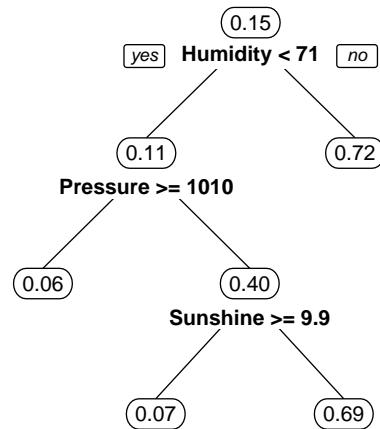
47 Enhanced Plots: Add Percentage Observations



```
prp(model, type=2, extra=106)
```

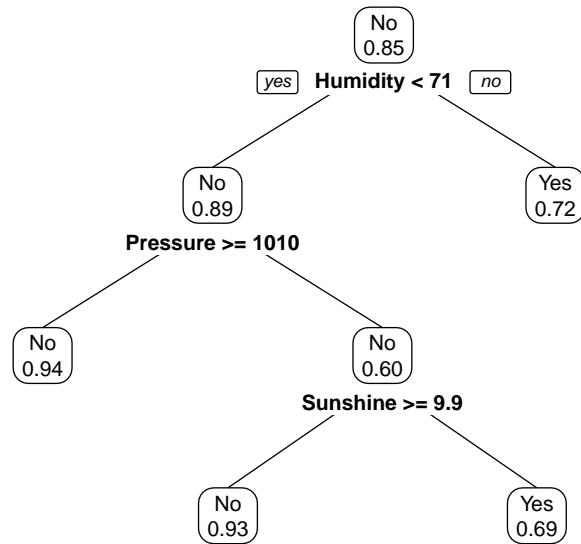
This is a particularly informative plot for binary classification tasks. Each node includes the probability of the second class, which is usually the positive class in a binary classification dataset. Each node also includes the percentage of the training dataset that corresponds to that node.

48 Enhanced Plots: Only Probability of Second Class



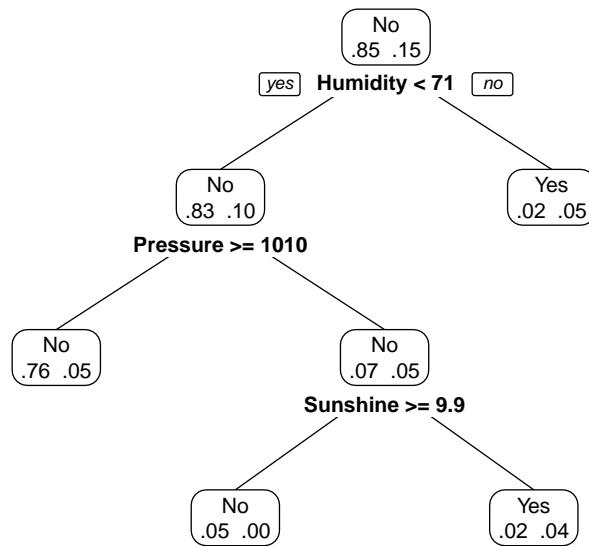
```
prp(model, type=2, extra=7)
```

49 Enhanced Plots: Probability of the Class



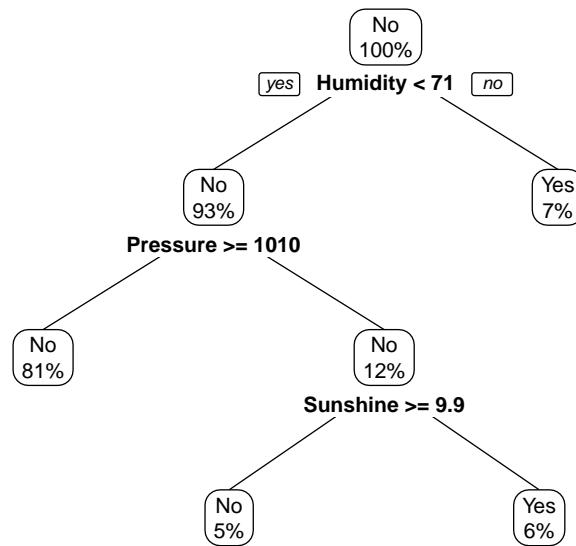
```
prp(model, type=2, extra=8)
```

50 Enhanced Plots: Overall Probability



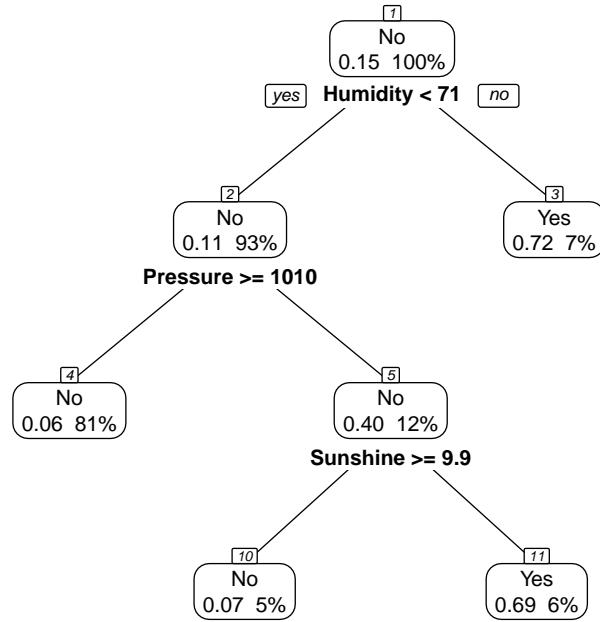
```
prp(model, type=2, extra=9)
```

51 Enhanced Plots: Percentage of Observations



```
prp(model, type=2, extra=100)
```

52 Enhanced Plots: Show the Node Numbers

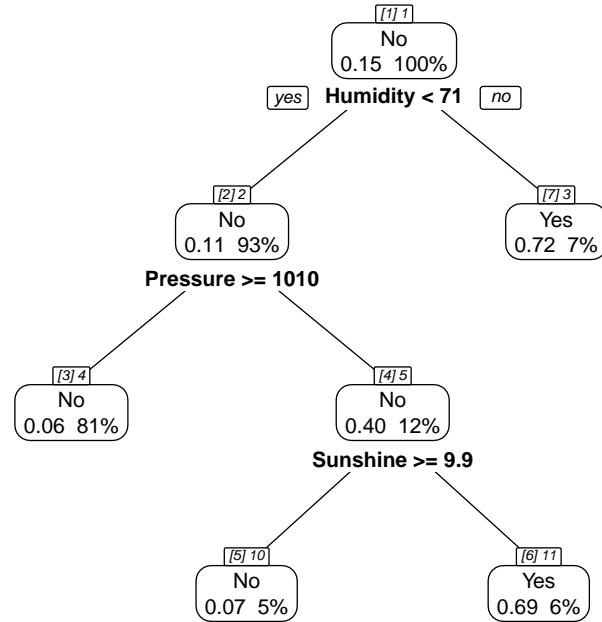


```
prp(model, type=2, extra=106, nn=TRUE)
```

We now take our favourite plot (`type=2` and `extra=106`) and explore some of the other options available for `prp()` from `rpart.plot`.

Here we add the node numbers as they appear in the textual version of the model, and also often for rule sets generated from the decision tree.

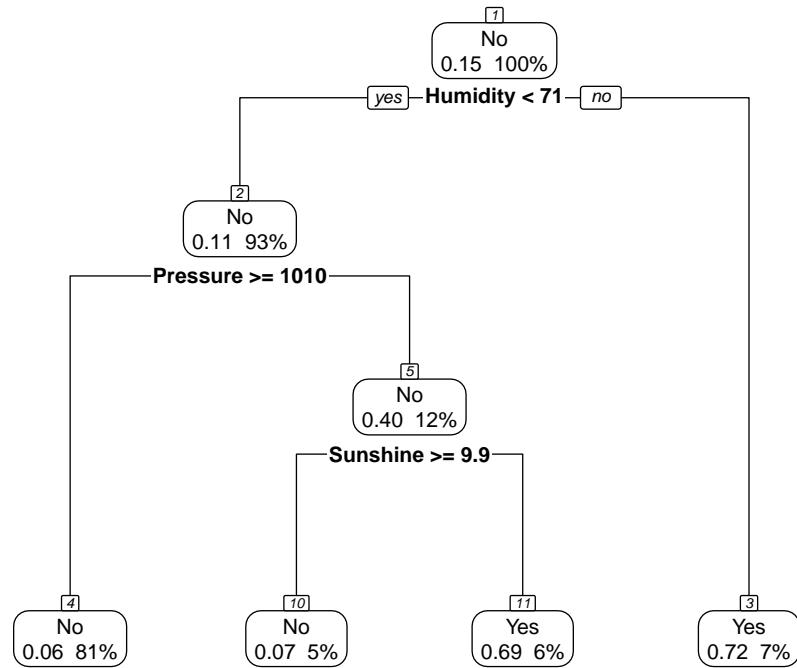
53 Enhanced Plots: Show the Node Indices



```
prp(model, type=2, extra=106, nn=TRUE, ni=TRUE)
```

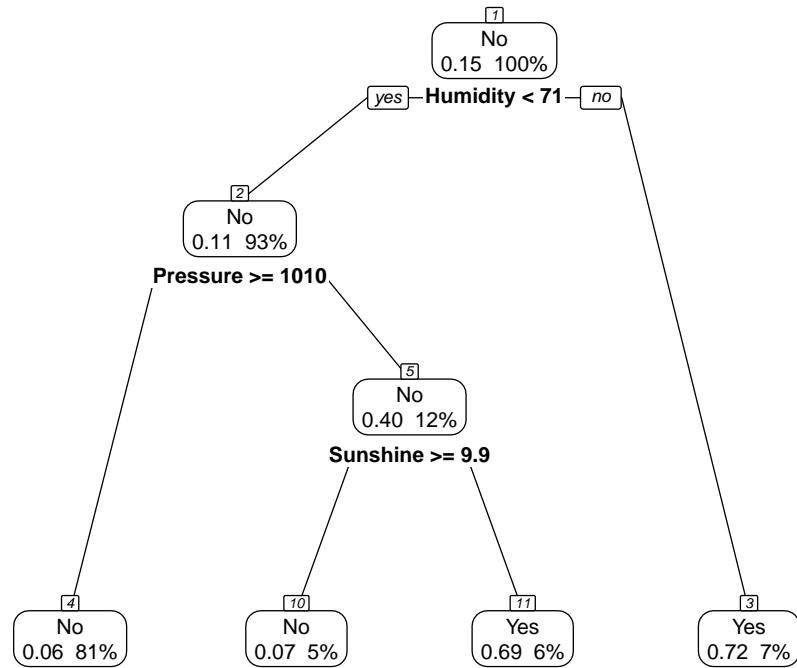
These are the row numbers of the nodes within the model object's `frame` component.

54 Enhanced Plots: Line up the Leaves



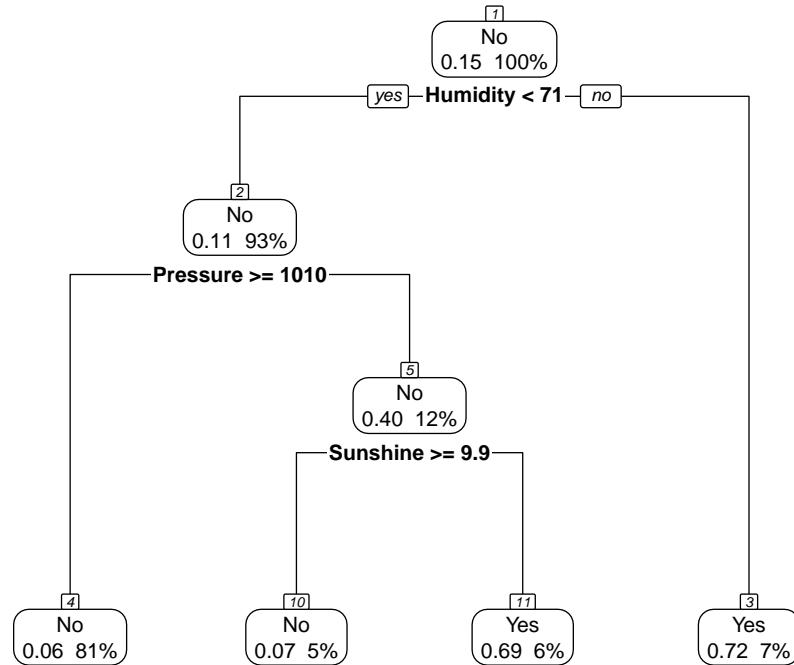
```
prp(model, type=2, extra=106, nn=TRUE, fallen.leaves=TRUE)
```

55 Enhanced Plots: Angle Branch Lines



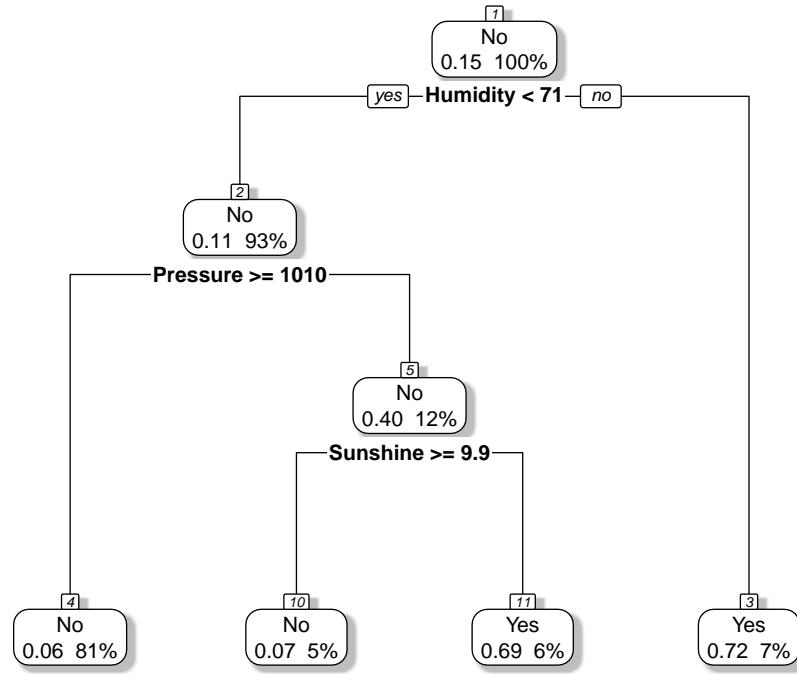
```
prp(model, type=2, extra=106, nn=TRUE, fallen.leaves=TRUE,  
branch=0.5)
```

56 Enhanced Plots: Do Not Abbreviate Factors



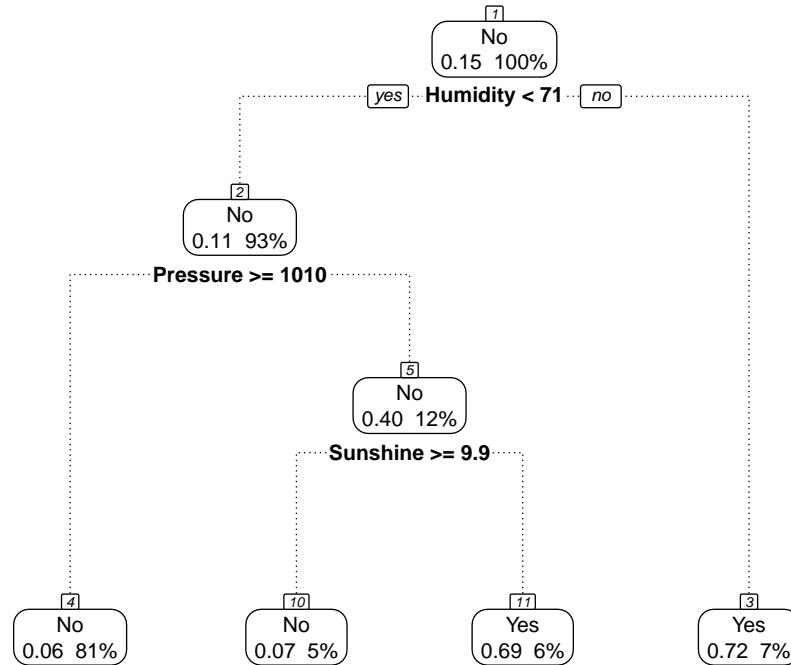
```
prp(model, type=2, extra=106, nn=TRUE, fallen.leaves=TRUE,  
    faclen=0)
```

57 Enhanced Plots: Add a Shadow to the Nodes



```
prp(model, type=2, extra=106, nn=TRUE, fallen.leaves=TRUE,  
    shadow.col="grey")
```

58 Enhanced Plots: Draw Branches as Dotted Lines



```
prp(model, type=2, extra=106, nn=TRUE, fallen.leaves=TRUE,
  branch.lty=3)
```

The `branch.lty=` option allows us to specify the type of line to draw for the branches. A dotted line is attractive as it reduces the dominance of the branches whilst retaining the node connections. Other options are just the standard values for line type in R:

0 blank	1 solid	2 dashed	3 dotted	4 dotdash	5 longdash	6 twodash
		"44"	"13"	"1343"	"73"	"2262"

More appropriate in Plots module.

The line type can also be specified as an even length string of up eight characters of the hex digits (0–9, a–f). The pairs specify the length in pixels of the line and the blank. Thus `lty="44"` is the same as `lty=2`:

```
plot(c(0,1), c(0,0), type="l", axes=FALSE, xlab=NA, ylab=NA, lty=2)
plot(c(0,1), c(0,0), type="l", axes=FALSE, xlab=NA, ylab=NA, lty="dashed")
plot(c(0,1), c(0,0), type="l", axes=FALSE, xlab=NA, ylab=NA, lty="44")
```

Add line example into each cell of table.

59 Enhanced Plots: Other Options

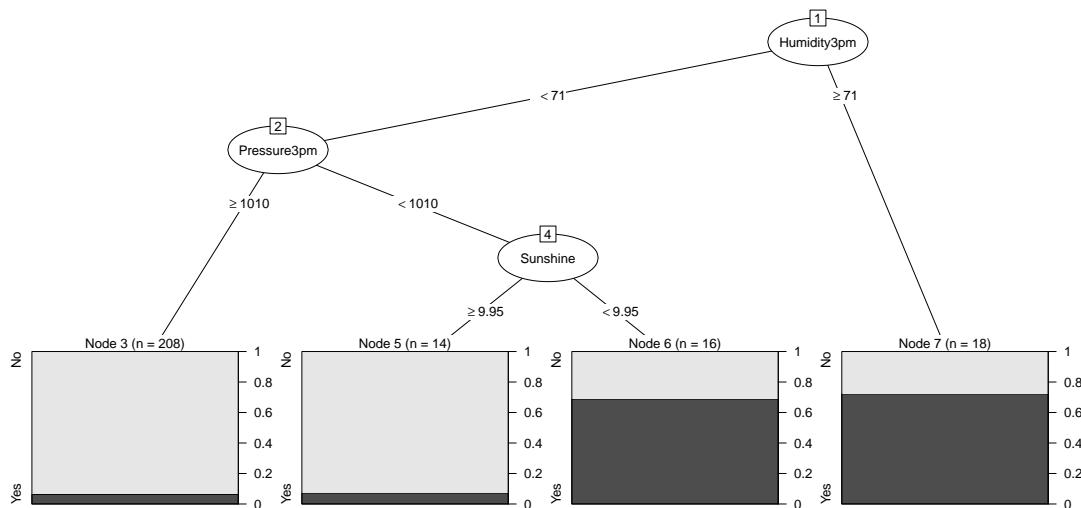
Exercise: Explore examples of split.cex=1.2, split.prefix="is ", split.suffix="?", col=cols, border.col=cols, split.box.col="lightgray", split.border.col="darkgray", split.round=.5, and other parameters you might discover.

60 Party Tree

The `party` (Hothorn *et al.*, 2013) package can be used to draw `rpart` decision trees using `as.party()` from `partykit` (Hothorn and Zeileis, 2014) which can be installed from R-Forge:

```
install.packages("partykit", repos="http://R-Forge.R-project.org")
library(partykit)

class(model)
## [1] "rpart"
plot(as.party(model))
```



The textual presentation of an `rpart` decision tree can also be improved using `party`.

```
print(as.party(model))

##
## Model formula:
## RainTomorrow ~ MinTemp + MaxTemp + Rainfall + Evaporation + Sunshine +
##           WindGustDir + WindGustSpeed + WindDir9am + WindDir3pm + WindSpeed9am +
##           WindSpeed3pm + Humidity9am + Humidity3pm + Pressure9am +
##           Pressure3pm + Cloud9am + Cloud3pm + Temp9am + Temp3pm + RainToday
##
## Fitted party:
## [1] root
## |   [2] Humidity3pm < 71
## |   |   [3] Pressure3pm >= 1010.25: No (n=208, err=6%)
## |   |   [4] Pressure3pm < 1010.25
## |   |   |   [5] Sunshine >= 9.95: No (n=14, err=7%)
## |   |   |   [6] Sunshine < 9.95: Yes (n=16, err=31%)
## |   |   [7] Humidity3pm >= 71: Yes (n=18, err=28%)
....
```

61 Conditional Decision Tree

Note that we are using the newer version of the ctree() function as is provided by partykit (Hothorn and Zeileis, 2014). One advantage of the newer version is that predict() with type="prob" works just like other predict() methods (returns a matrix rather than a list).

```
library(partykit)
model <- ctree(formula=form, data=ds[train, vars])

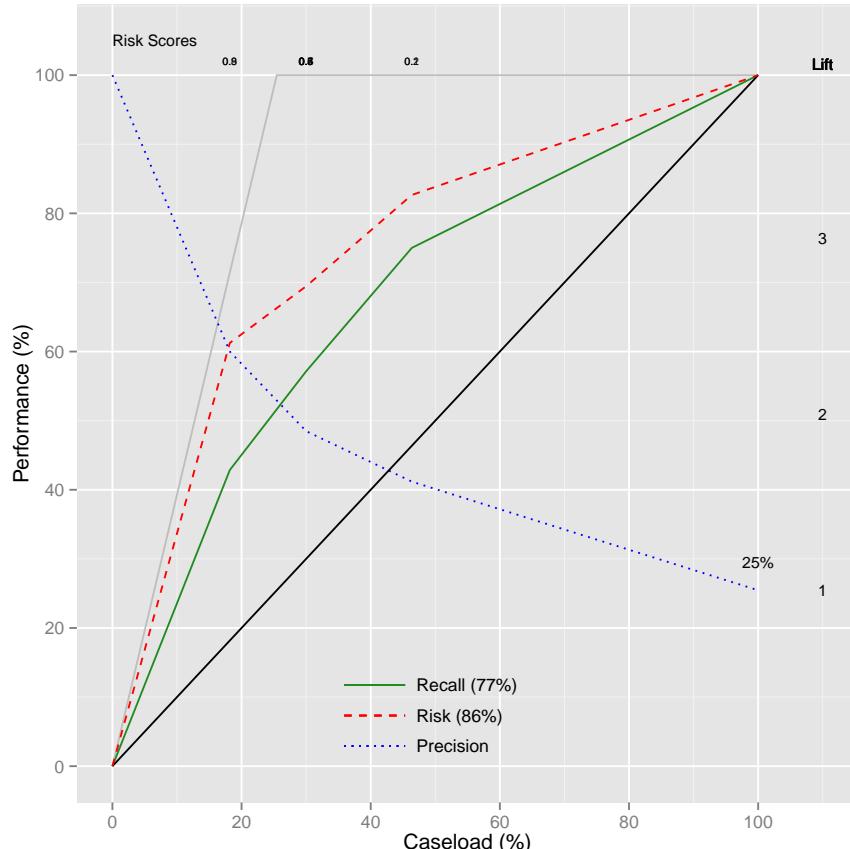
model

##
## Model formula:
## RainTomorrow ~ MinTemp + MaxTemp + Rainfall + Evaporation + Sunshine +
##     WindGustDir + WindGustSpeed + WindDir9am + WindDir3pm + WindSpeed9am +
##     WindSpeed3pm + Humidity9am + Humidity3pm + Pressure9am +
##     Pressure3pm + Cloud9am + Cloud3pm + Temp9am + Temp3pm + RainToday
##
## Fitted party:
## [1] root
## | [2] Sunshine <= 6.4
## | | [3] Pressure3pm <= 1015.9: Yes (n=29, err=24%)
## | | | [4] Pressure3pm > 1015.9: No (n=36, err=8%)
## | | [5] Sunshine > 6.4
## | | | [6] Pressure3pm <= 1010.2: No (n=25, err=28%)
## | | | [7] Pressure3pm > 1010.2: No (n=166, err=4%)
##
## Number of inner nodes: 3
## Number of terminal nodes: 4
```

62 Conditional Decision Tree Performance

Here we plot the performance of the decision tree, showing a risk chart. The areas under the recall and risk curves are also reported.

```
predicted <- predict(model, ds[test, vars], type="prob") [,2]
riskchart(predicted, actual, risks)
```

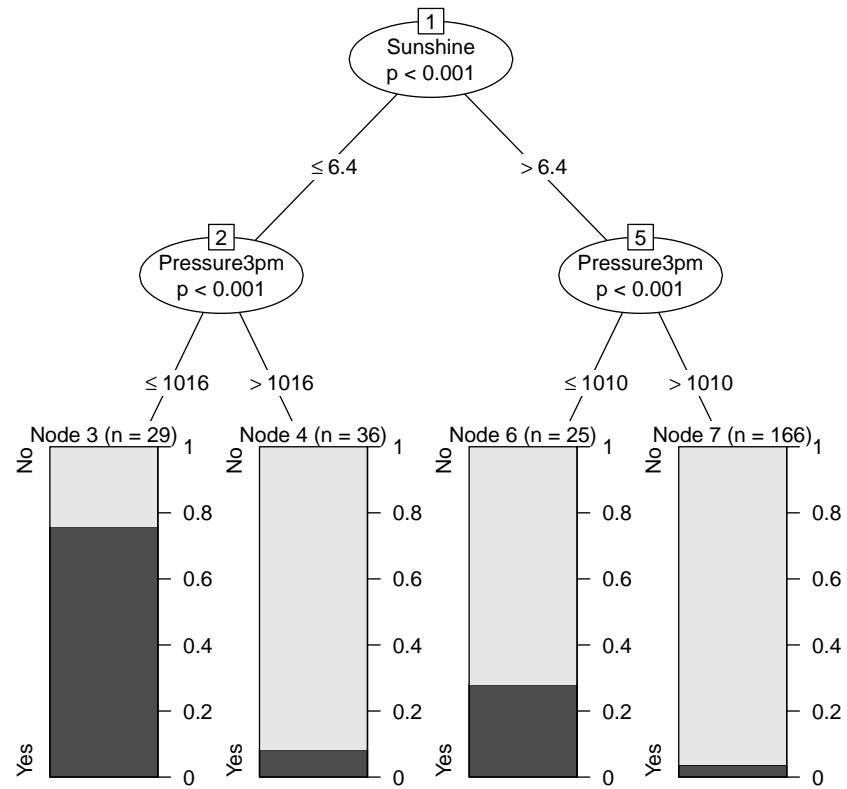


An error matrix shows, clockwise from the top left, the percentages of true negatives, false positives, true positives, and false negatives.

```
predicted <- predict(model, ds[test, vars], type="response")
sum(actual != predicted)/length(predicted) # Overall error rate
## [1] 0.2182
round(100*table(actual, predicted, dnn=c("Actual", "Predicted"))/length(predicted))
##          Predicted
## Actual   No  Yes
##   No    67     7
##   Yes   15    11
....
```

63 CTree Plot

```
plot(model)
```



64 Weka Decision Tree

The suite of algorithms implemented in Weka are also available to R thanks to RWeka (Hornik, 2014).

```
library(RWeka)
model <- J48(formula=form, data=ds[train, vars])

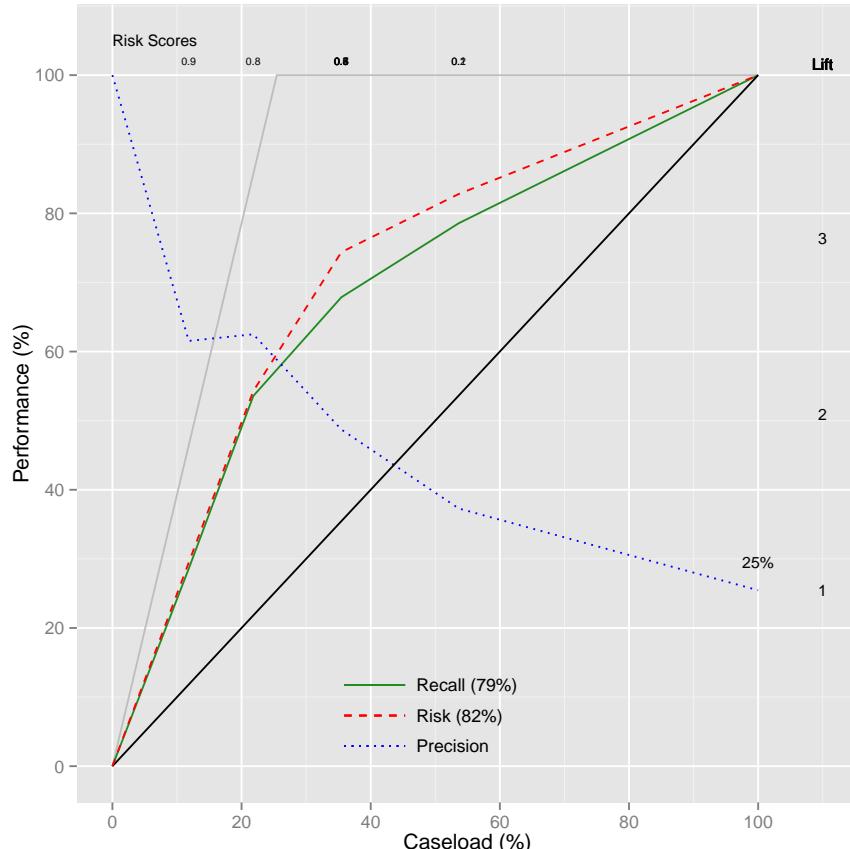
model

## J48 pruned tree
## -----
##
## 
## Humidity3pm <= 70
## |   Pressure3pm <= 1015.8
## |   |   Sunshine <= 9.2
## |   |   |   Evaporation <= 4.4: No (17.0/4.0)
## |   |   |   Evaporation > 4.4: Yes (17.0/2.0)
## |   |   Sunshine > 9.2: No (49.0/3.0)
## |   Pressure3pm > 1015.8: No (132.0/1.0)
## Humidity3pm > 70: Yes (16.0/4.0)
##
## Number of Leaves : 5
##
## Size of the tree : 9
```

65 Weka Decision Tree Performance

Here we plot the performance of the decision tree, showing a risk chart. The areas under the recall and risk curves are also reported.

```
predicted <- predict(model, ds[test, vars], type="prob") [,2]
riskchart(predicted, actual, risks)
```

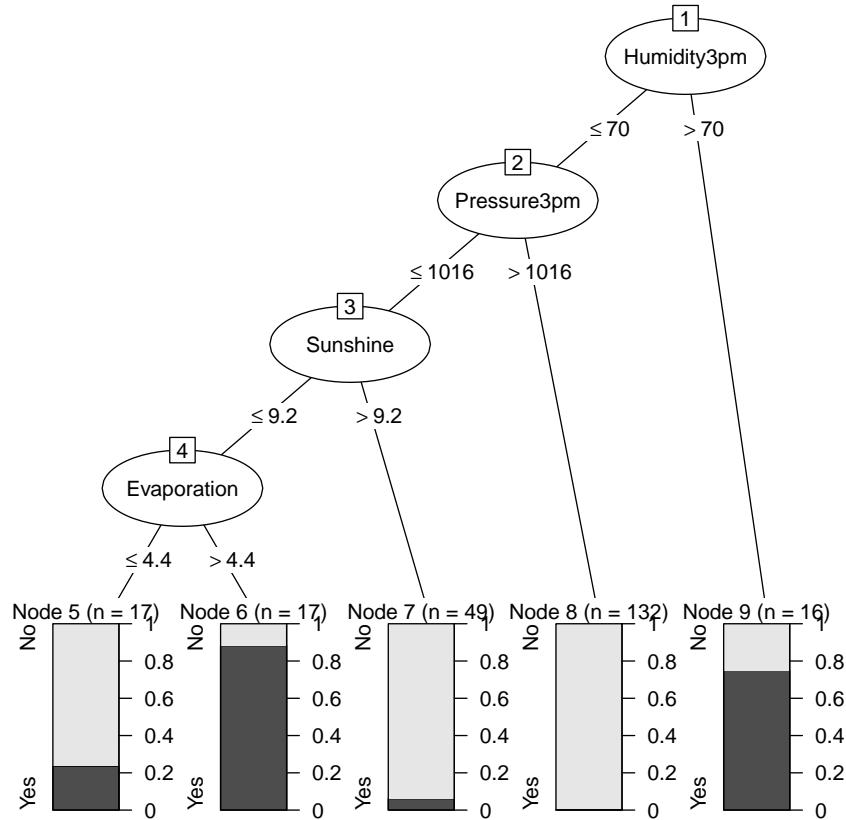


An error matrix shows, clockwise from the top left, the percentages of true negatives, false positives, true positives, and false negatives.

```
predicted <- predict(model, ds[test, vars], type="class")
sum(actual != predicted)/length(predicted) # Overall error rate
## [1] 0.2
round(100*table(actual, predicted, dnn=c("Actual", "Predicted"))/length(predicted))
##      Predicted
## Actual No Yes
##   No  66   8
##   Yes 12  14
....
```

66 Weka Decision Tree Plot Using Party

```
plot(as.party(model))
```



We can also display a textual version using `party`

```
print(as.party(model))

##
## Model formula:
## RainTomorrow ~ MinTemp + MaxTemp + Rainfall + Evaporation + Sunshine +
##           WindGustDir + WindGustSpeed + WindDir9am + WindDir3pm + WindSpeed9am +
##           WindSpeed3pm + Humidity9am + Humidity3pm + Pressure9am +
##           Pressure3pm + Cloud9am + Cloud3pm + Temp9am + Temp3pm + RainToday
##
## Fitted party:
## [1] root
## |   [2] Humidity3pm <= 70
## |   |   [3] Pressure3pm <= 1015.8
## |   |   |   [4] Sunshine <= 9.2
....
```

67 The Original C5.0 Implementation

The C50 ([Kuhn et al., 2014](#)) package interfaces the original C code of the C5.0 implementation by Ross Quinlan, the developer of the decision tree induction algorithm.

```
library(C50)
model <- C5.0(form, ds[train, vars])

model
##
## Call:
## C5.0.formula(formula=form, data=ds[train, vars])
##
## Classification Tree
## Number of samples: 256
## Number of predictors: 20
##
## Tree size: 8
##
## Non-standard options: attempt to group attributes

C5imp(model)

##          Overall
## Humidity3pm    100.00
## Pressure3pm     97.27
## Sunshine       34.77
## Evaporation     15.23
## WindGustSpeed    7.81
## WindDir3pm      7.03
## MinTemp         0.00
## MaxTemp         0.00
## Rainfall        0.00
## WindGustDir     0.00
## WindDir9am      0.00
## WindSpeed9am    0.00
## WindSpeed3pm    0.00
## Humidity9am     0.00
## Pressure9am     0.00
## Cloud9am        0.00
## Cloud3pm        0.00
## Temp9am         0.00
## Temp3pm         0.00
## RainToday        0.00
```

68 C5.0 Summary

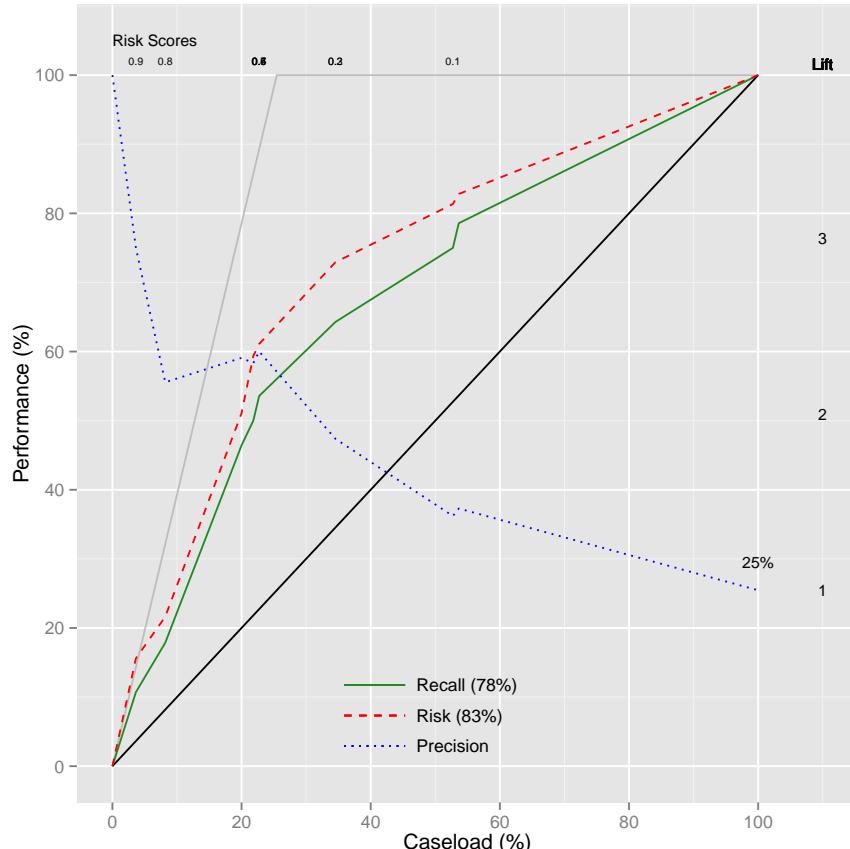
```
summary(model)

##
## Call:
## C5.0.formula(formula=form, data=ds[train, vars])
##
##
## C5.0 [Release 2.07 GPL Edition]   Sun Aug  3 17:59:19 2014
## -----
##
## Class specified by attribute `outcome'
##
## Read 256 cases (21 attributes) from undefined.data
##
## Decision tree:
##
## Humidity3pm > 70:
## ....WindDir3pm in [N-ESE]: Yes (7)
## :   WindDir3pm in [SE-NNW]:
## :     ....Pressure3pm <= 1014.5: Yes (7/1)
## :       Pressure3pm > 1014.5: No (4)
## Humidity3pm <= 70:
## ....Pressure3pm > 1015.8: No (149/3)
##     Pressure3pm <= 1015.8:
##       ....Sunshine > 9.2: No (50/3)
##         Sunshine <= 9.2:
##           ....Evaporation > 4.4: Yes (19/4)
##             Evaporation <= 4.4:
##               ....WindGustSpeed <= 30: Yes (2)
##                 WindGustSpeed > 30: No (18/2)
##
##
## Evaluation on training data (256 cases):
##
##      Decision Tree
## -----
##      Size      Errors
##
##      8      13( 5.1%)    <<
##
##
##      (a)      (b)      <-classified as
....
```

69 C5.0 Decision Tree Performance

Here we plot the performance of the decision tree, showing a risk chart. The areas under the recall and risk curves are also reported.

```
predicted <- predict(model, ds[test, vars], type="prob") [,2]
riskchart(predicted, actual, risks)
```



An error matrix shows, clockwise from the top left, the percentages of true negatives, false positives, true positives, and false negatives.

```
predicted <- predict(model, ds[test, vars], type="class")
sum(actual != predicted)/length(predicted) # Overall error rate
## [1] 0.2182
round(100*table(actual, predicted, dnn=c("Actual", "Predicted"))/length(predicted))
##      Predicted
## Actual No Yes
##   No   65   9
##   Yes  13  13
....
```

70 C5.0 Rules Model

```
library(C50)
model <- C5.0(form, ds[train, vars], rules=TRUE)

model
##
## Call:
## C5.0.formula(formula=form, data=ds[train, vars], rules=TRUE)
##
## Rule-Based Model
## Number of samples: 256
## Number of predictors: 20
##
## Number of Rules: 7
##
## Non-standard options: attempt to group attributes

C5imp(model)

##          Overall
## Humidity3pm    75.00
## Pressure3pm   74.61
## Sunshine      51.17
## WindDir3pm    44.53
## Evaporation   41.80
## WindGustSpeed 32.42
## MinTemp        0.00
## MaxTemp        0.00
## Rainfall       0.00
## WindGustDir   0.00
## WindDir9am    0.00
## WindSpeed9am  0.00
## WindSpeed3pm  0.00
## Humidity9am   0.00
## Pressure9am   0.00
## Cloud9am       0.00
## Cloud3pm       0.00
## Temp9am        0.00
## Temp3pm        0.00
## RainToday      0.00
```

71 C5.0 Rules Summary

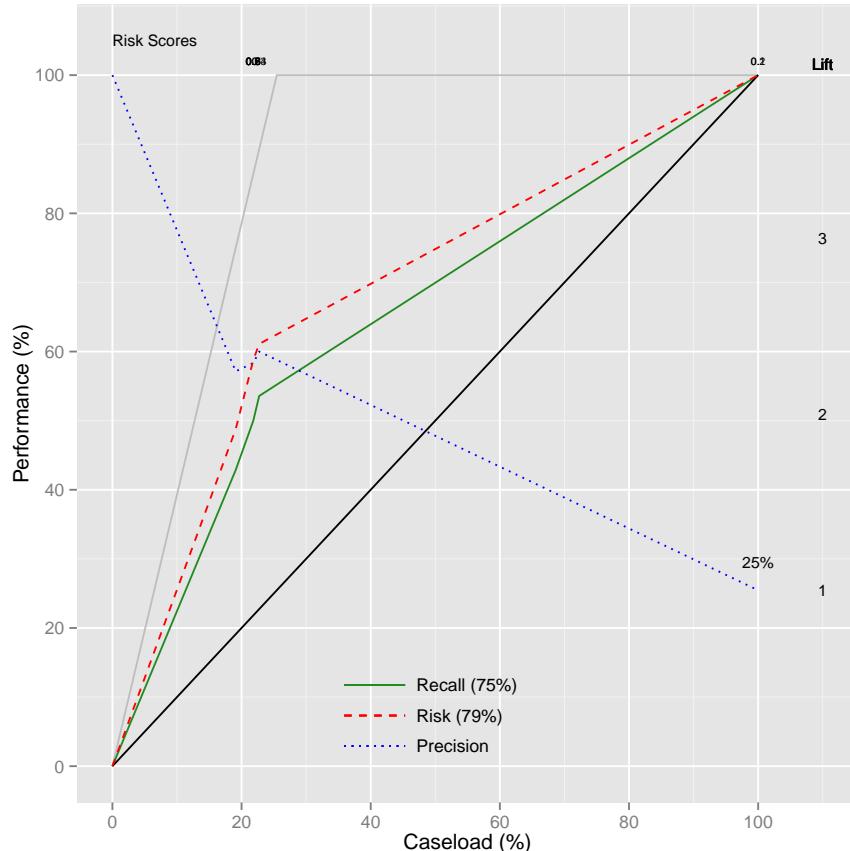
```
summary(model)

##
## Call:
## C5.0.formula(formula=form, data=ds[train, vars], rules=TRUE)
##
##
## C5.0 [Release 2.07 GPL Edition]   Sun Aug  3 17:59:20 2014
## -----
##
## Class specified by attribute `outcome'
##
## Read 256 cases (21 attributes) from undefined.data
##
## Rules:
##
## Rule 1: (149/3, lift 1.1)
## Humidity3pm <= 70
## Pressure3pm > 1015.8
## -> class No  [0.974]
##
## Rule 2: (107/3, lift 1.1)
## Sunshine > 9.2
## -> class No  [0.963]
##
## Rule 3: (107/3, lift 1.1)
## WindDir3pm in [SE-NNW]
## Pressure3pm > 1014.5
## -> class No  [0.963]
##
## Rule 4: (83/3, lift 1.1)
## Evaporation <= 4.4
## WindGustSpeed > 30
## Humidity3pm <= 70
## -> class No  [0.953]
##
## Rule 5: (7, lift 6.0)
## WindDir3pm in [N-ESE]
## Humidity3pm > 70
## -> class Yes  [0.889]
##
## Rule 6: (11/1, lift 5.7)
....
```

72 C5.0 Rules Performance

Here we plot the performance of the decision tree, showing a risk chart. The areas under the recall and risk curves are also reported.

```
predicted <- predict(model, ds[test, vars], type="prob") [,2]
riskchart(predicted, actual, risks)
```



An error matrix shows, clockwise from the top left, the percentages of true negatives, false positives, true positives, and false negatives.

```
predicted <- predict(model, ds[test, vars], type="class")
sum(ds[test, target] != predicted)/length(predicted) # Overall error rate
## [1] 0.2273

round(100*table(ds[test, target], predicted, dnn=c("Actual", "Predicted"))/length(predicted))
##      Predicted
## Actual No Yes
##   No   66   8
##   Yes  15  11
....
```

73 Regression Trees

The discussion so far has dwelt on classification trees. Regression trees are similarly well catered for in R.

We can plot regression trees as with classification trees, but the node information will be different and some options will not make sense. For example, `extra=` only makes sense for 100 and 101.

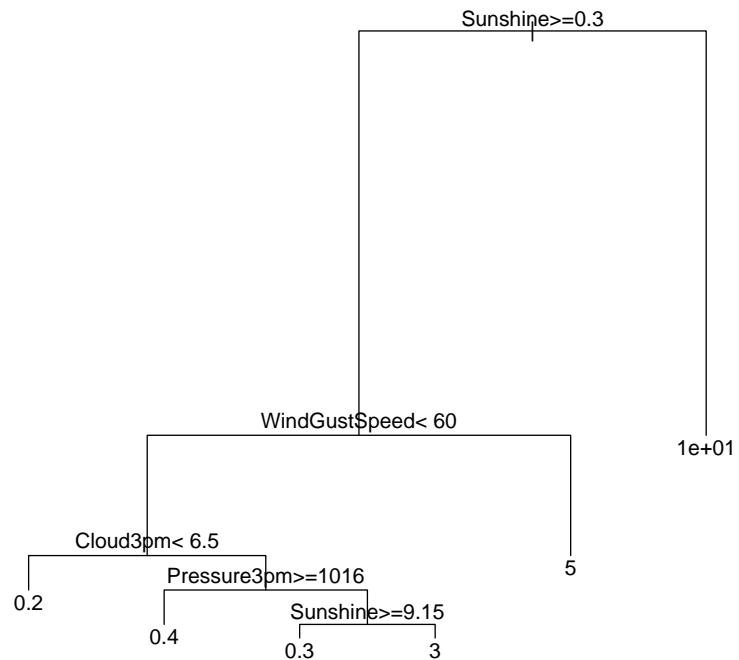
First we will build regression tree:

```
target <- "RISK_MM"
vars <- c(inputs, target)
form <- formula(paste(target, "~ ."))
(model <- rpart(formula=form, data=ds[train, vars]))

## n= 256
##
## node), split, n, deviance, yval
##      * denotes terminal node
##
## 1) root 256 2580.000  0.9656
##    2) Sunshine>=0.3 248 1169.000  0.6460
##      4) WindGustSpeed< 60 233  343.600  0.3957
##        8) Cloud3pm< 6.5 173   84.110  0.1653 *
##        9) Cloud3pm>=6.5 60   223.900  1.0600
##          18) Pressure3pm>=1016 36   65.320  0.4333 *
##          19) Pressure3pm< 1016 24   123.200  2.0000
##            38) Sunshine>=9.15 7   2.777  0.3429 *
##            39) Sunshine< 9.15 17   93.280  2.6820 *
##      5) WindGustSpeed>=60 15   584.200  4.5330 *
##    3) Sunshine< 0.3 8   599.800 10.8800 *
```

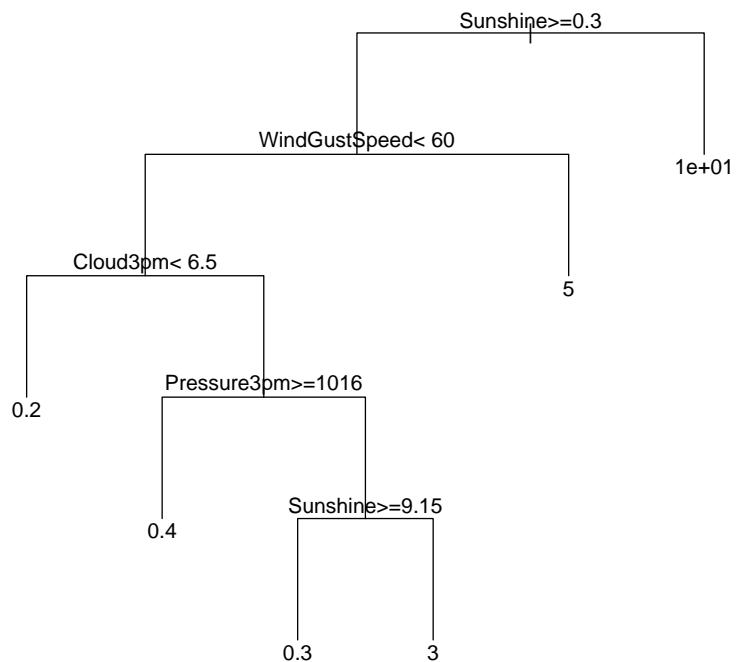
74 Visualise Regression Trees

```
plot(model)
text(model)
```



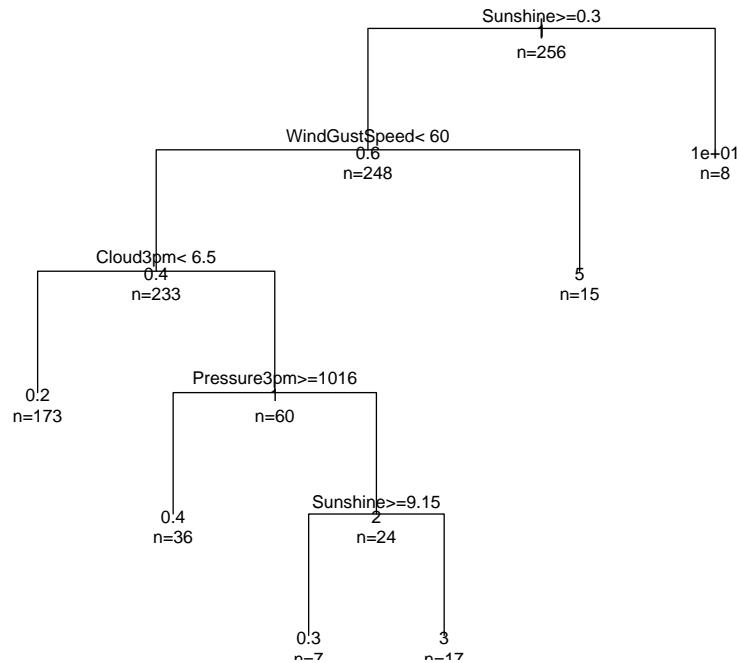
75 Visualise Regression Trees: Uniform

```
plot(model, uniform=TRUE)
text(model)
```



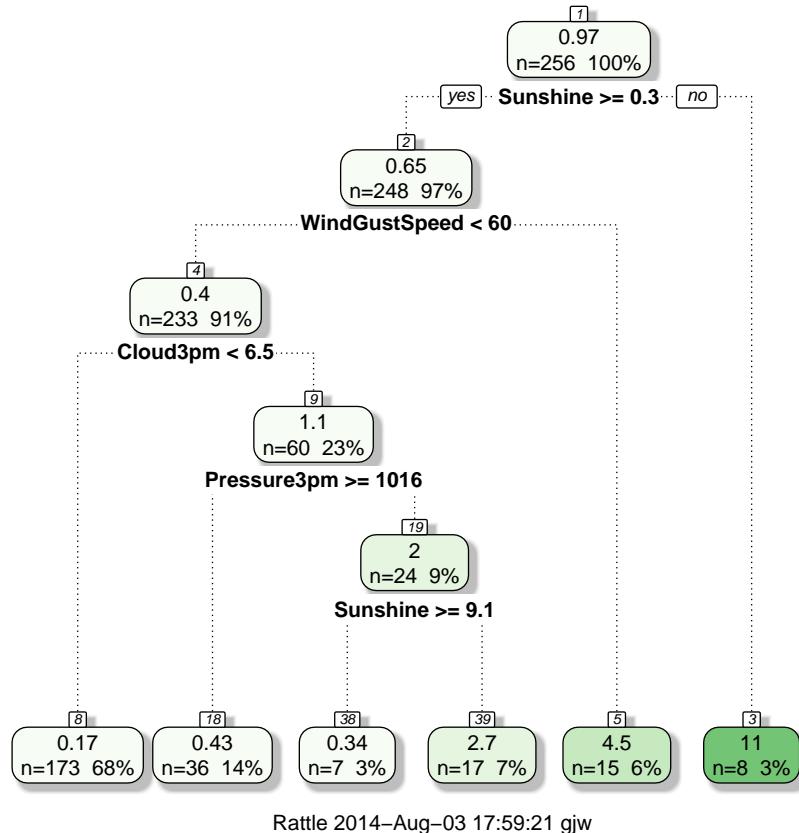
76 Visualise Regression Trees: Extra Information

```
plot(model, uniform=TRUE)
text(model, use.n=TRUE, all=TRUE, cex=.8)
```



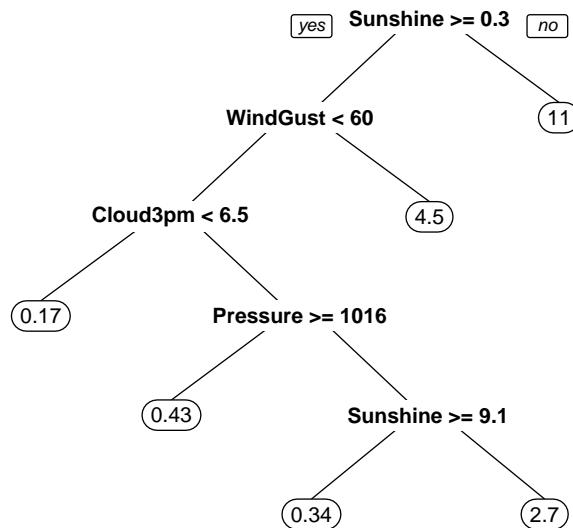
77 Fancy Plot of Regression Tree

```
fancyRpartPlot(model)
```



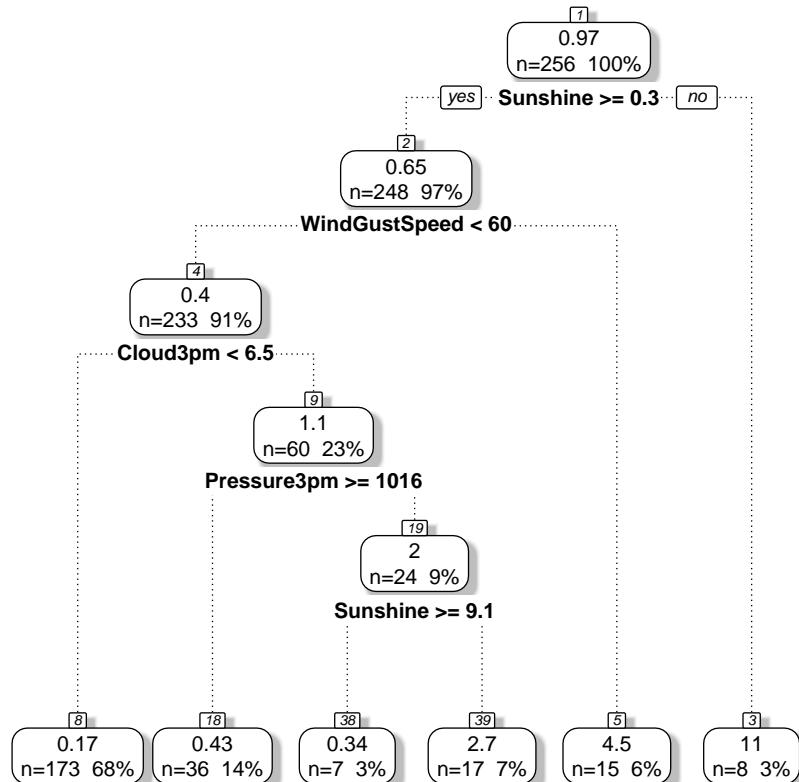
78 Enhanced Plot of Regression Tree: Default

```
prp(model)
```



79 Enhanced Plot of Regression Tree: Favourite

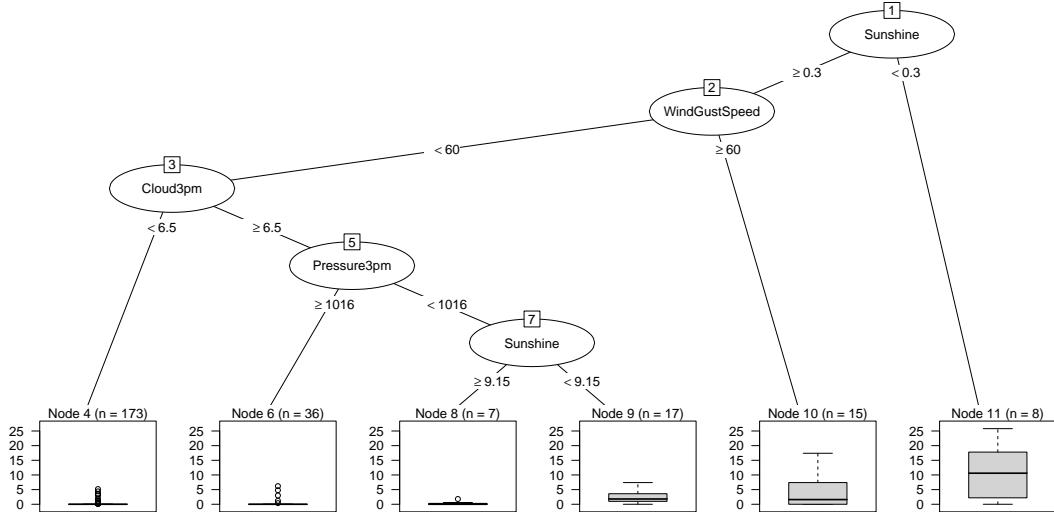
```
prp(model, type=2, extra=101, nn=TRUE, fallen.leaves=TRUE,
    faclen=0, varlen=0, shadow.col="grey", branch.lty=3)
```



80 Party Regression Tree

The tree drawing facilities of `party` (Hothorn *et al.*, 2013) can again be used to draw the `rpart` regression tree using `as.party()`.

```
class(model)
## [1] "rpart"
plot(as.party(model))
```



Notice the visualisation of the predictions—this is particularly informative. We have, in an instant, a view of the conditions when there is little or no rain, compared to significant rain.

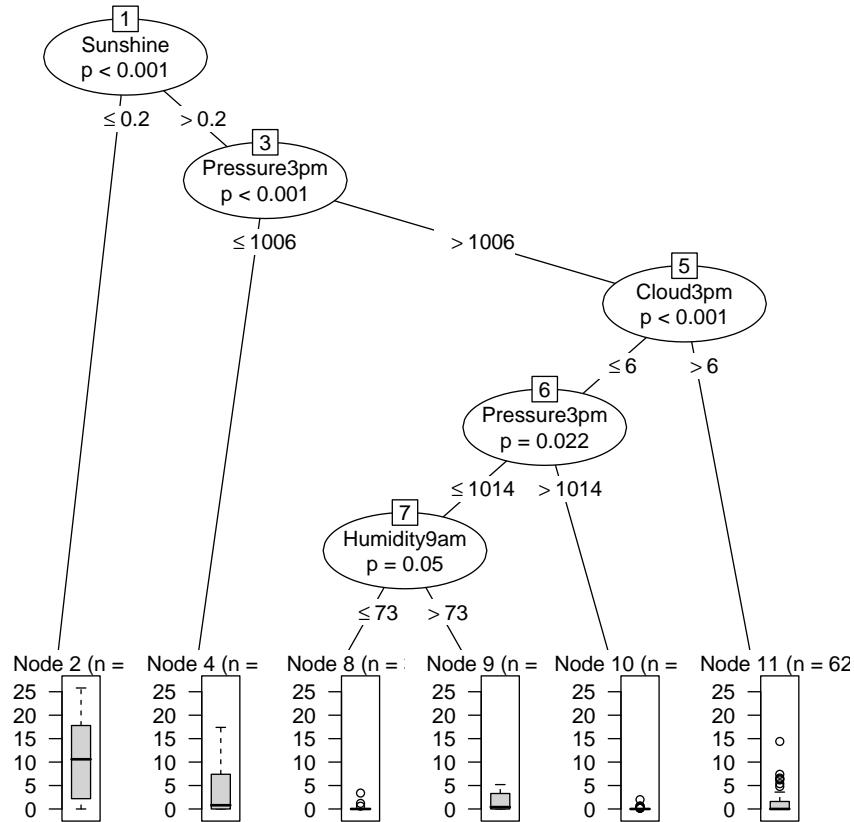
81 Conditional Regression Tree

We can also build a regression tree using `party` (Hothorn *et al.*, 2013).

```
model <- ctree(formula=form, data=ds[train, vars])  
  
model  
  
##  
## Model formula:  
## RISK_MM ~ MinTemp + MaxTemp + Rainfall + Evaporation + Sunshine +  
##      WindGustDir + WindGustSpeed + WindDir9am + WindDir3pm + WindSpeed9am +  
##      WindSpeed3pm + Humidity9am + Humidity3pm + Pressure9am +  
##      Pressure3pm + Cloud9am + Cloud3pm + Temp9am + Temp3pm + RainToday  
##  
## Fitted party:  
## [1] root  
## | [2] Sunshine <= 0.2: 11 (n=8, err=600)  
## | [3] Sunshine > 0.2  
## | | [4] Pressure3pm <= 1006.5: 5 (n=11, err=445)  
## | | [5] Pressure3pm > 1006.5  
## | | | [6] Cloud3pm <= 6  
## | | | | [7] Pressure3pm <= 1013.8  
## | | | | | [8] Humidity9am <= 73: 0 (n=31, err=12)  
## | | | | | [9] Humidity9am > 73: 2 (n=11, err=41)  
## | | | | [10] Pressure3pm > 1013.8: 0 (n=133, err=5)  
## | | | | [11] Cloud3pm > 6: 1 (n=62, err=399)  
##  
## Number of inner nodes: 5  
## Number of terminal nodes: 6
```

82 CTree Plot

```
plot(model)
```



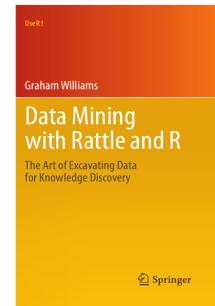
83 Weka Regression Tree

Weka's J48() does not support regression trees.

84 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.



- <http://www.milbo.org/rpart-plot/prp.pdf>: Plotting rpart trees with prp, by Stephen Milborrow. Stephen (author of rpart.plot) showcases the options of prp() and was the inspiration for the showcase of prp() included here.

85 References

- Hornik K (2014). *RWeka: R/Weka interface*. R package version 0.4-23, URL <http://CRAN.R-project.org/package=RWeka>.
- Hothorn T, Hornik K, Strobl C, Zeileis A (2013). *party: A Laboratory for Recursive Partitioning*. R package version 1.0-9, URL <http://CRAN.R-project.org/package=party>.
- Hothorn T, Zeileis A (2014). *partykit: A Toolkit for Recursive Partitioning*. R package version 0.8-0, URL <http://CRAN.R-project.org/package=partykit>.
- Kuhn M, Weston S, Coulter N, Quinlan JR (2014). *C50: C5.0 Decision Trees and Rule-Based Models*. R package version 0.1.0-19, URL <http://CRAN.R-project.org/package=C50>.
- Milborrow S (2014). *rpart.plot: Plot rpart models. An enhanced version of plot.rpart*. R package version 1.4-4, URL <http://CRAN.R-project.org/package=rpart.plot>.
- Neuwirth E (2011). *RColorBrewer: ColorBrewer palettes*. R package version 1.0-5, URL <http://CRAN.R-project.org/package=RColorBrewer>.
- 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/>.
- Therneau TM, Atkinson B (2014). *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-8, URL <http://CRAN.R-project.org/package=rpart>.
- 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/>.

This document, sourced from DTreesO.Rnw revision 473, was processed by KnitR version 1.6 of 2014-05-24 and took 39.9 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 2014-08-03 17:59:23.

Data Science with R

Decision Trees with Rattle

Graham.Williams@togaware.com

9th June 2014

Visit <http://onepager.togaware.com/> for more OnePageR's.

In this module we use the **weather** dataset to explore the building of decision tree models in **rattle** (Williams, 2014).

The required packages for this module include:

```
library(rpart)      # Popular recursive partitioning decision tree algorithm  
library(rattle)     # Graphical user interface for building decision trees
```

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 Loading the Data

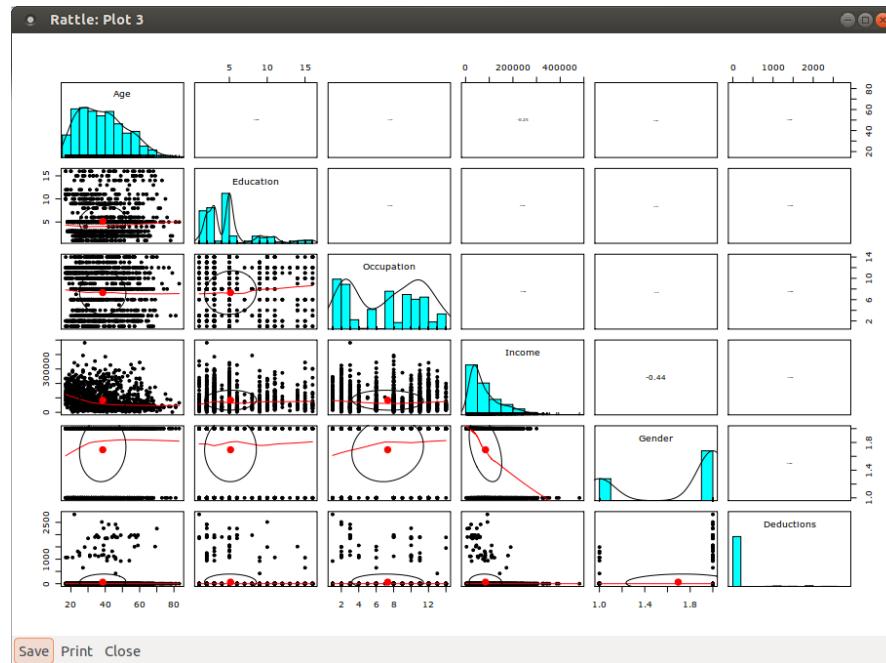
1. On the Data tab, click the Execute button to load the default `weather` dataset (which is loaded after clicking Yes).
2. Note that we can click on the `Filename` chooser box to find some other datasets. Assuming we have just loaded the default `weather` dataset, we should be taken to the folder containing the actual CSV data files provided with Rattle.
3. Load the `audit` dataset.
4. Note that the variable `TARGET_Adjusted` is selected as the Target variable, and that the variable `ID` is identified as having a role of Ident(ifier).
5. Note also the variable `RISK_Adjustment` is set to have a role of Risk (this is based on its name). **For now, choose to give it the role of an Input variable.**
6. Choose to Partition the data. In fact, leave the 70/15/15 percentage split in the Partition text box as it is. Also, ensure the Partition checkbox is checked. This results in a random 70% of the dataset being used for training or building our models. A 15% sample of the dataset is used for validation, and is used whilst we are building and comparing different decision trees through the use of different parameters. The final 15% is for testing.
7. **Exercise:** Research the issue of selecting a training/validation/testing dataset. Why is it important to partition the data in this way when data mining? Explain in a paragraph or two.
8. **Exercise:** Compare this approach to partitioning a dataset with the concept of cross fold validation. Report on this in one or two paragraphs.
9. Be sure you have clicked the Execute button whilst on the Data tab. This will ensure that the sampling, for example, has taken place.

No.	Variable	Data Type	Input	Target	Risk	Ident	Ignore	Weight	Comment
1	ID	Numeric	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 2000
2	Age	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 67
3	Employment	Categoric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 8 Missing: 100
4	Education	Categoric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 16
5	Marital	Categoric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 6
6	Occupation	Categoric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 14 Missing: 101
7	Income	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 2000
8	Gender	Categoric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 2
9	Deductions	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 41
10	Hours	Numeric	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 68
11	IGNORE_Accounts	Categoric	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 33 Missing: 43
12	RISK_Adjustment	Numeric	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 310
13	TARGET_Adjusted	Numeric	<input type="radio"/>	<input checked="" type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	Unique: 2

Roles noted. 2000 observations and 9 input variables. The target is TARGET_Adjusted. Categoric 2. Classification models enabled.

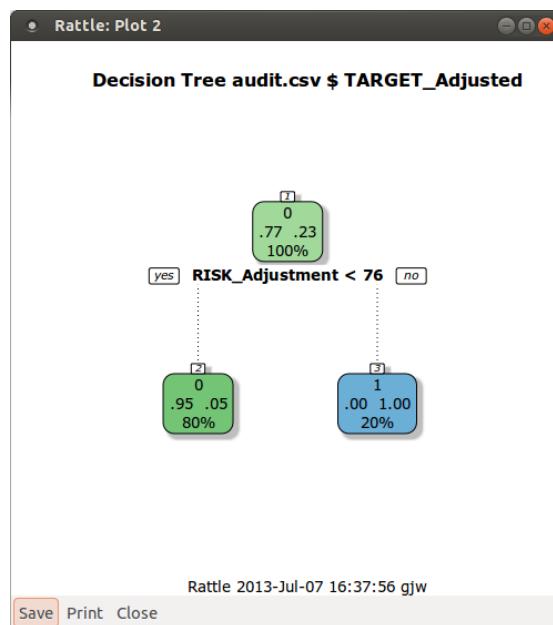
2 Exploring the Data

10. Click the View button to get a little familiar with the data.
11. **Exercise:** What values of the variable TARGET_Adjusted correspond to an adjustment? How does TARGET_Adjusted relate to RISK_Adjustment?
12. Explore the dataset further from the Explore tab.
13. Firstly, simply click Execute from the Summary option.
14. Explore some of the different summaries that we can generate.
15. Textual summaries are comprehensive, but sometimes take a little getting used to. Visual summaries can convey information more quickly. Select the Distributions option and then Execute (without any distributions being selected) to view a family of scatter plots.
16. **Exercise:** What do the numbers in the plots represent?
17. Choose one of each of the plot types for Income, then execute.
18. **Exercise:** Research the use of Benford's law. How might it be useful in the context of the Benford's plot of Income? Discuss in one or two paragraphs.
19. Rattle is migrating to a more sophisticated collection of graphics. To access this work in progress, from the Settings menu enable the Advanced Graphics option. Then select Execute again. It is experimental and if it fails, un-select the Option.
20. Now have a look at the distribution of Age against the target variable.
21. **Exercise:** Are the different distributions significant? Explain each of the different elements of the plot.



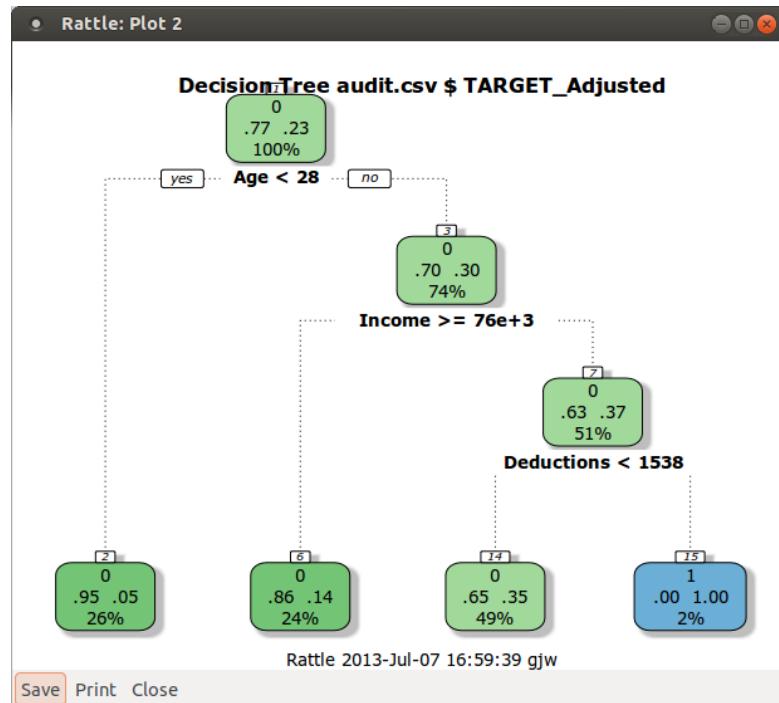
3 Naïvely Building Our First Decision Tree

22. Move to the Model tab, click the Execute button.
23. A decision tree is produced. Take a moment to understand what the description of the decision tree means. Click on the Draw button to see a visual presentation of the tree.
24. Go to the Evaluate tab and click the Execute button.
25. **Exercise:** How accurate is this model? How many true positives are there? How many false positives are there? Which are better—false positives or false negatives? What are the consequences of false positives and of true positives in the audit scenario?
26. **What is the fundamental flaw with the model we have just built?**
27. Go back to the Explore tab and have a look at the distribution of RISK_Adjustment against the target variable. Does this explain the model performance?
28. Go back to the Data tab and change RISK_Adjustment to be a Risk variable. Set to Ignore any variables that you feel are not suitable for decision tree classification. After having built further decision trees (or any models in fact) you might want to come back to the Data tab and change your variable selections. Be sure to click the Execute button each time.
29. **Exercise:** What do you notice about the variables chosen in the new model? Why are categoric variables favoured over numeric variables? Research this issue and discuss in one or two paragraphs.



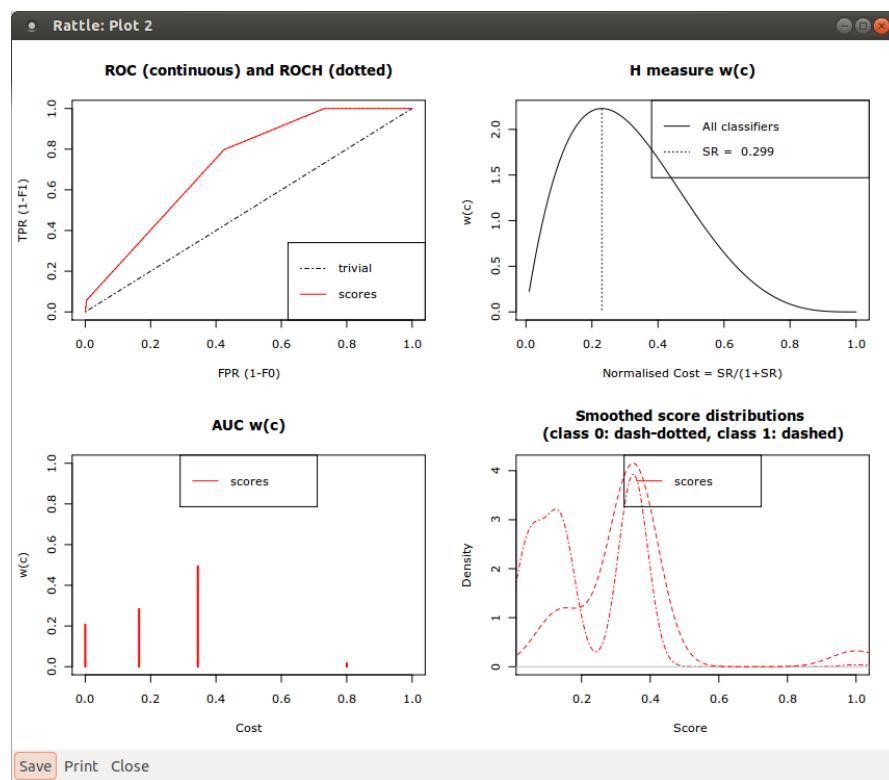
4 Building a Useful Decision Tree

30. Go to the Model tab and make sure the Tree radio button is selected.
31. Note the various parameters that can be set and modified. Read the Rattle documentation on Decision Trees for more information. You can also get additional help for these parameters from R by typing into the R console of RStudio: `help(rpart.control)`.
32. Which control options noted in the documentation of the `rpart()` command correspond to which Rattle options?
33. Generate a new decision tree by clicking on Execute and inspect what is printed into the Rattle textview.
34. Click on Draw and a window with a decision tree will be shown.
35. Which leaf node is most predictive of a tax return requiring an adjustment? Which is most predictive of not requiring an adjustment?
36. Compare the decision tree drawing with the Summary of the rpart model in the main Rattle textview. Each leaf node in the drawing has a coloured number (which corresponds to the leaf node number), a 0 or 1 (which is the class label from the audit data set according to the target variable `TARGET_Adjusted`), and a percentage number (which corresponds to the accuracy of the classified training records in this leaf node).



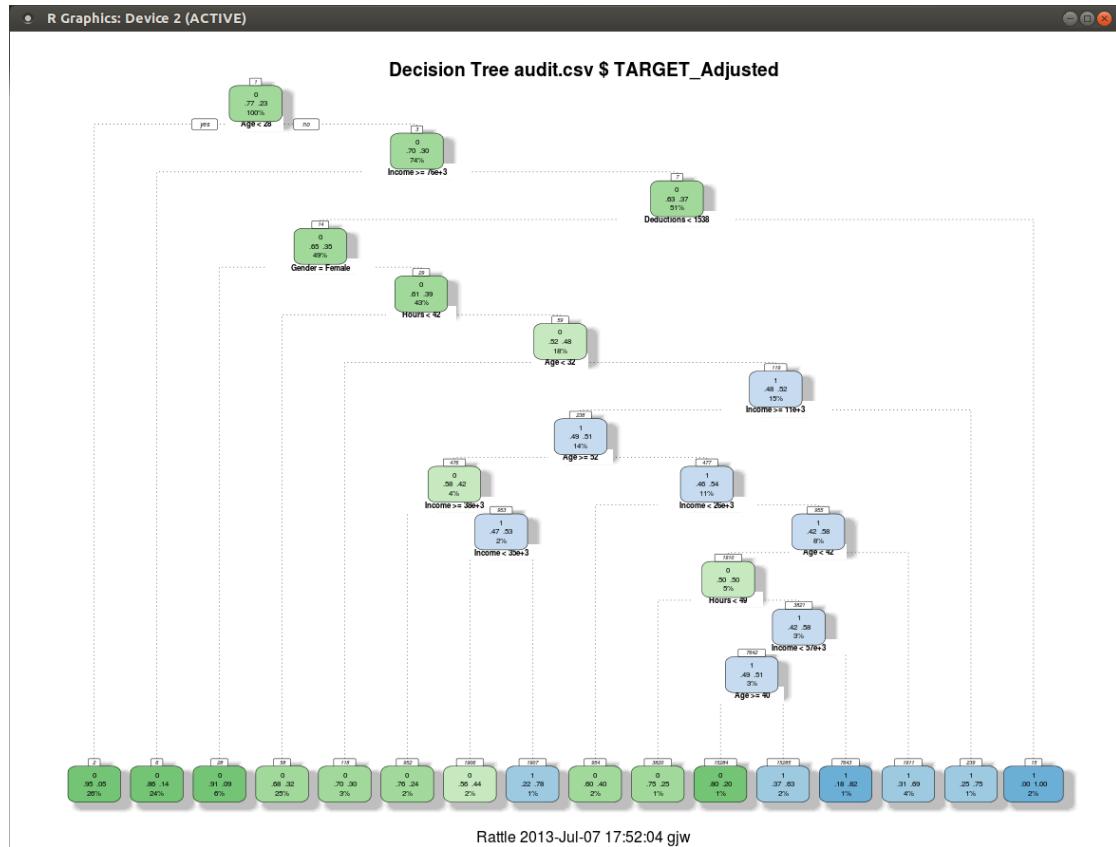
5 Evaluating the Model

37. On the **Evaluate** tab examine the different options to evaluate the accuracy of the decision tree we generated. Make sure the Validation and the Error Matrix radio buttons are both selected, and then click on Execute.
38. Check the error matrix that is printed (and write down the four numbers for each tree you generate).
39. **What is the error rate of the new decision tree? What is its accuracy?**
40. Click the **Training** radio button and again click on Execute.
41. **What is the error rate and what is the accuracy?**
42. **Why are the error rate and accuracy different between the validation and training settings?**
43. Select different Evaluate options then click on **Execute**.
44. You can read more on these evaluation measures in the Rattle book.
45. Investigate the ROC Curve graphical measure, as this is a widely used method to assess the performance of classifiers.



6 Fine Tuning the Decision Tree

46. Go back to the Model tab and experiment with different values for the parameters Complexity, Min Bucket, Min Split and Max Depth.
47. Which tree will give you the best accuracy, which one the worst? Which tree is the easiest to interpret? Which is hardest? Remember that you need to click on Execute each time you change a parameter setting and want to generate a new tree.
48. Click on the Rules button to see the rules generated from a given tree.
49. What is easier to understand, the tree diagram or the rules?
50. What is the highest accuracy you can achieve on the audit data set using the Rattle decision tree classifier? Which is the 'best' tree you can generate?



7 Finishing Up

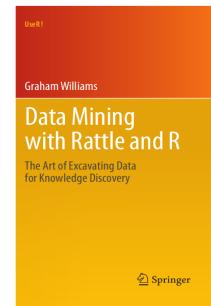
51. When finished we might like to save our project. Click on the **Save** icon.
52. We can give our project a name, like `audit_140609.rattle`.
53. Quit from Rattle and from R.
54. Now start up R and then Rattle again and load the project you saved.

8 Further Reading

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 module is one of many OnePageR modules available from <http://onepager.togaware.com>. In particular follow the links on the website with a * which indicates the generally more developed OnePageR modules.



9 References

- 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/>.
- 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 3.0.4, URL <http://rattle.togaware.com/>.

This document, sourced from DTreesG.Rnw revision 419, was processed by KnitR version 1.6 of 2014-05-24 and took 1 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-06-09 10:37:50.

Data Science with R

Ensemble of Decision Trees

Graham.Williams@togaware.com

3rd August 2014

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

The concept of building multiple decision trees to produce a better model can be dated back to the concept of Multiple Inductive Learning or the MIL algorithm (Williams, 1988). An ensemble of trees was found to produce a more accurate model than a single tree.

In this module we explore the use of `ada` (Culp *et al.*, 2012) and `randomForest` (Breiman *et al.*, 2012), as well as two newer packages `wsrpart` (Zhalama and Williams, 2014) and `wsrf` (Meng *et al.*, 2014).

The required packages for this module include:

```
library(rattle)      # The weather dataset.  
library(ada)         # Build boosted trees model with ada().  
library(randomForest) # Impute missing values with na.roughfix().  
library(wsrpart)     # Weighted subspace using RPart.  
library(wsrf)        # Weighted subspace implemented in Cpp.  
library(party)       # Conditional random forest cforest().
```

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 Prepare Weather Data for Modelling

See the separate Data and Model modules for template for preparing data and building models. In brief, we set ourselves up for modelling the **weather** dataset with the following commands, extending the simpler example we have just seen.

```
set.seed(1426)
library(rattle)
data(weather)
dsname      <- "weather"
ds          <- get(dsname)
id          <- c("Date", "Location")
target      <- "RainTomorrow"
risk         <- "RISK_MM"
ignore       <- c(id, if (exists("risk")) risk)
(vars        <- setdiff(names(ds), ignore))

## [1] "MinTemp"      "MaxTemp"      "Rainfall"      "Evaporation"
## [5] "Sunshine"     "WindGustDir"   "WindGustSpeed" "WindDir9am"
## [9] "WindDir3pm"   "WindSpeed9am"  "WindSpeed3pm"  "Humidity9am"
## [13] "Humidity3pm" "Pressure9am"   "Pressure3pm"   "Cloud9am"
....
inputs       <- setdiff(vars, target)
ds[vars]    <- na.roughfix(ds[vars]) # Impute missing values, roughly.
(nobs       <- nrow(ds))

## [1] 366

(numerics    <- intersect(inputs, names(ds)[which(sapply(ds[vars], is.numeric))]))
## [1] "MinTemp"      "MaxTemp"      "Rainfall"      "Sunshine"
## [5] "WindDir9am"   "WindDir3pm"   "WindSpeed9am" "WindSpeed3pm"
## [9] "Humidity9am" "Humidity3pm"  "Pressure9am"  "Pressure3pm"
## [13] "Cloud9am"    "Cloud3pm"
....
(categorics  <- intersect(inputs, names(ds)[which(sapply(ds[vars], is.factor))]))
## [1] "Evaporation" "WindGustDir"   "WindGustSpeed" "Temp9am"
## [5] "Temp3pm"

(form       <- formula(paste(target, "~ .")))
## RainTomorrow ~ .

length(train <- sample(nobs, 0.7*nobs))
## [1] 256
length(test  <- setdiff(seq_len(nobs), train))
## [1] 110
actual      <- ds[test, target]
```

2 Review the Dataset

It is always a good idea to review the data.

```

dim(ds)
## [1] 366 24

names(ds)
##  [1] "Date"          "Location"       "MinTemp"        "MaxTemp"
## [5] "Rainfall"       "Evaporation"    "Sunshine"       "WindGustDir"
## [9] "WindGustSpeed" "WindDir9am"     "WindDir3pm"     "WindSpeed9am"
## [13] "WindSpeed3pm"  "Humidity9am"   "Humidity3pm"   "Pressure9am"
## [17] "Pressure3pm"   "Cloud9am"      "Cloud3pm"      "Temp9am"
.....
head(ds)
##           Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 1 2007-11-01 Canberra     8.0    24.3     0.0      3.4      6.3
## 2 2007-11-02 Canberra    14.0    26.9     3.6      4.4     9.7
## 3 2007-11-03 Canberra   13.7    23.4     3.6      5.8     3.3
## 4 2007-11-04 Canberra   13.3    15.5    39.8      7.2     9.1
.....
tail(ds)
##           Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 361 2008-10-26 Canberra     7.9    26.1     0.0      6.8      3.5
## 362 2008-10-27 Canberra    9.0    30.7     0.0      7.6     12.1
## 363 2008-10-28 Canberra    7.1    28.4     0.0     11.6     12.7
## 364 2008-10-29 Canberra   12.5    19.9     0.0      8.4      5.3
.....
str(ds)
## 'data.frame': 366 obs. of  24 variables:
## $ Date : Date, format: "2007-11-01" "2007-11-02" ...
## $ Location : Factor w/ 49 levels "Adelaide","Albany",...: 10 10 10 10 ...
## $ MinTemp : num  8 14 13.7 13.3 7.6 6.2 6.1 8.3 8.8 8.4 ...
## $ MaxTemp : num  24.3 26.9 23.4 15.5 16.1 16.9 18.2 17 19.5 22.8 ...
.....
summary(ds)
##           Date                  Location        MinTemp        MaxTemp
##  Min.   :2007-11-01   Canberra   :366   Min.   :-5.30   Min.   : 7.6
##  1st Qu.:2008-01-31   Adelaide   : 0   1st Qu.: 2.30   1st Qu.:15.0
##  Median :2008-05-01   Albany     : 0   Median : 7.45   Median :19.6
##  Mean   :2008-05-01   Albury     : 0   Mean   : 7.27   Mean   :20.6
.....

```

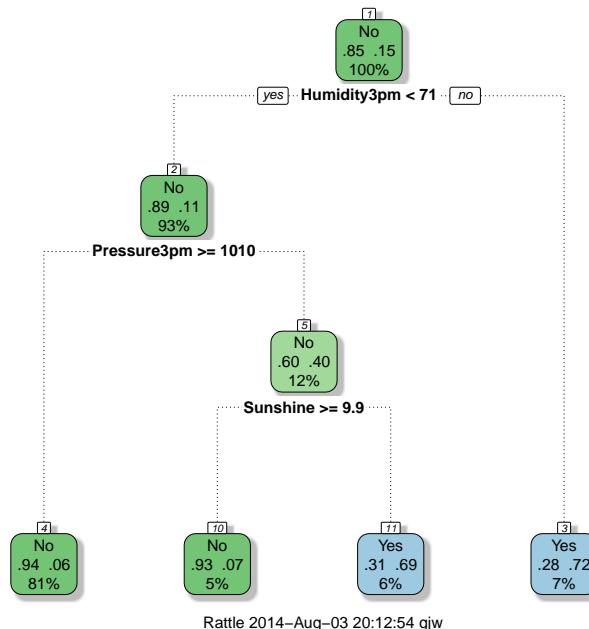
3 Decision Tree for Comparison

We begin by using our basic decision tree model as a base to compare the performance of the ensemble decision trees. See the DTrees module for details.

```
model <- m.rp <- rpart(form, ds[train, vars])

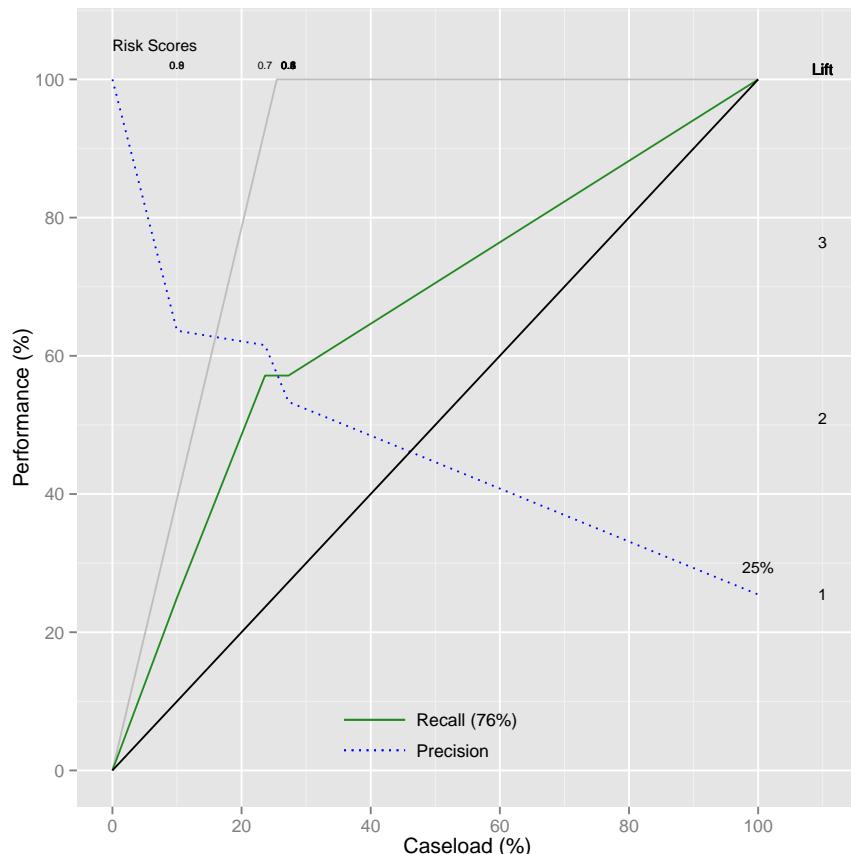
model
## n= 256
##
## node), split, n, loss, yval, (yprob)
##      * denotes terminal node
##
## 1) root 256 38 No (0.85156 0.14844)
##    2) Humidity3pm< 71 238 25 No (0.89496 0.10504)
##      4) Pressure3pm>=1010 208 13 No (0.93750 0.06250) *
##      5) Pressure3pm< 1010 30 12 No (0.60000 0.40000)
##        10) Sunshine>=9.95 14 1 No (0.92857 0.07143) *
##        11) Sunshine< 9.95 16 5 Yes (0.31250 0.68750) *
##    3) Humidity3pm>=71 18 5 Yes (0.27778 0.72222) *

fancyRpartPlot(model)
```



4 Decision Tree Performance

```
predicted <- predict(model, ds[test, vars], type="prob")[,2]
riskchart(predicted, actual)
```



5 Random Forest Model

```
model <- m.rf <- randomForest(form, ds[train, vars])  
  
model  
##  
## Call:  
##   randomForest(formula=form, data=ds[train, vars])  
##           Type of random forest: classification  
##                   Number of trees: 500  
## No. of variables tried at each split: 4  
##  
##       OOB estimate of  error rate: 12.11%  
## Confusion matrix:  
##      No Yes class.error  
## No  214   4    0.01835  
## Yes  27  11    0.71053
```

Notice the out-of-bag (OOB) estimate of the error rate.

Exercise: Explain what an out-of-bag estimate is. How is it calculated for the random forest?

6 Random Forest Performance—Error Matrix

An error matrix shows, clockwise from the top left, the percentages of true negatives, false positives, true positives, and false negatives.

```
predicted <- predict(model, ds[test, vars])
sum(actual != predicted)/length(predicted) # Overall error rate

## [1] 0.1545

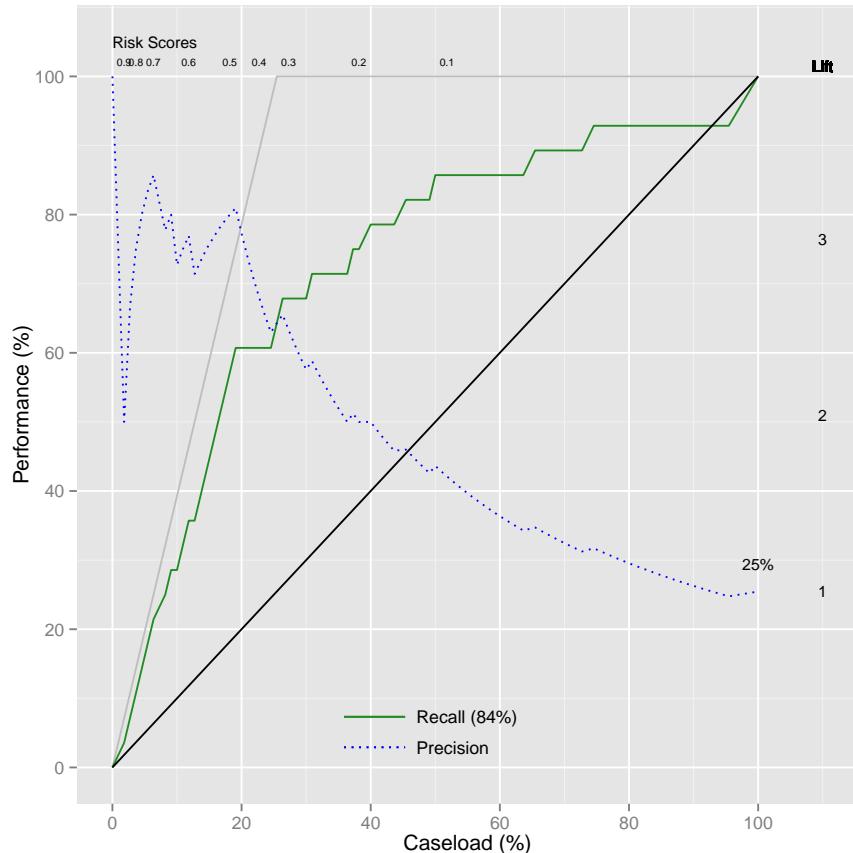
round(100*table(actual, predicted, dnn=c("Actual", "Predicted"))/length(predicted))

##      Predicted
## Actual No Yes
##   No    71   4
##   Yes   12  14
##   ...
```

Compare the matrix here with the OOB matrix from the randomForest() call itself. The data here is based on the 30% test dataset. The OOB estimate is based on the 70% sampled used as the training dataset.

7 Random Forest Performance—Risk Chart

```
predicted <- predict(model, ds[test, vars], type="prob")[,2]
riskchart(predicted, actual)
```



8 Conditional Random Forest

```
model <- m.cf <- cforest(form, ds[train, vars])
model

##
##  Random Forest using Conditional Inference Trees
##
## Number of trees:  500
.....


model <- m.cf <- cforest(form, ds[train, vars],
                           controls=cforest_control(ntree=500,
                                                     mtry=2,
                                                     replace=FALSE,
                                                     teststat="quad",
                                                     testtype = "Univ",
                                                     mincriterion=0,
                                                     fraction = 0.632,
                                                     minsplit=2,
                                                     minbucket=1))
model

##
##  Random Forest using Conditional Inference Trees
##
## Number of trees:  500
.....
```

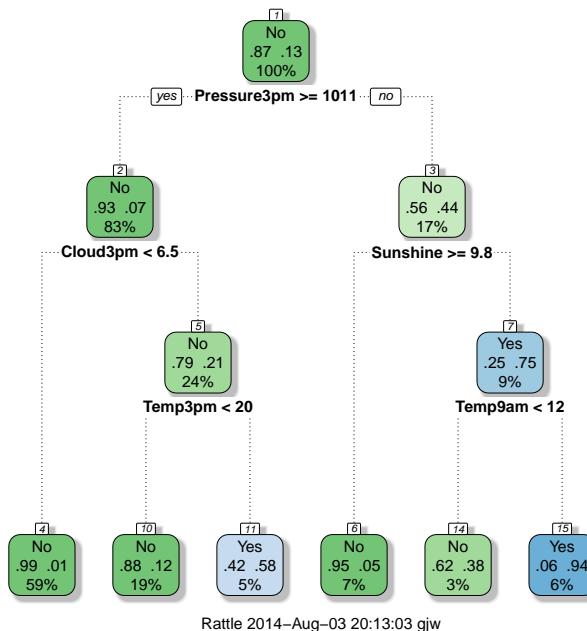
9 Weighted Subspace with RPart Decision Trees

```

model <- m.wsrp <- wsrpart(form, ds[train, vars], ntrees=100)

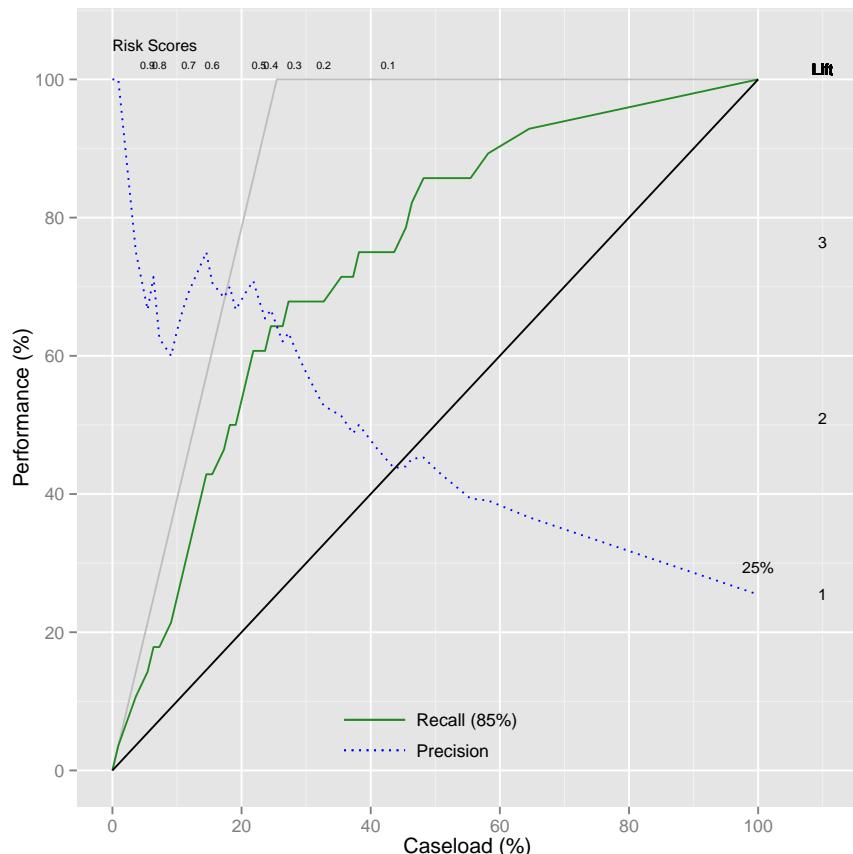
model
## A multiple rpart model with 100 trees.
##
## Variables used (11): MinTemp, Temp3pm, Rainfall, Cloud9am, Pressure3pm,
## WindSpeed9am, Pressure9am, Cloud3pm,
## Humidity3pm, Temp9am, Sunshine.
##
## n= 256
##
## node), split, n, loss, yval, (yprob)
##      * denotes terminal node
##
## 1) root 256 33 No (0.871094 0.128906)
##    2) Pressure3pm>=1011 213 14 No (0.934272 0.065728)
##       4) Cloud3pm< 6.5 152 1 No (0.993421 0.006579) *
##       5) Cloud3pm>=6.5 61 13 No (0.786885 0.213115)
## ...
## fancyRpartPlot(model[[1]]$model)

```



10 Weighted Subspace RPart Performance

```
predicted <- predict(model, ds[test, vars], type="prob")[,2]
riskchart(predicted, actual)
```



11 Weighted Subspace Random Forest

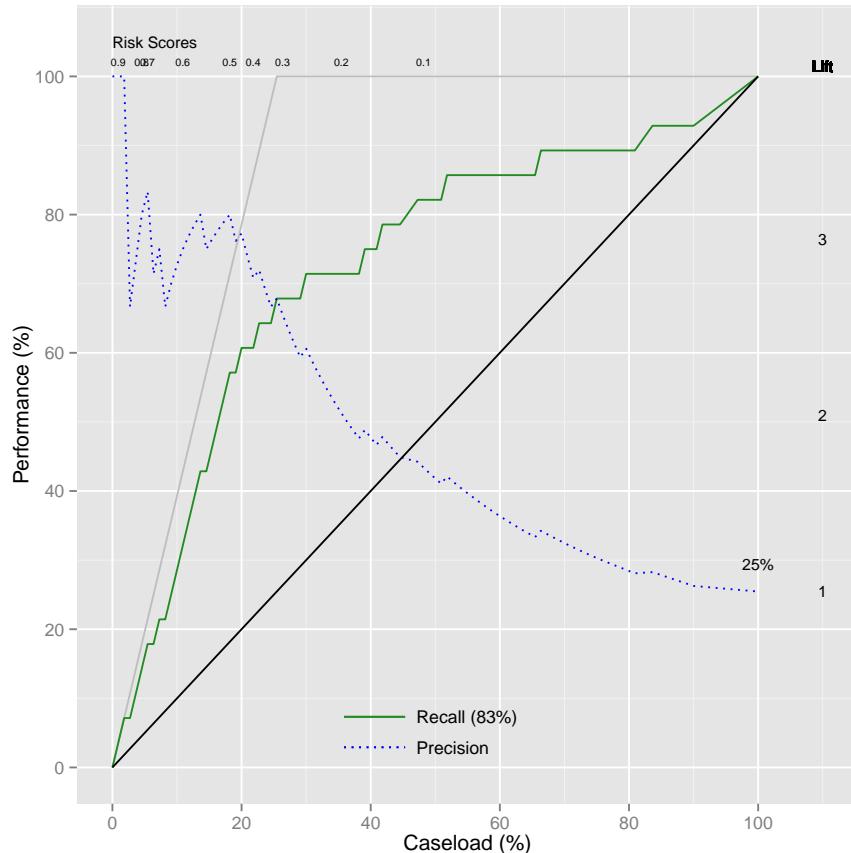
```
model <- m.wsrf <- wsrf(form, ds[train, vars], ntrees=500, nvars=20)

model

## 
## Tree 1 with 29 nodes:
## 1) root
##   ..2) Sunshine <= 6.15
##     ...3) Pressure3pm <= 1016.5
##       ...4) WindSpeed9am <= 2 No (1 0)
##       ...5) WindSpeed9am > 2
##         ...6) Temp3pm <= 7.75 No (0.5 0.5)
##         ...7) Temp3pm > 7.75 Yes (0 1)
##         ...8) Pressure3pm > 1016.5
##         ...9) WindGustSpeed <= 23
##           ...10) Pressure3pm <= 1021.6 Yes (0 1)
##           ...11) Pressure3pm > 1021.6 No (0.5 0.5)
##           ...12) WindGustSpeed > 23
##             ...13) Temp3pm <= 10.2 No (0.5 0.5)
##             ...14) Temp3pm > 10.2 No (1 0)
##             ...15) Sunshine > 6.15
##               ...16) Cloud3pm <= 3.5 No (1 0)
##               ...17) Cloud3pm > 3.5
##                 ...18) MaxTemp <= 28.85
##                   ...19) Humidity3pm <= 64
##                     ...20) Pressure3pm <= 1012.05
##                     ...21) Pressure3pm <= 1011.5
##                       ...22) Temp3pm <= 17.2 No (0.5 0.5)
##                       ...23) Temp3pm > 17.2 No (1 0)
##                         ...24) Pressure3pm > 1011.5 Yes (0 1)
##                         ...25) Pressure3pm > 1012.05 No (1 0)
##                         ...26) Humidity3pm > 64 Yes (0.333 0.667)
##                           ...27) MaxTemp > 28.85
##                           ...28) Sunshine <= 9.25 Yes (0 1)
##                           ...29) Sunshine > 9.25 No (1 0)
## 
## Tree 2 with 39 nodes:
## 1) root
##   ..2) Humidity3pm <= 72.5
##     ...3) Pressure3pm <= 1010.3
##     ...4) Humidity3pm <= 53.5
##       ...5) WindSpeed3pm <= 10 Yes (0 1)
##       ...6) WindSpeed3pm > 10
##         ...7) Cloud3pm <= 7.5
## 
```

12 Weighted Subspace Random Forest Performance

```
predicted <- predict(model, ds[test, vars], type="prob")[,2]
riskchart(predicted, actual)
```



13 Wide Datasets

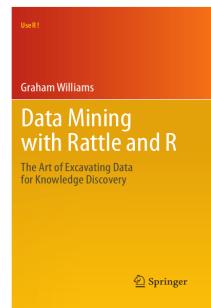
The weighted subspace algorithms target datasets with very many variables.

Exercise: Obtain a sample dataset with vary many variables, such as a text mining dataset, and compare the performances of rp, rf, wsprt, and wsrf.

14 Further Reading

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.



15 References

- Breiman L, Cutler A, Liaw A, Wiener M (2012). *randomForest: Breiman and Cutler's random forests for classification and regression*. R package version 4.6-7, URL <http://CRAN.R-project.org/package=randomForest>.
- Culp M, Johnson K, Michailidis G (2012). *ada: ada: an R package for stochastic boosting*. R package version 2.0-3, URL <http://CRAN.R-project.org/package=ada>.
- Meng Q, Zhao H, Williams GJ (2014). *wsrf: Weighted Subspace Random Forest*. R package version 1.3.17.
- 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/>.
- Williams GJ (1988). “Combining decision trees: Initial results from the MIL algorithm.” In JS Gero, RB Stanton (eds.), *Artificial Intelligence Developments and Applications: Selected papers from the first Australian Joint Artificial Intelligence Conference, Sydney, Australia, 2-4 November, 1987*, pp. 273–289. Elsevier Science Publishers B.V. (North-Holland). ISBN 0444704655.
- 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/>.
- Zhalama, Williams GJ (2014). *wsrpart: Build weighted subspace rpart decision trees*. R package version 1.2.151.

This document, sourced from EnsemblesO.Rnw revision 478, was processed by KnitR version 1.6 of 2014-05-24 and took 58.7 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 2014-08-03 20:13:52.

Support Vector Machines

Chapter pending / not available

Neural Networks

Chapter pending / not available

Data Science with R

Naive Bayes Classification

Graham.Williams@togaware.com

9th June 2014

Visit <http://onepager.togaware.com/> for more OnePageR's.

The required packages for this chapter include:

```
library(rattle)      # weather and normVarNames()
library(randomForest) # na.roughfix()
library(e1071)        # naiveBayes()
library(ROCR)         # prediction()
```

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 Prepare Weather Data for Modelling

See Chapters on Data and Model for the template for preparing data and building models. We repeat the setup here with little comment, except to note that we use the **weather** dataset from rattle (Williams, 2014).

```
library(rattle)          # Normalise names normVarNames() and weather dataset.
library(randomForest)    # Impute missing using na.roughfix().

dsname      <- "weather"
ds          <- get(dsname)
names(ds)   <- normVarNames(names(ds))
vars        <- names(ds)
target      <- "rain_tomorrow"
risk        <- "risk_mm"
id          <- c("date", "location")

ignore     <- union(id, if (exists("risk")) risk)
vars       <- setdiff(vars, ignore)

inputs     <- setdiff(vars, target)
numi       <- which(sapply(ds[inputs], is.numeric))
numc       <- names(numi)
cati       <- which(sapply(ds[inputs], is.factor))
catc       <- names(cati)

ds[numc]   <- na.roughfix(ds[numc]) # Impute missing values, roughly.
ds[target] <- as.factor(ds[[target]]) # Ensure the target is categoric.

nobs       <- nrow(ds)

form       <- formula(paste(target, "~ ."))

set.seed(42)

train     <- sample(nobs, 0.7*nobs)
test      <- setdiff(seq_len(nobs), train)
actual    <- ds[test, target]
risks     <- ds[test, risk]
```

2 Review the Dataset

It is always a good idea to review the data.

```
dim(ds)
## [1] 366 24

names(ds)

## [1] "date"          "location"       "min_temp"
## [4] "max_temp"       "rainfall"        "evaporation"
## [7] "sunshine"       "wind_gust_dir"  "wind_gust_speed"
## [10] "wind_dir_9am"   "wind_dir_3pm"   "wind_speed_9am"
## [13] "wind_speed_3pm" "humidity_9am"  "humidity_3pm"
.....
head(ds)

##           date location min_temp max_temp rainfall evaporation sunshine
## 1 2007-11-01 Canberra     8.0     24.3      0.0      3.4      6.3
## 2 2007-11-02 Canberra    14.0     26.9      3.6      4.4     9.7
## 3 2007-11-03 Canberra    13.7     23.4      3.6      5.8      3.3
## 4 2007-11-04 Canberra    13.3     15.5     39.8      7.2     9.1
.....
tail(ds)

##           date location min_temp max_temp rainfall evaporation sunshine
## 361 2008-10-26 Canberra     7.9     26.1      0       6.8      3.5
## 362 2008-10-27 Canberra    9.0     30.7      0       7.6     12.1
## 363 2008-10-28 Canberra    7.1     28.4      0      11.6     12.7
## 364 2008-10-29 Canberra   12.5     19.9      0       8.4      5.3
.....
str(ds)

## 'data.frame': 366 obs. of  24 variables:
## $ date           : Date, format: "2007-11-01" "2007-11-02" ...
## $ location        : Factor w/ 49 levels "Adelaide","Albany",...: 10 10 10 1...
## $ min_temp        : num  8 14 13.7 13.3 7.6 6.2 6.1 8.3 8.8 8.4 ...
## $ max_temp        : num  24.3 26.9 23.4 15.5 16.1 16.9 18.2 17 19.5 22.8 ...
## .....
summary(ds)

##           date                  location      min_temp      max_temp
## Min.   :2007-11-01   Canberra   :366   Min.   :-5.30   Min.   : 7.6
## 1st Qu.:2008-01-31   Adelaide    : 0   1st Qu.: 2.30   1st Qu.:15.0
## Median :2008-05-01   Albany      : 0   Median : 7.45   Median :19.6
## Mean    :2008-05-01   Albury      : 0   Mean    : 7.27   Mean    :20.6
.....
```

3 Naive Bayes Model

Here we use `naiveBayes()` from `e1071` (Meyer *et al.*, 2014).

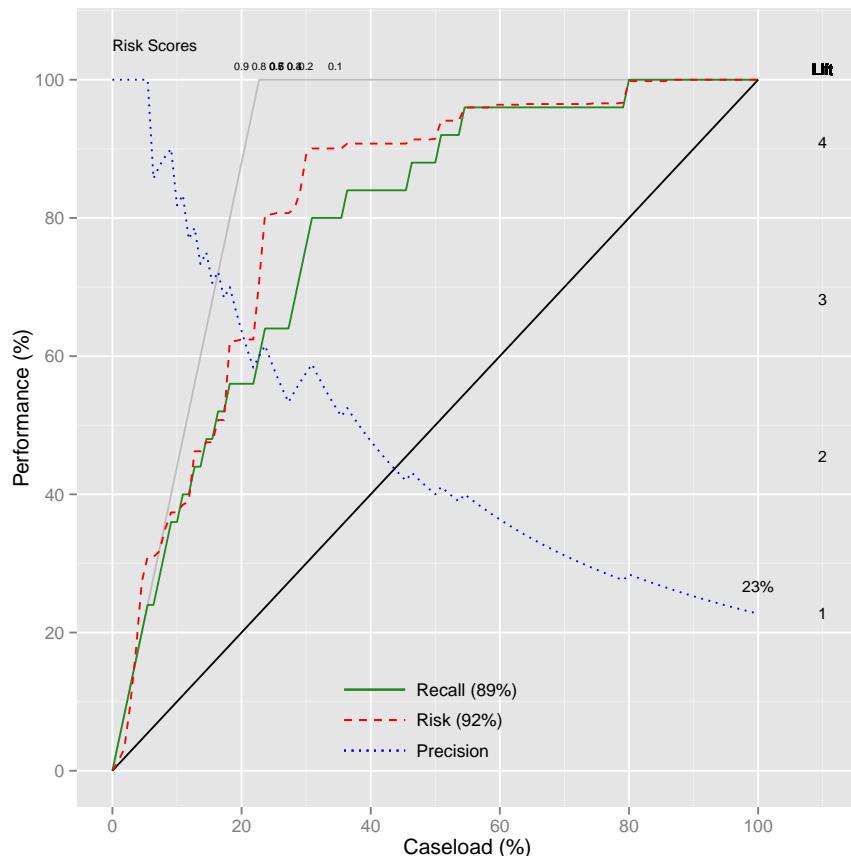
```
library(e1071)
model <- naiveBayes(form, data=ds[train, vars])
model

##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x=X, y=Y, laplace=laplace)
##
## A-priori probabilities:
## Y
##   No    Yes
## 0.8398 0.1602
##
## Conditional probabilities:
##   min_temp
## Y      [,1]  [,2]
## No    6.34 5.958
## Yes  10.53 6.239
##
##   max_temp
## Y      [,1]  [,2]
## No   19.94 6.730
## Yes  22.15 5.977
##
##   rainfall
## Y      [,1]  [,2]
## No   1.233 3.788
## Yes  2.190 4.377
....
```

4 Naive Bayes Model Evaluation

Next we evaluate the model.

```
classes    <- predict(model, ds[test, vars], type="class")
acc        <- sum(classes == actual, na.rm=TRUE)/length(actual)
err        <- sum(classes != actual, na.rm=TRUE)/length(actual)
predicted <- predict(model, ds[test, vars], type="raw")[,2]
pred       <- prediction(predicted, ds[test, target])
ate        <- attr(performance(pred, "auc"), "y.values")[[1]]
riskchart(predicted, actual, risks)
```



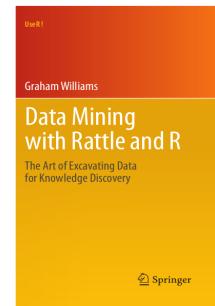
```
round(table(actual, classes, dnn=c("Actual", "Predicted"))/length(actual), 2)

##          Predicted
## Actual      No  Yes
##      No  0.67 0.10
##      Yes 0.08 0.15
```

5 Further Reading

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 module is one of many OnePageR modules available from <http://onepager.togaware.com>. In particular follow the links on the website with a * which indicates the generally more developed OnePageR modules.



6 References

- Meyer D, Dimitriadou E, Hornik K, Weingessel A, Leisch F (2014). *e1071: Misc Functions of the Department of Statistics (e1071)*, TU Wien. R package version 1.6-3, URL <http://CRAN.R-project.org/package=e1071>.
- 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/>.
- 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 3.0.4, URL <http://rattle.togaware.com/>.

This document, sourced from BayesO.Rnw revision 419, was processed by KnitR version 1.6 of 2014-05-24 and took 2.8 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-06-09 10:33:29.

Data Science with R

Multivariate Adaptive Regression Splines

Graham.Williams@togaware.com

3rd August 2014

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

MARS, or Multivariate Adaptive Regression Splines, constructs a linear combination of basis functions for logistic regression.

The required packages for this chapter include:

```
library(rattle)      # The weather dataset and normVarNames().  
library(randomForest) # Impute missing values using na.roughfix().  
library(dplyr)        # Data munging:tbl_df(), %>%.  
library(ROCR)         # Use prediction() to convert to measures.  
library(earth)        # An implementation of mars.
```

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 Data Preparation—Load and Configure

We use the **weather** dataset from **rattle** (Williams, 2014) to illustrate. Refer to Chapter [Data](#) for details.

```
library(rattle)          # Provides weather and normVarNames().
library(dplyr)           # Provides %>% andtbl_df().

dsname      <- "weather"
ds          <- get(dsname) %>%tbl_df()
names(ds)   <- normVarNames(names(ds))
vars        <- names(ds)
target      <- "rain_tomorrow"
risk        <- "risk_mm"
id          <- c("date", "location")

ds

## Source: local data frame [366 x 24]
##
##       date location min_temp max_temp rainfall evaporation sunshine
## 1 2007-11-01 Canberra     8.0     24.3     0.0      3.4      6.3
## 2 2007-11-02 Canberra    14.0     26.9     3.6      4.4     9.7
## 3 2007-11-03 Canberra    13.7     23.4     3.6      5.8     3.3
## 4 2007-11-04 Canberra    13.3     15.5    39.8      7.2     9.1
## 5 2007-11-05 Canberra    7.6     16.1     2.8      5.6    10.6
## 6 2007-11-06 Canberra    6.2     16.9     0.0      5.8     8.2
## 7 2007-11-07 Canberra    6.1     18.2     0.2      4.2     8.4
## 8 2007-11-08 Canberra    8.3     17.0     0.0      5.6     4.6
## 9 2007-11-09 Canberra    8.8     19.5     0.0      4.0     4.1
## 10 2007-11-10 Canberra   8.4     22.8    16.2      5.4     7.7
## ...
## Variables not shown: wind_gust_dir (fctr), wind_gust_speed (dbl),
## wind_dir_9am (fctr), wind_dir_3pm (fctr), wind_speed_9am (dbl),
## wind_speed_3pm (dbl), humidity_9am (int), humidity_3pm (int),
## pressure_9am (dbl), pressure_3pm (dbl), cloud_9am (int), cloud_3pm
## (int), temp_9am (dbl), temp_3pm (dbl), rain_today (fctr), risk_mm (dbl),
## rain_tomorrow (fctr)
```

2 Data Preparation—Variables to Ignore

Here we identify variables that we probably do not want to play a part in the modelling.

```
# Ignore the IDs and the risk variable.
ignore <- union(id, if (exists("risk")) risk)

# Ignore variables that look like identifiers.
ids <- which(sapply(ds, function(x) length(unique(x))) == nrow(ds))
ignore <- union(ignore, names(ids))

# Ignore variables which are completely missing.
mvc <- sapply(ds[vars], function(x) sum(is.na(x))) # Missing value count.
mvn <- names(ds)[(which(mvc == nrow(ds)))] # Missing var names.
ignore <- union(ignore, mvn)

# Ignore variables that are mostly missing - e.g., 70% or more missing
mvn <- names(ds)[(which(mvc >= 0.7*nrow(ds)))]
ignore <- union(ignore, mvn)

# Ignore variables with many levels.
factors <- which(sapply(ds[vars], is.factor))
lvls <- sapply(factors, function(x) length(levels(ds[[x]])))
many <- names(which(lvls > 20)) # Factors with too many levels.
ignore <- union(ignore, many)

# Ignore constants.
constants <- names(which(sapply(ds[vars], function(x) all(x == x[1L]))))
ignore <- union(ignore, constants)

# Initialise the variables
vars <- setdiff(vars, ignore)

vars
## [1] "min_temp"      "max_temp"      "rainfall"
## [4] "evaporation"   "sunshine"      "wind_gust_dir"
## [7] "wind_gust_speed" "wind_dir_9am"  "wind_dir_3pm"
## [10] "wind_speed_9am" "wind_speed_3pm" "humidity_9am"
## [13] "humidity_3pm"   "pressure_9am"  "pressure_3pm"
## [16] "cloud_9am"     "cloud_3pm"    "temp_9am"
## [19] "temp_3pm"      "rain_today"   "rain_tomorrow"

ignore
## [1] "date"       "location"    "risk_mm"
```

3 Data Preparation—Clean and Finalise

The dataset has missing values and the implementation of the algorithm does not support missing values so we impute the missing values here.

```
ds[vars] <- na.roughfix(ds[vars])
```

Now we finalise the meta-data.

```
# Variable roles.
inputc    <- setdiff(vars, target)
inputi    <- sapply(inputc, function(x) which(x == names(ds)), USE.NAMES=FALSE)
numi     <- intersect(inputi, which(sapply(ds, is.numeric)))
numc     <- names(numi)
cati     <- intersect(inputi, which(sapply(ds, is.factor)))
catc     <- names(cati)

# Remove all observations with a missing target.
ds       <- ds[!is.na(ds[target]),]

# Normalise factors.
factors   <- which(sapply(ds[vars], is.factor))
for (f in factors) levels(ds[[f]]) <- normVarNames(levels(ds[[f]]))

# Ensure the target is categoric.
ds[target] <- as.factor(ds[[target]])

# Number of observations.
nobs      <- nrow(ds)
```

4 Build Model

We use `earth` (Milborrow, 2014).

```
library(earth)          # Model builder

# Formula for modelling.
form      <- formula(paste(target, "~ ."))

# Training and test datasets.
seed      <- sample(1:1000000, 1)
set.seed(seed)
train     <- sample(nobs, 0.7*nobs)
test      <- setdiff(seq_len(nobs), train)
actual    <- ds[test, target]
risks     <- ds[test, risk]

# Build model.
m.earth   <- earth(form, data=ds[train, vars])
mtype    <- "earth"
model    <- m.earth

model
## Selected 21 of 94 terms, and 11 of 62 predictors
## Importance: wind_gust_speed, humidity_3pm, min_temp, max_temp, ...
## Number of terms at each degree of interaction: 1 20 (additive model)
## GCV 0.08528    RSS 15.4    GRSq 0.4259    RSq 0.5919
```

5 Evaluate Model with Error Matrix

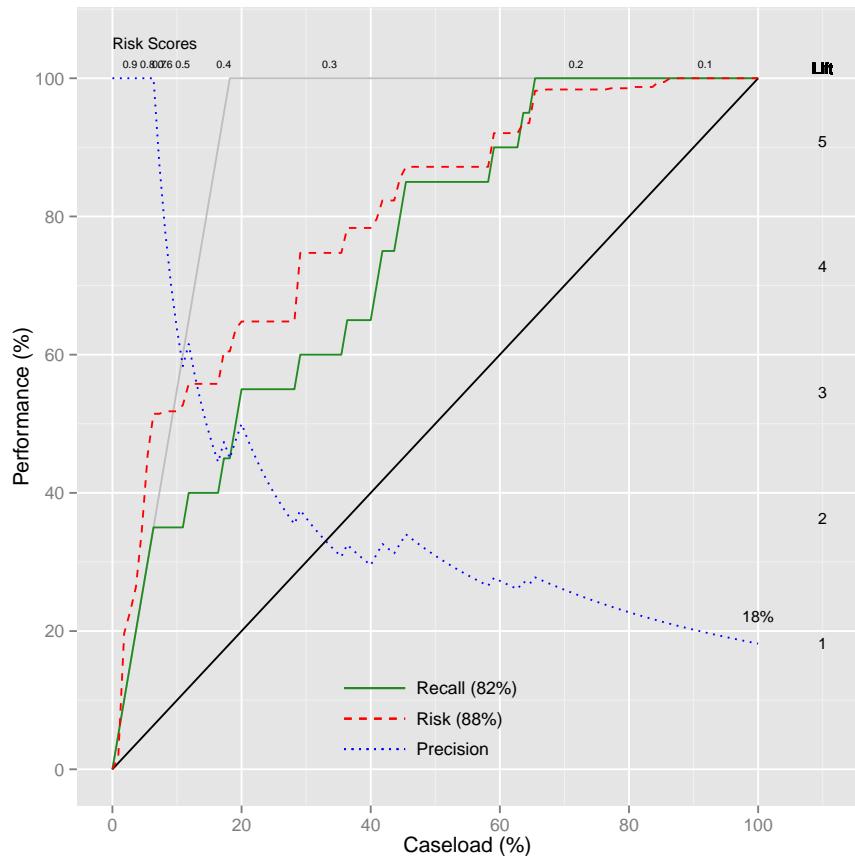
```
library(ROCR)          # prediction()

classes    <- predict(model, ds[test, vars], type="class")
acc        <- sum(classes == actual, na.rm=TRUE)/length(actual)
err        <- sum(classes != actual, na.rm=TRUE)/length(actual)
predicted  <- predict(model, ds[test, vars], type="response")
predicted  <- rescale(predicted, 0:1) # TRY THIS THEN READ DOCS
pred       <- prediction(predicted, ds[test, target])
ate        <- attr(performance(pred, "auc"), "y.values")[[1]]

round(table(actual, classes, dnn=c("Actual", "Predicted"))/length(actual), 2)

##      Predicted
## Actual   No Yes
##   No  0.78 0.04
##   Yes 0.12 0.06
```

6 Evaluate Model with Riskchart



```
library(rattle)      # riskchart()  
riskchart(predicted, actual, risks)
```

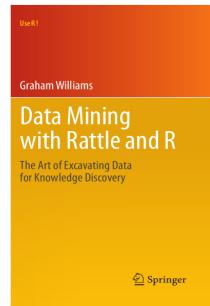
7 Further Reading and Acknowledgements

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:

- <http://www.milbo.org/doc/earth-notes.pdf>



8 References

- Milborrow S (2014). *earth: Multivariate Adaptive Regression Spline Models*. R package version 3.2-7, URL <http://CRAN.R-project.org/package=earth>.
- 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/>.
- 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/>.

This document, sourced from MarsO.Rnw revision 470, was processed by KnitR version 1.6 of 2014-05-24 and took 2.9 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 2014-08-03 17:34:24.

Data Science with R

Evaluating Model Performance

Graham.Williams@togaware.com

16th August 2014

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

This module explores the options for evaluating the performance of models. We introduce techniques for evaluating the performance on a testing dataset, as well as ongoing performance evaluation.

The required packages for this module include:

```
library(rattle)      # Weather, riskchart() and psfchart().
library(dplyr)        # Use tbl_df().
library(e1071)        # naiveBayes().
library(rpart)        # Decision tree model rpart().
library(randomForest) # Impute missing na.roughfix(), and randomForest().
library(wsrpart)      # Build weighted subpart rpart wsrpart().
library(wsrf)         # Build weighted subspace random forest wsrf().
library(gmodels)      # Generate cross-tabulation using CrossTable().
library(ROCR)         # Plot ROC curves.
library(ggplot2)       # Plot ROC curves.
```

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 Prepare Data for Modelling

```

# Identify the dataset
dsname    <- "weatherAUS"
ds         <- tbl_df(get(dsname))
names(ds)  <- normVarNames(names(ds)) # Lower case variable names.
vars       <- names(ds)
target     <- "rain_tomorrow"
risk       <- "risk_mm"
id         <- c("date", "location")

# Ignore the IDs and the risk variable.
ignore     <- c(id, if (exists("risk")) risk)

# Ignore variables which are completely missing.
mvc        <- sapply(ds[vars], function(x) sum(is.na(x)))
mvn        <- names(which(mvc == nrow(ds)))
ignore     <- union(ignore, mvn)

# Initialise the variables
vars       <- setdiff(vars, ignore)

# Variable roles.
inputs     <- setdiff(vars, target)
numi      <- which(sapply(ds[inputs], is.numeric))
numc      <- names(numi)
cati      <- which(sapply(ds[inputs], is.factor))
catc      <- names(cati)

# Remove all observations with a missing target.
ds         <- ds[!is.na(ds[target]),]

# Impute missing values needed for randomForest().
if (sum(is.na(ds[vars]))) ds[vars] <- na.roughfix(ds[vars])

# Ensure the target is categoric.
ds[target] <- as.factor(ds[[target]])

# Number of observations.
nobs      <- nrow(ds)

# Prepare for model building.
form      <- formula(paste(target, "~ ."))
seed      <- 328058
train     <- sample(nobs, 0.7*nobs)
test      <- setdiff(seq_len(nobs), train)
actual    <- ds[test, target]
risks     <- ds[test, risk]

```

2 Build Models

We build a selection of models, including a decision tree (Therneau and Atkinson, 2014), random forest (Breiman *et al.*, 2012), weighted subspace of rpart decision trees (Zhalama and Williams, 2014) and weighted subspace random forest (Meng *et al.*, 2014).

```
# Naive Bayes
library(e1071)
model      <- m.nb <- naiveBayes(form, ds[train, vars])          # 1s
cl.nb      <- predict(model, ds[test, vars], type="class")    # 20s
pr.nb      <- predict(model, ds[test, vars], type="raw")[,2] # 20s

# Decision tree
library(rpart)
model      <- m.rp <- rpart(form, ds[train, vars])           # 6s
cl.rp      <- predict(model, ds[test, vars], type="class")
pr.rp      <- predict(model, ds[test, vars], type="prob")[,2]

# Random forest
library(randomForest)
model      <- m.rf <- randomForest(form, ds[train, vars], ntree=100) # 20s
cl.rf      <- predict(model, ds[test, vars], type="class")
pr.rf      <- predict(model, ds[test, vars], type="prob")[,2]

# Weighted subspace rpart
library(wsrpart)
model      <- m.wsrp <- wsrpart(form, ds[train, vars], ntree=10) # 30s
cl.wsrp    <- predict(model, ds[test, vars], type="class")
pr.wsrp    <- predict(model, ds[test, vars], type="prob")[,2]

# Weighted subspace random forest
library(wsrf)
model      <- m.wsrf <- wsrf(form, ds[train, vars], ntree=10) # 30s
cl.wsrf    <- predict(model, ds[test, vars], type="class")
pr.wsrf    <- predict(model, ds[test, vars], type="prob")[,2]
```

3 Confusion Matrix

A [confusion matrix](#) reports textually the performance of a predictive model's predictions against the actual classes.

We can easily generate a confusion matrix using `table()`. Here we report the actual number of predictions in the four categories of true/false positive/negative.

```
table(actual, cl.nb, dnn=c("Actual", "Predicted"))

##      Predicted
## Actual    No    Yes
##   No 17913 2954
##   Yes 2301 3886

table(actual, cl.rp, dnn=c("Actual", "Predicted"))

##      Predicted
## Actual    No    Yes
##   No 20171 696
##   Yes 3997 2190

table(actual, cl.rf, dnn=c("Actual", "Predicted"))

##      Predicted
## Actual    No    Yes
##   No 19775 1092
##   Yes 2866 3321

table(actual, cl.wsrp, dnn=c("Actual", "Predicted"))

##      Predicted
## Actual    No    Yes
##   No 20143 724
##   Yes 3928 2259

table(actual, cl.wsrf, dnn=c("Actual", "Predicted"))

##      Predicted
## Actual    No    Yes
##   No 19947 920
##   Yes 3328 2859
```

4 Confusion Matrix with Percentage

We can convert to percentages, and include a column to report on the class error rate:

```
pcme <- function(actual, cl)
{
  x <- table(actual, cl)
  tbl <- cbind(round(x/length(actual), 2),
               Error=round(c(x[1,2]/sum(x[1,]), x[2,1]/sum(x[2,])), 2))
  names(attr(tbl, "dimnames")) <- c("Actual", "Predicted")
  tbl
}
pcme(actual, cl.nb)

##      Predicted
## Actual  No  Yes Error
##   No  0.66 0.11  0.14
##   Yes 0.09 0.14  0.37

pcme(actual, cl.rp)

##      Predicted
## Actual  No  Yes Error
##   No  0.75 0.03  0.03
##   Yes 0.15 0.08  0.65

pcme(actual, cl.rf)

##      Predicted
## Actual  No  Yes Error
##   No  0.73 0.04  0.05
##   Yes 0.11 0.12  0.46

pcme(actual, cl.wsrp)

##      Predicted
## Actual  No  Yes Error
##   No  0.74 0.03  0.03
##   Yes 0.15 0.08  0.63

pcme(actual, cl.wsrf)

##      Predicted
## Actual  No  Yes Error
##   No  0.74 0.03  0.04
##   Yes 0.12 0.11  0.54
```

5 Overall and Average Class Error

The overall error is simply the number of observations mis-classified divided by the total number of observations.

```
overall <- function(x) round((x[1,2] + x[2,1]) / sum(x), 2)
overall(table(actual, cl.nb)/length(actual))

## [1] 0.19

overall(table(actual, cl.rp)/length(actual))

## [1] 0.17

overall(table(actual, cl.rf)/length(actual))

## [1] 0.15

overall(table(actual, cl.wsdp)/length(actual))

## [1] 0.17

overall(table(actual, cl.wsrf)/length(actual))

## [1] 0.16
```

The overall error rate is sometimes quite a blunt measure of the performance of a model, and is particularly misleading when the classes are unbalanced. Consider the case where the majority class has an error rate of 10% and the minority class has an error rate of 30%. Overall the error rate will be closer to the 10% error rate because of the sheer number of observations of this class. That is quite misleading given our usual interest in the minority class.

The averaged class error rate is simply the average of the class errors.

```
avgerr <- function(x) round(mean(c(x[1,2], x[2,1])) / apply(x, 1, sum)), 2)
avgerr(table(actual, cl.nb)/length(actual))

## [1] 0.26

avgerr(table(actual, cl.rp)/length(actual))

## [1] 0.34

avgerr(table(actual, cl.rf)/length(actual))

## [1] 0.26

avgerr(table(actual, cl.wsdp)/length(actual))

## [1] 0.33

avgerr(table(actual, cl.wsrf)/length(actual))

## [1] 0.29
```

6 Cross Tabulation Confusion Matrix

A cross-tabulation can be used to generate a confusion-matrix to present the performance of a model. Here we use `CrossTable()` from `gmodels` ([source code and/or documentation contributed by Ben Bolker et al., 2013](#)) to generate the table. This includes the Chi-square test of the independence of all table factors

```
library(gmodels)
CrossTable(actual, cl.nb)

##
##
##   Cell Contents
## |-----|-----|
## |           N |
## | Chi-square contribution |
## |           N / Row Total |
## |           N / Col Total |
## |           N / Table Total |
## |-----|
##
##
## Total Observations in Table:  27054
##
##
##          | cl.nb
## actual |      No |      Yes | Row Total |
## -----+-----+-----+-----+
##       No | 17913 |    2954 | 20867 |
##             | 345.742 | 1021.758 | |
##             | 0.858 | 0.142 | 0.771 |
##             | 0.886 | 0.432 | |
##             | 0.662 | 0.109 | |
## -----+-----+-----+-----+
##       Yes |   2301 |    3886 | 6187 |
##             | 1166.090 | 3446.102 | |
##             | 0.372 | 0.628 | 0.229 |
##             | 0.114 | 0.568 | |
##             | 0.085 | 0.144 | |
## -----+-----+-----+-----+
## Column Total | 20214 |    6840 | 27054 |
##                 | 0.747 | 0.253 | |
## -----+-----+-----+-----|
##
##
```

7 Cross Tabulation: Random Forest

```
CrossTable(actual, cl.rf)

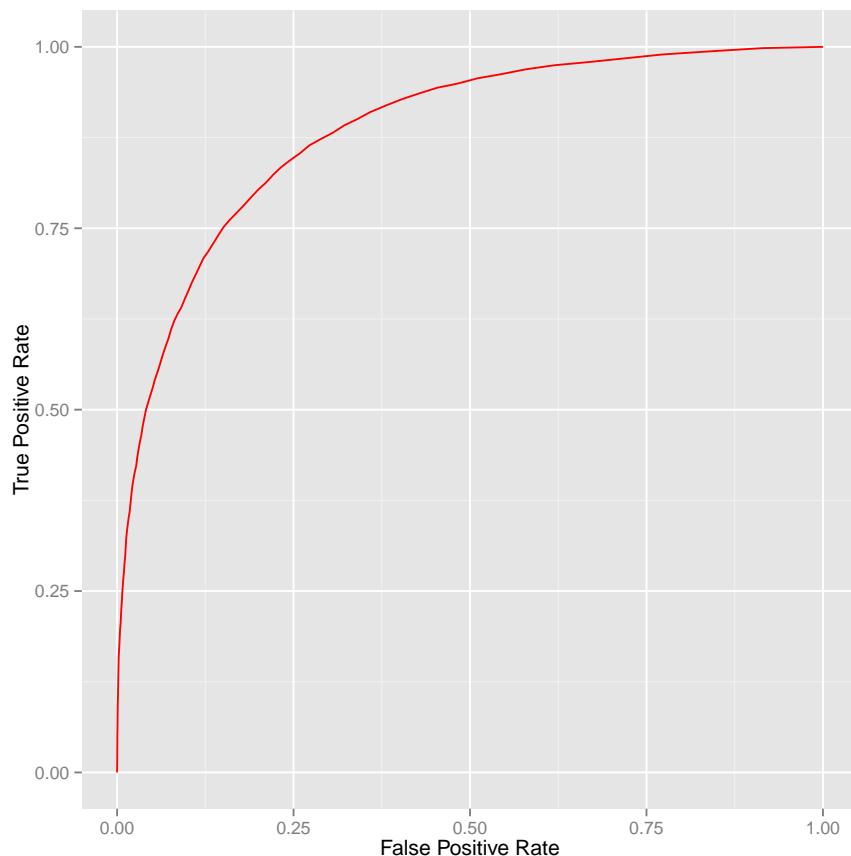
##
##
##      Cell Contents
## |-----|-----|
## |           N |-----|
## | Chi-square contribution |-----|
## |           N / Row Total |-----|
## |           N / Col Total |-----|
## |           N / Table Total |-----|
## |-----|-----|
## 
## 
## Total Observations in Table: 27054
##
##
##          | cl.rf
## actual |-----|-----|-----|-----|
##       No | 19775 | 1092 | 20867 |-----|
##       | 306.035 | 1570.122 |-----|
##       | 0.948 | 0.052 | 0.771 |-----|
##       | 0.873 | 0.247 |-----|
##       | 0.731 | 0.040 |-----|
## -----|-----|-----|-----|
##       Yes | 2866 | 3321 | 6187 |-----|
##       | 1032.171 | 5295.577 |-----|
##       | 0.463 | 0.537 | 0.229 |-----|
##       | 0.127 | 0.753 |-----|
##       | 0.106 | 0.123 |-----|
## -----|-----|-----|-----|
## Column Total | 22641 | 4413 | 27054 |-----|
##       | 0.837 | 0.163 |-----|
## -----|-----|-----|-----|
## 
##
```

8 ROC Curves

The Receiver Operating Characteristics (ROC) curve provides a visual guide to performance. The ROCR ([Sing et al., 2013](#)) package provides the functionality.

```
library(ROCR)
pr <- prediction(pr.rf, actual)
pe <- performance(pr, "tpr", "fpr")
pd <- data.frame(fpr=unlist(pe@x.values), tpr=unlist(pe@y.values))

p <- ggplot(pd, aes(x=fpr, y=tpr))
p <- p + geom_line(colour="red")
p <- p + xlab("False Positive Rate") + ylab("True Positive Rate")
p
```



The ROC curve plots the true positive rate against the false positive rate. Here we see the curve and note that the more area under the curve (AUC) the better is the performance. To interpret, consider the false positive rate of 0.2 which corresponds to a true positive rate of about 0.8. Using the model to prioritise observations, to identify 80% of the true positives, we falsely include 20% of the false positives.

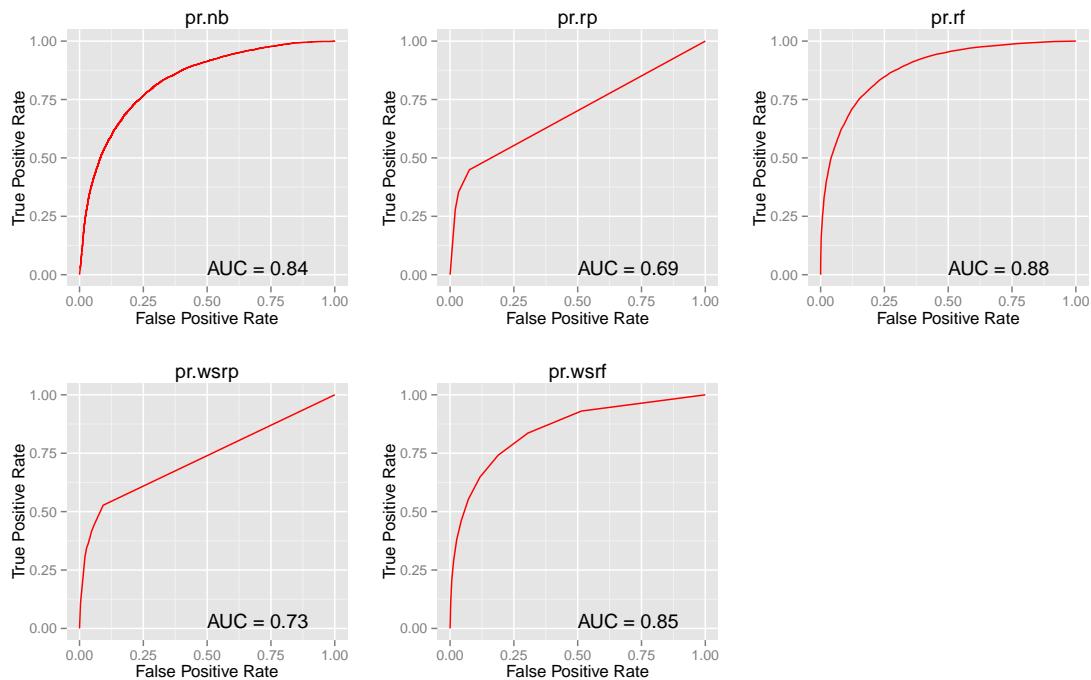
9 ROC Curves—All Models

Here we display all ROC curves. A simple function is used to capture the code that generates the plot from the predictions provided by the model.

```
library(ROCR)
library(gridExtra)

proc <- function(pr.m)
{
  pr <- prediction(pr.m, actual)
  pe <- performance(pr, "tpr", "fpr")
  au <- performance(pr, "auc")@y.values[[1]]
  pd <- data.frame(fpr=unlist(pe@x.values), tpr=unlist(pe@y.values))

  p <- ggplot(pd, aes(x=fpr, y=tpr))
  p <- p + geom_line(colour="red")
  p <- p + xlab("False Positive Rate") + ylab("True Positive Rate")
  p <- p + ggtitle(deparse(substitute(pr.m)))
  p <- p + annotate("text", x=0.50, y=0.00, hjust=0, vjust=0, size=5,
                     label=paste("AUC = ", round(au, 2)))
  return(p)
}
grid.arrange(proc(pr.nb), proc(pr.rp), proc(pr.rf),
             proc(pr.wsrp), proc(pr.wsrf), ncol=3)
```

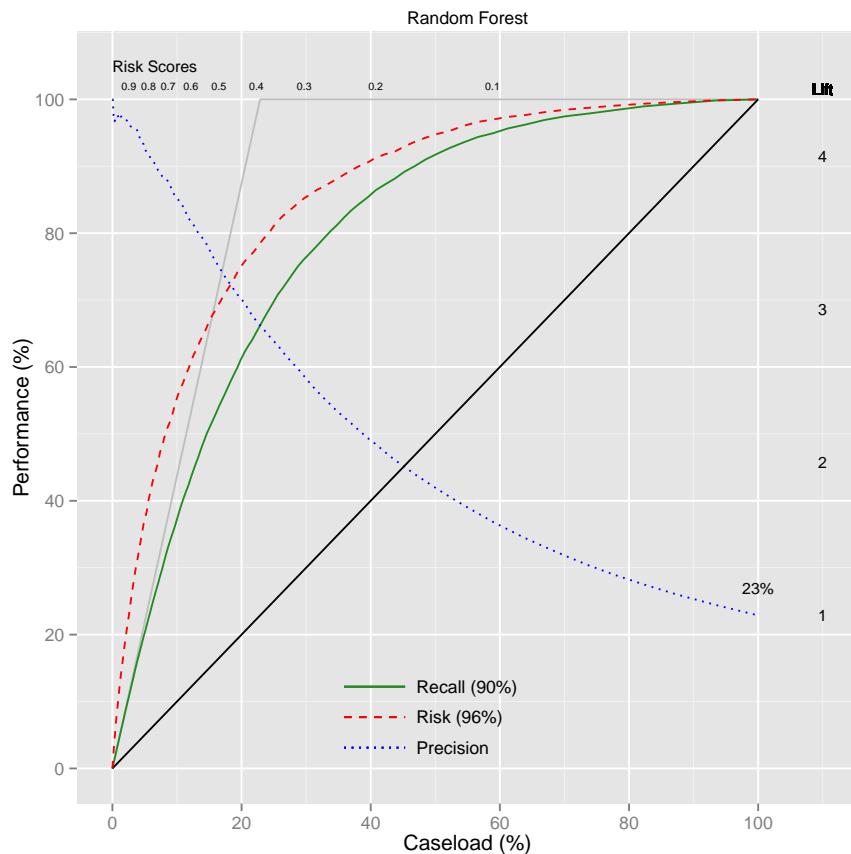


10 ROC Curves—Applied to In-Production Results

ROC curves and the common measure of the area under the curve (AUC) have been shown by Hand (2009) and Fawcett (2003) to have a number of deficiencies. Hari Koesmarno (2013) notes: *Hand (2009) illustrated that the area under the ROC curve (AUC) has a serious deficiency, especially the AUC using different misclassification cost distributions for different classifiers. This means that using the AUC is equivalent to using different metrics to evaluate different classification rules. ROC curves are commonly used in medical decision making and in recent years have been increasingly adopted in machine learning and data mining research communities (Fawcett, 2003). The incoherency of the AUC has been studied by Fawcett (2006) and Hand (2009). However ROC is suited to model comparison and selection with constant cost over a training (rather than an in-production) population.*

Exercise: Include some ROC Curves to illustrate this point.

11 Risk Chart

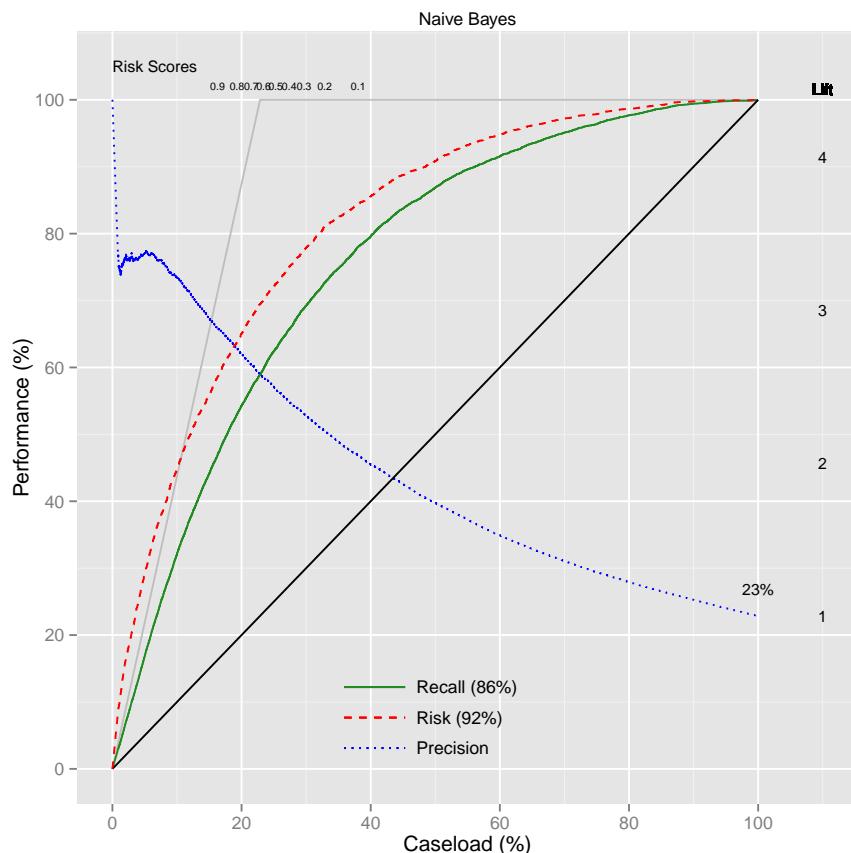


The concept of a risk chart was developed for the Australian Taxation Office, and implemented in `rattle` (Williams, 2014). It is also similar to a cumulative gain chart. We use the random forest model built earlier to generate the risk chart from the test dataset using `riskchart()` from Rattle.

```
riskchart(pr.rf, actual, risks, "Random Forest")
```

The risk chart plots the performance (the true positive rate) against the caseload (the combined observations).

12 Risk Chart: Naive Bayes



```
riskchart(pr.rf, actual, risks, "Random Forest")
```

13 Risk Chart — Interpreting Based on Financial Risk

Although the previous risk chart is actually based on the weather dataset, we will describe it in terms of predicting the risk that a income tax return is not correct. We might imagine 100 tax returns have been randomly chosen for auditing purposes. Of those just 23 have had some requirement to change their tax return and to pay additional taxes. This is the 23% label on the right hand end of the precision plot.

The x-axis is the case load. It represents the ordering by risk score of the tax payers. Those tax payers with the highest score begin the queue at the left hand end of the axis and those with the lowest scores at the right hand end of the axis. Thinking of the 100 taxpayers that have been risk scored standing in a queue, they might be numbered from 1 to 100, and form the queue from the highest risk score (on the left) to the lowest (on the right).

The distribution of the risk scores generated by the model are shown along the top of the chart. We can see the higher risk scores on the left and the lower scores to the right.

The y-axis is a measure of the performance of the model in identifying the tax payers who required an adjustment to be made to their tax return. A performance of 50%, for example, is then 50% of the 23 tax payers (11 or 12 tax payers) whose tax return required an adjustment.

The black diagonal line is the performance we obtain if we randomly selected tax payers from among the 100 chosen for auditing. If we randomly chose 40 tax payers (a caseload of 40%) then we would expect to have a performance of 40%. That is, we expect to identify 40% of the 23 tax payers requiring adjustments (9 tax payers)

The solid green and the dashed red lines are then the performance lift we obtain by using the model. It is measured as the recall (of the tax payers requiring adjustments) and the risk (which is the additional amount of tax the tax payer needs to pay). We use the model to prioritise the tax payers rather than selecting them at random. If we now select 40% of the tax payers again, but we chose those with the highest risk score, then we now expect to get about 84% of the 23 tax payers requiring adjustments (i.e., 19 of these tax payers). That is quite a lift over the 9 adjustment tax payers identified if we had selected the 40 to audit at random.

As we can see, using the model to select 40 tax payers delivers us twice as many of the tax payers that we are really interested in. This is a lift of 2, and we can read the lift off of the dotted blue precision line against the right hand axis. The precision or strike rate is actually the proportion of the 40 cases selected that are the target tax payers. We can see that it is close the 50%, using the left hand y-axis labels.

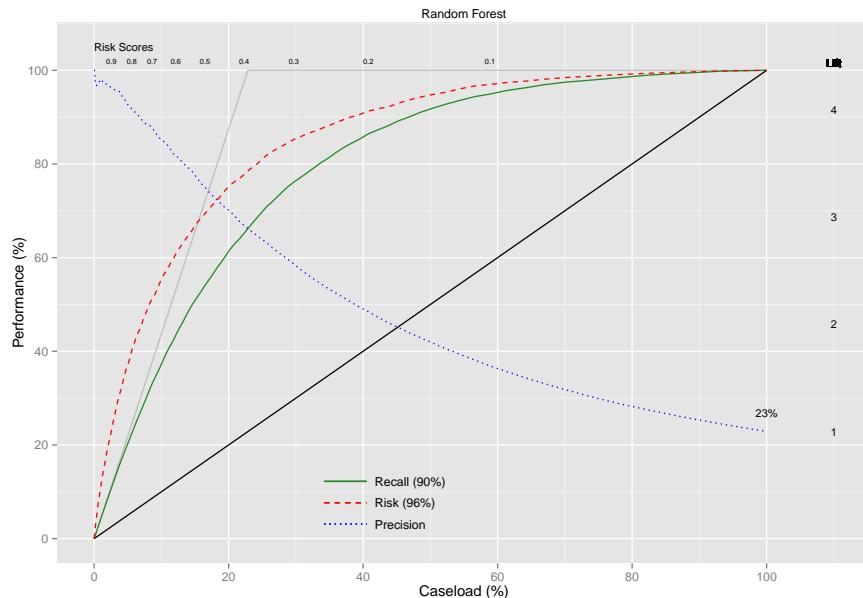
14 Risk Chart—AUC Interpretation

The area under a risk chart curve is calculated relative to a perfect model. The performance of a perfect model is captured as the grey line we see in the risk chart. The grey line will have two segments. The first segment is drawn from the origin to an intercept on the line where $y=100$. The intercept is at the overall strike rate of the training data. The second segment simply terminates at the top right corner of the plot.

The grey line would be the plot we would obtain for a perfectly accurate model. All and only positive observations are given the higher scores by the model, and then the lower scores are given only to negative observations. If this were the case for a model, then by the time we have processed the strike rate number of highest scored observations, we will have covered all of the positive observations.

In the risk chart below the strike rate is 23%. That is, 23% of the observations (which is of course also 23% of caseload) are positive observations. Thus, the grey line's internal point is at 100% performance after 23% of the caseload—a perfect model.

```
print(riskchart(pr.rf, actual, risks, "Random Forest"))
```



The area under the curve that is reported for a risk chart is then the area under the curve relative to this grey line, rather than relative to the whole chart. For an ROC curve, the perfect model would be one where the first segment of the grey line runs along the y-axis to 100, and with $x=0$. Note that for an ROC curve, x is the false positive rate, rather than the caseload. Thus the AUC for and ROC is relative to the whole chart (which is the same as being relative to the grey line on the ROC chart since it actually encompasses the whole chart).

15 Risk Chart—Actual Interpretation for Weather

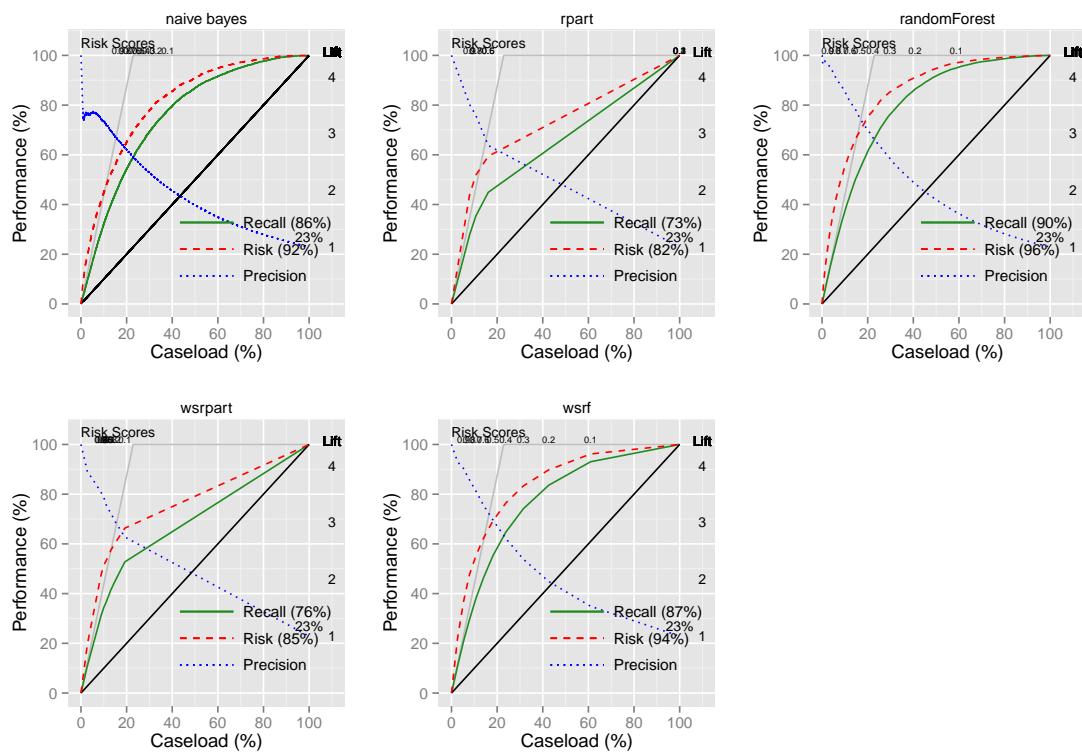
The models we have built are actually based on the weatherAUS dataset.

Exercise: Interpret the model based on the domain of predicting rain_tomorrow.

16 Risk Chart—All Models

Display all Risk Charts as separate plots on a grid.

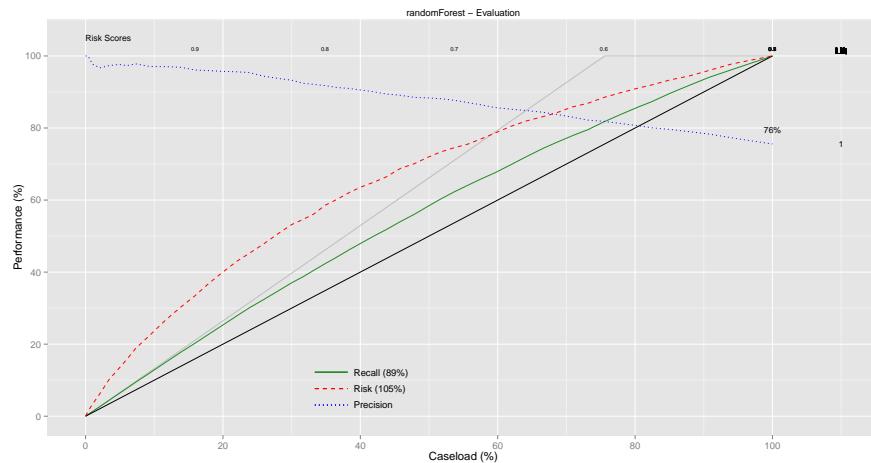
```
library(gridExtra)
rc.nb <- riskchart(pr.nb, actual, risks, "naive bayes")
rc.rp <- riskchart(pr.rp, actual, risks, "rpart")
rc.rf <- riskchart(pr.rf, actual, risks, "randomForest")
rc.wsrp <- riskchart(pr.wsrp, actual, risks, "wsrpart")
rc.wsrft <- riskchart(pr.wsrft, actual, risks, "wsrf")
grid.arrange(rc.nb, rc.rp, rc.rf, rc.wsrp, rc.wsrft, ncol=3)
```



17 Risk Charts—Evaluate Ongoing Model Deployment

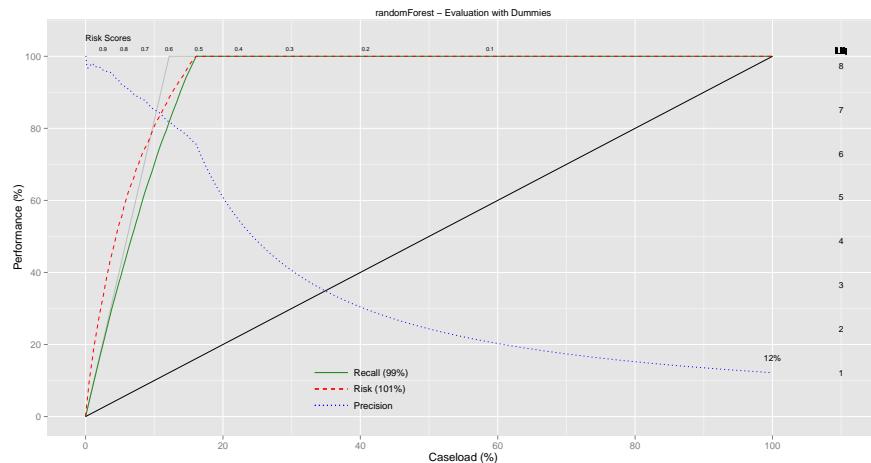
A risk chart evaluates a model based on test data. When deployed the model may be used to filter out observations (with a low risk score) and so they have no outcome recorded (some may be false negatives). A risk chart of just the (high risk) audited cases can be misleading as in this example where only those observations with a risk score greater than 0.5 are selected.

```
pi <- which(pr.rf > 0.5)
riskchart(pr.rf[pi], actual[pi], risks[pi], "randomForest - Evaluation")
```



If we have available all scored observations and treat those not audited as true negatives, we obtain a clearly too optimistic chart.

```
actual0 <- actual; risks0 <- risks; actual0[-pi] <- "No"; risks0[-pi] <- 0
riskchart(pr.rf, actual0, risks0, "randomForest - Evaluation with Dummies")
```



The key is to note the presence of the grey line (the maximal performance line) and the corresponding relative area under the curve calculation. These provide a clearer understanding of the performance of the model on this censored evaluation dataset.

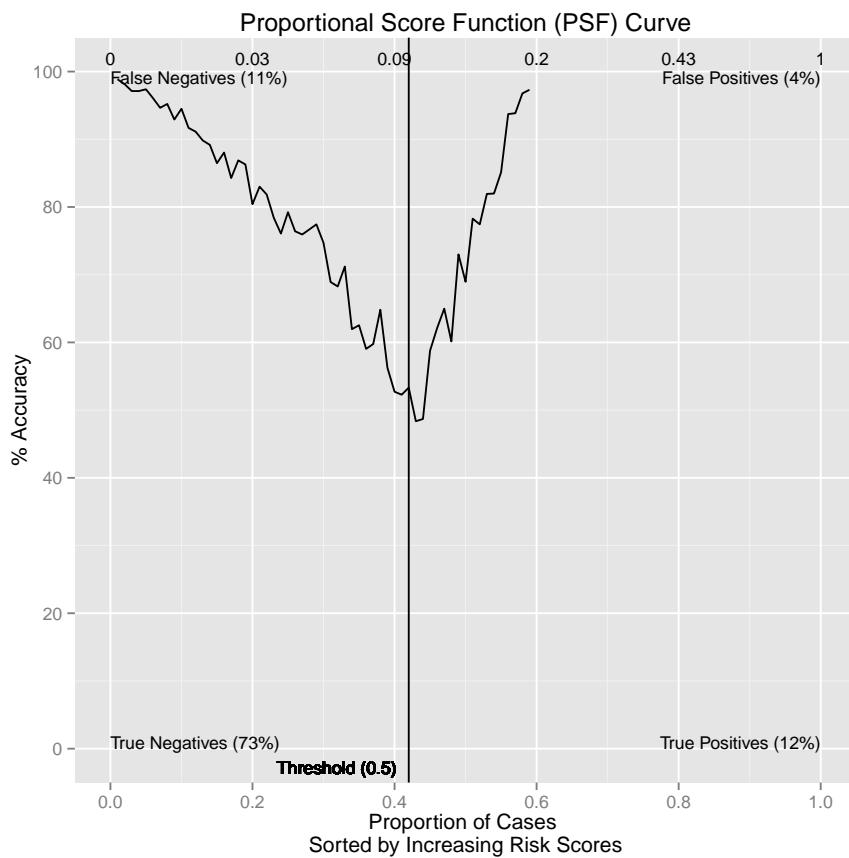
18 PSF Chart

The Proportional Score Function (PSF) chart is a useful tool for evaluating the ongoing performance of a model. We can think of it as a visualisation of a confusion matrix, dividing the plot into 4 regions corresponding to true/false positives/negatives. The idea was developed from the statistical techniques developed by [Koesmarno \(1996\)](#).

We saw previously that a Risk Chart is not appropriate for the visualisation of the ongoing performance of a deployed model. What we see in such a risk chart is the performance on the high risk cases (those selected for action). A PSF Chart is a good alternative for measuring classifier performance.

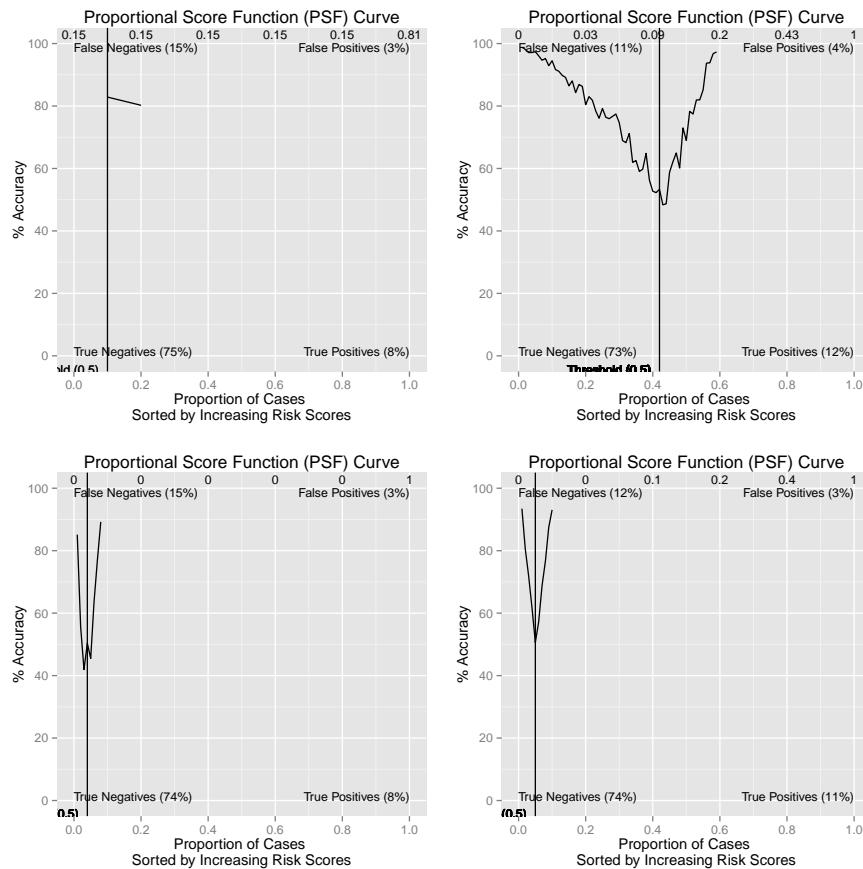
A PSF Chart displays the performance of a model for a chosen cutoff or threshold above which risk scores are regarded to predict the positive class. By default, that the threshold is the traditional 0.5. The curve can provide insights into accuracy and the degradation in the performance of the classifier.

```
print(psfchart(pr.rf, actual))
```



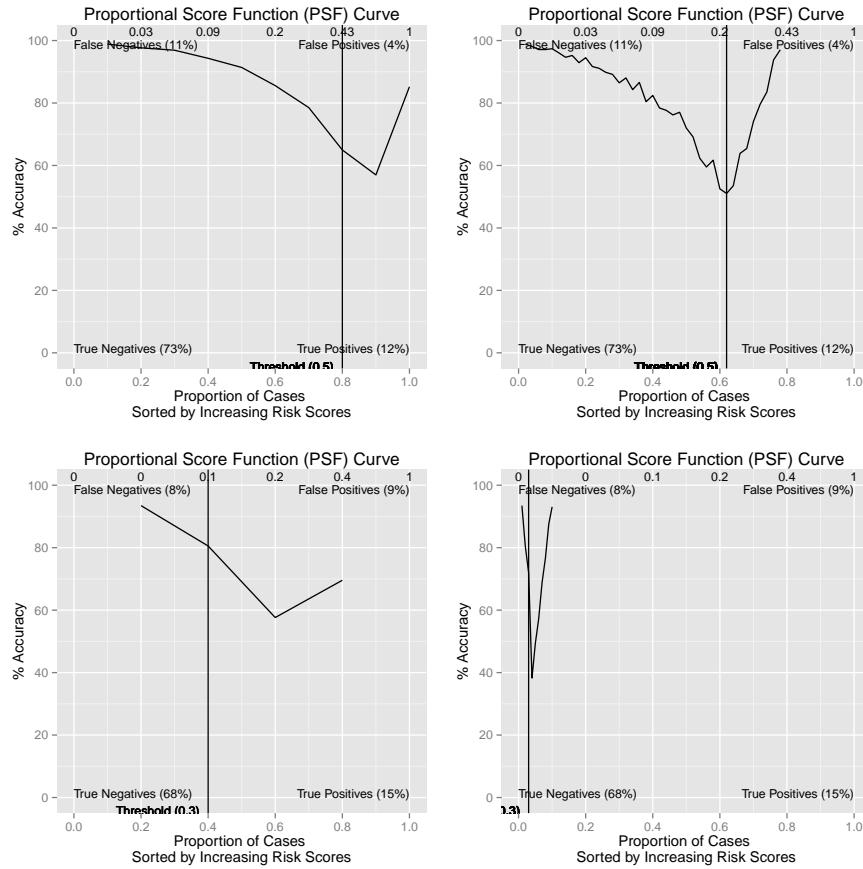
19 PSF Chart—All Models

```
psf.rp <- psfchart(pr.rp, actual, bins=10)
psf.rf <- psfchart(pr.rf, actual)
psf.wsrp <- psfchart(pr.wsrp, actual)
psf.wsrf <- psfchart(pr.wsrf, actual)
grid.arrange(psf.rp, psf.rf, psf.wsrp, psf.wsrf)
```



20 PSF Chart—Options

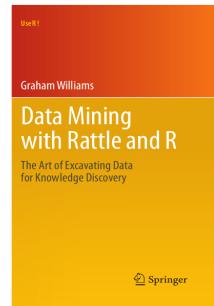
```
psf.rf1 <- psfchart(pr.rf, actual, bins=10)
psf.rf2 <- psfchart(pr.rf, actual, bins=50)
psf.wsr1 <- psfchart(pr.wsr, actual, bins=5, threshold=0.3)
psf.wsr2 <- psfchart(pr.wsr, actual, bins=100, threshold=0.3)
grid.arrange(psf.rf1, psf.rf2, psf.wsr1, psf.wsr2)
```



21 Further Reading

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.



22 References

- Breiman L, Cutler A, Liaw A, Wiener M (2012). *randomForest: Breiman and Cutler's random forests for classification and regression*. R package version 4.6-7, URL <http://CRAN.R-project.org/package=randomForest>.
- Koesmarno HK (1996). “Class-size percentile transformation for reconstructing a distribution function.” *Journal of Applied Statistics*, **23**(4), 423–434.
- Meng Q, Zhao H, Williams GJ (2014). *wsrf: Weighted Subspace Random Forest*. R package version 1.3.17.
- 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/>.
- Sing T, Sander O, Beerenwinkel N, Lengauer T (2013). *ROCR: Visualizing the performance of scoring classifiers*. R package version 1.0-5, URL <http://CRAN.R-project.org/package=ROCR>.
- source code and/or documentation contributed by Ben Bolker GRWIR, Lumley T, Johnson RC, are Copyright SAIC-Frederick RCJ, by the Intramural Research Program IF, of the NIH, Institute NC, for Cancer Research under NCI Contract NO1-CO-12400 C (2013). *gmodels: Various R programming tools for model fitting*. R package version 2.15.4.1, URL <http://CRAN.R-project.org/package=gmodels>.
- Therneau TM, Atkinson B (2014). *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-8, URL <http://CRAN.R-project.org/package=rpart>.
- 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/>.
- Zhalama, Williams GJ (2014). *wsrpart: Build weighted subspace rpart decision trees*. R package version 1.2.151.

This document, sourced from EvaluateO.Rnw revision 484, was processed by KnitR version 1.6 of 2014-05-24 and took 188 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 2014-08-16 11:14:12.

Scoring

Chapter pending / not available

PMML Exporting Models for Deployment

Chapter pending / not available

One Page R Data Science Strings

Graham.Williams@togaware.com

29th July 2018

Visit <https://essentials.togaware.com/onepagers> for more Essentials.

Wrangling strings of characters is something we will find ourselves doing often as data scientists. R provides a comprehensive set of tools for handling and processing strings. In this chapter we review the functionality provided by R for managing and manipulating strings.

20180602

Through this guide new R commands will be introduced. The reader is encouraged to review the command's documentation and understand what the command does. Help is obtained using the ? command as in:

```
?read.csv
```

Documentation on a particular package can be obtained using the help= option of library():

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

This chapter is intended to be hands on. To learn effectively you are encouraged to run R (e.g., RStudio or Emacs with ESS mode) and to replicate the commands. Check that output is the same and that you understand how it is generated. Try some variations. Explore.

Copyright © 2000-2018 Graham Williams. This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#) allowing this work to be copied, distributed, or adapted, with attribution and provided under the same license.



1 Packages Used

Packages used in this chapter include `dplyr` (Wickham *et al.*, 2018), `glue` (Hester, 2017), `magrittr` (Bache and Wickham, 2014), `stringr` (Wickham, 2018), `stringi` (Gagolewski *et al.*, 2018), `scales` (Wickham, 2017), and `rattle.data` (Williams, 2017b).

```
# Load required packages from local library into R session.

library(dplyr)          # Wrangling: mutate().
library(stringi)         # The string concat operator %s+%.
library(stringr)         # String manipulation.
library(glue)            # Format strings.
library(magrittr)        # Pipelines for data processing: %>% %T>% %<%.
library(rattle.data)     # Weather dataset.
library(scales)          # commas(), percent().
```

2 Concatenate Strings

One of the most basic operations in string manipulation is the concatenate operation. R provides alternatives for doing so but a modern favourite is the `stringi::%s+%` operator.

```
"abc" %s+% "def" %s+% "ghi"
## [1] "abcdefghi"

c("abc", "def", "ghi", "jkl") %s+% c("mno")
## [1] "abcmno" "defmno" "ghimno" "jklmno"

c("abc", "def", "ghi", "jkl") %s+% c("mno", "pqr")
## [1] "abcmno" "defpqr" "ghimno" "jklpqr"

c("abc", "def", "ghi", "jkl") %s+% c("mno", "pqr", "stu", "vwx")
## [1] "abcmno" "defpqr" "ghistu" "jklvwx"
```

The tidy function for concatenating strings is `stringr::str_c()`. A `sep=` can be used to specify a separator for the concatenated strings.

```
str_c("hello", "world")
## [1] "helloworld"

str_c("hello", "world", sep=" ")
## [1] "hello world"
```

We can also concatenate strings using `glue::glue()`.

```
glue("hello", "world")
## helloworld
```

The traditional `base::cat()` function returns the concatenation of the supplied strings. Numeric and other complex objects are converted into character strings.

```
cat("hello", "world")
## hello world

cat ("hello", 123, "world")
## hello 123 world
```

Yet another alternative (and there are many) is the function `base::paste()`. Notice that it separates the concatenated strings with a space.

```
paste("hello", "world")
## [1] "hello world"
```

3 Concatenate Strings Special Cases

Each operator/functions treats NULL differently. Note the convenience for `base::cat()` to add a space between the strings, and that `base::paste()` treats NULL as a zero length string, and thus there are two spaces between the words concatenated.

```
"hello" %s+% NULL %s+% "world"
## character(0)

str_c("hello", NULL, "world")
## [1] "helloworld"

glue("hello", NULL, "world")

cat("hello", NULL, "world")
## hello world

paste("hello", NULL, "world")
## [1] "hello world"
```

NA tends to be treated differently too.

```
"hello" %s+% NA %s+% "world"
## [1] NA

str_c("hello", NA, "world")
## [1] NA

glue("hello", NA, "world")
## helloNAworld

cat("hello", NA, "world")
## hello NA world

paste("hello", NA, "world")
## [1] "hello NA world"
```

The examples becomes more interesting in the context that the arguments to the functions might be string returning functions. If that function returns NULL or NA, purposely or accidentally then it is useful to know the consequences.

4 String Length

The tidy way to get the length of a string is `stringr::str_length()`.

20180606

```
str_length("hello world")
## [1] 11
str_length(c("hello", "world"))
## [1] 5 5
str_length(NULL)
## integer(0)
str_length(NA)
## [1] NA
```

The function `base::nchar()` is the traditional approach.

```
nchar("hello world")
## [1] 11
nchar(c("hello", "world"))
## [1] 5 5
nchar(NULL)
## integer(0)
nchar(NA)
## [1] NA
```

5 Case Conversion

Often during data transformations strings have to be converted from one case to the other. These simple transformations can be achieved by `base::tolower()` and `base::toupper()`. The `base::casefold()` function can also be used as a wrapper to the two functions.

```
toupper("String Manipulation")
## [1] "STRING MANIPULATION"
tolower("String Manipulation")
## [1] "string manipulation"
casefold("String Manipulation")
## [1] "string manipulation"
casefold("String Manipulation", upper=TRUE)
## [1] "STRING MANIPULATION"
```

6 Tidy Sub-String Operations

We will find ourselves often wanting to extract or modify sub-strings within a string. The tidy way to do this is with the `stringr::str_sub()` function. We can specify the `start=` and the `end=` of the string. The indices of the string start from 1.

```
s <- "string manipulation"
str_sub(s, start=3, end=6)

## [1] "ring"

str_sub(s, 3, 6)
## [1] "ring"
```

A negative is used to count from the end of the string.

```
str_sub(s, 1, -8)
## [1] "string manip"
```

Replacing a sub-string with another string is straightforward using the assignment operator.

```
str_sub(s, 1, -8) <- "stip"
s

## [1] "stipation"
```

The function also operates over a vector of strings.

```
v <- c("string", "manipulation", "always", "fascinating")
str_sub(v, -4, -1)

## [1] "ring" "tion" "ways" "ting"

str_sub(v, -4, -1) <- "RING"
v

## [1] "stRING"        "manipulaRING" "alRING"       "fascinaRING"
```

7 Base Sub-String Operations

The base function `base::substr()` can be used to extract and replace parts of a string similar to `stringr::str_sub()`. Note however that it does not handle negative values and that string replacement only replaces the same length as the replacement string, without changing the length of the original string.

```
s <- "string manipulation"
substr(s, start=3, stop=6)

## [1] "ring"

substr(s, 3, 6)

## [1] "ring"

substr(s, 1, 12) <- "stip"
s

## [1] "stipng manipulation"
```

The `base::substring()` function performs similarly though uses `last=` rather than `stop=`.

```
s <- "string manipulation"
substring(s, first=3, last=6)

## [1] "ring"

x <- c("abcd", "aabcb", "babcc", "cabcd")
substring(x, 2, 4)

## [1] "bcd" "abc" "abc" "abc"

substring(x, 2, 4) <- "AB"
x

## [1] "aABd"  "aABcb" "bABcc" "cABcd"
```

8 Trim and Pad

One of the major challenges of string parsing is removing and adding whitespaces and wrapping text.

Additional white space can be present on the left, right or both sides of the word. The `stringr::str_trim()` function offers an effective way to get rid of these whitespaces.

```
ws <- c(" abc", "def ", " ghi ")
str_trim(ws)

## [1] "abc" "def" "ghi"

str_trim(ws, side="left")
## [1] "abc" "def" "ghi "

str_trim(ws, side="right")
## [1] " abc" "def" " ghi"

str_trim(ws, side="both")
## [1] "abc" "def" "ghi"
```

Conversely we can also pad a string with additional characters for up to a specified width using `stringr::str_pad()`. The default padding character is a space but we can override that.

```
str_pad("abc", width=7)
## [1] "     abc"

str_pad("abc", width=7, side="left")
## [1] "     abc"

str_pad("abc", width=7, side="right")
## [1] "abc     "

str_pad("abc", width=7, side="both", pad="#")
## [1] "##abc##"
```

9 Wrapping and Words

Formatting a text string into a neat paragraph of defined maximum width is another operation we often find ourselves wanting. The `stringr::str_wrap()` function will do this for us.

```
st <- "All the Worlds a stage, All men are merely players"
cat(str_wrap(st, width=25))

## All the Worlds a stage,
## All men are merely
## players
```

Words of course form the basis for wrapping a sentence. We may wish to extract words from a sentence ourselves for further processing. Here we us `stringr::word()` to do so. We specify the positions of the word to be extracted from the sentence. The default separator value is space.

```
st <- c("The quick brown fox", "jumps on the brown dog")
word(st, start=1, end=2)

## [1] "The quick" "jumps on"

word(st, start=1, end=-2)

## [1] "The quick brown"      "jumps on the brown"
```

10 Glue Strings Together

The `glue` package provides a mechanism for building output strings from a collection of strings and variables. The basic use of `glue::glue()` will concatenate its string arguments with variable substitution identified using curly braces. In this example we use the `rattle.data::weatherAUS` and format large numbers using `scales::comma()`.

```
dsname <- "weatherAUS"
nobs   <- nrow(weatherAUS)
starts <- min(weatherAUS$Date)
glue("The {dsname} dataset",
     " has just less than {comma(nobs + 1)} observations,",
     " starting from {format(starts, '%-d %B %Y')}.")
## The weatherAUS dataset has just less than 145,461 observations, starting f...
```

We can manually wrap the sentence.

```
glue("
  The {dsname} dataset has just
  less than {comma(nobs + 1)} observations
  starting from {format(starts, '%-d %B %Y')}.
")

## The weatherAUS dataset has just
## less than 145,461 observations
## starting from 1 November 2007.
```

Notice how the initial and last empty lines are handled “as expected”, and the line split is maintained.

Named arguments within the function call can be used to assign values to variables that only exist in the scope of the function call.

```
glue("
  The {dsname} dataset has just
  less than {comma(nobs + 1)} observations
  starting from {format(starts, '%-d %B %Y')}.
  ",
  dsname = "weather",
  nobs   = nrow(weather),
  starts = min(weather$Date))

## The weather dataset has just
## less than 367 observations
## starting from 1 November 2007.
```

We can also see the effect of indenting lines in this example, where the indentation is retained.

11 Pipeline Glue

We can use `glue::glue_data()` within pipes and operate over the rows of the data that is piped into the operator.

```
weatherAUS %>%
  sample_n(6) %>%
  glue_data("Observation",
            "rownames(.) %>% as.integer() %>% comma() %>% sprintf('%7s', .)",
            "location {Location %>% sprintf('%-14s', .)}",
            "max temp {MaxTemp %>% sprintf('%5.1f', .)}")

## Observation 133,068 location Launceston      max temp 23.8
## Observation 136,307 location AliceSprings    max temp 23.2
## Observation 41,622 location Williamtown      max temp 23.2
## Observation 120,795 location Perth           max temp 24.2
## Observation 93,346 location Townsville       max temp 29.6
## Observation 75,506 location Portland          max temp 14.3
....
```

It can also be useful with the tidyverse work flow.

```
weatherAUS %>%
  sample_n(6) %>%
  mutate(TempRange = glue("{MinTemp}-{MaxTemp}")) %>%
  glue_data("Observed temperature range at {Location} of {TempRange}")

## Observed temperature range at Woomera of 6.5-17.6
## Observed temperature range at NorahHead of 17.7-26.3
## Observed temperature range at Townsville of 19.8-26.7
## Observed temperature range at Nuriootpa of 9.8-33.1
## Observed temperature range at MelbourneAirport of 15.3-26.6
## Observed temperature range at Nuriootpa of 3.6-22.1
....
```

12 Pattern Matching with Regular Expressions

One of the most powerful string processing concepts is the concept of regular expressions. A regular expression is a sequence of characters that describe a pattern. The concept was formalized by American mathematician Stephen Cole Kleene. A regular expression pattern can contain a combination of alphanumeric and special characters. It is a complex topic and we take an introductory look at it here to craft regular expressions in R.

An important concept is that of metacharacters which have special meaning within a regular expression. Unlike other characters that are used to match themselves, metacharacters have a specific meaning. The following table shows a list of metacharacters used in regular expressions.

	Metacharacter	Description
1	<code>^</code>	Matches at the start of the string
2	<code>\$</code>	Matches at the end of the string
3	<code>()</code>	Defines a subexpression to be matched and retrieved later.
4	<code> </code>	Matches the pattern before or pattern after
5	<code>[]</code>	Matches a single character that is contained within bracket
6	<code>.</code>	Matches any single character

Such metacharacters are used to match different patterns.

```
s <- c("hands", "data", "on", "data$cienc", "handsondata$cienc", "handson")
grep(pattern="^data", s, value=TRUE)
## [1] "data"      "data$cienc"
grep(pattern="on$", s, value=TRUE)
## [1] "on"        "handson"
grep(pattern="(nd)..(nd)", s, value=TRUE)
## [1] "handsondata$cienc"
```

In order to match a metacharacter in R we need to escape it with `\` (double backslash).

```
grep(pattern="\$\$", s, value=TRUE)
## [1] "data$cienc"      "handsondata$cienc"
```

13 Regular Expressions: Quantifiers

Quantifiers are used to match repetition of a pattern within a string. The following table shows a list of quantifiers.

	Quantifier	Description
1	*	The preceding item is matched 0 or more times
2	+	The preceding item is matched 1 or more times
3	?	The preceding item is matched at most 1 times.
4	{n}	The preceding item is matched n times.
5	{n,}	The preceding item is matched atleast n times.

Some examples will illustrate.

```
s <- c("aaab", "abb", "bc", "abcd", "bbc", "bab", "caa")
grep(pattern="ab*b", s, value=TRUE)
## [1] "aaab"   "abb"    "abcd"   "bab"
grep(pattern="abbc?", s, value=TRUE)
## [1] "abb"    "abcd"
grep(pattern="b{2,}?", s, value=TRUE)
## [1] "abb"    "abcd"   "bbc"
```

14 Regular Expressions: Character Classes

A character class is a collection of characters that are in some way grouped together. We enclose the characters to be grouped within square brackets `[]`. The pattern then matches any one of the characters in the set. For example, the character class `[0-9]` matches any of the digits from 0 to 9.

	Character Class	Description
1	<code>[0-9]</code>	Digits
2	<code>[a-z]</code>	Lower-case letters
3	<code>[A-Z]</code>	Upper-case letters
4	<code>[a-zA-Z]</code>	Alphabetic characters
5	<code>[^a-zA-Z]</code>	Non-alphabetic characters
6	<code>[a-zA-Z0-9]</code>	Alphanumeric characters
7	<code>[\n\t\f\v]</code>	Space characters
8	<code>[!,:;`)]{@-}\$_?[^{ (\\"#\&~_/<=>)]</code>	Punctuation characters

```
s <- c("abc12", "@#$", "345", "ABcd")
grep(pattern="[0-9]+", s, value=TRUE)
## [1] "abc12" "345"

grep(pattern="[A-Z]+", s, value=TRUE)
## [1] "ABcd"

grep(pattern="[^@#]+", s, value=TRUE)
## [1] "abc12" "345"    "ABcd"
```

R also supports the use of POSIX character classes which are represented within `{}[]` (double braces).

```
grep(pattern="[:alpha:]", s, value=TRUE)
## [1] "abc12" "ABcd"

grep(pattern="[:upper:]", s, value=TRUE)
## [1] "ABcd"
```

15 Generate Strings for Testing

It is sometimes very useful to be able to test out some code using some test data. A simple way to generate test strings us with `stringi::stri_rand_lipsum()`.

```
stri_rand_lipsum(20)

## [1] "Lorem ipsum dolor sit amet, posuere at in in id ligula sodales eget ..."
## [2] "Sapien augue dignissim, vulputate, montes ipsum rutrum eu eu porta f..."
## [3] "Ultrices ante in commodo eu id elementum velit ut bibendum. Nisl, la...
## [4] "Dis aptent senectus rhoncus et sed donec, vitae posuere, neque. Arcu...
## [5] "Vestibulum molestie in donec tincidunt, eu sapien. Quam in curae cla...
## [6] "Leo, ac integer sed penatibus. Curabitur, neque habitant quam, dui c...
## [7] "Vulputate elementum in urna. Ut nunc sed, imperdiet. Suscipit eu int...
## [8] "Senectus justo. Lobortis mauris praesent taciti. Massa ultrices in v...
## [9] "Nec at sapien phasellus nec eros quis ligula ac vestibulum, eu lorem...
## [10] "Sit, vestibulum est, velit ultrices nisi porta aliquam non in. Monte...
## [11] "Risus, sit metus augue non. Quisque sed amet, ac libero tempus sed n...
## [12] "Sed porttitor, eu amet ex amet nibh mauris, venenatis sed nec. Netus...
## [13] "Tellus himenaeos at convallis tincidunt sit. Metus sit mauris mus si...
## [14] "Massa in in potenti tellus rutrum orci donec fames. Ut elit in moles...
## [15] "In ac fermentum amet mus. In ridiculus augue elit fermentum, ornare....
## [16] "Sed ultricies vel consequat aliquet magnis nisl tortor. Nisl eu, gra...
## [17] "Scelerisque sagittis consequat tempor iaculis sociis commodo. Hac ma...
## [18] "Non a interdum per malesuada enim potenti cum. Auctor ut purus egest...
## [19] "Interdum vel ut eros. Sed, dictumst laoreet curabitur nec, cursus ar...
## [20] "Leo sed ut at lacinia lacus enim felis in ultrices. Imperdiet feugia...

stri_rand_lipsum(2)

## [1] "Lorem ipsum dolor sit amet, sem, aliquam duis arcu. Nam magna, non ve...
## [2] "Sed, ac hac primis aenean. Fames neque maecenas sed ligula velit. Eu ...

sapply(stri_rand_lipsum(10), nchar, USE.NAMES=FALSE)

## [1] 514 527 617 740 653 630 690 680 789 417

sapply(stri_rand_lipsum(10), nchar, USE.NAMES=FALSE)

## [1] 502 473 615 360 813 761 629 547 290 301
```

The strings generated are of different lengths and each call generates different strings.

16 Read a File as Vector of Strings

There may be occasions where we would like to load a dataset from a file as strings, one line as a string, returning a vector of strings. We can achieve using the function `base::readLines()`. IN the following example we access the system file `weather.csv` that is provided by the `rattle` (?) package.

```
dsname <- "weather" # Dataset name.
ftype <- "csv"      # Source dataset file type.
dsname %s+%
".%" %s+%
ftype %T>%
print() %>%
system.file(ftype, ., package="rattle") %>%
readLines() ->
ds
## [1] "weather.csv"
```

A sample of the data.

```
head(ds)

## [1] "\"Date\"", \"Location\", \"MinTemp\", \"MaxTemp\", \"Rainfall\", \"Evaporat...
## [2] "2007-11-01,\\\"Canberra\\\",8,24.3,0,3.4,6.3,\\\"NW\\\",30,\\\"SW\\\",\\\"NW\\\",6,20...
## [3] "2007-11-02,\\\"Canberra\\\",14,26.9,3.6,4.4,9.7,\\\"ENE\\\",39,\\\"E\\\",\\\"W\\\",4,....
## [4] "2007-11-03,\\\"Canberra\\\",13.7,23.4,3.6,5.8,3.3,\\\"NW\\\",85,\\\"N\\\",\\\"NNE\\\"...
## [5] "2007-11-04,\\\"Canberra\\\",13.3,15.5,39.8,7.2,9.1,\\\"NW\\\",54,\\\"WNW\\\",\\\"W\\\"...
## [6] "2007-11-05,\\\"Canberra\\\",7.6,16.1,2.8,5.6,10.6,\\\"SSE\\\",50,\\\"SSE\\\",\\\"ES...
....
```

Find those strings that contain a specific pattern using `base::grep()`.

```
grep("ENE", ds)

## [1]   3   10  23  26  28  36  37  42  43  49  50  54  68  69  71  76  86  91
## [19]  97 101 103 106 108 109 110 118 129 132 133 135 138 145 160 171 176 215
## [37] 222 278 303 304 310 323 341 348 351 357 365

grep("ENE", ds, value=TRUE)

## [1] "2007-11-02,\\\"Canberra\\\",14,26.9,3.6,4.4,9.7,\\\"ENE\\\",39,\\\"E\\\",\\\"W\\\",4...
## [2] "2007-11-09,\\\"Canberra\\\",8.8,19.5,0,4,4.1,\\\"S\\\",48,\\\"E\\\",\\\"ENE\\\",19,1...
## [3] "2007-11-22,\\\"Canberra\\\",16.4,19.4,0.4,9.2,0,\\\"E\\\",26,\\\"ENE\\\",\\\"E\\\",6...
## [4] "2007-11-25,\\\"Canberra\\\",15.4,28.4,0,4.4,8.1,\\\"ENE\\\",33,\\\"SSE\\\",\\\"NE\\\"...
## [5] "2007-11-27,\\\"Canberra\\\",13.3,22.2,0.2,6.6,2.3,\\\"ENE\\\",39,\\\"E\\\",\\\"E\\\"...
## [6] "2007-12-05,\\\"Canberra\\\",14.5,21.8,0,8.4,9.8,\\\"ENE\\\",43,\\\"ESE\\\",\\\"E\\\"...
....
```

17 Command Summary

This chapter has introduced, demonstrated and described the following R packages, functions, commands, operators, and datasets:

Currently incomplete.

%s+% *Operator from stringr.* Concatenate strings with no separator between the strings.

cat() *Function from base.* Concatenate strings with a space separator between the strings by default. Add **sep=** to specify a different or no separator.

gregexpr() *Function from base.* Returns all matches of pattern in string.

grep() *Function from base.* Returns index of elements that matched.

grepl() *Function from base.* Returns boolean values indicating if a pattern exist in the string.

gsub() *Function from base.* Replaces all matches of pattern with replacement.

paste() *Function from base.* Concatenate strings by pasting them together.

regexec() *Function from base.* Combines results of `regexp()` and `gregexpr()`.

regexp() *Function from base.* Returns the first match of the pattern in string.

str_c() *Function from stringr.* Tidy version of concatenate strings.

strsplit() *Function from data.table.* Split string in to vector according to pattern match.

str_detect() *Function from stringr.* Detect a presence or absence of a pattern in a string.

str_extract() *Function from stringr.* Extracts first occurrence of pattern in string..

str_extract_all() *Function from stringr.* Extracts all occurrence of pattern in string..

str_match() *Function from stringr.* Extract first matched group from a string.

str_match_all() *Function from stringr.* Extract all matched groups from a string.

str_locate() *Function from stringr.* Locate the position of the frst occurence of a pattern in a string.

str_locate_all() *Function from stringr.* Locate the position of all occurrences of a pattern in a string.

str_replace() *Function from stringr.* Returns the first match of the pattern in string.

str_replace_all() *Function from stringr.* Returns all matches of pattern in string.

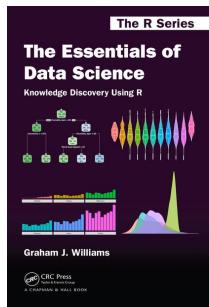
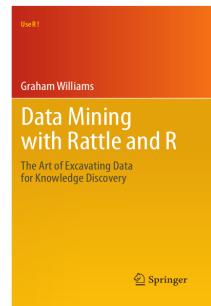
str_split() *Function from stringr.* Split up a string into a variable number of pieces.

str_split_fixed() *Function from stringr.* Split up a string into a fixed number of pieces.

sub() *Function from base.* Replaces the first match of pattern with replacement.

18 Further Reading and Acknowledgements

The [Rattle](#) book ([Williams, 2011](#)), published by Springer, provides a comprehensive introduction to data mining and analytics using Rattle and R. It is available from [Amazon](#). Rattle provides a graphical user interface through which the user is able to load, explore, visualise, and transform data, and to build, evaluate, and export models. Through its Log tab it specifically aims to provide an R template which can be exported and serve as the starting point for further programming with data in R.



The [Essentials of Data Science](#) book ([Williams, 2017a](#)), published by CRC Press, provides a comprehensive introduction to data science through programming with data using R. It is available from [Amazon](#). The book provides a template based approach to doing data science and knowledge discovery. Templates are provided for data wrangling and model building. These serve as generic starting points for programming with data, and are designed to require minimal effort to get started. Visit <https://essentials.togaware.com> for further guides and templates.

Other resources include:

- [Handling and Processing Strings in R](#), a freely available ebook by Gaston Sanchez from 2013.
- <http://www.rexamine.com/2013/04/properly-internationalized-regular-expressions-in-r/>

Some of the material has been updated from material collected by Karthik Bharadwaj.

19 References

- Bache SM, Wickham H (2014). *magrittr: A Forward-Pipe Operator for R*. R package version 1.5, URL <https://CRAN.R-project.org/package=magrittr>.
- Gagolewski M, Tartanus B, , other contributors; IBM, other contributors; Unicode, Inc (2018). *stringi: Character String Processing Facilities*. R package version 1.2.3, URL <https://CRAN.R-project.org/package=stringi>.
- Hester J (2017). *glue: Interpreted String Literals*. R package version 1.2.0, URL <https://CRAN.R-project.org/package=glue>.
- R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- Wickham H (2017). *scales: Scale Functions for Visualization*. R package version 0.5.0, URL <https://CRAN.R-project.org/package=scales>.
- Wickham H (2018). *stringr: Simple, Consistent Wrappers for Common String Operations*. R package version 1.3.1, URL <https://CRAN.R-project.org/package=stringr>.
- Wickham H, François R, Henry L, Müller K (2018). *dplyr: A Grammar of Data Manipulation*. R package version 0.7.6, URL <https://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.
- Williams GJ (2017a). *The Essentials of Data Science: Knowledge discovery using R*. The R Series. CRC Press.
- Williams GJ (2017b). *rattle.data: Rattle Datasets*. R package version 1.0.2, URL <https://CRAN.R-project.org/package=rattle.data>.

This document, sourced from StringsO.Rnw bitbucket revision 276, was processed by KnitR version 1.20 of 2018-02-20 10:11:46 UTC and took 7.4 seconds to process. It was generated by gjw on Ubuntu 18.04 LTS.

One PageR Data Science

Dates and Time

Graham.Williams@togaware.com

16th May 2018

Visit <https://essentials.togaware.com/onepagers> for more Essentials.

Date and time data is common in many disciplines, particularly where our observations are of some event over time. R has well developed support for dealing with such data, in wrangling the data, summarising the data and analysing the data. In this chapter we review the common tasks when dealing with date and time data.

20180513

Packages used in this chapter include tidyverse (Wickham, 2017), lubridate (Spinu *et al.*, 2018), magrittr (Bache and Wickham, 2014), rattle (Williams, 2017b), WDI (Arel-Bundock, 2018b), countrycode (Arel-Bundock, 2018a),

```
# Load required packages from local library into the R session.

library(tidyverse)      # ggplot2, tibble, tidyr, readr, purrr, dplyr
library(lubridate)       # Dates and time.
library(magrittr)        # Pipe operator %>% %<>% %T>% equals().
library(rattle)          # cocomcat().
library(WDI)             # World Bank Data
library(countrycode)
library(gridExtra)
```

Through this guide new R commands will be introduced. The reader is encouraged to review the command's documentation and understand what the command does. Help is obtained using the ? command as in:

```
?read.csv
```

Documentation on a particular package can be obtained using the help= option of library():

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

This chapter is intended to be hands on. To learn effectively the reader is encouraged to run R locally (e.g., RStudio or Emacs with ESS mode) and to replicate all commands as they appear here. Check that output is the same and it is clear how it is generated. Try some variations. Explore.

Copyright © 2000-2018 Graham Williams. This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#) allowing this work to be copied, distributed, or adapted, with attribution and provided under the same license.



1 Dataset with Dates

Often when ingesting data into R we will have a dataset in a CSV file that contains dates. An example is the **stroke** dataset. Here we set up template variables (`dsname`, `dsloc`, and `dspath`), and whilst constructing the value of the path variable we print the constructed path and display the first few lines from the CSV file for our observation.

```
# Name of the dataset.

dsname <- "stroke"

# Identify the source location of the dataset.

dsloc <- "data"

# Construct the path to the dataset and display some if it.

dsname %>%
  paste0(".csv") %>%
  file.path(dsloc, .) %T>%
  cat("Dataset:", ., "\n\n") %T>%
{
  paste("head", .) %>%
  system(intern=TRUE) %>%
  sub("\r", "\n", .) %>%
  print()
} ->
dspath

## Dataset: data/stroke.csv
##
## [1] "SEX;DIED;DSTR;AGE;DGN;COMA;DIAB;MINF;HAN\n"
## [2] "1;7.01.1991;2.01.1991;76;INF;0;0;1;0\n"
## [3] "1;.;3.01.1991;58;INF;0;0;0;0\n"
## [4] "1;2.06.1991;8.01.1991;74;INF;0;0;1;1\n"
## [5] "0;13.01.1991;11.01.1991;77;ICH;0;1;0;1\n"
## [6] "0;23.01.1996;13.01.1991;76;INF;0;1;0;1\n"
## [7] "1;13.01.1991;13.01.1991;48;ICH;1;0;0;1\n"
## [8] "0;1.12.1993;14.01.1991;81;INF;0;0;0;1\n"
## [9] "1;12.12.1991;14.01.1991;53;INF;0;0;1;1\n"
## [10] "0;.;15.01.1991;73;ID;0;0;0;1\n"
```

Observe that this CSV file actually uses semicolons rather than commas to separate the fields. This is common in countries where the comma is used to separate the decimal digits from the whole digits. We can use `readr::read_csv2()` to ingest such a CSV file.

Also observe that the two date columns `DIED` and `DSTR` are in a particular (even peculiar) format which we might determine to be day then month then year, separated by a period.

Finally, observe that missing values appear to be represented as a single period.

2 Ingest Semicolon Separated Dataset

We can now ingest the data into R using `readr::read_csv2()` with an appropriate `na=".,"`.

20180514

```
dspath %>%
  read_csv2(na=".,") %T>%
  glimpse() %>%
  assign(dsname, ., .GlobalEnv)

## Using ',' as decimal and '.' as grouping mark.
## Use read_delim() for more control.

## Parsed with column specification:
## cols(
##   SEX = col_integer(),
##   DIED = col_number(),
##   DSTR = col_number(),
##   AGE = col_integer(),
##   DGN = col_character(),
##   COMA = col_integer(),
##   DIAB = col_integer(),
##   MINF = col_integer(),
##   HAN = col_integer()
## )

## Observations: 829
## Variables: 9
## $ SEX <int> 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, ...
## $ DIED <dbl> 7011991, NA, 2061991, 13011991, 23011996, 130...
## $ DSTR <dbl> 2011991, 3011991, 8011991, 11011991, 13011991...
## $ AGE <int> 76, 58, 74, 77, 76, 48, 81, 53, 73, 69, 86, 7...
## $ DGN <chr> "INF", "INF", "INF", "ICH", "INF", "ICH", "IN...
## $ COMA <int> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ DIAB <int> 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ MINF <int> 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, ...
## $ HAN <int> 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, ...
```

Notice the message indicating that comma is interpreted as a decimal separator and period as a grouping mark. This is based on the fact of the use of semicolon as the field separator and the usual motivation for doing so (comma used for decimal). For this dataset those assumptions don't actually hold. Observe that the two date columns have been treated as numbers under this assumption where the string of numbers with multiple periods is interpreted as numeric.

We decide to follow the suggestion to use `readr::read_delim()` which provides more control over the ingestion.

Nonetheless we observe for the first time that the dataset consists of 829 observations of 9 variables. There appear to be a number of binary encoded variables, that may indicate they represent TRUE and FALSE, whilst SEX would appear to encode male/female as 0/1 or 1/0, though without further information we do not know which of these encoding it is.

3 Basic Dataset Ingestion

The function `readr::read_delim()` makes fewer assumptions about the data and in this case will be a better option. We can specify the field delimiter using `delim=";"` whilst also retaining the missing value argument.

```
dspath %>%
  read_delim(delim=";", na=".") %T>%
  glimpse() %>%
  assign(dsname, ., .GlobalEnv)

## Parsed with column specification:
## cols(
##   SEX = col_integer(),
##   DIED = col_character(),
##   DSTR = col_character(),
##   AGE = col_integer(),
##   DGN = col_character(),
##   COMA = col_integer(),
##   DIAB = col_integer(),
##   MINF = col_integer(),
##   HAN = col_integer()
## )

## Observations: 829
## Variables: 9
## $ SEX <int> 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, ...
## $ DIED <chr> "7.01.1991", NA, "2.06.1991", "13.01.1991", ...
## $ DSTR <chr> "2.01.1991", "3.01.1991", "8.01.1991", "11.01...
## $ AGE <int> 76, 58, 74, 77, 76, 48, 81, 53, 73, 69, 86, 7...
....
```

That is now a good start to ingesting this data into R. The dates will be wrangled shortly but ingesting them as character strings retains their format.

Following our template approach we copy the dataset to our template variable (`ds`) so that we can work on the data using the generic template name.

```
# Prepare the dataset for usage with our template.

ds <- get(dsname)
```

4 Normalise the Dataset

```
# Review the variables to optionally normalise their names.

names(ds)

## [1] "SEX"   "DIED"  "DSTR"  "AGE"   "DGN"   "COMA"  "DIAB"  "MINF"
## [9] "HAN"

# Normalise the variable names.

names(ds) %<>% normVarNames() %T>% print()

## [1] "sex"   "died"  "dstr"  "age"   "dgn"   "coma"  "diab"  "minf"
## [9] "han"

# Confirm the results are as expected.

glimpse(ds)

## Observations: 829
## Variables: 9
## $ sex <int> 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, ...
## $ died <chr> "7.01.1991", NA, "2.06.1991", "13.01.1991", ...
## $ dstr <chr> "2.01.1991", "3.01.1991", "8.01.1991", "11.01...
## $ age <int> 76, 58, 74, 77, 76, 48, 81, 53, 73, 69, 86, 7...
....
```

5 Text to Date Conversion Using Lubridate

We notice that there are two variables that look like dates: `died` and `dstr`.

They have been read in as character strings. Because the format of the dates is not an obvious date format the `readr::read_delim()` has not recognised them as dates. For more standard formats any date columns will be automatically identified.

The `lubridate` ([Spinu et al., 2018](#)) package can perform the conversion into a date format for us here. We can convert the dates using `lubridate::dmy()` since the source format appears to be day, month, the year.

```
ds$died %<>% dmy() %T>% {class(.) %>% print()}

## [1] "Date"

ds$dstr %<>% dmy() %T>% {class(.) %>% print()}

## [1] "Date"

glimpse(ds)

## Observations: 829
## Variables: 9
## $ sex <int> 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, ...
## $ died <date> 1991-01-07, NA, 1991-06-02, 1991-01-13, 1996...
## $ dstr <date> 1991-01-02, 1991-01-03, 1991-01-08, 1991-01-...
## $ age <int> 76, 58, 74, 77, 76, 48, 81, 53, 73, 69, 86, 7...
....
```

The data types are now `Date`.

6 Text to Date Conversion Using Base R

As an alternative we could have used `as.Date()` to convert them into a Date class. Because the original format is not automatically recognised by `as.Date()` we need to tell it the format using `format=`.

```
tmp <- stroke

tmp$DIED %>% as.Date(format = "%d.%m.%Y") %T>% {class(.) %>% print()}

## [1] "Date"

tmp$DSTR %>% as.Date(format = "%d.%m.%Y") %T>% {class(.) %>% print()}

## [1] "Date"

glimpse(tmp)

## Observations: 829
## Variables: 9
## $ SEX <int> 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, ...
## $ DIED <date> 1991-01-07, NA, 1991-06-02, 1991-01-13, 1996...
## $ DSTR <date> 1991-01-02, 1991-01-03, 1991-01-08, 1991-01...
## $ AGE <int> 76, 58, 74, 77, 76, 48, 81, 53, 73, 69, 86, 7...
...
rm(tmp)
```

Notice now that the dates are printed in a standard ISO format which is `%Y-%m-%d` and is strongly suggested as the preferred format so as to remove any ambiguity .

7 POSIXct and POSIXlt

Objects of class `POSIXct` (calendar time) and `POSIXlt` (local time) represent calendar dates and times. They both represent the same information, but in different ways, calendar time as a single number and local time as a vector of the components making up the date/time. Both `POSIXct` and `POSIXlt` objects are also `POSIXt` objects, thus effectively inheriting from the common class `POSIXt`, allowing operations on mixed class (`POSIXct` and `POSIXlt`) objects. Generally, for data frames we use `POSIXct`. `POSIXlt` is more directly accessible for us to read.

`POSIXct` is simply the number of seconds since 1 January 1970.

```
(ct <- Sys.time())
## [1] "2018-05-16 12:37:47 +08"
class(ct)
## [1] "POSIXct" "POSIXt"
str(ct)
##  POSIXct[1:1], format: "2018-05-16 12:37:47"
unclass(ct)
## [1] 1526445467
```

`POSIXlt` (local time) represents the date and time as a named list of vectors.

```
(ct <- as.POSIXlt(ct))
## [1] "2018-05-16 12:37:47 +08"
class(ct)
## [1] "POSIXlt" "POSIXt"
str(ct)
##  POSIXlt[1:1], format: "2018-05-16 12:37:47"
unclass(ct)
## $sec
## [1] 47.44999
##
## $min
## [1] 37
##
## $hour
## [1] 12
##
## $mday
## [1] 16
##
....
```

8 Formatting Dates

A wide variety of formats are supported in printing a date and time. The format string is a common standard used with many applications.

To print a date/time to a specific format we specify the format with in the call to `format()`:

```
format(Sys.time(), "%a %d %b %Y %H:%M:%S %Z")
## [1] "Wed 16 May 2018 12:37:47 +08"
```

The table below illustrates many of the available options.

<code>%c</code>	date and time	Wed 16 May 2018 12:37:47 +08
<code>%x</code>	date	16/05/18
<code>%F</code>	ISO 8601	2018-05-16
<code>%d/%m/%Y</code>	day/month/year	16/05/2018
<code>%a %e %m %Y</code>	day month year	Wed 16 May 2018
<code>%A %d %B %Y</code>	day month year	Wednesday 16 May 2018
<code>Day %j and Week %U of %Y</code>	day/week of the year	Day 136 and Week 19 of 2018
<code>%A: Day %w of Week %U</code>	day of week	Wednesday: Day 3 of Week 19
<code>%y%m%d</code>	two digit date stamp	180516
<code>%X</code>	time	12:37:47
<code>%r</code>	time	12:37:47 PM
<code>%k.%M %p</code>	24 hour time	12.37 PM
<code>%l.%M %p</code>	12 hour time	12.37 PM
<code>%H%M%S</code>	timestamp	123747
<code>%I:%M:%S %p</code>	time 12 hour clock	12:37:47 PM
<code>%H:%M:%S %z</code>	time and UTC offset	12:37:47 +0800
<code>%H:%M:%S %Z</code>	time and timezone	12:37:47 +08

There are more! See the help page for `strptime()` for details.

9 Computing on Dates and Times: difftime

R Dates can be used in computations quite naturally.

```
ds$lived <- ds$died - ds$dstr
head(ds$lived)

## Time differences in days
## [1] 5 NA 145 2 1836 0

class(ds$lived)

## [1] "difftime"
```

Similarly POSIXct representations can be computed on, though the results are reported in seconds rather than days, by default. A Date does not include a time, hence we might expect Date calculations to be in days.

```
ds$lived <- ds$died - ds$dstr
head(ds$lived)

## Time differences in days
## [1] 5 NA 145 2 1836 0

class(ds$lived)

## [1] "difftime"

as.integer(ds$lived[1])/60/60/24
## [1] 5.787037e-05
```

We can change the default displayed units if desired.

```
units(ds$lived)

## [1] "days"

units(ds$lived) <- "days"
units(ds$lived)

## [1] "days"

head(ds$lived)

## Time differences in days
## [1] 5 NA 145 2 1836 0
```

10 Lubridate Intervals

```
ds$interval <- with(ds, interval(dstr, died))
head(ds$interval)

## [1] 1991-01-02 UTC--1991-01-07 UTC
## [2] 1991-01-03 UTC--NA
## [3] 1991-01-08 UTC--1991-06-02 UTC
## [4] 1991-01-11 UTC--1991-01-13 UTC
## [5] 1991-01-13 UTC--1996-01-23 UTC
## [6] 1991-01-13 UTC--1991-01-13 UTC
...
class(ds$interval)

## [1] "Interval"
## attr(,"package")
## [1] "lubridate"

max(ds$interval, na.rm=TRUE)

## [1] 158630400

min(ds$interval, na.rm=TRUE)

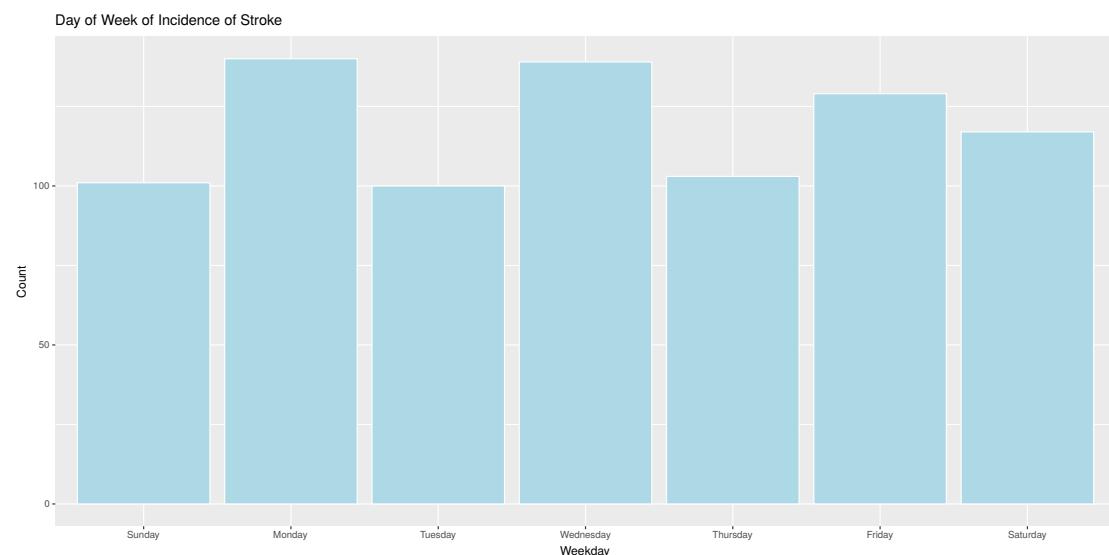
## [1] 0

head(as.duration(ds$interval))

## [1] "432000s (~5 days)"      NA
## [3] "12528000s (~20.71 weeks)" "172800s (~2 days)"
## [5] "158630400s (~5.03 years)" "0s"
```

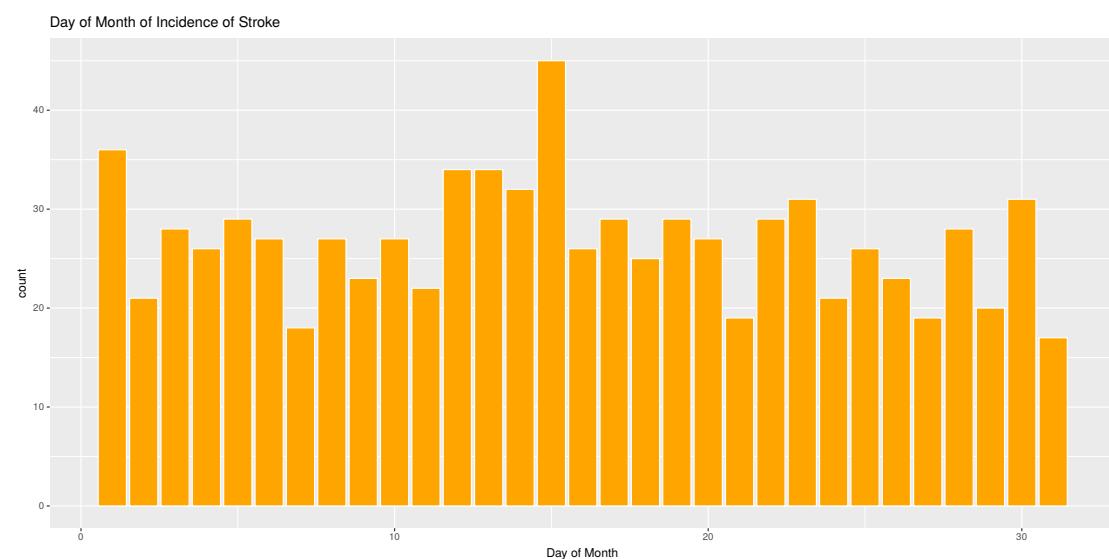
11 Plot Day of Week Frequencies

```
ds %>%
  mutate(weekday=wday(dstr, label=TRUE, abbr=FALSE)) %>%
  ggplot(aes(weekday)) +
  geom_bar(colour="white", fill="lightblue") +
  labs(title="Day of Week of Incidence of Stroke", x="Weekday", y="Count")
```



12 Plot Day of Month Frequencies

```
ds %>%
  mutate(mday=mday(dstr)) %>%
  ggplot(aes(mday)) +
  geom_bar(colour="white", fill="orange") +
  labs(title="Day of Month of Incidence of Stroke", x="Day of Month")
```



13 Plot Daily Observations

```
data.frame(L1=100+c(0, cumsum(runif(99, -20, 20))),  
           L2=150+c(0, cumsum(runif(99, -10, 10))),  
           Date=seq.Date(as.Date("2000-01-01"),  
                         by="1 month", length.out=100)) %>%  
  melt(id="Date") %>%  
  ggplot(aes(x=Date, y=value, colour=variable)) +  
  geom_line() +  
  ylab("Observation") +  
  labs(colour="Location")  
  
## Error in melt(., id = "Date"): could not find function "melt"
```

14 Plot World Bank Data: Obtain Data

This example was inspired by the [ProgrammingR](#) blog post of 14 May 2013.

The World Bank provide economic indicators on the Internet available via an API. We can access the data using `WDI` ([Arel-Bundock, 2018b](#)). We also use `countrycode` ([Arel-Bundock, 2018a](#)) to map the country codes.

We search the World Bank data for the fertility rate data using `WDIsearch()`. We identify the countries we are interested in, convert them to their two character country codes and then extract the country data from the World Bank for a ten year period.

```
(meta.data <- WDIsearch("Fertility rate", field="name", short=FALSE))

##      indicator
## [1,] "SP.FER.TOTL.ZR"
## [2,] "SP.DYN.WFRT.Q5"
## [3,] "SP.DYN.WFRT.Q4"
## [4,] "SP.DYN.WFRT.Q3"
## [5,] "SP.DYN.WFRT.Q2"
...
(indicators <- meta.data[1:2, 1])
## [1] "SP.FER.TOTL.ZR" "SP.DYN.WFRT.Q5"

countries <- c("United States", "Britain", "India", "China", "Australia")
(iso2char <- countrycode(countries, "country.name", "iso2c"))

## [1] "US" "GB" "IN" "CN" "AU"

(wdids <- WDI(iso2char, meta.data[1:2,1], start=2001, end=2011))

## Warning in WDI(iso2char, meta.data[1:2, 1], start = 2001, end = 2011): Unable
to download indicators SP.FER.TOTL.ZR

##      iso2c      country SP.DYN.WFRT.Q5 year
## 1      AU      Australia      NA 2011
## 2      AU      Australia      NA 2010
## 3      AU      Australia      NA 2009
## 4      AU      Australia      NA 2008
## 5      AU      Australia      NA 2007
...
```

15 Plot World Bank Data: Multiple Plots

Generate the plots. We generate a list of plots, by applying a function to each indicator. Notice inside the function the call to `ggplot()` uses `environment=environment()` to ensure the variable `nm` is available to the `aes()`.

```
plots <- lapply(indicators, function(nm)
{
  p <- ggplot(wdids, aes(x=year, y=wdids[,nm], group=country, color=country),
               environment=environment())
  p <- p + geom_line(size=1)
  p <- p + scale_x_continuous(name="Year", breaks=c(unique(wdids[, "year"])))
  p <- p + scale_y_continuous(name=nm)
  p <- p + scale_linetype_discrete(name="Country")
  p <- p + theme(legend.title=element_blank())
  p <- p + labs(title=paste(meta.data[meta.data[, 1]==nm, "name"], "\n"))
})
```

Once we have our list of plots, we can call `grid.arrange()` to arrange the plots to be displayed.

```
do.call(grid.arrange, plots)
## Error in '[.data.frame'(wdids, , nm): undefined columns selected
```

16 Time Series Plot

We will prepare a dataset to illustrate a number of options for plotting. We first pick a few variables to plot.

```
vars <- c("Date", "MinTemp", "MaxTemp", "Sunshine", "Rainfall", "Evaporation")
ds <- weather[vars]

## Error in eval(expr, envir, enclos): object 'weather' not found
```

We want to illustrate a common issue with different scales on the one plot, so we convert the hours of sunshine into seconds.

```
ds$Sunshine <- ds$Sunshine * 60

## Warning: Unknown or uninitialized column: 'Sunshine'.

## Error in '$<-.data.frame`('*tmp*', Sunshine, value = numeric(0)): replacement has 0 rows, data has 829
```

We will also accumulate the amount of rainfall and the amount of evaporation over the period:

```
ds$CumRainfall <- cumsum(ds$Rainfall)

## Warning: Unknown or uninitialized column: 'Rainfall'.

## Error in '$<-.data.frame`('*tmp*', CumRainfall, value = numeric(0)): replacement has 0 rows, data has 829

ds$CumEvaporation <- cumsum(ds$Evaporation)

## Warning: Unknown or uninitialized column: 'Evaporation'.

## Error in '$<-.data.frame`('*tmp*', CumEvaporation, value = numeric(0)): replacement has 0 rows, data has 829
```

We now also melt the dataset into a form that will facilitate plotting all of the variables.

```
dsm <- melt(ds, id="Date")

## Error in melt(ds, id = "Date"): could not find function "melt"

g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))

## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)): object 'dsm' not found

g <- g + geom_point()

## Error in eval(expr, envir, enclos): object 'g' not found

print(g)

## Error in print(g): object 'g' not found
```

That's a start, but not real good. The very large numbers swamp the rest. Notice also the warning regarding observations with missing values. We'll ignore that (and turn the warning off for the following plots).

17 Rescale with a Log10 Transform

We can perform a log (base 10) transform to ensure the low valued variables get some resolution in the plot.

```
g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))
## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)): object 'dsm'
not found

g <- g + geom_point()
## Error in eval(expr, envir, enclos): object 'g' not found
g <- g + scale_y_log10()
## Error in eval(expr, envir, enclos): object 'g' not found
print(g)
## Error in print(g): object 'g' not found
```

So that is a little better but note the warnings. We can not take the log of numbers less than or equal to zero. These data are ignored in plotting. That is not really what we wanted to do.

18 Rescale with an asinh Transform for Negatives

We can use alternative transformations and one good transformation for rescaling positive and negative data is based on `asinh` (the inverse hyperbolic sine of the data). This handles negatives and zero and serves a similar purpose to the log transforms.

```
asinh_trans <- function() trans_new(name="asinh",
                                      transform=asinh,
                                      inverse=sinh)
g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))
## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)): object 'dsm'
not found
g <- g + geom_point()
## Error in eval(expr, envir, enclos): object 'g' not found
g <- g + scale_y_continuous(trans="asinh")
## Error in eval(expr, envir, enclos): object 'g' not found
print(g)
## Error in print(g): object 'g' not found
```

We now get the negatives and zeros into the picture.

19 Scale Options: Setting Limits on the Y Axis

The y axis is unbalanced above and below zero. That is usually just fine, but we can also balance it up if desired.

```
g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))
## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)): object 'dsm'
not found

g <- g + geom_point()
## Error in eval(expr, envir, enclos): object 'g' not found

g <- g + scale_y_continuous(trans="asinh",
                             limits=c(-1e4, 1e4))
## Error in eval(expr, envir, enclos): object 'g' not found

print(g)
## Error in print(g): object 'g' not found
```

Actually though, there's quite a bit of wasted space now, so we'll drop the limits for the following plots. There is no point really in taking up precious real estate for no particular purpose.

20 Scale Options: Specify Breaks Along the Y Axis

The y axis labels are somewhat sparse, with no indications between 0 and 1,000. We can spice that up a little by specifying where the breaks along the axis should be labelled.

```
g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))
## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)): object 'dsm'
not found

g <- g + geom_point()
## Error in eval(expr, envir, enclos): object 'g' not found

g <- g + scale_y_continuous(trans="asinh",
                             breaks=c(-10, 0, 10, 1e2, 1e3))

## Error in eval(expr, envir, enclos): object 'g' not found

print(g)
## Error in print(g): object 'g' not found
```

This does add value to the plot. The actual gradation of points along the y axis is now much easier to perceive.

21 Scale Options: Label the Breaks

As well as specifying the breaks we can also specify how they are to be labelled. This could be useful when we want to abbreviate the labels in some standard way, if that improves the readability.

```
g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))

## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)): object 'dsm'
not found

g <- g + geom_point()

## Error in eval(expr, envir, enclos): object 'g' not found

g <- g + scale_y_continuous(trans="asinh",
                             breaks=c(-10, 0, 10, 1e2, 1e3),
                             labels=c("-10", "0", "10", "100", "1K"))

## Error in eval(expr, envir, enclos): object 'g' not found

print(g)
## Error in print(g): object 'g' not found
```

22 Plot Lines instead of Points

```
g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))

## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)): object 'dsm'
not found

g <- g + geom_line()

## Error in eval(expr, envir, enclos): object 'g' not found

g <- g + scale_y_continuous(trans="asinh",
                             breaks=c(-10, 0, 10, 1e2, 1e3),
                             labels=c("-10", "0", "10", "100", "1K"))

## Error in eval(expr, envir, enclos): object 'g' not found

print(g)

## Error in print(g): object 'g' not found
```

That is pretty messy looking and the story is hard to tell.

23 Plot Points and Lines

The two cumulative plots might be better as lines and the others as points. Thus we will have a mixture of point and line geometries.

```
draw.lines <- c("CumRainfall", "CumEvaporation")

g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))

## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)): object 'dsm' not found

g <- g + geom_point(data=subset(dsm, !variable %in% draw.lines))

## Error in eval(expr, envir, enclos): object 'g' not found

g <- g + geom_line(data=subset(dsm, variable %in% draw.lines))

## Error in eval(expr, envir, enclos): object 'g' not found

g <- g + scale_y_continuous(trans="asinh",
                             breaks=c(-10, 0, 10, 1e2, 1e3),
                             labels=c("-10", "0", "10", "100", "1K"))

## Error in eval(expr, envir, enclos): object 'g' not found

print(g)

## Error in print(g): object 'g' not found
```

24 Vertical Lines and Text

There may be significant dates we wish to note on the plot. Here we add two vertical lines that may be of some relevance. We use `geom_vline()` to do so but note that the intercept must be numeric. We'll use a dotted line (`linetype=3`) so the vertical lines are dominating the plot.

```
events <- as.Date(c("2007-12-25", "2008-03-22"))

g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))

## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)): object 'dsm' not found

g <- g + geom_point(data=subset(dsm, !variable %in% draw.lines))

## Error in eval(expr, envir, enclos): object 'g' not found

g <- g + geom_line(data=subset(dsm, variable %in% draw.lines))

## Error in eval(expr, envir, enclos): object 'g' not found

g <- g + scale_y_continuous(trans="asinh",
                             breaks=c(-10, 0, 10, 1e2, 1e3),
                             labels=c("-10", "0", "10", "100", "1K"))

## Error in eval(expr, envir, enclos): object 'g' not found

g <- g + geom_vline(xintercept=as.numeric(events), linetype=3)

## Error in eval(expr, envir, enclos): object 'g' not found

g <- g + annotate("text", events[1], -9, label="Christmas", size=3, colour="blue")

## Error in eval(expr, envir, enclos): object 'g' not found

g <- g + annotate("text", events[2], -9, label="Easter", size=3, colour="purple")

## Error in eval(expr, envir, enclos): object 'g' not found

print(g)

## Error in print(g): object 'g' not found
```

25 Finishing Touches

Add a title. Place the legend at the bottom.

```

g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))
## Error in ggplot(dsm, aes(x = Date, y = value, colour = variable)): object 'dsm' not found

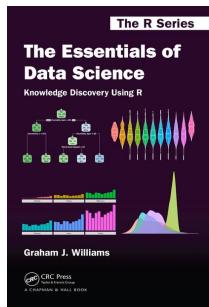
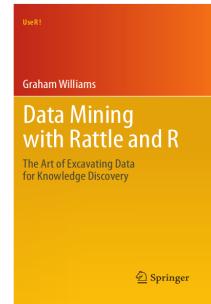
g <- g + geom_point(data=subset(dsm, !variable %in% draw.lines))
## Error in eval(expr, envir, enclos): object 'g' not found
g <- g + geom_line(data=subset(dsm, variable %in% draw.lines))
## Error in eval(expr, envir, enclos): object 'g' not found
g <- g + scale_y_continuous(trans="asinh",
                             breaks=c(-10, 0, 10, 1e2, 1e3),
                             labels=c("-10", "0", "10", "100", "1K"))

## Error in eval(expr, envir, enclos): object 'g' not found
g <- g + geom_vline(xintercept=as.numeric(events), linetype=3)
## Error in eval(expr, envir, enclos): object 'g' not found
g <- g + annotate("text", events[1], -9, label="Christmas", size=3, colour="blue")
## Error in eval(expr, envir, enclos): object 'g' not found
g <- g + annotate("text", events[2], -9, label="Easter", size=3, colour="purple")
## Error in eval(expr, envir, enclos): object 'g' not found
g <- g + labs(title=sprintf("Weather pattern for %s", weather$Location[1]))
## Error in eval(expr, envir, enclos): object 'g' not found
g <- g + theme(legend.direction="horizontal", legend.position="bottom")
## Error in eval(expr, envir, enclos): object 'g' not found
print(g)
## Error in print(g): object 'g' not found

```

26 Further Reading and Acknowledgements

The [Rattle Book](#) (Williams, 2011), published by Springer, provides a comprehensive introduction to data mining and analytics using Rattle and R. It is available from [Amazon](#). Rattle provides a graphical user interface through which the user is able to load, explore, visualise, and transform data, and to build, evaluate, and export models. Through its Log tab it specifically aims to provide an R template which can be exported and serve as the starting point for further programming with data in R.



The [Essentials of Data Science](#) book (Williams, 2017a), published by CRC Press, provides a comprehensive introduction to data science through programming with data using R. It is available from [Amazon](#). The book provides a template based approach to doing data science and knowledge discovery. Templates are provided for data wrangling and model building. These serve as generic starting points for programming with data, and are designed to require minimal effort to get started. Visit <https://essentials.togaware.com> for further guides and templates.

Other resources include:

- Garrett Grolemund and Hadley Wickham's paper, *Dates and Time Made Easy with lubridate*, published in the Journal of Statistical Software, April 2011, Volume 40, Issue 3, provides a great introduction to effectively using lubridate. It is freely available at <http://www.jstatsoft.org/v40/i03/paper>.

27 References

- Arel-Bundock V (2018a). *countrycode: Convert Country Names and Country Codes*. R package version 1.00.0, URL <https://CRAN.R-project.org/package=countrycode>.
- Arel-Bundock V (2018b). *WDI: World Development Indicators (World Bank)*. R package version 2.5, URL <https://CRAN.R-project.org/package=WDI>.
- Bache SM, Wickham H (2014). *magrittr: A Forward-Pipe Operator for R*. R package version 1.5, URL <https://CRAN.R-project.org/package=magrittr>.
- R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- Spinu V, Grolemund G, Wickham H (2018). *lubridate: Make Dealing with Dates a Little Easier*. R package version 1.7.4, URL <https://CRAN.R-project.org/package=lubridate>.
- Wickham H (2017). *tidyverse: Easily Install and Load the 'Tidyverse'*. R package version 1.2.1, URL <https://CRAN.R-project.org/package=tidyverse>.
- 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.
- Williams GJ (2017a). *The Essentials of Data Science: Knowledge discovery using R*. The R Series. CRC Press.
- Williams GJ (2017b). *rattle: Graphical User Interface for Data Science in R*. R package version 5.1.0, URL <https://CRAN.R-project.org/package=rattle>.

This document, sourced from DateTimeO.Rnw bitbucket revision 234, was processed by KnitR version 1.20 of 2018-02-20 10:11:46 UTC and took 9.9 seconds to process. It was generated by gjw on Ubuntu 18.04 LTS.

Hands-On Data Science with R Maps

Graham.Williams@togaware.com

28th October 2014

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

Spatial data is central to many of our tasks as Data Scientists. Identifying patterns and then correlations and relationships between those patterns delivers opportunities for delivering new services. Imagine predicting common routes for travellers this morning, and dynamically routing public transport to meet those needs. Fundamental to the Data Scientist is the ability to process, visualize and then model spatial data. Done right maps can be a very effective communications tool. Numerous R packages work together to bring us a sophisticated mapping and spatial analysis capability.

The required packages for this module include:

```
library(ggplot2)      # Plotting maps.  
library(maps)        # Map data.  
library(oz)          # Map data for Australia.  
library(scales)       # Functions: alpha() comma()  
library(ggmap)        # Google maps.
```

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 Google Maps: Geocoding

One of the fundamental things about spatial data and mapping is the geographic coordinate system used to uniquely identify locations. We use longitude (x axis, abbreviated lon) and latitude (y axis, abbreviated lat) for locations on our planet. The longitude is the angle from the meridian through Greenwich and the latitude is the angle from the equator. We can use `ggmap` (?) to `geocode()` street addresses and locations. Here are a few examples.

```
library(ggmap)
geocode("New York")

##           lon      lat
## 1 -74.00594 40.71278

geocode("Qiushi Road, Shenzhen")

##           lon      lat
## 1 113.9751 22.59073

geocode("Canberra")

##           lon      lat
## 1 149.1287 -35.282

geocode("Gus' Cafe, Garema Place, Canberra, Australia")

##           lon      lat
## 1 149.1321 -35.27839

geocode("9 Bunda Street, Canberra")

##           lon      lat
## 1 149.1312 -35.27757

geocode("11 Bunda Street, Canberra")

##           lon      lat
## 1 149.1313 -35.27758
```

For later use we will save some locations.

```
(syd <- as.numeric(geocode("Sydney")))

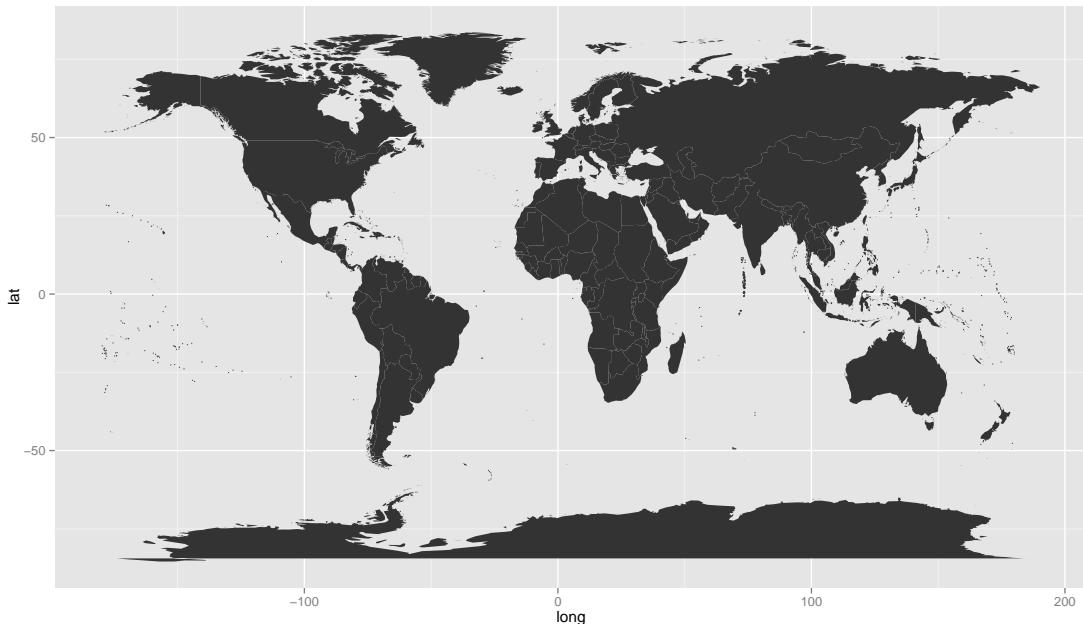
## [1] 151.20699 -33.86749

(cbr <- as.numeric(geocode("Canberra")))

## [1] 149.1287 -35.2820

syddb <- c(151.15, -33.88, 151.25, -33.84)
```

2 World Map



The data here comes from the `maps` (Brownrigg, 2014) package. We load the vector data for plotting a world map using `map_data()`.

```
ds <- map_data("world")
class(ds)

## [1] "data.frame"

str(ds)

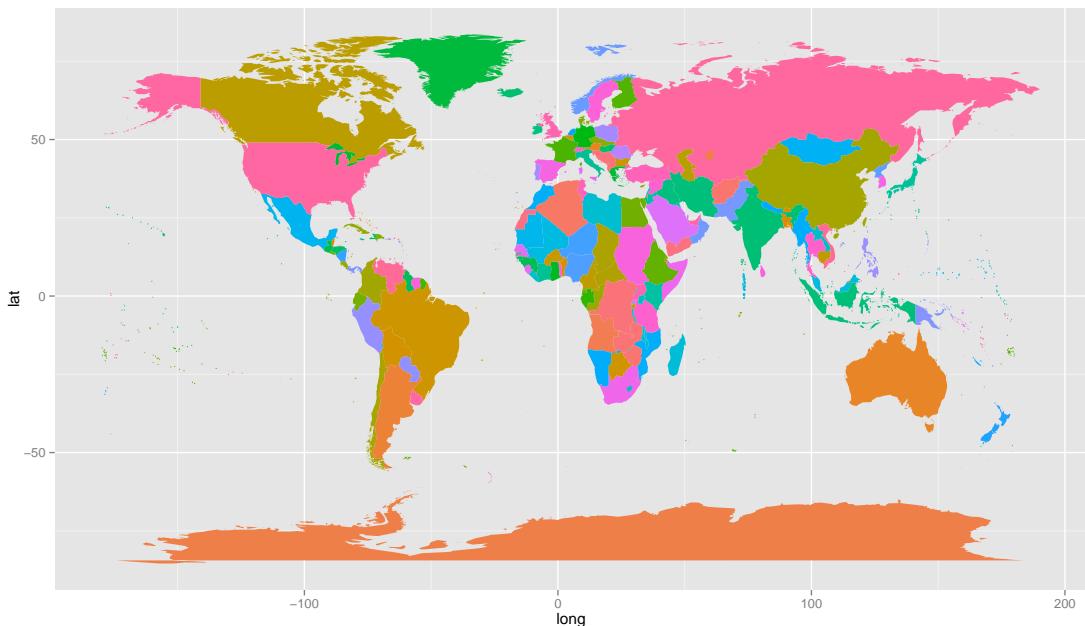
## 'data.frame': 25553 obs. of  6 variables:
## $ long      : num  -133 -132 -132 -132 -130 ...
## $ lat       : num  58.4 57.2 57 56.7 56.1 ...
## $ group     : num  1 1 1 1 1 1 1 1 1 1 ...
## ...
## #> #> #> head(ds)

##      long   lat group order region subregion
## 1 -133.4 58.42     1     1 Canada    <NA>
## 2 -132.3 57.16     1     2 Canada    <NA>
## 3 -132.0 56.99     1     3 Canada    <NA>
## 4 -131.8 56.83     1     4 Canada    <NA>
## 5 -131.6 56.67     1     5 Canada    <NA>
## 6 -131.4 56.51     1     6 Canada    <NA>
```

It is then quite simple to plot the world map using `ggplot2` (Wickham and Chang, 2014).

```
p <- ggplot(ds, aes(x=long, y=lat, group=group))
p <- p + geom_polygon()
p
```

3 World Map with Coloured Regions



We can specify a fill for the map, based on the regions.

```
p <- ggplot(ds, aes(x=long, y=lat, group=group, fill=region))
p <- p + geom_polygon()
p <- p + theme(legend.position = "none")
p
```

Note there are very many regions, and so the legend would be too large, so we turn that off.

```
length(unique(ds$region))
## [1] 234
```

4 Obtain Map Data

Visit <http://www.gadm.org/country> to download administrative vector data for any region of the world. The coordinate reference system is latitude/longitude and the WGS84 datum. The file formats include shapefiles, ESRI data files, Google Earth, and RData files.

From the website: A "shapefile" consist of at least three actual files. This is a commonly used format that can be directly used in Arc-anything, DIVA-GIS, and many other programs. Unfortunately, many of the non standard latin (roman / english) characters are lost in the shapefile. Even if you use the shapefile for mapping, you can use the .csv file that comes with the shapefiles, or the attribute data in the geodatabase for the correct attributes (the geodatabase is a MS Access database that (on windows) can be accessed via ODBC).

An "ESRI personal geodatabase" is a MS Access file that can be opened in ArcGIS (version 10). One of its advantages, compared to a shapefile, is that it can store non-latin characters (e.g. Cyrillic and Chinese characters). You can also query the (attribute) data in Access or via ODBC.

An "ESRI file geodatabase" can be opened in ArcGIS (version 10). It can also store non-latin characters (e.g. Cyrillic and Chinese characters).

A "Google Earth .kmz" file can be opened in Google Earth.

A "RData" file can be used in R (with the sp package loaded). See the CRAN spatial task view

5 Australian Map Data

A data frame version of the Australian map data that comes originally from oz (?) is available at <http://www.elaliberte.info/software>. We load the CSV version of the data frame here.

```
ds <- read.csv(file.path("data", "ozdata.csv"))
dim(ds)

## [1] 5852    7

head(ds, 2)

##   X long lat group order state border
## 1 1 129.0 -31.58 1 1 WA coast
## 2 2 128.7 -31.69 1 2 WA coast

tail(ds, 2)

##           X long lat group order state border
## 5851 5851 146.9 -43.59 7 660 TAS coast
## 5852 5852 146.9 -43.59 7 661 TAS coast

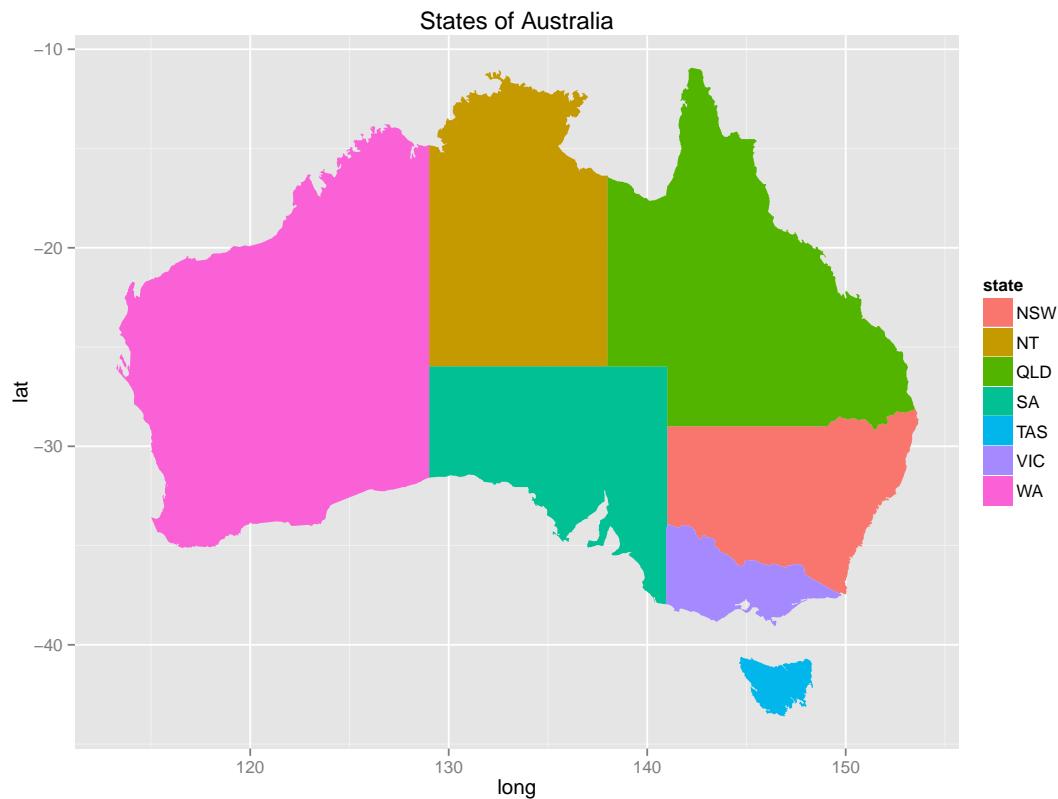
str(ds)

## 'data.frame': 5852 obs. of 7 variables:
## $ X      : int  1 2 3 4 5 6 7 8 9 10 ...
## $ long   : num  129 129 129 128 128 ...
## $ lat    : num  -31.6 -31.7 -31.8 -31.8 -31.9 ...
## $ group  : int  1 1 1 1 1 1 1 1 1 ...
## $ order  : int  1 2 3 4 5 6 7 8 9 10 ...
## $ state  : Factor w/ 7 levels "NSW","NT","QLD",...: 7 7 7 7 7 7 7 7 7 ...
## $ border : Factor w/ 19 levels "coast","NSW.QLD",...: 1 1 1 1 1 1 1 1 1 ...

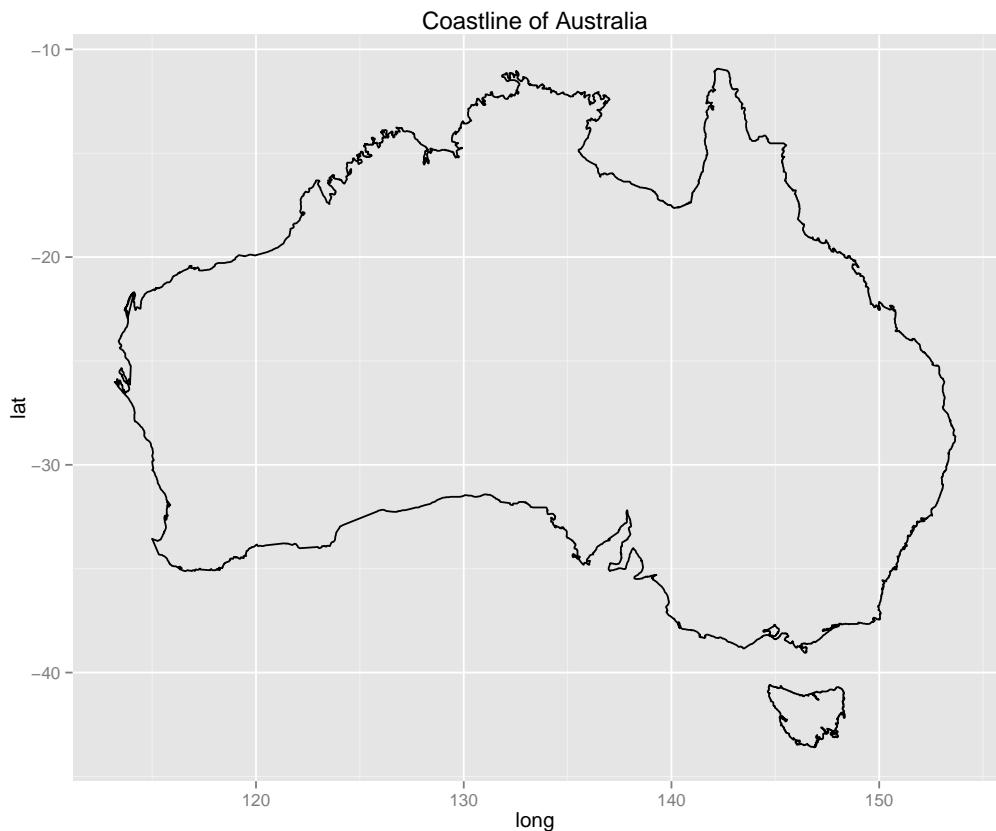
summary(ds)

##          X            long         lat        group
## Min.   : 1   Min.   :113   Min.   :-43.6   Min.   :1.00
## 1st Qu.:1464 1st Qu.:131  1st Qu.:-35.6  1st Qu.:2.00
## Median :2926 Median :142   Median :-28.9   Median :3.00
## Mean   :2926 Mean   :139   Mean   :-27.2   Mean   :3.52
## 3rd Qu.:4389 3rd Qu.:147  3rd Qu.:-16.1  3rd Qu.:5.00
## Max.   :5852  Max.   :154   Max.   :-10.9  Max.   :7.00
##
##          order      state       border
## Min.   : 1   NSW: 856   coast :4904
## 1st Qu.: 210 NT : 743   NSW.VIC: 282
## Median : 418 QLD:1118   VIC.NSW: 282
## Mean   : 459 SA : 504   NSW.QLD: 176
## 3rd Qu.: 648 TAS: 661   QLD.NSW: 176
## Max.   :1298  VIC: 672   QLD.SA : 3
##                   WA :1298   (Other): 29
```

6 Australian Map with States



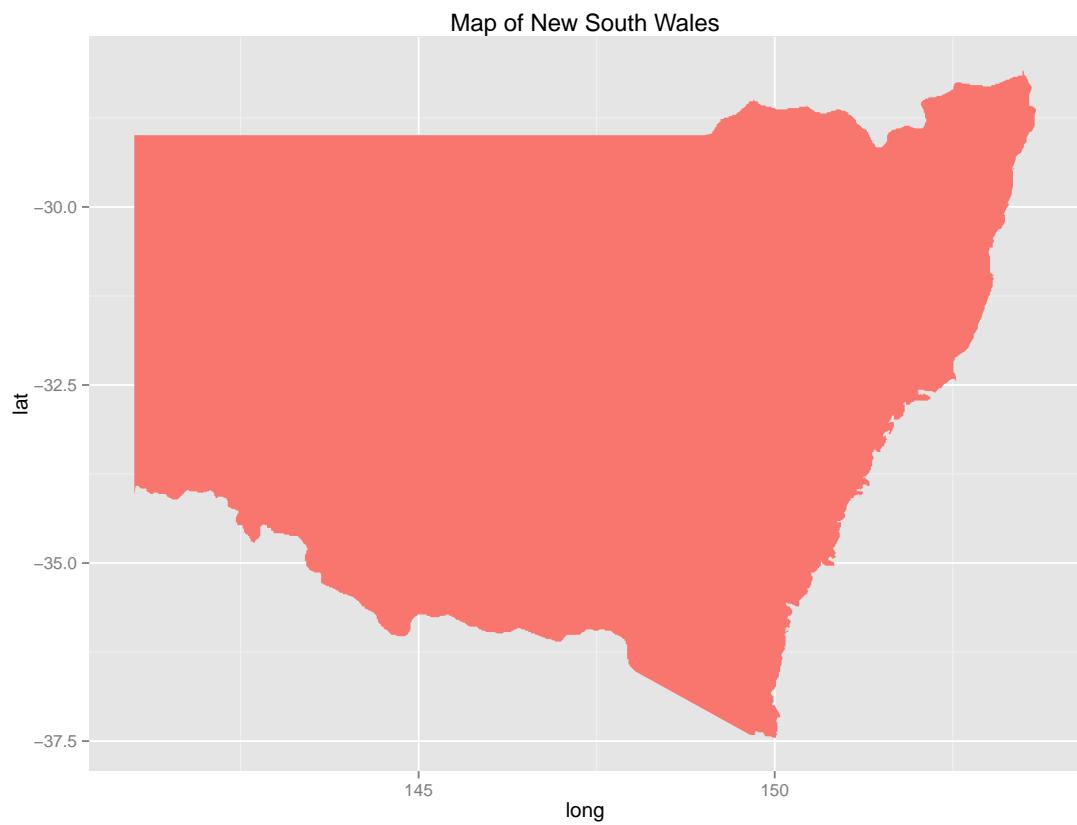
7 Map a Subset — Australian Coastline



```
p <- ggplot(subset(ds, border=="coast"), aes(long, lat, fill=state))
p <- p + geom_path()
p <- p + coord_equal()
p <- p + ggtitle("Coastline of Australia")
p
```

This example was motivated by the example using `qqplot()` at <http://www.elaliberte.info/software>.

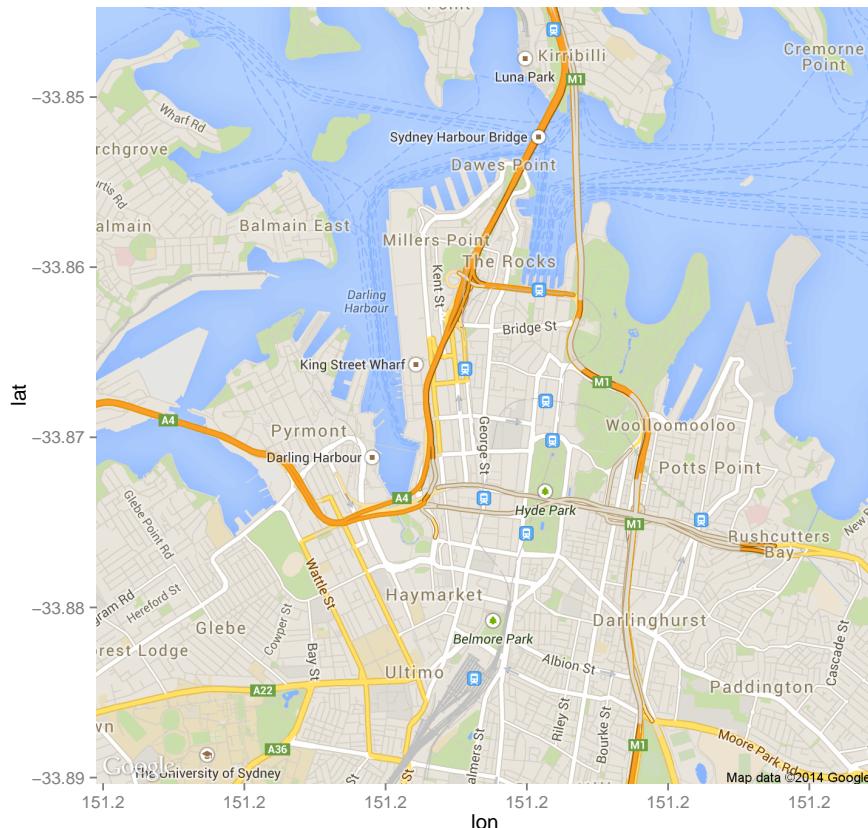
8 Map a Subset — New South Wales



```
p <- ggplot(subset(ds, state=="NSW"), aes(long, lat, fill=state))
p <- p + geom_polygon()
p <- p + coord_equal()
p <- p + ggtitle("Map of New South Wales")
p <- p + theme(legend.position="none")
p
```

This example was motivated by the example using `qqplot()` at <http://www.elaliberte.info/software>.

9 Google Map of Sydney



Here we download the map data for Sydney from Google using `get_map()` from `ggmap` (?). The data is transformed into a raster object for plotting.

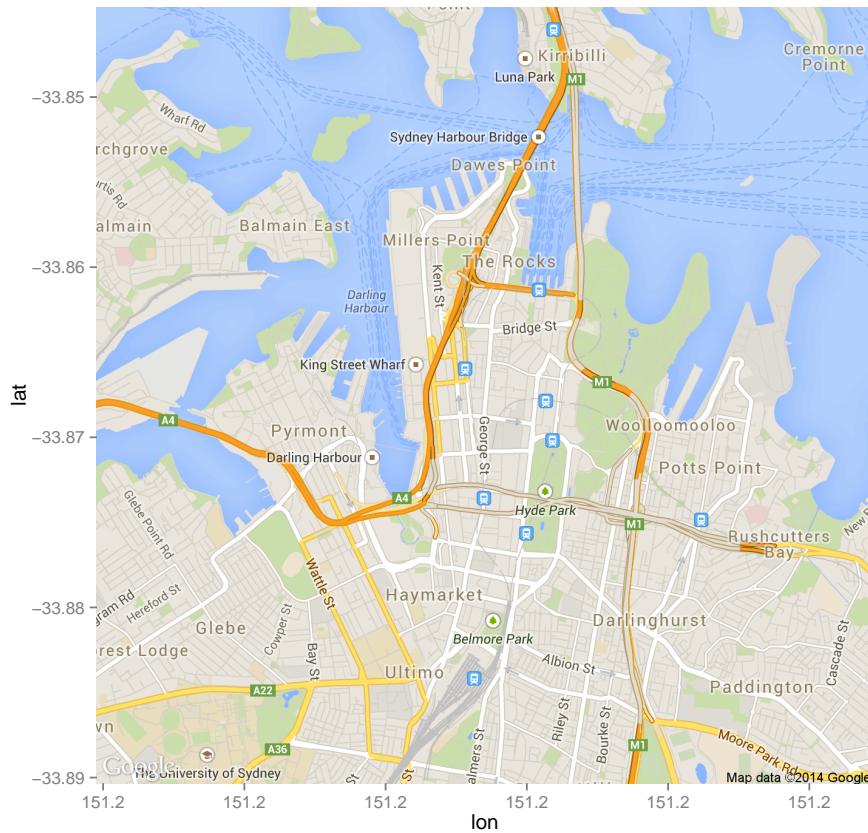
```
map <- get_map(location="Sydney", zoom=14, maptype="roadmap", source="google")
p <- ggmap(map)
p
```

The `zoom=` option is an integer. A value of 0 returns a map of the whole world, centred around the location. The maximum value is 21 and returns a map down to the building. Choosing 4 is usually good for continents, 14 for a city.

Notice that the object returned by `ggmap()` is a `ggplot` object, and so all the usual `ggplot2` (Wickham and Chang, 2014) functions apply.

```
class(p)
## [1] "gg"      "ggplot"
```

10 Google Map by Geocode of Sydney



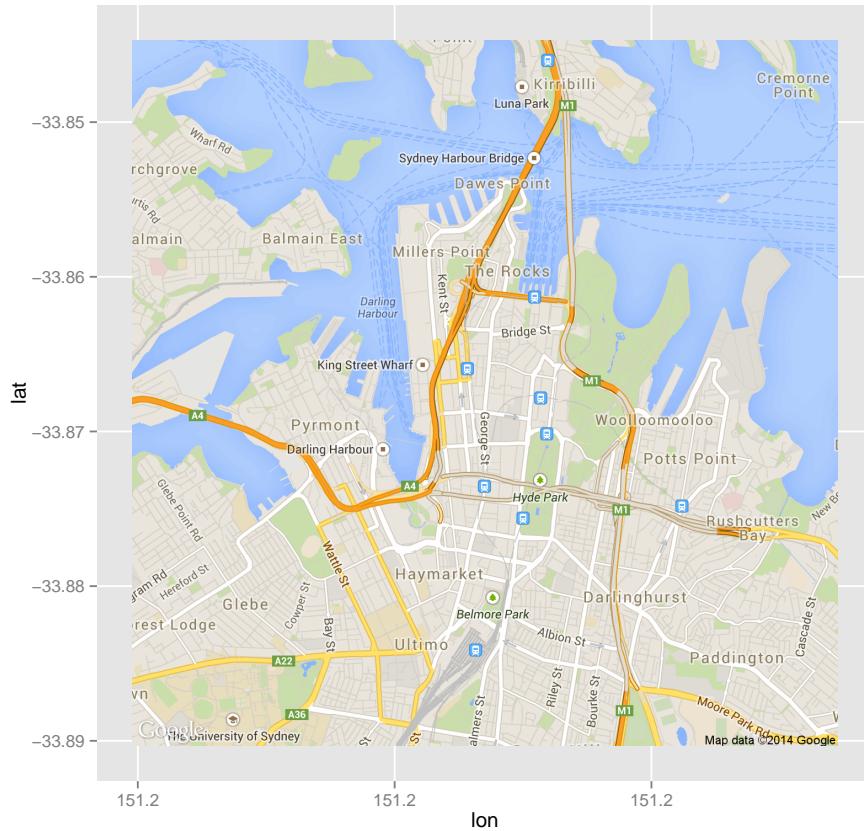
We will generally provide geo-codes to extract maps.

```
map <- get_map(location=syd, zoom=14, maptype="roadmap", source="google")
p <- ggmap(map)
p
```

Notice we have previously saved the location of Sydney into the variable *syd* and we have used that here to extract the same map.

```
(syd <- as.numeric(geocode("Sydney")))
## [1] 151.21 -33.87
```

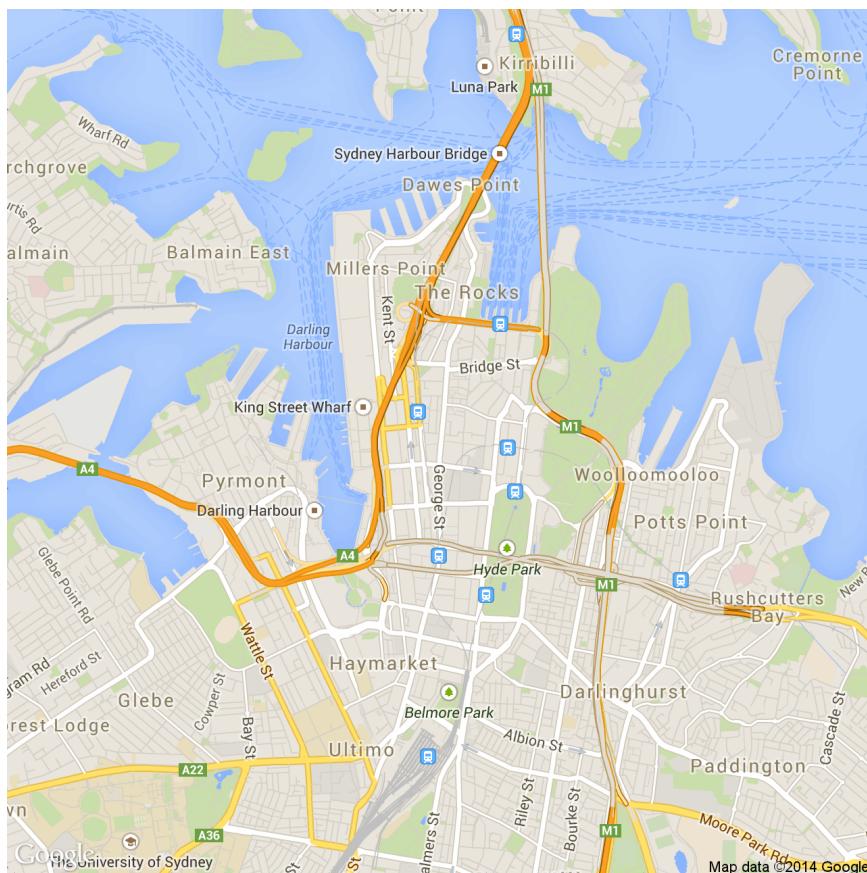
11 Map with Normal Extent of Sydney



```
map <- get_map(location="Sydney", zoom=14, maptype="roadmap", source="google")
p <- ggmap(map, extent="normal")
p
```

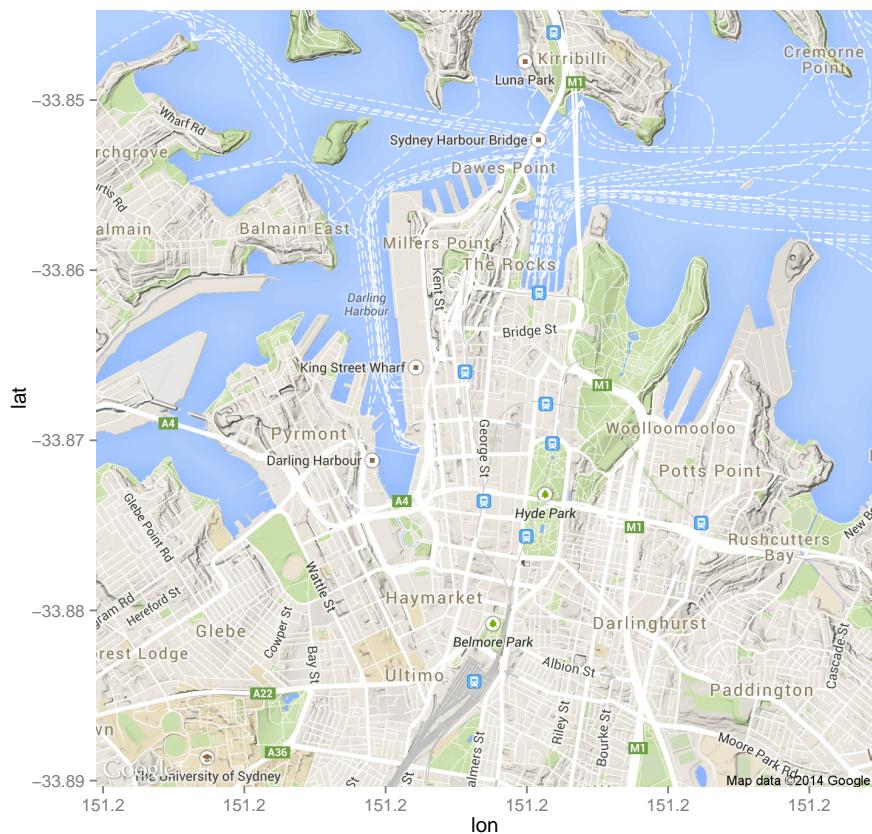
The default is `extent="panel"`.

12 Map with Device Extent of Sydney



```
map <- get_map(location="Sydney", zoom=14, maptype="roadmap", source="google")
p <- ggmap(map, extent="device")
p
```

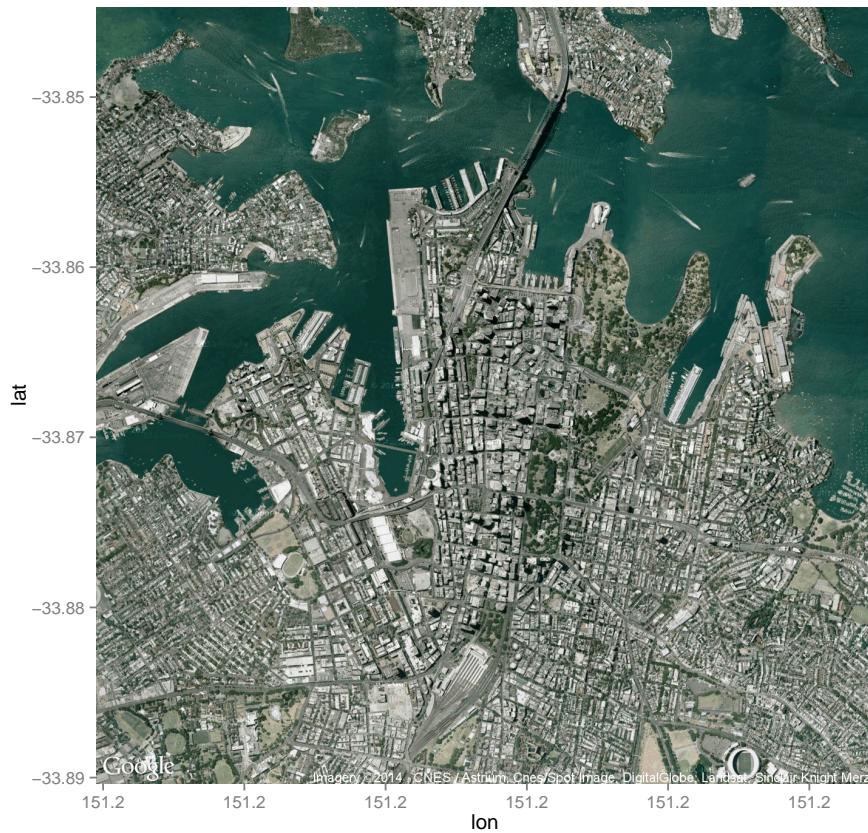
13 Google Terrain Map of Sydney



Here we see another type of map available from Google.

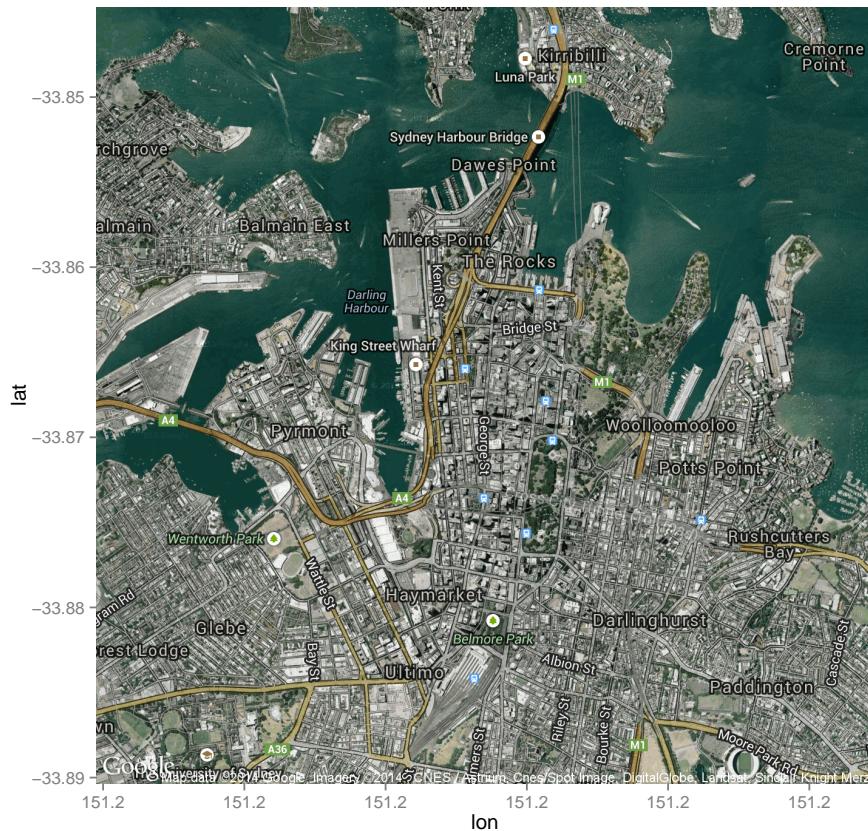
```
map <- get_map(location="Sydney", zoom=14, maptype="terrain", source="google")
p <- ggmap(map)
p
```

14 Google Satellite Map of Sydney



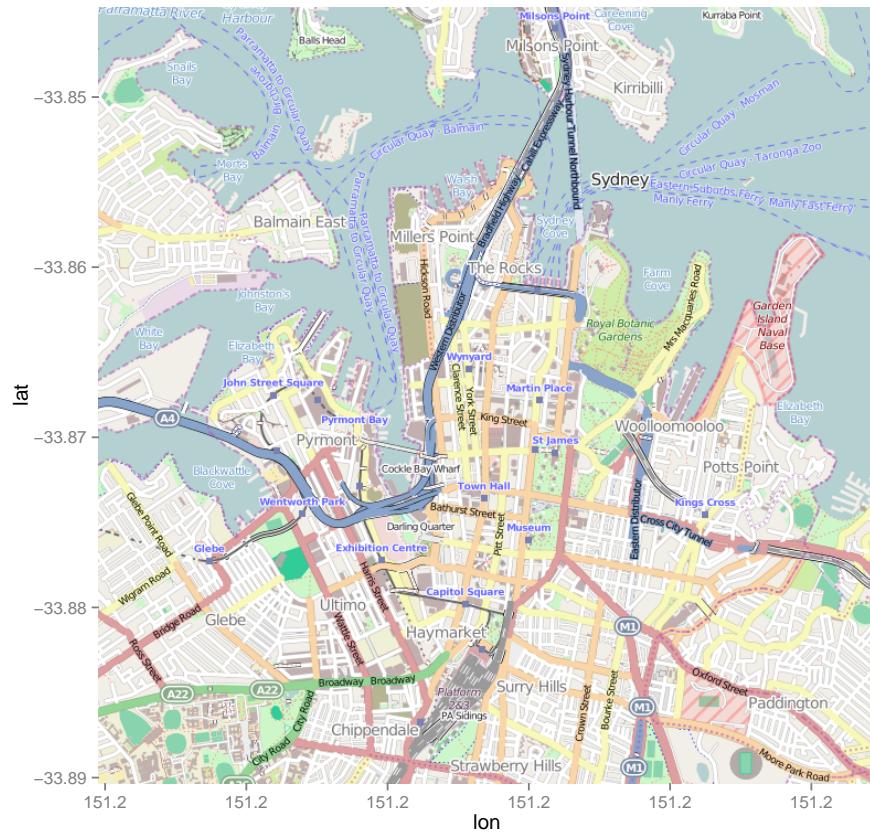
```
map <- get_map(location="Sydney", zoom=14, maptype="satellite", source="google")
p <- ggmap(map)
p
```

15 Google Hybrid Map of Sydney



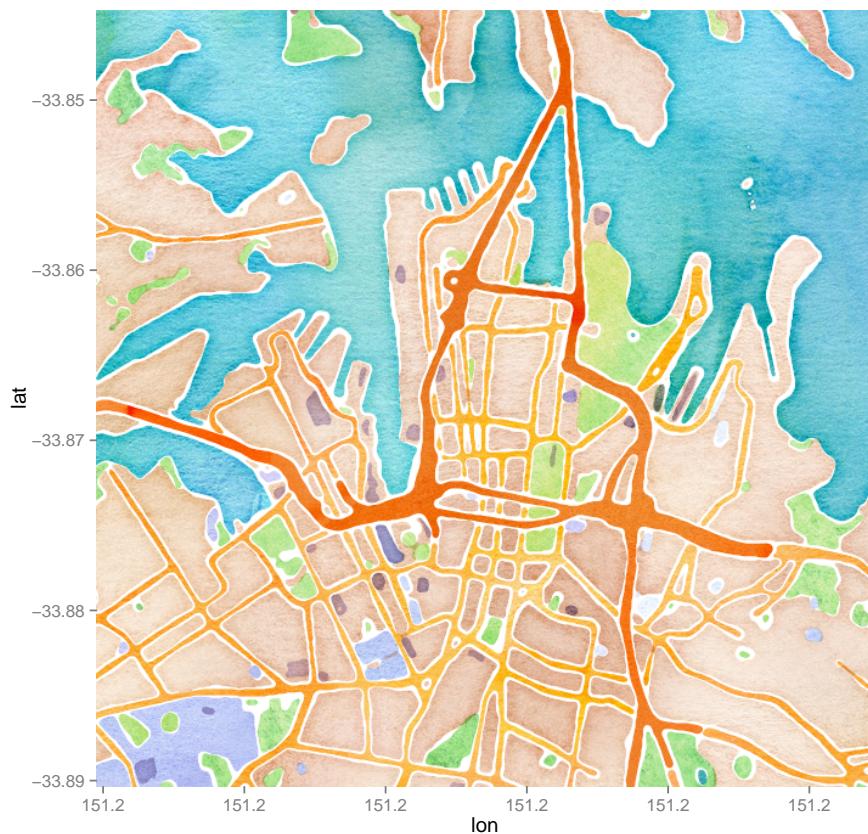
```
map <- get_map(location="Sydney", zoom=14, maptype="hybrid", source="google")
p <- ggmap(map)
p
```

16 OpenStreetMap of Sydney



```
map <- get_map(location="Sydney", zoom=14, source="osm")
p <- ggmap(map)
p
```

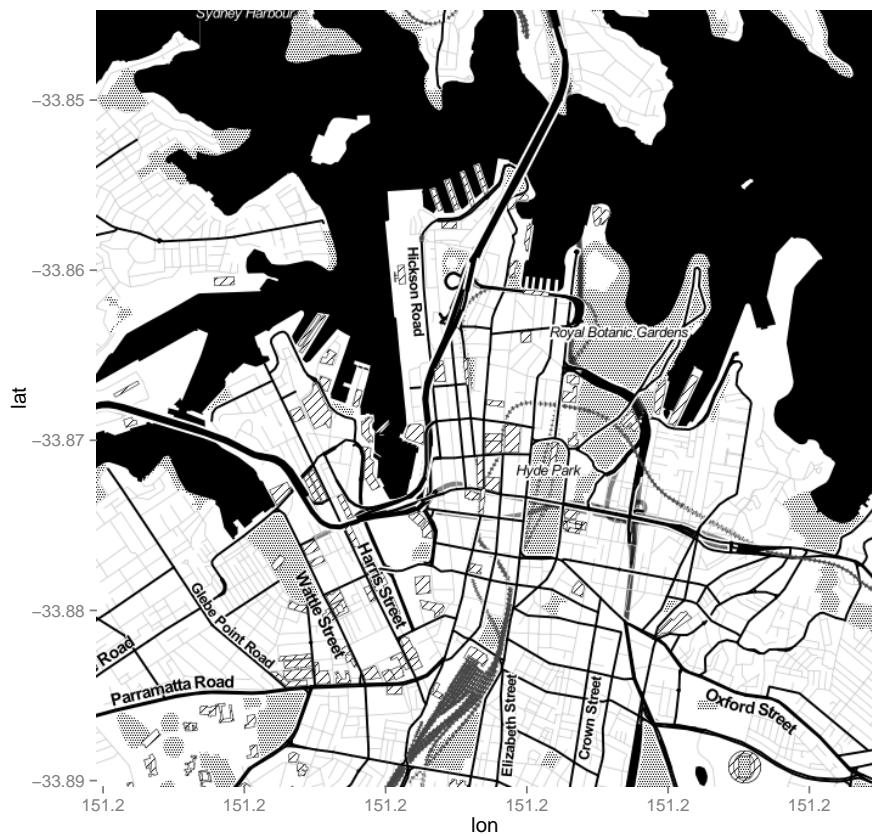
17 Stamen Watercolour Map of Sydney



Here we downloaded the map data for Sydney from other sources. This watercolor comes from [Stamen](#).

```
map <- get_map(location="Sydney", zoom=14, maptype="watercolor", source="stamen")
p <- ggmap(map)
p
```

18 Stamen Toner Map of Sydney



```
map <- get_map(location="Sydney", zoom=14, maptype="toner", source="stamen")
p <- ggmap(map)
p
```

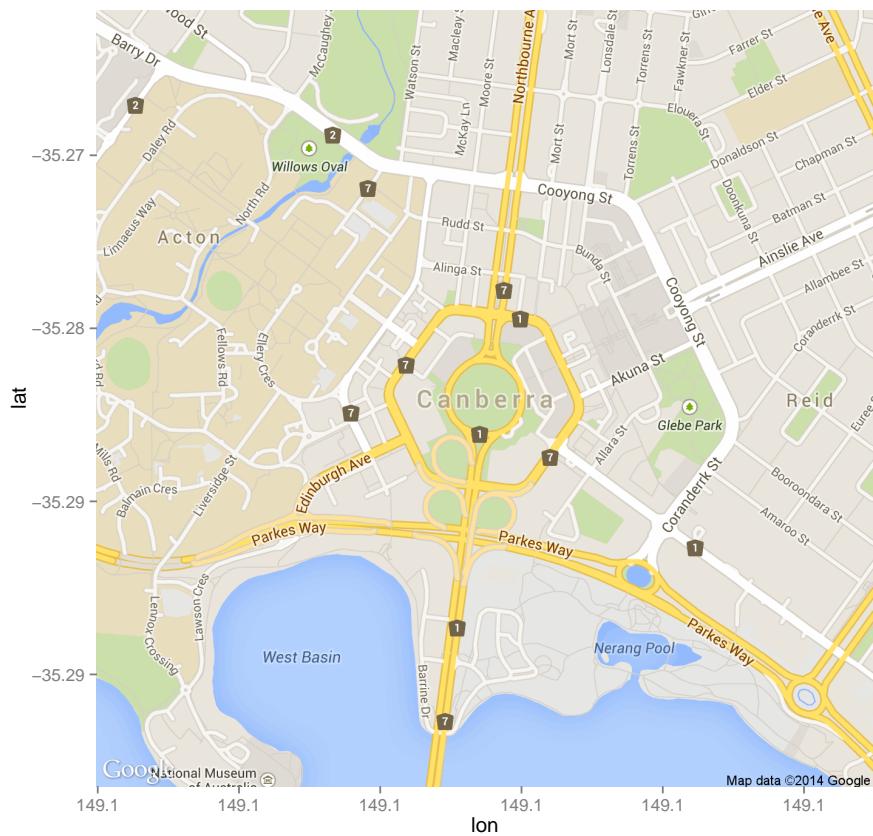
19 Bounding Box to Specify Sydney Map

Some sources also support bounding boxes rather than a centroid and zoom. Stamen, for example, supports bounding boxes, but Google does not. Here we use a bounding box (*sydbb*) that we defined earlier.



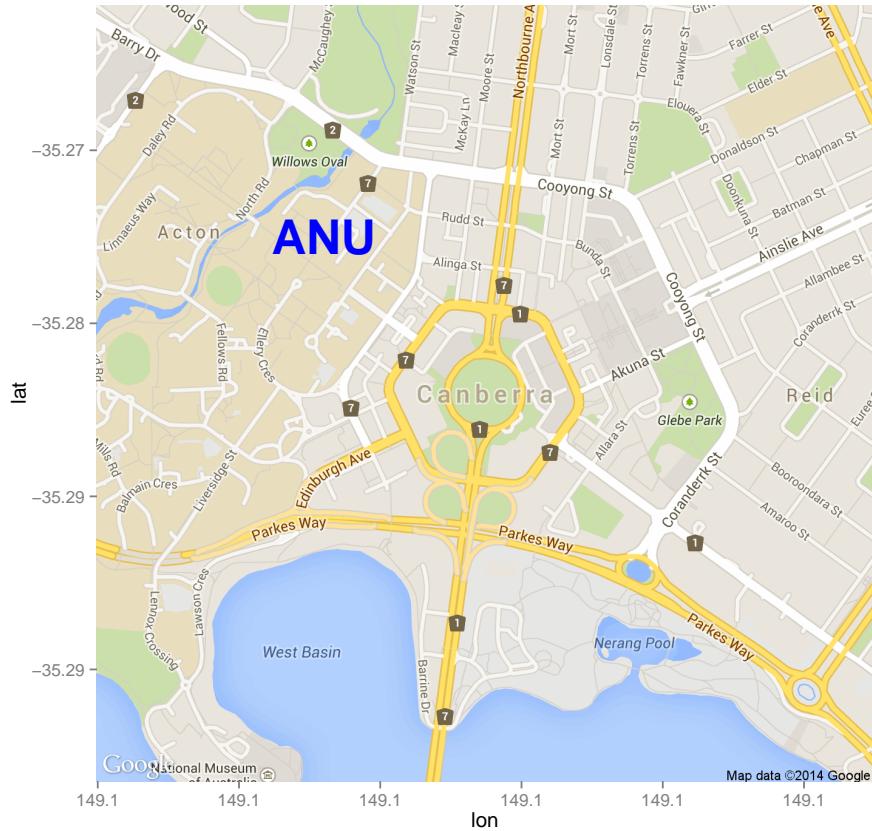
```
map <- get_map(location=sydbb, maptype="watercolor", source="stamen")
p <- ggmap(map)
p
```

20 Google Map of Canberra



```
map <- get_map(location="Canberra", zoom=15, maptype="roadmap", source="google")
p <- ggmap(map)
p
```

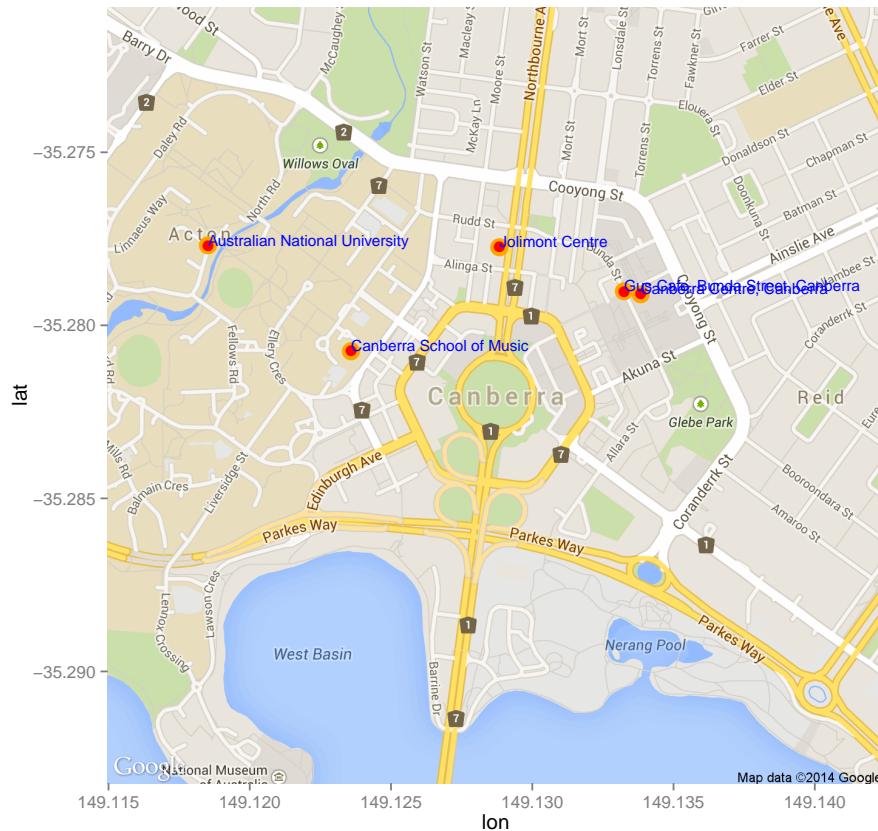
21 Annotating a Map with Text



Here we add a blue ANU label (Australian National University).

```
map <- get_map(location="Canberra", zoom=15, maptype="roadmap", source="google")
dflbl <- data.frame(lon=149.1230, lat=-35.2775, text="ANU")
p <- ggmap(map)
p <- p + geom_text(data=dflbl, aes(x=lon, y=lat, label=text),
                     size=10, colour="blue", fontface="bold")
p
```

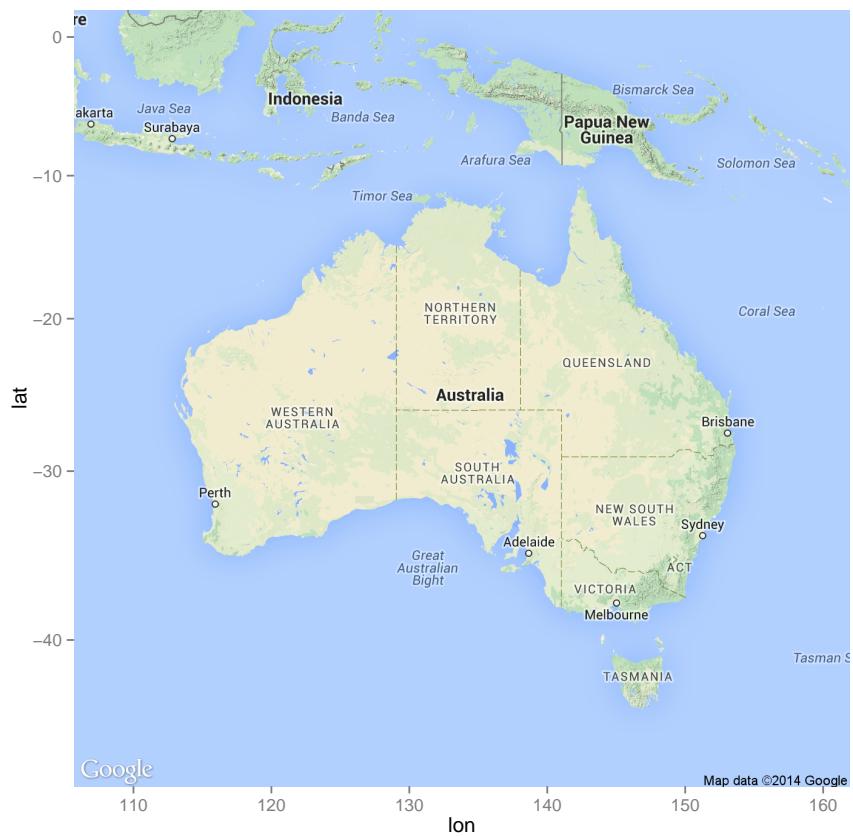
22 Annotating a Map with Landmarks



Here we have geocoded some landmarks and then added them to the map.

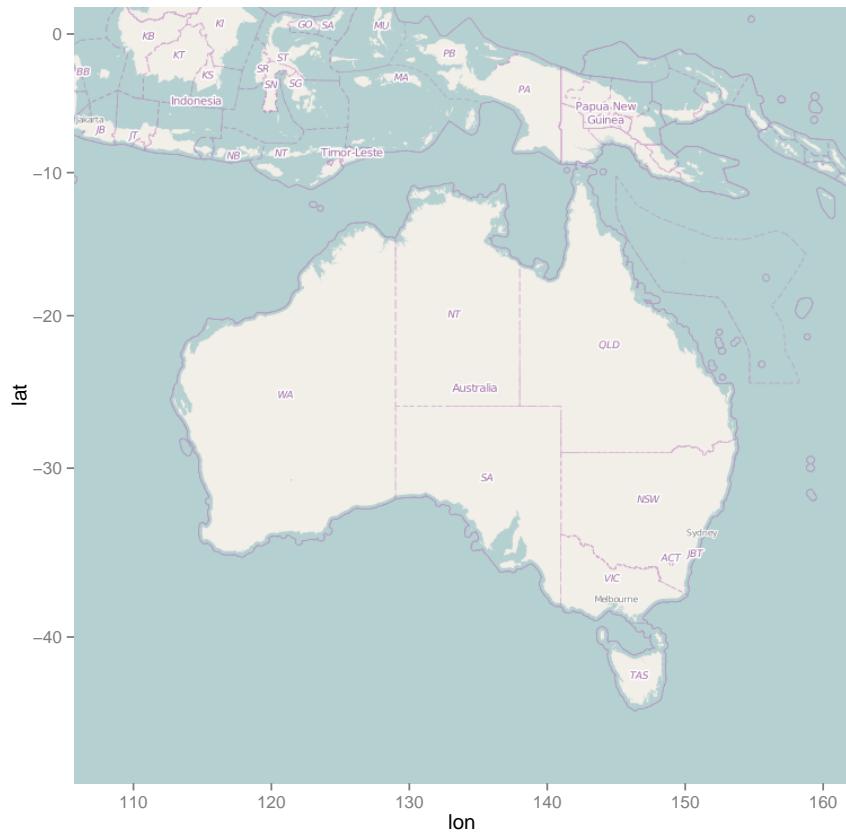
```
landmarks <- c("Gus Cafe, Bunda Street, Canberra", "Canberra Centre, Canberra",
  "Canberra School of Music", "Jolimont Centre",
  "Australian National University")
lbls <- cbind(geocode(landmarks), text=landmarks)
p <- ggmap(map)
p <- p + geom_point(data=lbls, aes(x=lon, y=lat), size=5, colour="orange")
p <- p + geom_point(data=lbls, aes(x=lon, y=lat), size=3, colour="red")
p <- p + geom_text(data=lbls, aes(x=lon, y=lat, label=text),
  size=3, colour="blue", hjust=0, vjust=0)
p
```

23 Google Map of Australia



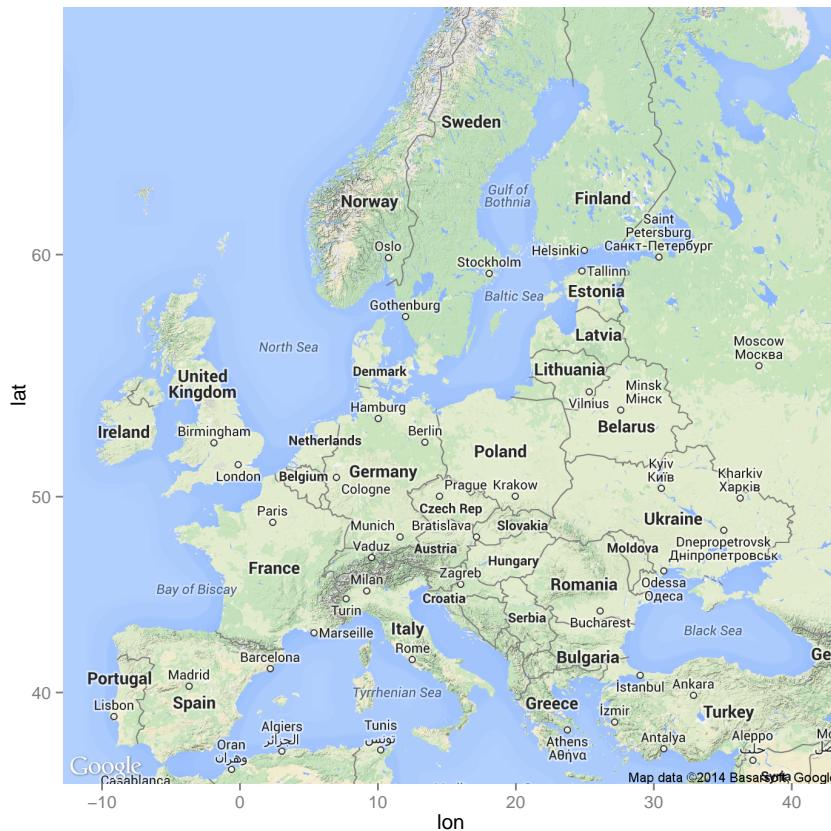
```
map <- get_map(location=as.numeric(geocode("Australia")),
                 zoom=4, source="google")
p <- ggmap(map)
p
```

24 OpenStreetMap of Australia



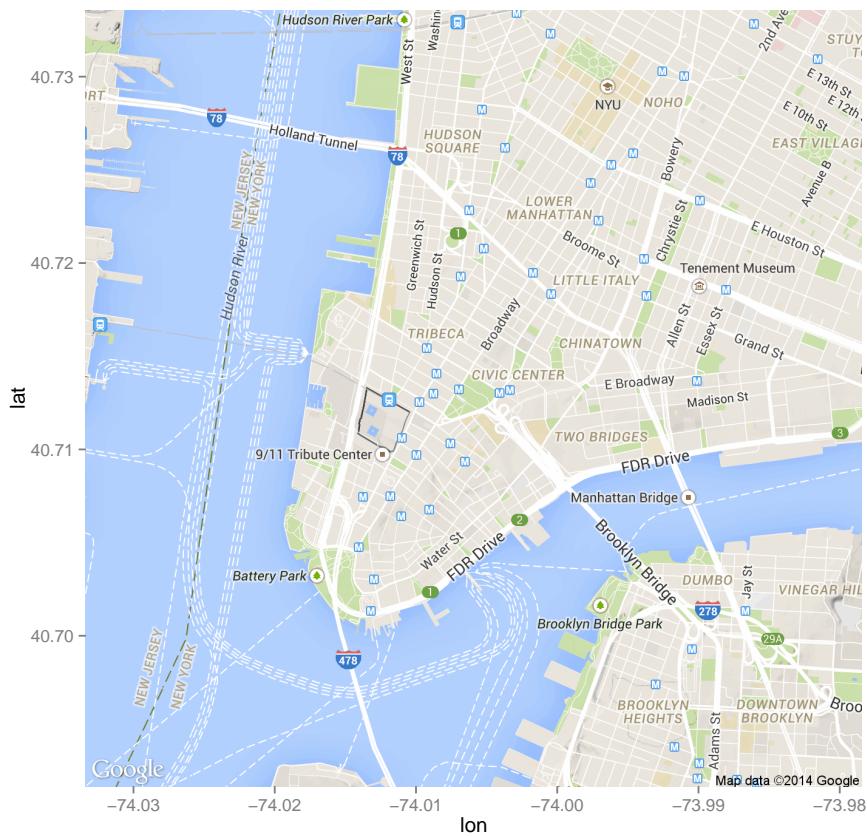
```
map <- get_map(location="Australia", zoom=4, source="osm")
p <- ggmap(map)
p
```

25 Google Map of Europe



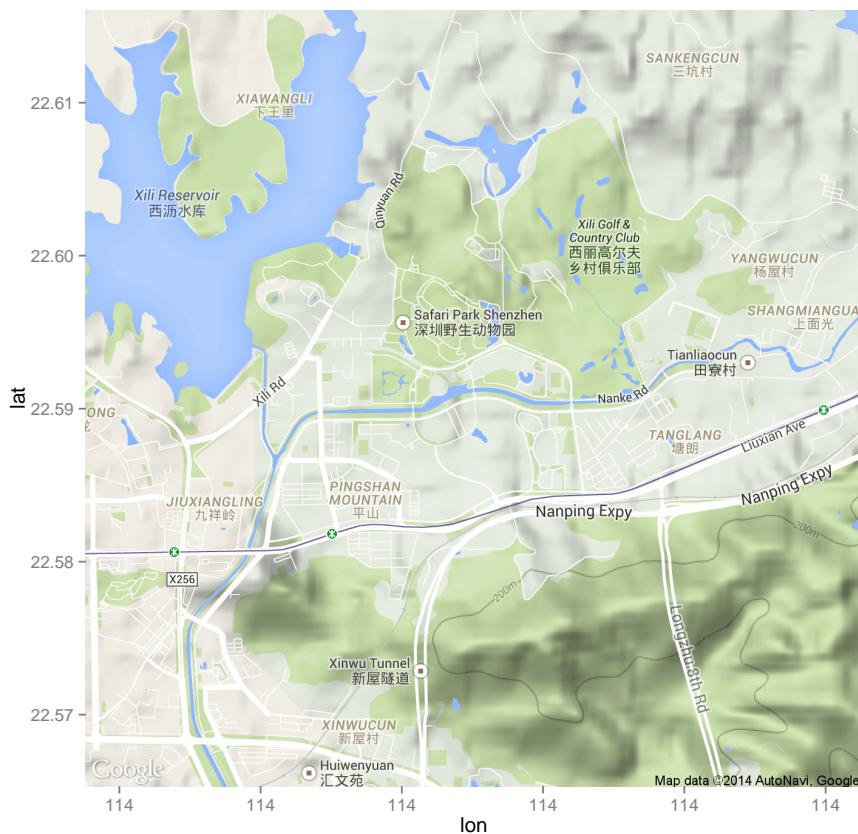
```
map <- get_map(location="Europe", zoom=4)
p <- ggmap(map)
p
```

26 New York: Google Maps



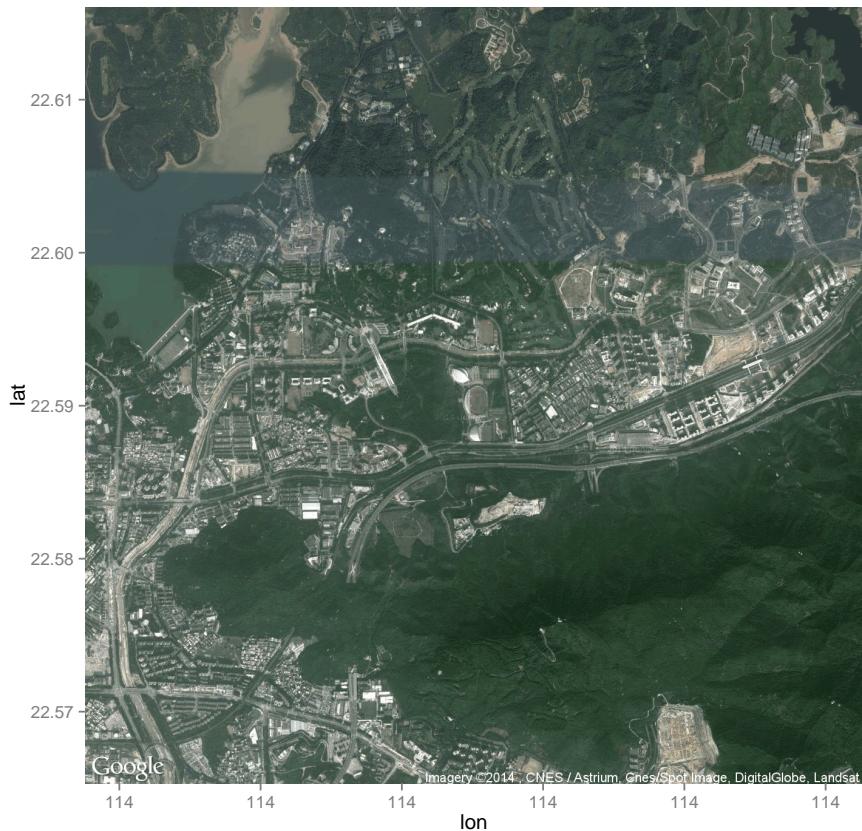
```
map <- get_map(location=as.numeric(geocode("New York")),
                 zoom=14, source="google")
p <- ggmap(map)
p
```

27 Google Maps: Shenzhen University Town



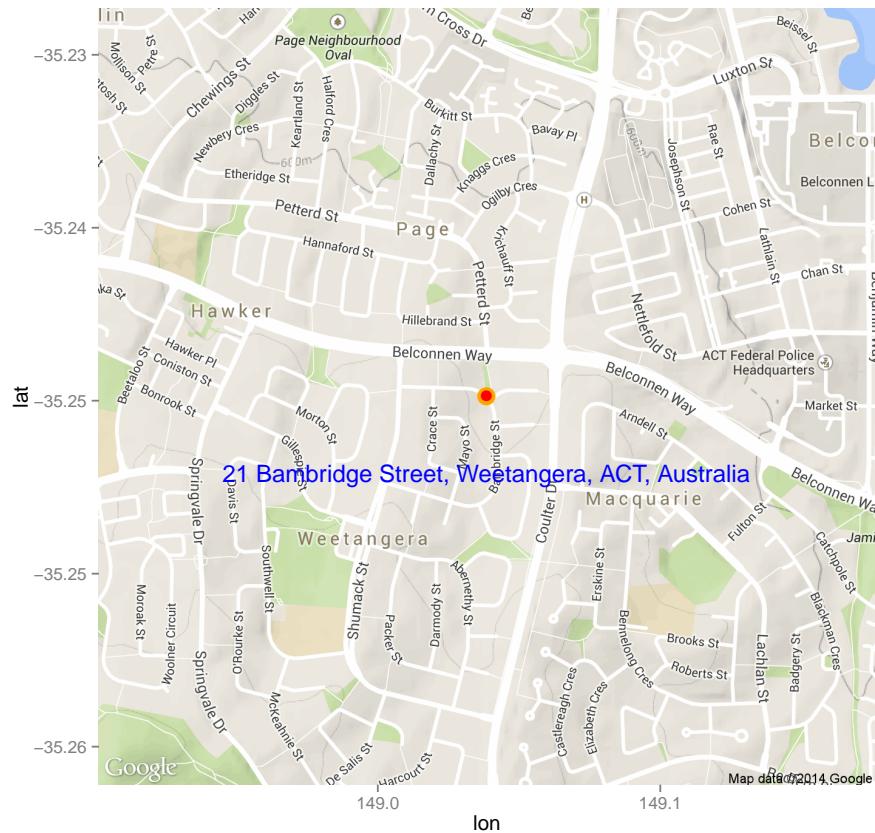
```
map <- get_map(location=as.numeric(geocode("Qiushi Road, Shenzhen")),  
                 zoom=14, source="google")  
p <- ggmap(map)  
p
```

28 Google Maps: Shenzhen Satellite



```
map <- get_map(location=as.numeric(geocode("Qiu Shi Road, Shenzhen")),  
                 zoom=14, maptype="satellite", source="google")  
p <- ggmap(map)  
p
```

29 Plot an Address



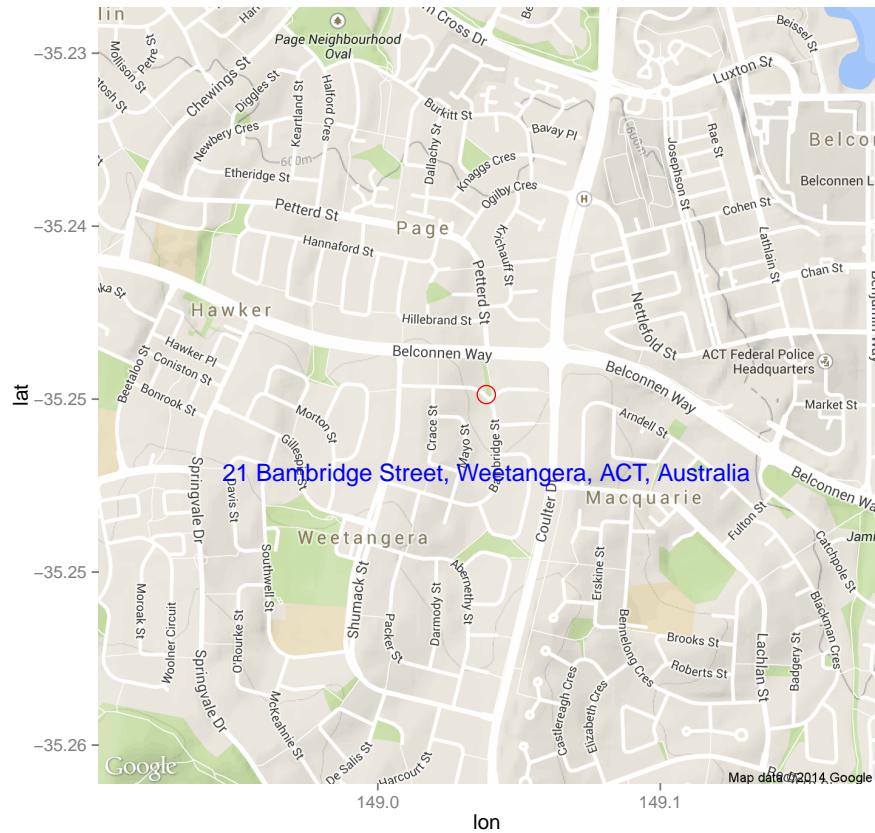
```

addr <- "21 Bambridge Street, Weetangera, ACT, Australia"
loc <- as.numeric(geocode(addr))
lbl <- data.frame(lon=loc[1], lat=loc[2], text=addr)
map <- get_map(location=loc, zoom=15, source="google")
p <- ggmap(map)
p <- p + geom_point(data=lbl, aes(x=lon, y=lat), size=5, colour="orange")
p <- p + geom_point(data=lbl, aes(x=lon, y=lat), size=3, colour="red")
p <- p + geom_text(data=lbl, aes(x=lon, y=lat, label=text),
                     size=5, colour="blue", hjust=0.5, vjust=5)
p

```

The location is pinpointed with a red dot overlaying an orange dot. We also add the actual address to the plot.

30 Plot an Address Using Circle



```

addr <- "21 Bambridge Street, Weetangera, ACT, Australia"
loc <- as.numeric(geocode(addr))
lbl <- data.frame(lon=loc[1], lat=loc[2], text=addr)
map <- get_map(location=loc, zoom=15, source="google")
p <- ggmap(map)
p <- p + geom_point(data=lbl, aes(x=lon, y=lat), size=5,
                      shape=1, colour="red")
p <- p + geom_text(data=lbl, aes(x=lon, y=lat, label=text),
                     size=5, colour="blue", hjust=0.5, vjust=5)
p

```

Here we have replaced the dot with a circle to pinpoint the location, using `shape=1` to choose a circle rather than a filled dot.

31 Plot an Address on a Satellite Image



```
addr <- "21 Bambridge Street, Weetangera, ACT, Australia"
loc <- as.numeric(geocode(addr))
lbl <- data.frame(lon=loc[1], lat=loc[2], text=addr)
map <- get_map(location=loc, zoom=15, maptype="hybrid", source="google")
p <- ggmap(map)
p <- p + geom_point(data=lbl, aes(x=lon, y=lat),
                      alpha=I(0.5), size=I(5), colour="red")
p <- p + geom_text(data=lbl, aes(x=lon, y=lat, label=text),
                     size=5, colour="white", hjust=0.5, vjust=5)
p
```

The underlying map is now a hybrid satellite and road map, with a transparent red dot to pinpoint the location. The transparency is controlled by `alpha=I(0.5)`.

32 USA Arrests: Assaults per Murder Data

This example comes from the help page for `map_data()` from `ggplot2` (Wickham and Chang, 2014). It shows the number of assaults per murder in each US state, though it is quite easy to modify the code to display various statistics from the data.

First we take a copy of the **USArrests** dataset and lowercase the variables and the state names to make the matching across different datasets uniform.

```
arrests <- USArrests
names(arrests) <- tolower(names(arrests))
arrests$region <- tolower(rownames(USArrests))
head(arrests)

##          murder assault urbanpop rape      region
## Alabama    13.2     236      58 21.2    alabama
## Alaska     10.0     263      48 44.5    alaska
## Arizona     8.1     294      80 31.0   arizona
....
```

Then we merge the statistics with the spatial data, in readiness for mapping.

```
states <- map_data("state")
head(states)

##    long   lat group order region subregion
## 1 -87.46 30.39     1     1 alabama      <NA>
## 2 -87.48 30.37     1     2 alabama      <NA>
## 3 -87.53 30.37     1     3 alabama      <NA>
.....

ds <- merge(states, arrests, sort=FALSE, by="region")
head(ds)

##    region   long   lat group order subregion murder assault urbanpop rape
## 1 alabama -87.46 30.39     1     1      <NA>   13.2     236      58 21.2
## 2 alabama -87.48 30.37     1     2      <NA>   13.2     236      58 21.2
## 3 alabama -87.95 30.25     1    13      <NA>   13.2     236      58 21.2
.....

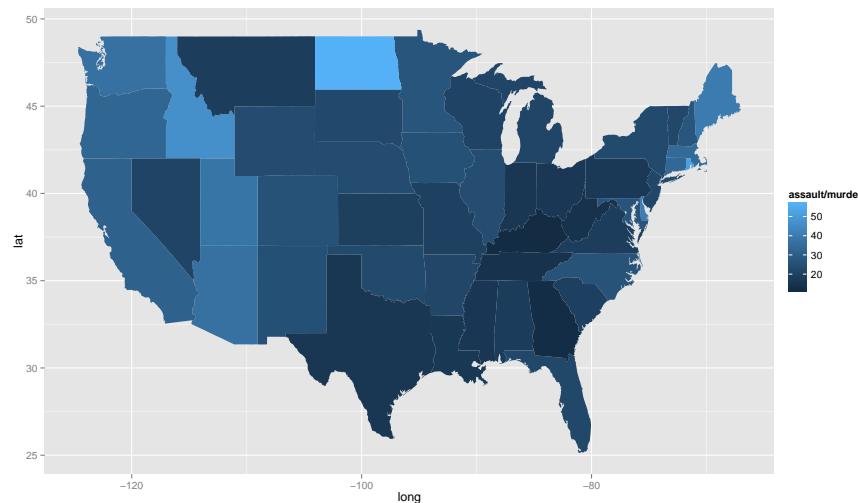
ds <- ds[order(ds$order), ]
head(ds)

##    region   long   lat group order subregion murder assault urbanpop rape
## 1 alabama -87.46 30.39     1     1      <NA>   13.2     236      58 21.2
## 2 alabama -87.48 30.37     1     2      <NA>   13.2     236      58 21.2
## 6 alabama -87.53 30.37     1     3      <NA>   13.2     236      58 21.2
.....
```

33 USA Arrests: Assaults per Murder Map

Once we have the data ready, plotting it simply requires nominating the dataset, and identifying the x and y as long and lat respectively. We also need to identify the grouping, which is by state, and so the fill is then specified for each state to indicate the statistic of interest.

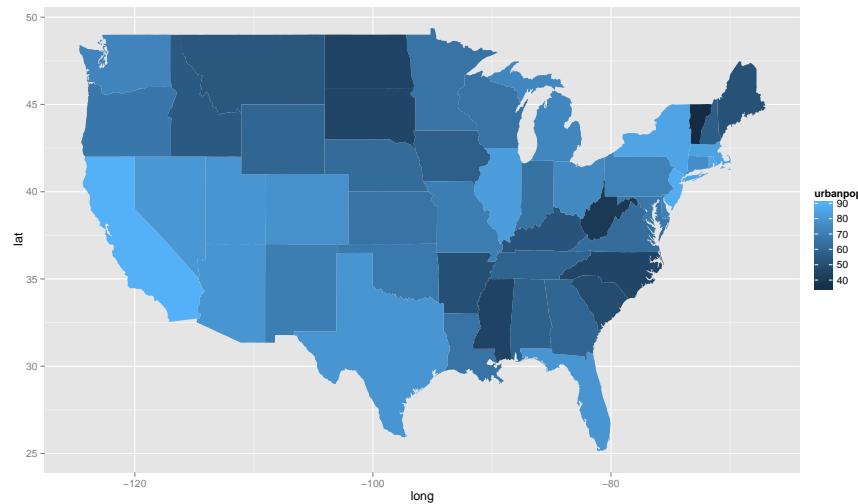
```
g <- ggplot(ds, aes(long, lat, group=group, fill=assault/murder))
g <- g + geom_polygon()
print(g)
```



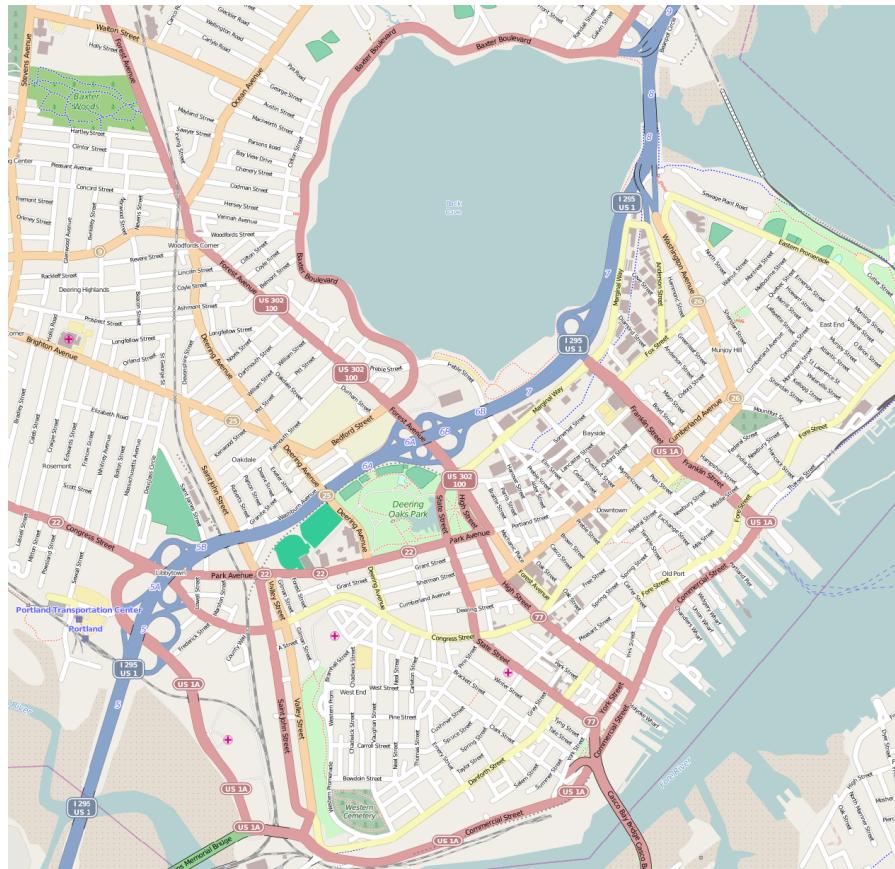
Exercise:
How can
we change
the colour
of the
scale?

We might also be interested in plotting the percentage of urban population in each state.

```
g <- ggplot(ds, aes(long, lat, group=group, fill=urbanpop))
g <- g + geom_polygon()
print(g)
```

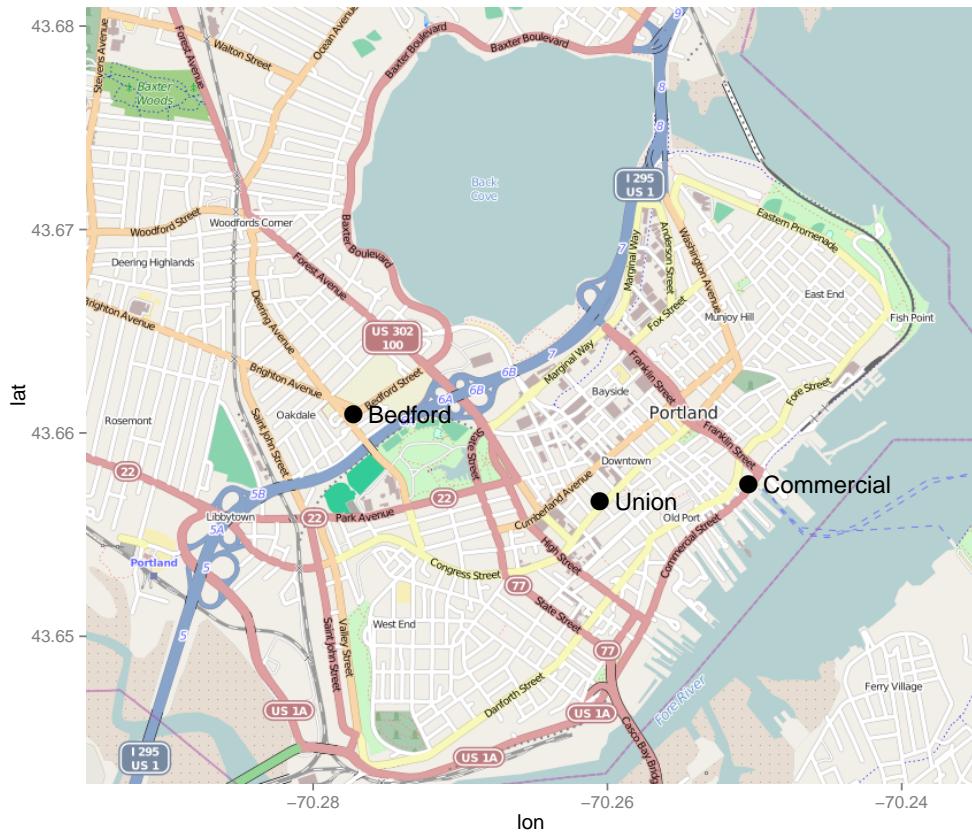


34 Portland: Open Street Maps



```
library(OpenStreetMap)
stores <- data.frame(name=c("Commercial", "Union", "Bedford"),
                      lon=c(-70.25042295455, -70.26050806045, -70.27726650238),
                      lat=c(43.657471302616, 43.65663299041, 43.66091757424))
lat <- c(43.68093, 43.64278)
lon <- c(-70.29548, -70.24097)
portland <- openmap(c(lat[1],lon[1]),c(lat[2],lon[2]), zoom=15, 'osm')
plot(portland, raster=TRUE)
```

35 Portland: Annotated Maps



From <http://stackoverflow.com/questions/10686054/outlined-text-with-ggplot2>

```

stores <- data.frame(name=c("Commercial", "Union", "Bedford"),
                      lon=c(-70.25042295455, -70.26050806045, -70.27726650238),
                      lat=c( 43.65747130261, 43.656632990041, 43.66091757424))

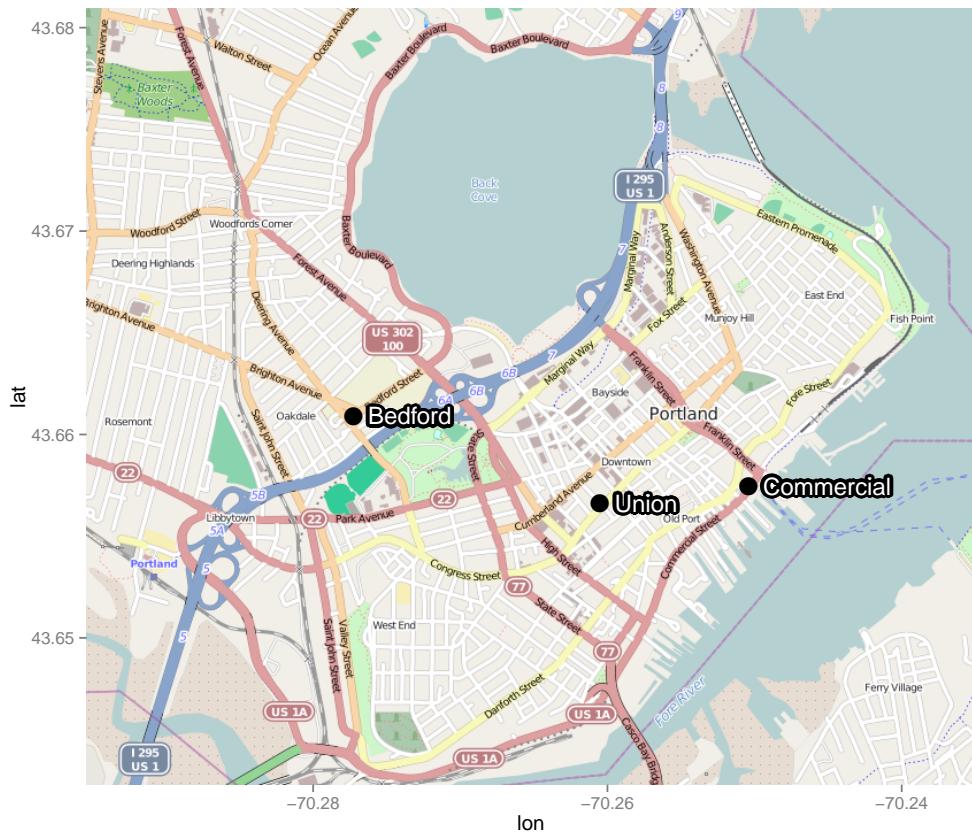
portland <- c(-70.2954, 43.64278, -70.2350, 43.68093)

library(ggmap)
map <- get_map(location=portland, source="osm")

g <- ggmap(map)
g <- g + geom_point(data=stores, aes(x=lon, y=lat), size=5)
g <- g + geom_text(data=stores, aes(label=name, x=lon+.001, y=lat), hjust=0)
print(g)

```

36 Portland: Standout Text Annotation



This map uses outlined text to ensure we can better read text on the map.

```
g <- ggmap(map)
g <- g + geom_point(data=stores, aes(x=lon, y=lat), size=5)

theta <- seq(pi/16, 2*pi, length.out=32)
xo <- diff(portland[c(1,3)])/250
yo <- diff(portland[c(2,4)])/250

for(i in theta)
  g <- g + geom_text(data=stores, bquote(aes(x=lon + .001 + .(cos(i) * xo),
                                             y=lat + .(sin(i) * yo), label = name)),
                      size=5, colour='black', hjust=0)

g <- g + geom_text(data=stores, aes(x=lon+.001, y=lat, label=name),
                     size=5, colour='white', hjust=0)
print(g)
```

37 Animated Maps

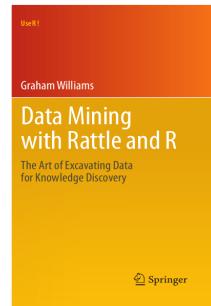
See <http://rmaps.github.io/blog/posts/animated-choropleths/>.

DRAFT

38 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 packages `RgoogleMaps` (?) and `OpenStreetMap` (?) provide alternative interfaces to Google Maps, etc., whereby we can download maps and annotate them. Much of the functionality we have already seen is provided by `ggmap` (?). `OpenStreetMap` provides `automap()`. `RgoogleMaps` provides `GetMap()` to download a map from Google, and `PlotOnStaticMap()` for displaying the map and overlaying plots on the map.
- The package `osmar` (?) provides an interactive environment based on OpenStreetMap.

Tony's ToDo

stats between locations like distance
 calculate an area on a map
 colour code regions
 regions like ABS has
 ability to plot graphs either on map or in a new window with stats as meta data
 ability to have a region shaded with locations pinpointed on top
 dialogue box which describes data on location
 have images on a map, and you can filter out, with a simple scroll option.
 a 3d options on a property like bing maps has
 the ability to make a move to fly through way points on a map
 radius zones around a point.
 radius points around multiple points, like a distorted ven diagram
 stats that change over time like a movie but static location, but graphs change
 time change colour for night and day or both

39 References

- Brownrigg R (2014). *maps: Draw Geographical Maps*. R package version 2.3-9, URL <http://CRAN.R-project.org/package=maps>.
- 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/>.
- 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>.
- 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 3.0.4, URL <http://rattle.togaware.com/>.

This document, sourced from Maps.Rnw revision 534, was processed by Knitr version 1.6 of 2014-05-24 and took 86.4 seconds to process. It was generated by gjw on theano running Ubuntu 14.04.1 LTS with Intel(R) Core(TM) i7-3517U CPU @ 1.90GHz having 4 cores and 3.9GB of RAM. It completed the processing 2014-10-28 14:54:04.

DRAFT

Hands-On Data Science with R

Dealing with Big Data

Graham.Williams@togaware.com

27th November 2014

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

In this module we explore how to load larger datasets into R and provide operations for processing larger data within R. Whilst we will demonstrate some timing differences between approaches, the purpose of this chapter is to present efficient approaches for dealing with large datasets rather than a systematic comparison of alternatives.

The required packages for this module include:

```
library(data.table)    # Efficient data storage
library(dplyr)         # Efficient data manipulation
library(rattle)        # For normVarNames.
#library(rbenchmark)
#library(sqlite)
```

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 Loading Data From CSV

Loading data from a CSV file is one of the simplest ways to load data into R. The standard function to do so is `read.csv()` but it has some inherent default inefficiencies. We can instead use `fread()` from `data.table` (Dowle *et al.*, 2014) to more quickly load data into R. The time differences can be significant, and indeed, can be the difference between being able to load big data and not being able to.

To illustrate we will use a small dataset, and so we expect to see small absolute improvements in time, although the relative improvements are demonstrated as significant. The dataset we use is the `weatherAUS` dataset from `rattle` (Williams, 2014), available as a CSV file.

```
fnwa <- file.path("data", "weatherAUS.csv")
cat(format(file.info(fnwa)$size, big.mark=", "), "\n")
## 7,426,513
```

Loading the dataset from `data/weatherAUS.csv` using `read.csv()` takes about 1 second.

```
system.time(ds <- read.csv(file=fnwa))

##    user  system elapsed
##   1.152   0.004   1.157

dim(ds)
## [1] 66672     24

class(ds)
## [1] "data.frame"
```

Using `fread()` we see that the time taken is quite a bit less:

```
system.time(ds <- fread(input=fnwa))

##    user  system elapsed
##   0.127   0.000   0.127

dim(ds)
## [1] 66672     24

class(ds)
## [1] "data.table" "data.frame"
```

Loading the same dataset from `data/weatherAUS.csv` using `fread()` takes just 15% of the time required by `read.csv()`. If this translated through to larger datasets, then something that might take a day to load (e.g., 100 million observations of 1,000 variables) using `read.csv()` will take 4 hours (though generally less) using `fread()`.

2 Loading Big Data from CSV—Options For `read.csv()`

As the datasets get larger, `read.csv()` becomes less useful for reading from file. When reading very much larger CSV files the times can be quite substantial. We can still use `read.csv()` but with some options we can do better.

One suggestion is to tell `read.csv()` the number of rows that we wish to load, using `nrow=`. Even though we might set this to a value slightly larger than the number of rows in the dataset, it is useful to do so, since it seems to let R know not to allocate too much unnecessary memory. In general R will not know how much memory will be needed to load the dataset into, and so may attempt to allocate much more than actually required, as it is processing the data. Without telling R an expected number of rows we can run out of memory on smaller memory machines. Setting it to a size slightly larger than the number of rows to be read is advised.

```
ds <- read.csv(file=fnbd, nrow=1.1e6)
```

We can also slightly improve `read.csv()` load times by not converting strings to factors.

```
ds <- read.csv(file=fnbd, nrow=1.1e6, stringsAsFactors=FALSE)
```

Using `fread()` we don't need to take the above precautions, such as specifying a row number, since `fread()` is well tuned to loading large datasets fast.

```
ds <- fread(input=fnbd, showProgress=FALSE))
```

We can help `read.csv()` quite a bit more, and avoid a lot of extra processing and memory usage by telling `read.csv()` the data types of the columns of the CSV file. We do this using `colClasses=`.

Often though we are not all that sure what the classes should be, and there may be many of them, and it would not be reasonable to have to manually specify each one of them. We can get a pretty good idea by reading only a part of the CSV file as to the data types of the various columns.

A common approach is to make use of `nrows=` to read a limited number of rows. From the first 5000 rows, for example, we can extract the classes of the columns as automatically determined by R and then use that information to read the whole dataset.

Note firstly the expected significant reduction in time in reading just the first 5,000 rows.

```
system.time(dsf <- read.csv(file=fnwa, nrows=5e3))

##    user  system elapsed
##  0.037   0.000   0.036

classes <- sapply(dsf, class)
classes

##          Date      Location      MinTemp      MaxTemp      Rainfall
## "factor" "factor" "numeric" "numeric" "numeric"
##   Evaporation      Sunshine WindGustDir WindGustSpeed WindDir9am
## "numeric" "numeric" "factor"   "integer"   "factor"
## ...
##
```

Now we can read the full dataset without R having to do so much parsing:

```
system.time(dsf <- read.csv(file=fnwa, colClasses=classes))  
##    user  system elapsed  
##  0.610   0.004   0.614
```

For a small dataset, as used here for illustration, the timing differences are in fractions of a second. Here we use the larger dataset and we again communicate the number of rows to avoid R seeking too much memory:

```
ds <- read.csv(file=fnbd, nrows=5e3)  
classes <- sapply(ds, class)  
ds <- read.csv(file=fnbd, nrows=1.1e6, colClasses=classes)  
class(ds)  
dim(ds)
```

For our 10GB dataset, with 2 million observations and 1000 variables, this approach reduces the read time from 30 minutes to 10 minutes.

3 Efficient Data Manipulation with DPLYR

The `dplyr` (Wickham and Francois, 2014) package has focused on providing very efficient data manipulations over data frames and data tables. We will use our small data table to illustrate.

```
ds <- fread(input=fnwa)
setnames(ds, names(ds), normVarNames(names(ds)))

## Loading required package: stringr
```

The basic concept for the grammar defined by dplyr is to split the data, apply some operation to the groups, then combine the data.

Here we first split the data using `group_by`:

```
g <- ds %>% group_by(location)
g

## Source: local data table [66,672 x 24]
## Groups: location
##
##       date location min_temp max_temp rainfall evaporation sunshine
## 1 2008-07-01 Adelaide    8.8     15.7      5.0      1.6     2.6
## 2 2008-07-02 Adelaide   12.7     15.8      0.8      1.4     7.8
## 3 2008-07-03 Adelaide    6.2     15.1      0.0      1.8     2.1
## 4 2008-07-04 Adelaide    5.3     15.9      0.0      1.4     8.0
## 5 2008-07-05 Adelaide    9.8     15.4      0.0      NA     0.9
## 6 2008-07-06 Adelaide   11.3     15.7      NA      NA     1.5
## 7 2008-07-07 Adelaide    7.6     11.2     16.2      4.6     1.1
## 8 2008-07-08 Adelaide    5.3     13.5     17.0      0.6     2.1
## 9 2008-07-09 Adelaide    8.4     14.3      1.8      1.6     0.8
## 10 2008-07-10 Adelaide   9.5     13.1      9.0      1.2     7.2
## ...
## Variables not shown: wind_gust_dir (chr), wind_gust_speed (int),
## wind_dir_9am (chr), wind_dir_3pm (chr), wind_speed_9am (int),
## wind_speed_3pm (int), humidity_9am (int), humidity_3pm (int),
## pressure_9am (dbl), pressure_3pm (dbl), cloud_9am (int), cloud_3pm
## (int), temp_9am (dbl), temp_3pm (dbl), rain_today (chr), risk_mm (dbl),
## rain_tomorrow (chr)
```

Notice the compact form for printing a human readable for of the dataset.

For each group we might want to report the average daily rainfall:

```
g <- g %>% summarise(daily_rainfall = mean(rainfall, na.rm=TRUE))
g

## Source: local data table [46 x 2]
##
##       location daily_rainfall
## 1      Adelaide     1.5569191
## ...
```

We can of course combine this in one pipeline:

```
ds %>%
  group_by(location) %>%
  summarise(daily_rainfall = mean(rainfall, na.rm=TRUE))

## Source: local data table [46 x 2]
##
##           location daily_rainfall
## 1          Adelaide      1.5569191
## ...
```

We can use either format, depending on circumstance - both can be quite readable.

4 Archetypes—Overview

We introduce here a new approach to handling big data for model building. The approach is called Acrhetypes.

DRAFT

5 Archetypes Phase 1—Cluster the Population

For example, cluster on missing values. This can be interpreted as a behavioural group. We illustrate using `wskm` (?).

DRAFT

6 Archetypes Phase 2—Rule Induction to Nuggets

For example, build decision trees for each cluster, convert to rules, and each rule is then a Nugget. We illustrate with `wsrpart` (?).

DRAFT

7 Archetypes Phase 3—Local Model Build per Nugget

Now build a predictive model for each nugget. If not a predictive model then perhaps at least a formulation of the interestingness of the nugget. We illustrate with `wsrf` (?).

DRAFT

8 Archetypes Phase 4—Global Model

Now build a single formula to combine the local models as they pertain to scoring new observations. We could use linear regression or use genetic programming. We illustrate with `rgp`(?).

DRAFT

9 Working with Data Tables

Whereas we index a data frame by the observation (i) and the columns (j), the basic syntax of the data table takes a very different view, and one more familiar to database users. In terms of the familiar SQL syntax, we might think of a data table access as specifying the condition to be met for the observations we wish to extract (a WHERE clause), the columns we wish to extract (a SELECT clause) and how the data is to be aggregated (a GROUP BY clause).

```
ds [WHERE, SELECT, by=GROUPBY]
```

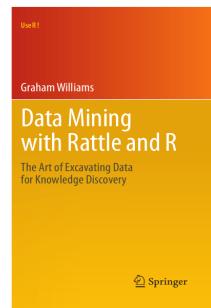
10 Further Reading and Acknowledgements

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:

- A good overview of `data.table` is available through the [useR!2014 Tutorial](#).
- An extensive cheat sheet is available from <http://blog.datacamp.com/data-table-cheat-sheet/>.



11 References

- Dowle M, Short T, Lianoglou S, with contributions from R Saporta AS, Antonyan E (2014). *data.table: Extension of data.frame*. R package version 1.9.4, URL <http://CRAN.R-project.org/package=data.table>.
- 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/>.
- Wickham H, Francois R (2014). *dplyr: A Grammar of Data Manipulation*. R package version 0.3, 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 3.0.4, URL <http://rattle.togaware.com/>.

This document, sourced from BigDataO.Rnw revision 542, was processed by KnitR version 1.8 of 2014-11-11 and took 3.6 seconds to process. It was generated by gjw on theano running Ubuntu 14.04.1 LTS with Intel(R) Core(TM) i7-3517U CPU @ 1.90GHz having 4 cores and 3.9GB of RAM. It completed the processing 2014-11-27 14:01:11.

DRAFT

Hands-On Data Science with R

Miscellaneous Plots in R

Graham.Williams@togaware.com

26th August 2014

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

In this chapter we explore a variety of plots generated using R. For an introduction to plots based on ggplot2 see the specific GGPLOT2 chapter.

The required packages for this module include:

```
library(ipplots)
library(ggplot2)
library(tabplot)
library(rattle)
library(dplyr)
```

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.

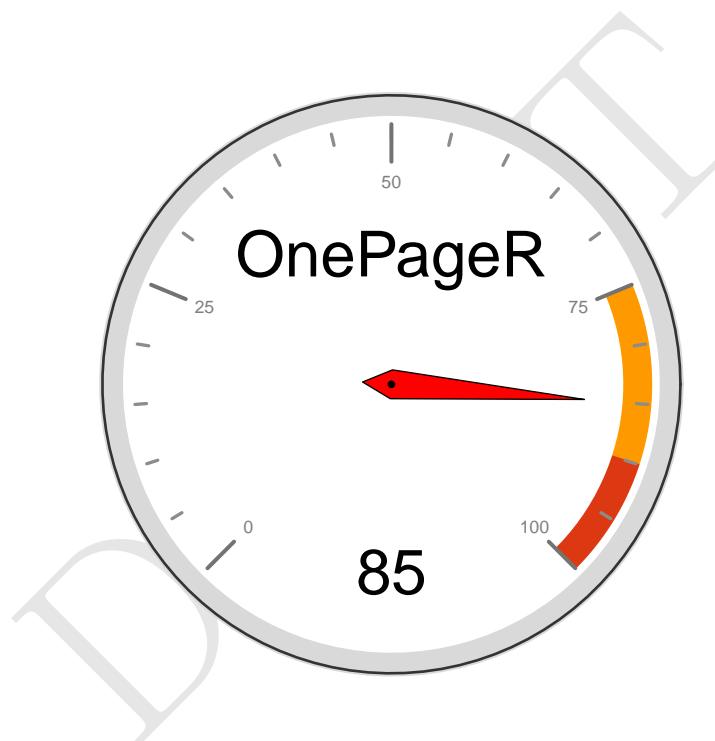


1 Dial Plot

The dial plot is available as the [Google Gauge Plot](#). Pentaho Business Intelligence provides the dial plot for dashboards. However, [Hadley Wickham](#) suggests we “are trying to understand your data, not driving a racing car or aeroplane.” Hadley points us to the work of [Stephen Few](#) who presents the “powerful and eloquent” arguments and suggests alternatives for the most effective presentation of data.

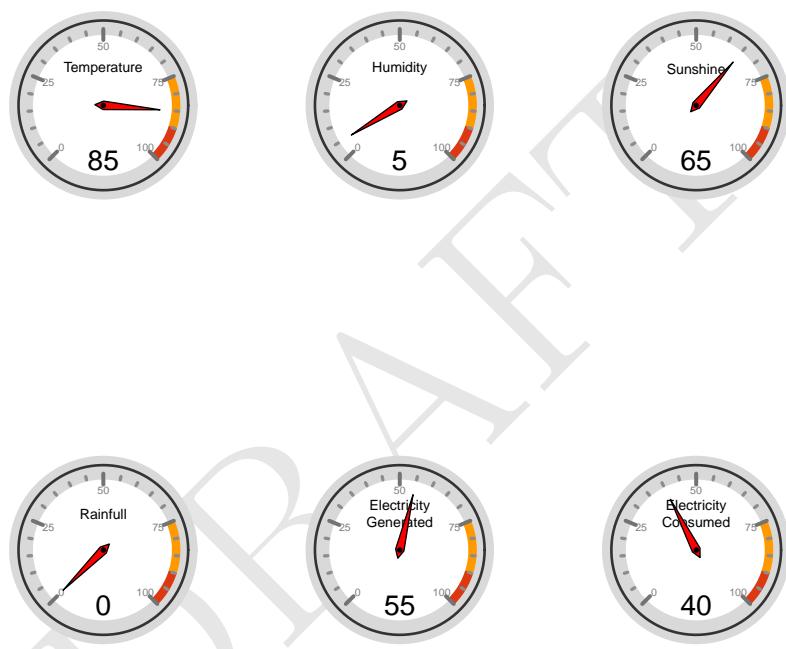
Nonetheless, [Gaston Sanchez](#) wrote and Jeff Hemsley modified a version of `dial.plot()` for R.

```
source("http://onepager.togaware.com/dial.plot.R")
dial.plot(label="OnePageR", value=85)
```

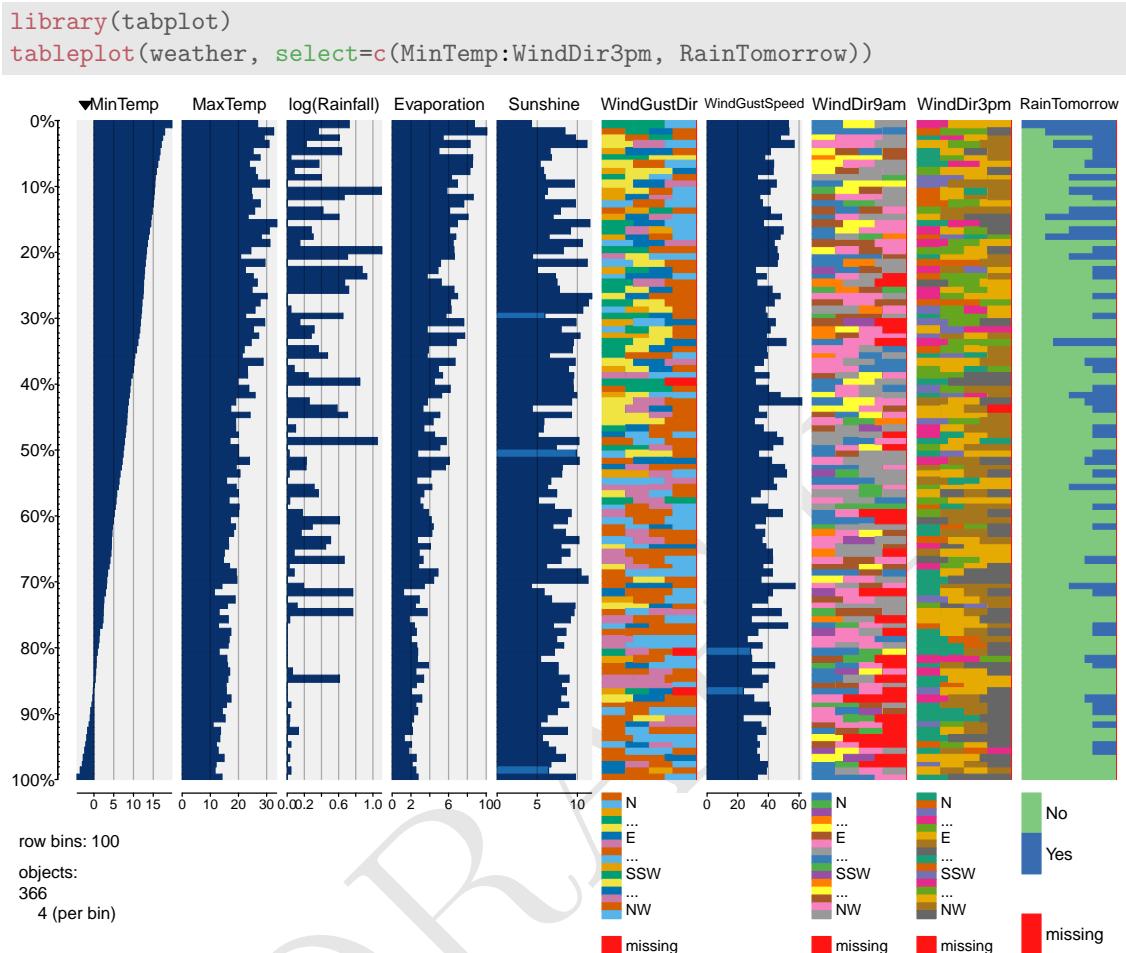


2 Dashboard

```
opar <- par(mfrow=c(2,3))
dial.plot(label="Temperature", label.cex=1, value=85, value.cex=2)
dial.plot(label="Humidity", label.cex=1, value=5, value.cex=2)
dial.plot(label="Sunshine", label.cex=1, value=65, value.cex=2)
dial.plot(label="Rainfull", label.cex=1, value=0, value.cex=2)
dial.plot(label="Electricity\nGenerated", label.cex=1, value=55, value.cex=2)
dial.plot(label="Electricity\nConsumed", label.cex=1, value=40, value.cex=2)
```

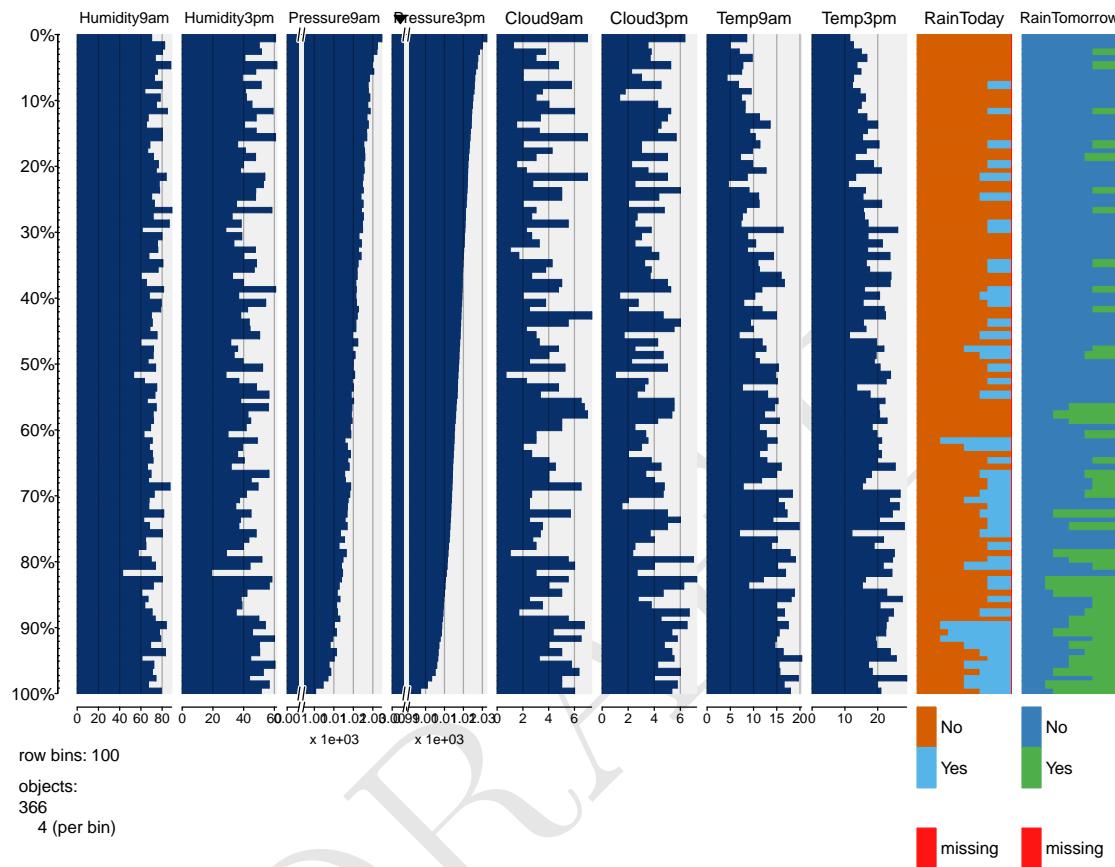


3 Table Plots



4 Table Plots

```
library(tabplot)
tableplot(weather, select=c(Humidity9am:RainToday, RainTomorrow), sortCol=Pressure3pm)
```



5 Visually Weighted Regression

From Nicrebrede www.nicebread.de (and posted on Bloggers on R) by Felix Schoenbrodt 30 August 2012 addressing Solomon Hsiang's proposal of an appealing method for visually displaying the uncertainty in regressions and using shading in response to Gelman's note that traditional statistical summaries such as 95% intervals give too much weight to the edges.

DRAFT

6 F1: Exploring the Dataset

We can now explore a particular dataset using `ggplot2` graphics to get an understanding of the story behind the data. The data and the original plots (some are now modified) are from [Tony Hirst's blog](#).

```
(load("data/f1.RData"))
## [1] "f1"

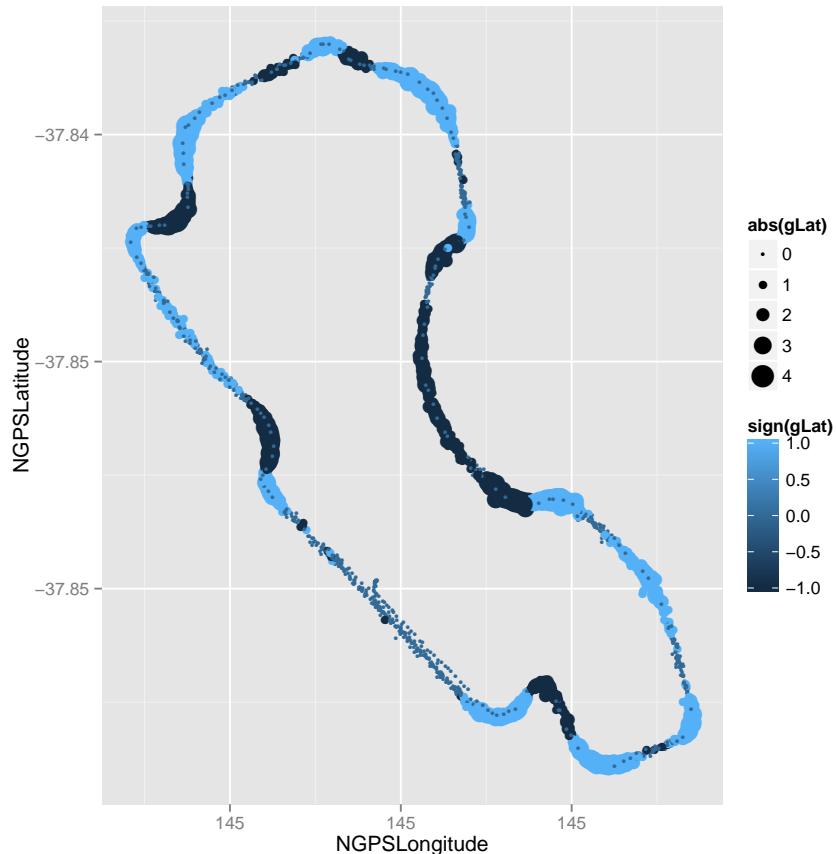
head(f1)

##      file timestamp NGPSLatitude NGPSLongitude NGear nEngine
## 1 1269758114 17:35:10       -37.85          145     4    13422
## 2 1269758115 17:35:11       -37.85          145     3    13383
## 3 1269758116 17:35:12       -37.85          145     3    14145
....
```

7 F1: Simple Map

We can draw the particular F1 circuit using the longitude and latitude as the x and y coordinates, and using the sign of the latitudinal g-force on the driver. We believe that a positive value of `gLat` indicates force to the left and a negative value indicates a force to the right.

```
p <- ggplot(f1, aes(NGPSLongitude, NGPSLatitude))
p <- p + geom_point(aes(col=sign(gLat), size=abs(gLat)))
p
```



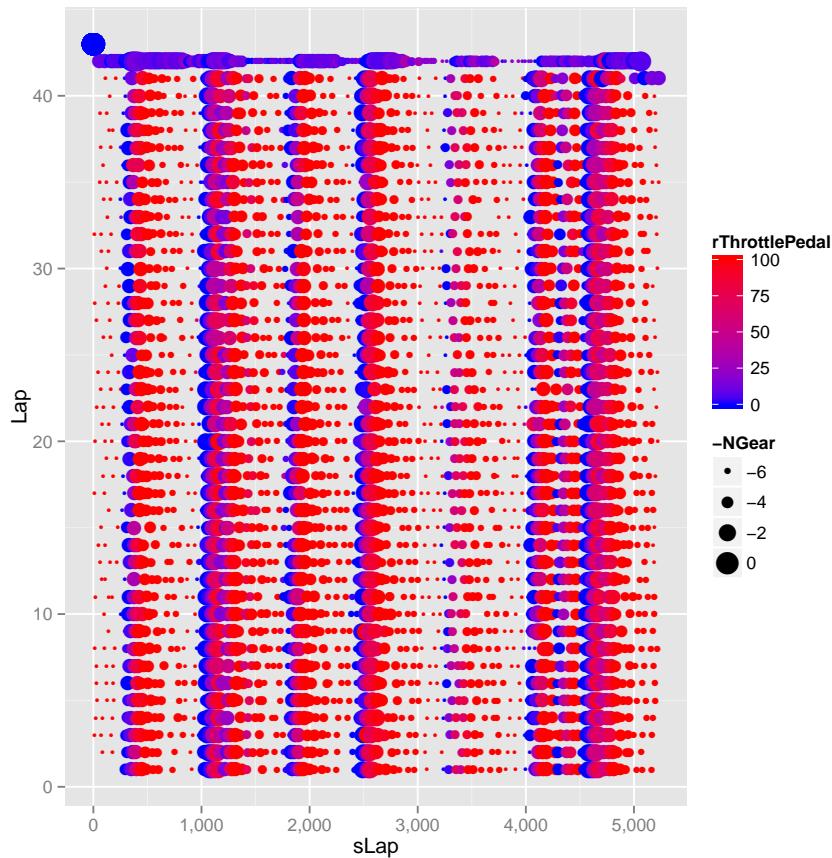
Based on Tony Hirst's Blog Post, March 2012

We should be able to see from the plot the forces on the driver on the left and right hand corners, and see how tight the corner is based on the size of the dots.

8 F1: Driver Behaviour

We can explore the driver's behaviour in using low gear and throttle. The distance around the track is plotted on the x-axis and the lap number on y axis. The node size is inversely proportional to gear number (low gear, large point size) and the colour is the relative amount of throttle pedal depression.

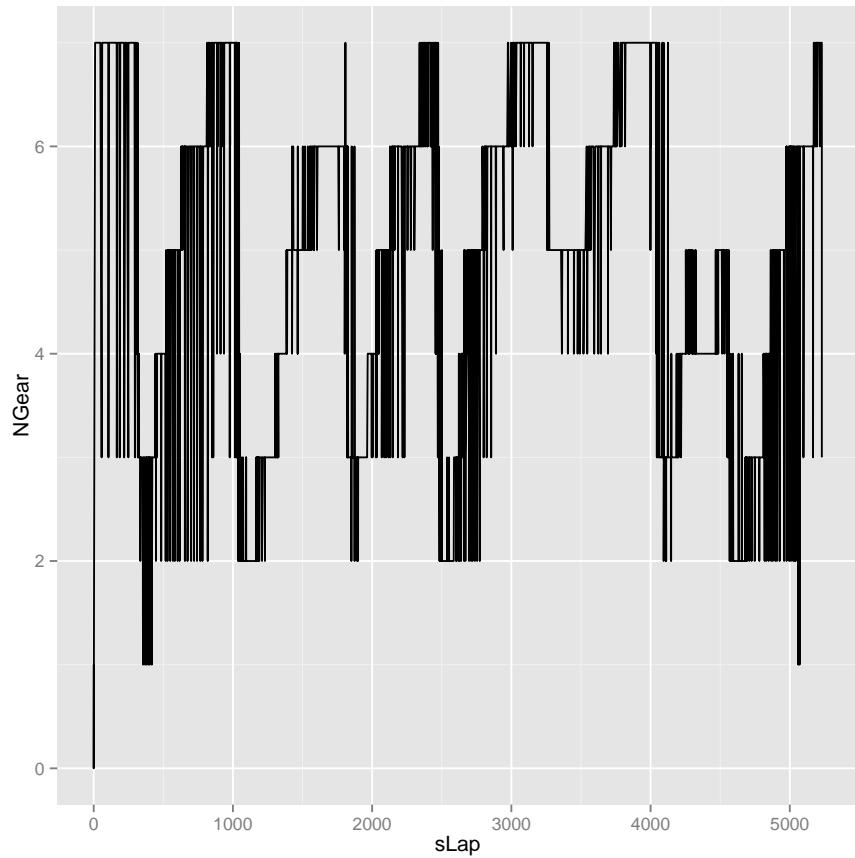
```
library(scales)
p <- ggplot(f1, aes(sLap, Lap))
p <- p + geom_point(aes(col=rThrottlePedal, size=-NGear))
p <- p + scale_colour_gradient(low="blue", high="red")
p <- p + scale_x_continuous(labels=comma)
p
```



Based on Tony Hirst's Blog Post, March 2012

9 F1: Gear Usage Around the Track

```
p <- ggplot(f1, aes(sLap, NGear))
p <- p + geom_line()
p
```

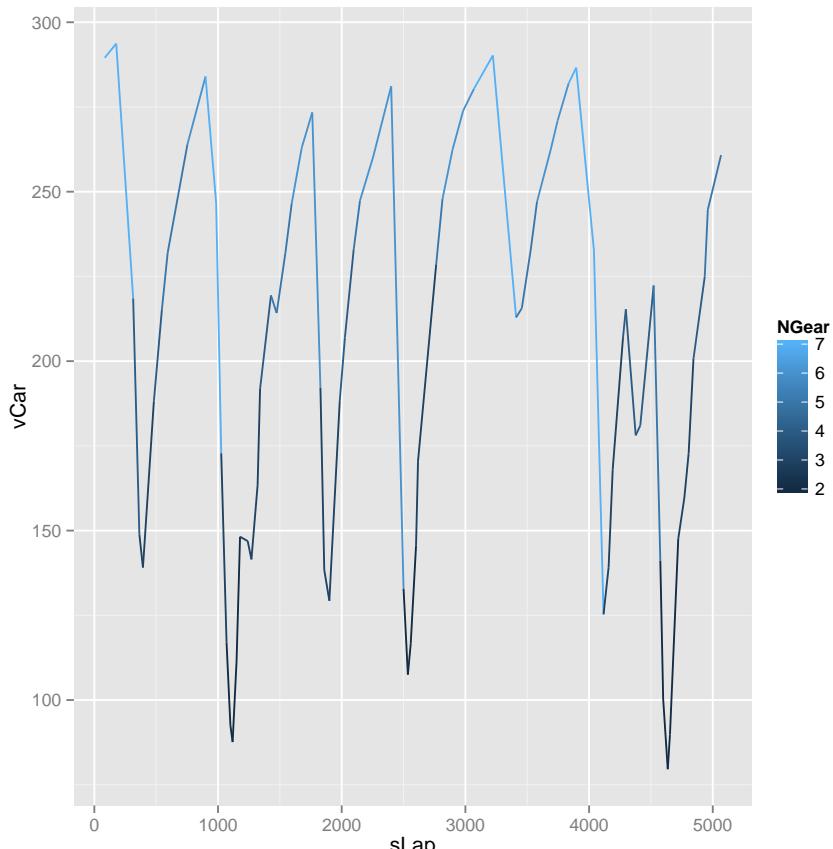


Based on Tony Hirst's Blog Post, March 2012

10 F1: Trace a Single Lap

We can trace a single lap to display the speed (y-axis) coloured by gear as the vehicle travels around the circuit:

```
ggplot(subset(f1, Lap==2), aes(sLap, vCar)) +  
  geom_line(aes(colour=NGear))
```

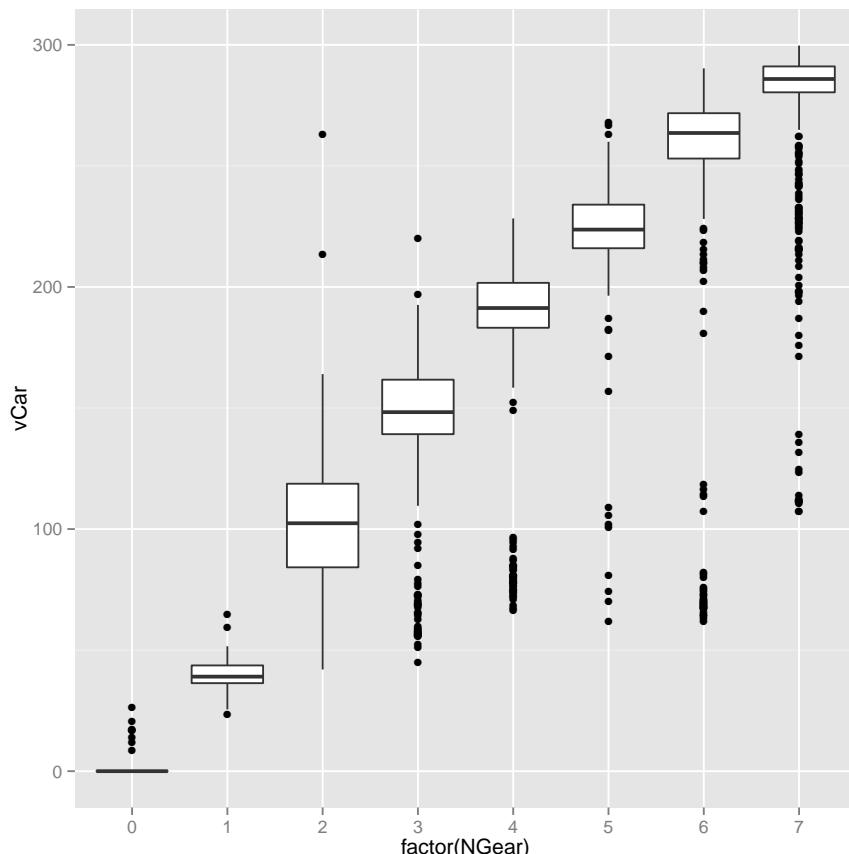


Based on Tony Hirst's Blog Post, March 2012

11 F1: Box Plot of Speed by Gear

Statistical graphics provide important insights. The box plot here makes sense, in that higher gears correspond to higher speeds.

```
ggplot(f1, aes(factor(NGear), vCar)) +  
  geom_boxplot()
```

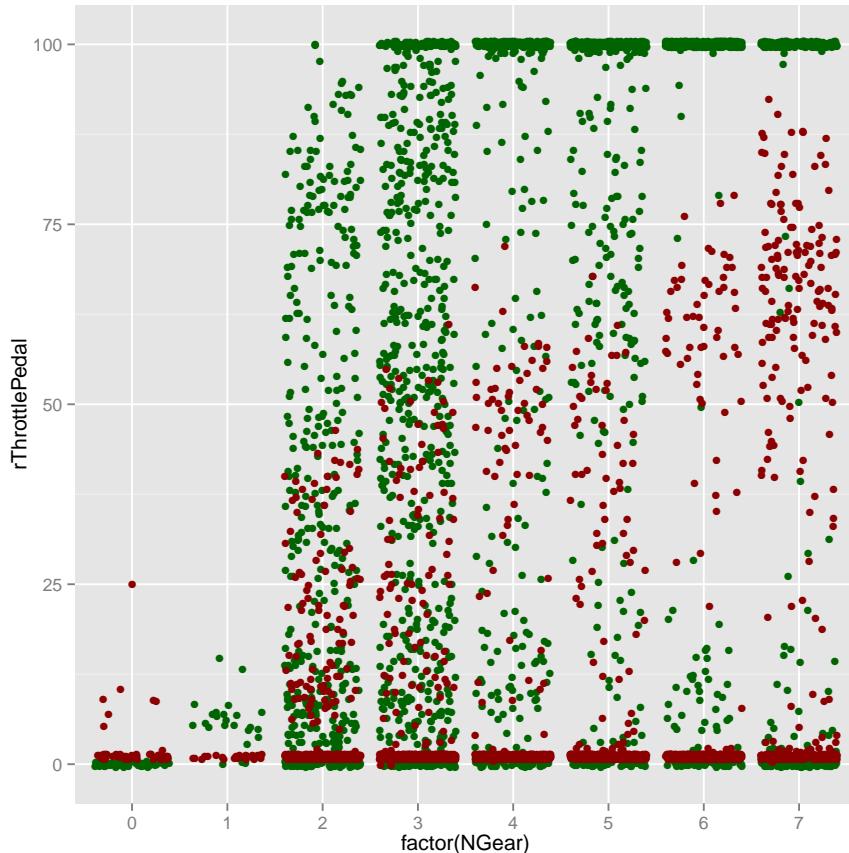


Based on Tony Hirst's Blog Post, March 2012

12 F1: Footwork

How busy are the feet? We can summarise the brake (red) and throttle (green) depression based on gear.

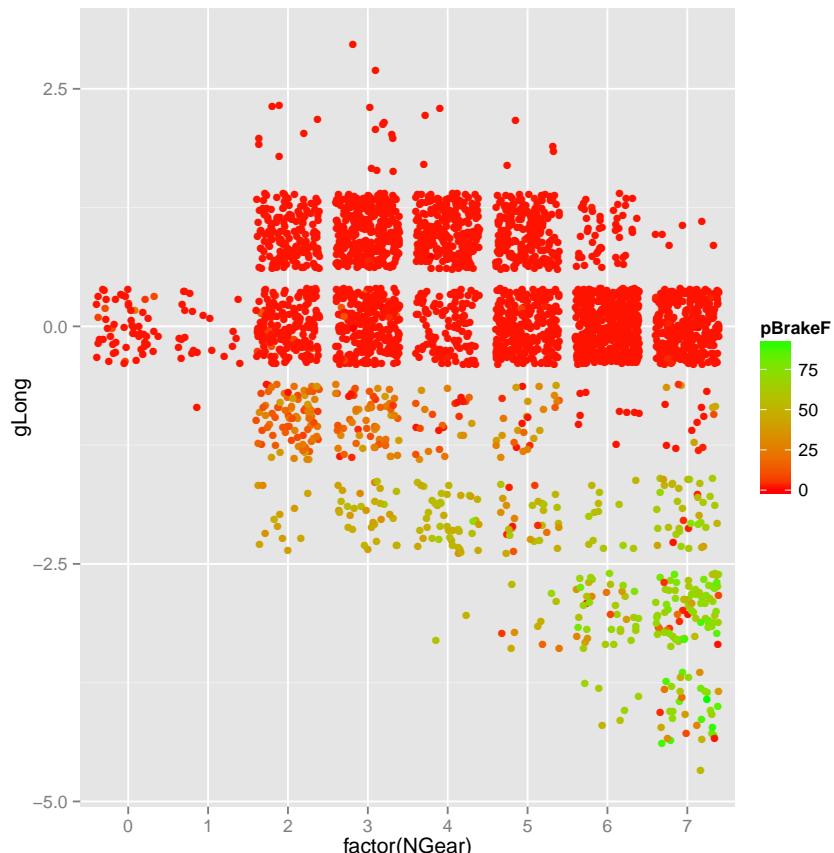
```
ggplot(f1, aes(factor(NGear))) +  
  geom_jitter(aes(y=rThrottlePedal), colour='darkgreen') +  
  geom_jitter(aes(y=pBrakeF), colour='darkred')
```



Based on Tony Hirst's Blog Post, March 2012

13 F1: Forces on the Driver

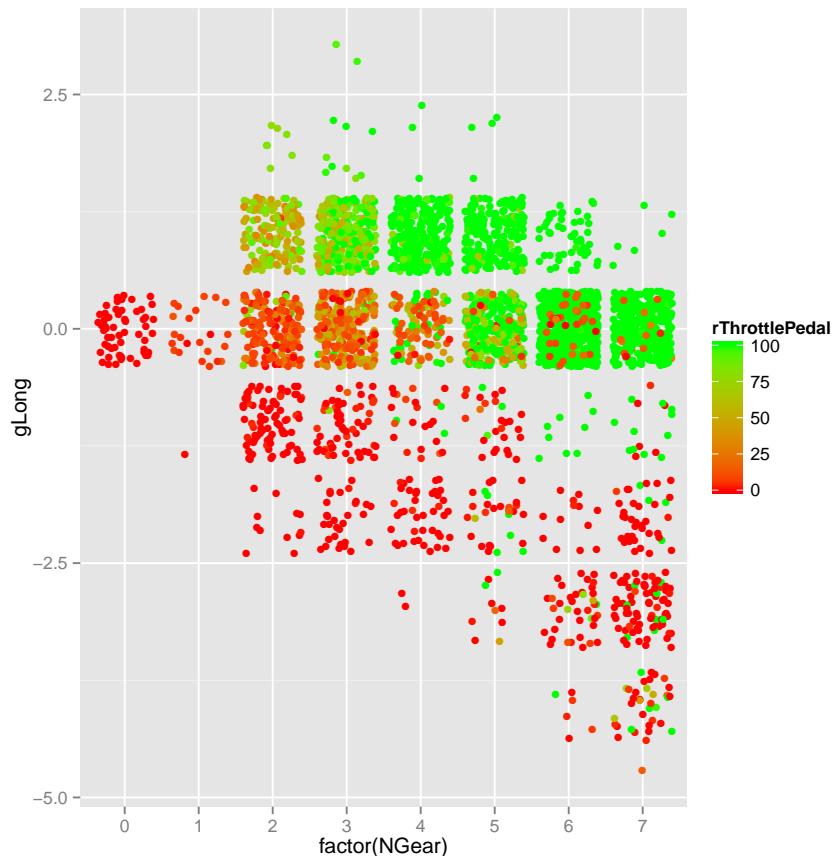
```
ggplot(f1, aes(factor(NGear), gLong)) +  
  geom_jitter(aes(col=pBrakeF)) +  
  scale_colour_gradient(low='red', high='green')
```



Based on Tony Hirst's Blog Post, March 2012

14 F1: More Forces

```
ggplot(f1, aes(factor(NGear), gLong)) +  
  geom_jitter(aes(col=rThrottlePedal)) +  
  scale_colour_gradient(low='red', high="green")
```

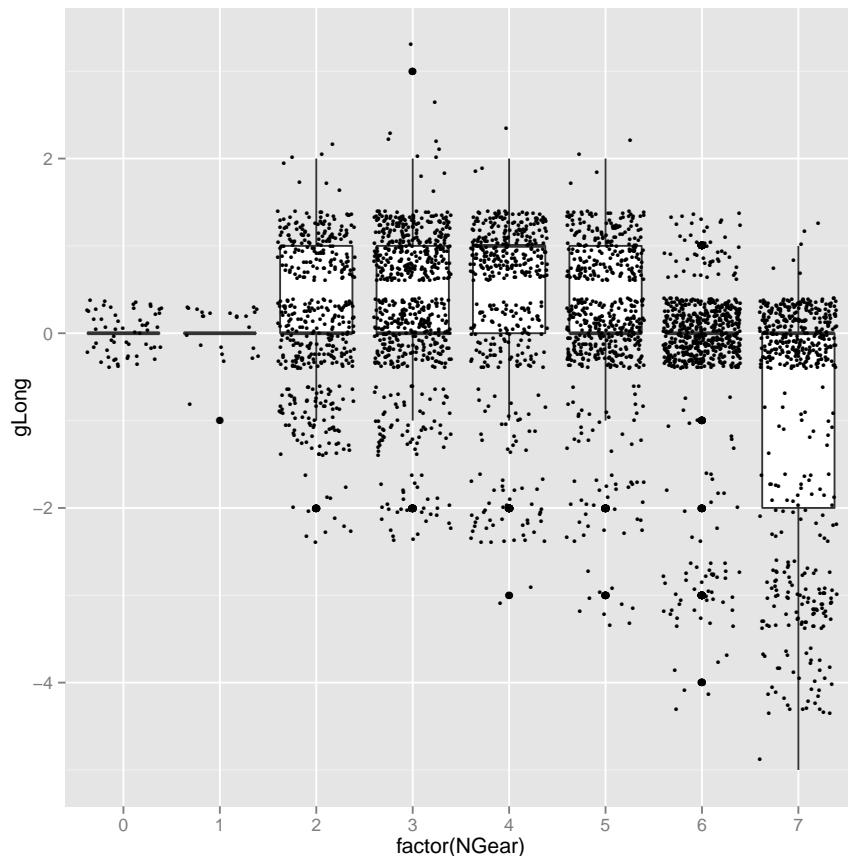


Based on Tony Hirst's Blog Post, March 2012

15 F1: Box Plot of Forces

We can use a box plot to investigate the longitudinal g-force's relationship with acceleration or braking by gear. Note that a random jitter is used to scatter points around their actual integer values.

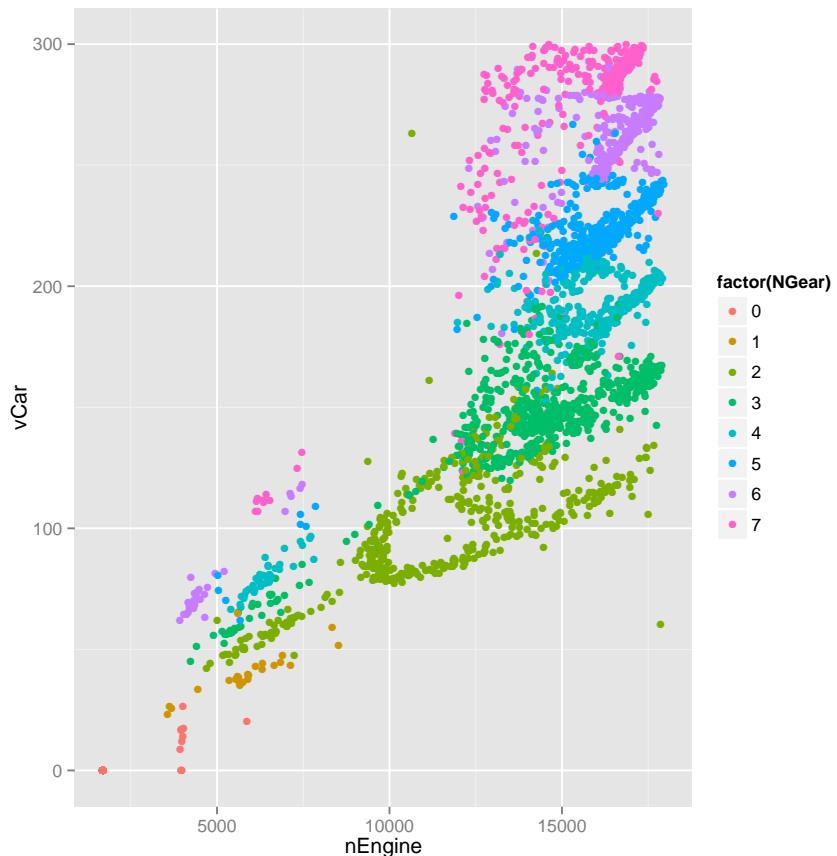
```
ggplot(f1, aes(factor(NGear), gLong)) +  
  geom_boxplot() +  
  geom_jitter(size=1)
```



Based on Tony Hirst's Blog Post, March 2012

16 F1: RPM and Speed in Relation to Gear

```
ggplot(f1, aes(nEngine, vCar)) +  
  geom_point(aes(col=factor(NGear)))
```



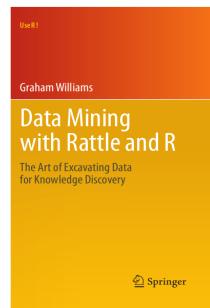
Based on Tony Hirst's Blog Post, March 2012

17 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:



18 References

- 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/>.
- 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>.
- 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.

This document, sourced from PlotsO.Rnw revision 509, was processed by KnitR version 1.6 of 2014-05-24 and took 6.8 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 2014-08-26 20:45:20.

Data Science with R

Writing Functions in R

Graham.Williams@togaware.com

22nd May 2014

Visit <http://onepager.togaware.com/> for more OnePageR's.

The required packages for this module include:

```
library(rattle)
```

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 Functions

```
mult10 <- function(x)
{
  if (is.character(x))
  {
    result <- apply(sapply(x, rep, 10), 2, paste, collapse="")
    names(result) <- NULL
  }
  else
  {
    result <- x * 10
  }

  return(result)
}
```

This page
is under
develop-
ment.

2 Function Calls

```
4 + 5
## [1] 9
"+"(4, 5)
## [1] 9

1 + 2 + 3 + 4 + 5
## [1] 15
Reduce("+", 1:5)
## [1] 15

cmd <- "1 + 2 + 3 + 4 + 5"
eval(parse(text=cmd))
## [1] 15
```

This page
is under
develop-
ment.

3 Flow Control

```
for (i in 0:4)
  for (j in 5:9)
    print(paste0(i, j))

## [1] "05"
## [1] "06"
## [1] "07"
## [1] "08"
....
```

This page
is under
develop-
ment.

4 Exercise Dataset: WeatherAUS

For our exercises we will use the **weatherAUS** dataset from `rattle` (Williams, 2014). We will essentially follow the template presented in the Models module.

```
ds <- read.csv(file="data/weatherAUS.csv")

dim(ds)
## [1] 66672    24

head(ds)

##           Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 1 2008-12-01   Albury    13.4    22.9     0.6        NA       NA
## 2 2008-12-02   Albury     7.4    25.1     0.0        NA       NA
## 3 2008-12-03   Albury    12.9    25.7     0.0        NA       NA
## 4 2008-12-04   Albury     9.2    28.0     0.0        NA       NA
## 5 2008-12-05   Albury    17.5    32.3     1.0        NA       NA
## 6 2008-12-06   Albury    14.6    29.7     0.2        NA       NA
...
tail(ds)

##           Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 66667 2012-11-24 Darwin    25.5    34.1     0.0      5.0      6.2
## 66668 2012-11-25 Darwin    24.4    35.7     0.2      4.8     11.7
## 66669 2012-11-26 Darwin    25.0    35.4     0.0      7.4     11.7
## 66670 2012-11-27 Darwin    26.5    35.9     0.0      8.0     10.3
## 66671 2012-11-28 Darwin    27.4    35.0     0.0      7.8      6.5
## 66672 2012-11-29 Darwin    24.8    33.5     3.4      7.4      4.7
...
str(ds)

## 'data.frame': 66672 obs. of  24 variables:
## $ Date        : Factor w/ 1826 levels "2007-11-01","2007-11-02",...: 397 ...
## $ Location    : Factor w/ 46 levels "Adelaide","Albany",...: 3 3 3 3 3 3 ...
## $ MinTemp     : num  13.4 7.4 12.9 9.2 17.5 14.6 14.3 7.7 9.7 13.1 ...
## $ MaxTemp     : num  22.9 25.1 25.7 28 32.3 29.7 25 26.7 31.9 30.1 ...
## $ Rainfall    : num  0.6 0 0 0 1 0.2 0 0 0 1.4 ...
## $ Evaporation : num  NA NA NA NA NA NA NA NA NA ...
...
summary(ds)

##           Date          Location        MinTemp        MaxTemp
## 2009-01-01: 46  Canberra: 1826  Min.  :-8.5  Min.  :-3.1
## 2009-01-02: 46  Sydney   : 1734  1st Qu.: 7.3  1st Qu.:17.6
## 2009-01-03: 46  Adelaide: 1583  Median :11.7  Median :22.1
## 2009-01-04: 46  Brisbane: 1583  Mean   :11.9  Mean   :22.7
## 2009-01-05: 46  Darwin   : 1583  3rd Qu.:16.6  3rd Qu.:27.5
## 2009-01-06: 46  Hobart   : 1583  Max.   :33.9  Max.   :48.1
...
```

5 Exercises: Prepare for Modelling

Following the template presented in the Models module, we continue with setting up some of the modelling parameters.

```

target <- "RainTomorrow"
risk   <- "RISK_MM"
dsname <- "weather"

ds[target] <- as.factor(ds[[target]])
summary(ds[target])

## RainTomorrow
## No :50187
## Yes :15259
## NA's: 1226
.....
vars     <- colnames(ds)
ignore   <- vars[c(1, 2, if (exists("risk")) which(risk==vars))]
vars     <- setdiff(vars, ignore)
(inputs <- setdiff(vars, target))

## [1] "MinTemp"      "MaxTemp"      "Rainfall"      "Evaporation"
## [5] "Sunshine"     "WindGustDir"   "WindGustSpeed" "WindDir9am"
## [9] "WindDir3pm"    "WindSpeed9am"  "WindSpeed3pm"  "Humidity9am"
## [13] "Humidity3pm"   "Pressure9am"   "Pressure3pm"   "Cloud9am"
.....
nobs    <- nrow(ds)
dim(ds[vars])

## [1] 66672    21

(form <- formula(paste(target, "~ .")))
## RainTomorrow ~ .

set.seed(142)

length(train <- sample(nobs, 0.7*nobs))
## [1] 46670
length(test  <- setdiff(seq_len(nobs), train))
## [1] 20002

```

6 Exercise: varWeights()

The first exercise is to write a function to take a dataset and return probabilities associated with each input variable in the dataset, that relate to the correlation between the input variable and the target variable.

```
varw <- varWeights(form, ds)
```

We will use the R correlation functions to calculate the correlation between each column (variable) of the `data` frame and the values of the `target` vector. Below are some hints.

```
n1 <- ds[["Temp3pm"]]
c1 <- ds[["WindGustDir"]]
t1 <- ds[[target]]

cor(as.numeric(n1), as.numeric(t1), use="pairwise.complete.obs")
## [1] -0.1857
cor(as.numeric(c1), as.numeric(t1), use="pairwise.complete.obs")
## [1] 0.04414
```

The template for the function is:

```
varWeights <- function(formula, data)
{
  ...
}
```

The actual solution will produce the following output:

```
varWeights(form, ds)
##          Date      Location     MinTemp     MaxTemp     Rainfall
## 0.0028545 0.0004961  0.0210742  0.0336550  0.0544825
## Evaporation Sunshine WindGustDir WindGustSpeed WindDir9am
## 0.0249688   0.1006235   0.0096812   0.0518932   0.0067119
....
```

It is time now to write that function.

7 Exercise: selectVars()

The next exercise is to write a function that will return `n` variables chosen at random from all of the variables in a `dataset`, but chosen with a probability proportional to the correlation of the target variable.

```
vars <- selectVars(form, ds, 3)
```

The `sample()` function might come in use for this function. Note the `prob=` argument of `sample`. We will, of course, also make use of the `varWeights()` function we defined previously.

The template for the function is:

```
selectVars <- function(formula, data, n)
{
  ...
}
```

The actual solution will produce the following output:

```
selectVars(form, ds, 3)
## [1] "Pressure3pm" "MaxTemp"      "RISK_MM"
selectVars(form, ds, 3)
## [1] "Cloud9am"   "RISK_MM"      "Sunshine"
selectVars(form, ds, 3)
## [1] "WindDir9am" "Humidity3pm" "Cloud9am"
selectVars(form, ds, 3)
## [1] "Cloud3pm"    "Evaporation"  "Humidity3pm"
```

8 Exercise: wsrpart()

This exercise is to write a function to build a subspace decision tree. The function `wsrpart()` (for weighted subspace rpart) will take a dataset (`data`) and return a decision tree (built using `rpart()`) that uses only a subspace of the variables available. The number of variables to use is, by default, $\text{trunc}(\log_2(n + 1))$ (overridden by `nvars=`) and the variables are chosen according to the weighted selection implemented through `selectVars()`.

The idea is similar, but not identical to, the concept of random forests developed by Leo Breiman. Note that Breiman's original random forest paper (on which this idea is based) specifies $\text{trunc}(\log_2 n + 1)$ which is ambiguous in terms of being either $\text{trunc}(\log_2(n + 1))$ or $\text{trunc}(\log_2(n) + 1)$, although he probably meant the latter.

```
dt <- wsrpart(form, data)
```

The template for the function is:

```
wsrpart <- function(formula, data, nvars, ...)
{
  ...
}
```

Notice the use of “...” in the argument list. This allows us to pass other arguments on through to `rpart()`.

The actual solution will produce the following output:

```
## system.time(model <- wsrpart(form, ds[train, vars]))
```

```
##    user  system elapsed
##   0.159   0.011   2.497
```

```
model
```

```
## A multiple rpart model with 1 tree.
##
```

```
## Variables used (11): Pressure9am, Cloud3pm, Sunshine, Rainfall, WindGustDir,
##                      Humidity3pm, WindDir9am, RainToday,
```

```
....
```

9 Exercise: Multiple Decision Trees

Now extend the function `wsrpart()` to build multiple decision trees. The function will take a formula, a dataset (`data`) and a number of trees to build (`ntrees`), and returns a list of `ntrees` decision trees. Each element of the list will itself be a list, with at least one element named `model`. This is the actual rpart model. The result should be of class `mrpart` for “multiple rpart.”

The actual solution will produce the following output:

```
system.time(model <- wsrpart(form, ds[train, vars], 4))

##    user  system elapsed
##  94.024   2.833 34.958

class(model)

## [1] "mrpart"

length(model)

## [1] 50

class(model[[1]]$model)

## [1] "rpart"

model[[1]]$model

## n=45731 (939 observations deleted due to missingness)
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##
## 1) root 45731 10680 No (0.7665 0.2335)
##    2) RainToday=No 35102  5572 No (0.8413 0.1587) *
##    3) RainToday=Yes 10629  5106 No (0.5196 0.4804)
##      6) Cloud3pm< 6.5 4728  1531 No (0.6762 0.3238) *
##      7) Cloud3pm>=6.5 5901  2326 Yes (0.3942 0.6058) *
```

10 Exercise: predict.mrpart()

Score a Dataset Using the Forest

Define a function `predict.mrpart()`. Returns the proportion of trees voting for the positive case, assuming binary classification models.

This page
is under
develop-
ment.

The actual solution will produce the following output:

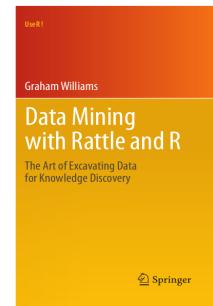
```
predict(model, ds[test,vars])  
##    8     9    12    16    19    27    36    45    46    51    55    59  
##  No   No  Yes   No   No   No   No   No   No   No   No   No  
##  61   62   66   68   78   83   86   87   88   89   93   95  
##  No   No  
....
```

11 Further Reading

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 module is one of many OnePageR modules available from <http://onepager.togaware.com>. In particular follow the links on the website with a * which indicates the generally more developed OnePageR modules.

Other resources include:



12 References

- 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/>.
- 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 3.0.4, URL <http://rattle.togaware.com/>.

This document, sourced from Functions.Rnw revision 370, was processed by KnitR version 1.5 of 2013-09-28 and took 47.4 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-05-22 22:49:43.

Hands-On Data Science with R

Parallel Execution

Graham.Williams@togaware.com

9th December 2014

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

R supports several levels of parallel execution, starting with executing code on multiple cores, and going up to executing code in massively parallel Hadoop platforms. Since R Version 2.14.0 `parallel` has provided support for parallel computation through forking (c.f. `multicore`) and sockets (c.f. `snow`).

To illustrate parallel computation we will build `rpart` decision trees in parallel.

The required packages for this module include:

```
library(parallel)
library(rpart)
```

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 Weather Data

We will model the **weatherAUS** dataset. We choose this dataset since it is reasonably large, and takes quite a few seconds to build a decision tree.

We have a CSV version of the dataset available in the local data folder.

```
dir(path="data", pattern="*.csv")
## [1] "dvdtrans.csv"          "heart.csv"           "ozdata.csv"
## [4] "stroke.csv"            "WDI_Country.csv"     "WDI_CS_Notes.csv"
## [7] "WDI_csv.zip"            "WDI_Data.csv"         "WDI_Description.csv"
## [10] "WDI_Footnotes.csv"      "WDI_Series.csv"       "WDI_ST_Notes.csv"
....
```

The data is directly read into a data frame.

```
ds <- read.csv(file="data/weatherAUS.csv")
```

As always, we first check the contents of the dataset to ensure everything looks okay:

```
dim(ds)
## [1] 66672    24

head(ds)

##           Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 1 2008-12-01   Albury    13.4    22.9     0.6        NA        NA
## 2 2008-12-02   Albury     7.4    25.1     0.0        NA        NA
## 3 2008-12-03   Albury    12.9    25.7     0.0        NA        NA
## 4 2008-12-04   Albury     9.2    28.0     0.0        NA        NA
....
tail(ds)

##           Date Location MinTemp MaxTemp Rainfall Evaporation Sunshine
## 66667 2012-11-24   Darwin    25.5    34.1     0.0      5.0      6.2
## 66668 2012-11-25   Darwin    24.4    35.7     0.2      4.8     11.7
## 66669 2012-11-26   Darwin    25.0    35.4     0.0      7.4     11.7
## 66670 2012-11-27   Darwin    26.5    35.9     0.0      8.0     10.3
....
str(ds)

## 'data.frame': 66672 obs. of  24 variables:
## $ Date      : Factor w/ 1826 levels "2007-11-01","2007-11-02",...: 397 ...
## $ Location   : Factor w/ 46 levels "Adelaide","Albany",...: 3 3 3 3 3 3 ...
## $ MinTemp    : num  13.4 7.4 12.9 9.2 17.5 14.6 14.3 7.7 9.7 13.1 ...
## $ MaxTemp    : num  22.9 25.1 25.7 28 32.3 29.7 25 26.7 31.9 30.1 ...
## ...
summary(ds)

##           Date             Location          MinTemp          MaxTemp
## 2009-01-01: 46   Canberra: 1826   Min.   :-8.5   Min.   :-3.10
```

```
## 2009-01-02: 46 Sydney : 1734 1st Qu.: 7.3 1st Qu.:17.60
## 2009-01-03: 46 Adelaide: 1583 Median :11.7 Median :22.10
## 2009-01-04: 46 Brisbane: 1583 Mean   :11.9 Mean   :22.68
....
```

DRAFT

2 Prepare for Modelling

Following the template presented in the Models module, we continue with setting up some of the modelling parameters.

```

target <- "RainTomorrow"
risk   <- "RISK_MM"
dsname <- "weather"

ds[target] <- as.factor(ds[[target]])
summary(ds[target])

## RainTomorrow
## No :50187
## Yes :15259
## NA's: 1226
.....

vars    <- colnames(ds)
ignore  <- vars[c(1, 2, if (exists("risk")) which(risk==vars))]
vars   <- setdiff(vars, ignore)
(inputs <- setdiff(vars, target))

## [1] "MinTemp"      "MaxTemp"      "Rainfall"      "Evaporation"
## [5] "Sunshine"     "WindGustDir"   "WindGustSpeed" "WindDir9am"
## [9] "WindDir3pm"   "WindSpeed9am"  "WindSpeed3pm"  "Humidity9am"
## [13] "Humidity3pm"  "Pressure9am"   "Pressure3pm"   "Cloud9am"
.....
nobs    <- nrow(ds)
dim(ds[vars])

## [1] 66672    21

(form <- formula(paste(target, "~ .")))
## RainTomorrow ~ .

(seed <- sample(1:1000000, 1))
## [1] 84938
set.seed(seed)

length(train <- sample(nobs, 0.7*nobs))
## [1] 46670
length(test  <- setdiff(seq_len(nobs), train))
## [1] 20002

```

3 Build a Model

An exercise in the Functions module developed `wsrpart()` to build one or more `rpart` (Therneau and Atkinson, 2014) decision trees based on a random subset of the data and a weighted random subset of the variables. We can use this function here, and will do so to distribute computation first over multiple cores and then over multiple servers.

Each time `wsrpart()` builds a decision tree it selects a different random training dataset and a different random choice of variables to use in the tree building. The processing for each call to the function to build the decision tree (`rpart()`) will be distributed across multiple cores or servers.

Building a single model returns the model and other information.

```
set.seed(42)
system.time(model <- wsrpart(form, ds[train, vars], ntrees=1))

##    user  system elapsed
##  0.117   0.009   2.420

model[[1]]$model

## n=45816 (854 observations deleted due to missingness)
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
##       .....

model[[1]]$vars

## [1] "Humidity9am"    "Humidity3pm"    "Cloud9am"      "Pressure9am"
## [5] "Evaporation"    "MinTemp"       "WindGustSpeed" "WindGustDir"
## [9] "Temp3pm"         "Sunshine"      "RainToday"

model[[1]]$accuracy

## NULL
```

4 Build a Second Model

We can call it again to obtain another model:

```
set.seed(84)
system.time(model <- wsrpart(form, ds[train, vars], ntrees=1))

##    user  system elapsed
##  2.368   0.068   2.369

model[[1]]$model

## n=45834 (836 observations deleted due to missingness)
##
## node), split, n, loss, yval, (yprob)
##      * denotes terminal node
##      ...

model[[1]]$vars

## [1] "Cloud9am"      "Sunshine"       "MaxTemp"        "Humidity3pm"
## [5] "Rainfall"       "WindSpeed9am"  "Cloud3pm"       "RainToday"
## [9] "Humidity9am"   "WindSpeed3pm"  "Pressure9am"

model[[1]]$oob.error

## [1] 0.1757465
```

5 Build Models in Parallel

The `parallel` (?) package provides functions to distribute the computation across multiple cores and servers.

We first determine the number of cores available on the computer we are processing our data on:

```
cores <- detectCores()
cores
## [1] 8
```

We can then start a parallel run of building models using `mcpallel()`. This command forks the current process to build the tree (and hence will not work on MS/Windows). Here we build one tree for each core.

```
jobs <- lapply(1:cores,
               function(x) mcpallel(wsrpart(form, ds[train,vars], ntrees=1),
                                     name=sprintf("dt%02d", x)))
```

We can inspect the first two processes:

```
jobs[1:2]
## [[1]]
## $pid
## [1] 2879
##
## $fd
## [1] 4 7
##
## $name
## [1] "dt01"
##
## attr(,"class")
## [1] "parallelJob" "childProcess" "process"
##
## [[2]]
## $pid
## [1] 2880
##
## $fd
## [1] 5 9
##
## $name
## [1] "dt02"
##
## attr(,"class")
## [1] "parallelJob" "childProcess" "process"
```

6 Collect Results

We now wait for the jobs to finish:

```
system.time(model <- mcollect(jobs, wait=TRUE))
##    user  system elapsed
##  4.710   0.588   6.813
```

The decision trees will then be available in the resulting list:

```
length(model)
## [1] 8

model[[1]][[1]]$model
## n=45845 (825 observations deleted due to missingness)
##
## node), split, n, loss, yval, (yprob)
##      * denotes terminal node
##
## 1) root 45845 10809 No (0.7642273 0.2357727)
##    2) Humidity3pm< 71.5 37958 5782 No (0.8476737 0.1523263) *
##    3) Humidity3pm>=71.5 7887 2860 Yes (0.3626220 0.6373780)
##      6) Humidity3pm< 82.5 4314 2111 No (0.5106630 0.4893370)
##      12) Pressure9am>=1013.95 2746 1101 No (0.5990532 0.4009468) *
...
.

model[[2]][[1]]$model
## n=45826 (844 observations deleted due to missingness)
##
## node), split, n, loss, yval, (yprob)
##      * denotes terminal node
##
## 1) root 45826 10773 No (0.7649151 0.2350849)
##    2) Humidity3pm< 68.5 36052 5140 No (0.8574282 0.1425718) *
##    3) Humidity3pm>=68.5 9774 4141 Yes (0.4236751 0.5763249)
##      6) Humidity3pm< 82.5 6277 2793 No (0.5550422 0.4449578)
##      12) RainToday=No 3688 1298 No (0.6480477 0.3519523) *
...
.
```

7 Exercise: Multiple Cores

Our exercise here is to write a function to build decision trees on multiple cores. We will modify the function `wsrpart` we built in the Models module. It will use multiple cores in building the forest of trees in parallel. We specify the number of trees to build (`ntrees=`) and optionally the maximum number of cores to use `parallel=`. At most, and by default, one less than the number of available cores (but at least 1) will be used. Thus, that many trees will be built in parallel at any time. More trees than cores can be specified, and as trees finish being built across the cores, further trees can start being built. We do this to allow the user to decide how best to manage the parallel execution across the cores, without swamping the server with too many processes.

The actual solution will produce the following output, showing also some timings:

```
system.time(model <- wsrpart(form, ds, ntrees=4, parallel=2))

##    user  system elapsed
##  6.257   0.293   4.210

num.trees <- cores
set.seed(42)
system.time(model <- wsrpart(form, ds, ntrees=num.trees, parallel=2))

##    user  system elapsed
## 17.404   0.516 10.381

model[[1]]$model

## n=65476 (1196 observations deleted due to missingness)
##
## node), split, n, loss, yval, (yprob)
##       * denotes terminal node
....
```

We might do some more timings:

```
set.seed(42)
system.time(model <- wsrpart(form, ds, ntrees=num.trees, parallel=2))

##    user  system elapsed
## 20.149   0.595 12.374

set.seed(42)
system.time(model <- wsrpart(form, ds, ntrees=num.trees, parallel=2))

##    user  system elapsed
## 16.564   0.600   8.221
```

8 Parallel Processes Through Local Sockets

Before we proceed to run parallel processes over a network of workers (remote servers) we will do the same, but have a single node cluster (the current server). We use `makeCluster()` from `parallel` to do this.

We begin with a simple example. Here we create a cluster of as many nodes as there are cores on the local host.

```
cl <- makeCluster(rep("localhost", cores))
cl

## socket cluster with 8 nodes on host 'localhost'
```

Now we ask each node of the cluster to do something. In this case we get the addition function with the additional argument 3. The 1:2 are the arguments passed to each node, so that node 1 gets 1 and node 2 gets 2.

```
clusterApply(cl, 1:2, get("+"), 3)

## [[1]]
## [1] 4
##
## [[2]]
...
.
```

Because the nodes are each on the local host we don't need to export data to the nodes. Also, each node has the current working directory set appropriately. We can use `clusterEvalQ()` to have the same expression executed on each node.

```
clusterEvalQ(cl, getwd())

## [[1]]
## [1] "/home/gjw/projects/onepager"
##
## [[2]]
...
.
```

We should close the cluster once we are finished with it, though this is optional since the nodes will terminate themselves when the associated socket becomes unavailable.

```
stopCluster(cl)
```

9 Build Models Through Local Sockets

Create a new cluster of as many nodes as there are cores on the local host.

```
cl <- makeCluster(rep("localhost", cores))
cl

## socket cluster with 8 nodes on host 'localhost'
```

We load `rpart` and `rattle` on each node of the cluster.

```
clusterEvalQ(cl, {library(parallel); library(rpart); library(rattle)})

## [[1]]
## [1] "rattle"      "rpart"       "parallel"     "methods"     "stats"
## [6] "graphics"    "grDevices"   "utils"        "datasets"    "base"
##
## ...
##
```

Note that we might not have previously noticed that `library()` returns the current library search path as its value. It is returned silently and so normally we don't see the return result. With `clusterEvalQ()` the final result is displayed.

We need to load the dataset onto each node:

```
clusterExport(cl, c("ds", "form", "train", "vars"))
```

Now to build the decision trees. We need to export the definition of `wsrpart()` (and its support functions) to each node of the cluster. We can then call on the nodes to run the command:

```
clusterExport(cl, c("varWeights", "selectVars", "wsrpart"))
system.time(model <- clusterCall(cl, wsrpart, form, ds[train, vars], ntrees=4))

##    user  system elapsed
##  1.398   0.060  27.830

length(model)
## [1] 8
```

Clean up after ourselves:

```
stopCluster(cl)
```

10 Build Models Through Multiple Servers

We call `makeCluster()` to build a cluster of servers to run our decision trees on the nodes.

```
nodes <- paste("node", 1:10, sep="")
cl <- makeCluster(nodes)
cl
```

This page
under
con-
struc-
tion.

If there are errors, particularly about ssh askpass, then chances are you can not connect to the remote nodes using ssh without a password. You will need to set up a public ssh key (using `ssh-keygen -t dsa` without a pass phrase) on node1 and copy the resulting file `.ssh/id_dsa.pub` to each of the nodes as `.ssh/authorized_keys`.

Now we build just a single decision tree on each node:

```
clusterEvalQ(cl, {library(parallel); library(rpart); library(rattle)})
clusterExport(cl, c("varWeights", "selectVars", "wsrpart"))
clusterExport(cl, c("varWeights", "selectVars", "wsrpart"))
system.time(model <- clusterCall(cl, wsrpart, form, ds[train, vars], ntrees=4))

stopCluster(cl)
```

11 Build Models Through Multiple Servers and Multiple Cores

We call `makeCluster()` again to build a cluster of servers to run our decision trees on the nodes. We then ask each node to use `mcparallel()` to use all cores on each node, through `mcwsrpart()`.

```
nodes <- paste("node", 1:10, sep="")
cl <- makeCluster(nodes)
cl

clusterEvalQ(cl, {library(rpart); library(rattle)})
clusterExport(cl, "weatherDS")
clusterExport(cl, c("varWeights", "selectVars", "wsrpart", "mcwsrpart"))
system.time(forest <- clusterCall(cl, mcwsrpart, weatherDS, 8))

stopCluster(cl)
```

12 Exercise: Build Decision Trees on Multiple Servers

Extend the function `wsrpart()` to take an argument, `parallel=`, which lists a cluster of servers on which we are to build the decision trees in parallel. On each server we use as many cores as available in building trees.

This page
under con-
struction.

13 Installing Packages Across Servers

THIS PAGE UNDER CONSTRUCTION

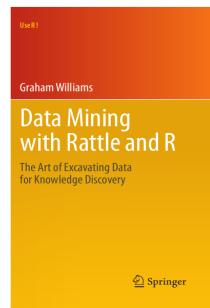
```
nodes <- paste("node", 2:10, sep="")
cl <- makeCluster(nodes)
clusterEvalQ(cl,
             install.packages("rattle",
                             lib="/usr/local/lib/R/site-library",
                             repos="http://rattle.togaware.com"))
stopCluster(cl)
```

14 Further Reading

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:



15 References

- 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/>.
- Therneau TM, Atkinson B (2014). *rpart: Recursive Partitioning and Regression Trees*. R package version 4.1-8, URL <http://CRAN.R-project.org/package=rpart>.
- 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.

DRAFT

This document, sourced from GeneticProgrammingO.Rnw revision 550, was processed by KnitR version 1.8 of 2014-11-11 and took 83.5 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 8 cores and 12.3GB of RAM. It completed the processing 2014-12-09 06:49:10.

Data Science with R Environments

Graham.Williams@togaware.com

19th July 2014

Visit <http://onepager.togaware.com/> for more OnePageR's.

The advanced topic of R environments are covered in this chapter.

The required packages for this chapter include:

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 Environment Basics

R references all objects within environments. An environment itself is an object in R. It is a special object in that it consists of a frame and a link to its enclosing environment. The frame is a collection of named objects, and it is within the frame where the object lives.

We can easily create one using `new.env()`:

```
my.env = new.env()
my.env
## <environment: 0x10c28a8>
```

The hexadecimal code listed here is the address of the environment in the computers memory. It is where the data is actually stored in memory and is used to access the data.

Environments are contained within other environments, called the enclosure, and identified as the parent environment. The R manual page suggests avoiding confusion by not calling the enclosing environment the parent environment—it is easily confused with the concept of a parent frame. We do use `parent.env()` to identify the enclosing environment though!

```
parent.env(my.env)
## <environment: R_GlobalEnv>
```

Every environment has an enclosure except for one environment, which is the empty environment `R_EmptyEnv`.

```
emptyenv()
## <environment: R_EmptyEnv>
parent.env(emptyenv())
## Error: the empty environment has no parent
```

We note that `my.env`'s parent is `R_GlobalEnv`. This is the global environment and is commonly referred to as the user's workspace.

```
.GlobalEnv
## <environment: R_GlobalEnv>
globalenv()
## <environment: R_GlobalEnv>
```

By default, the enclosure of any environment created using `new.env()` is the current environment. We can override this.

```
your.env <- new.env()
parent.env(your.env)

## <environment: R_GlobalEnv>
parent.env(your.env) <- my.env
parent.env(your.env)

## <environment: 0x10c28a8>
```

2 Enclosures

The enclosure of the global environment is often the stats package, named `package:stats`.

```
parent.env(parent.env(my.env))  
## <environment: package:stats>  
## attr(,"name")  
## [1] "package:stats"  
## attr(,"path")  
## [1] "/usr/lib/R/library/stats"
```

It is not always the case. Within the production of this document that you are currently reading, whilst the code is being run by the `knitr` package, the enclosure is the `xtable` package!

```
parent.env(parent.env(my.env))  
## <environment: package:xtable>  
## attr(,"name")  
## [1] "package:xtable"  
## attr(,"path")  
## [1] "/usr/lib/R/site-library/xtable"
```

When running R within the ESS package for Emacs we see the following:

```
parent.env(parent.env(my.env))  
## <environment: 0x383ef98>  
## attr(,"name")  
## [1] "ESSR"
```

And within RStudio we see:

```
parent.env(parent.env(my.env))  
## <environment: 0x3745410>  
## attr(,"name")  
## [1] "tools:rstudio"
```

Generally the next level of enclosure is the stats package.

```
parent.env(parent.env(globalenv()))  
## <environment: package:knitr>  
## attr(,"name")  
## [1] "package:knitr"  
## attr(,"path")  
## [1] "/usr/lib/R/site-library/knitr"
```

3 Naming Environments

From the various environments we have just seen, we might notice that environments can have a name. The name is stored as an attribute of the objects. We access the name of the environment using `attr()` and can set the name with assignment to this function:

```
attr(my.env, "name")
## NULL
attr(my.env, "name") <- "MyEnv"
attr(my.env, "name")
## [1] "MyEnv"
my.env
## <environment: 0x10c28a8>
## attr(,"name")
## [1] "MyEnv"
```

We can also retrieve the environment's name using `environmentName()`:

```
environmentName(my.env)
## [1] "MyEnv"
environmentName(globalenv())
## [1] "R_GlobalEnv"
environmentName(emptyenv())
## [1] "R_EmptyEnv"
```

4 Objects within Environments

In R we create objects through assignment. Here we create a new named object, called `m`, and store as the value of the object the numeric value 42.

```
m <- 42
```

This object is automatically placed into the current environment. By default the current environment is the global environment, which we noted above is also referred to as the user's workspace. We can check which environment is current using `environment()`:

```
environment()
## <environment: R_GlobalEnv>
```

To identify the contents of the frame of an environment we use `ls()`:

```
ls()
## [1] "hook_output" "hook_source" "m"           "Module"      "my.env"
## [6] "start.time"  "your.env"
```

By default `ls()` lists the contents of the current environment. We can specify a particular environment using the optional `name=` argument:

```
ls(name=my.env)
## character(0)

ls(my.env)
## character(0)

ls(globalenv())
## [1] "hook_output" "hook_source" "m"           "Module"      "my.env"
## [6] "start.time"  "your.env"

ls("package:stats")
##    [1] "acf"                  "acf2AR"                "add1"
##    [4] "addmargins"            "add.scope"              "aggregate"
##    [7] "aggregate.data.frame" "aggregate.ts"          "AIC"
##   [10] "alias"                 "anova"                 "ansari.test"
##    ...
## 
ls(parent.env(parent.env(environment())))
## [1] "all_labels"        "all_patterns"       "asis_output"
## [4] "current_input"     "dep_auto"           "dep_prev"
## [7] "eclipse_theme"     "fig_path"           "hook_ffmpeg_html"
## [10] "hook_movecode"     "hook_optipng"       "hook_pdfcrop"
##    ...
```

5 Adding Objects to Environments

To add an object to a specific environment we use `assign()`:

```
ls(my.env)
## character(0)

assign("n", c("a", "b"), envir=my.env)
ls(my.env)
## [1] "n"
```

Notice that the object is not in the current, global environment:

```
ls()
## [1] "hook_output" "hook_source" "m"           "Module"      "my.env"
## [6] "start.time"  "your.env"

n

## Error: object 'n' not found
```

To retrieve the value of this object from the correct environment we use `get()`:

```
get("n", envir=my.env)
## [1] "a" "b"
```

6 Environments within Functions

On invoking a function, a new environment is created, enclosed by the current environment. That environment becomes the current environment whilst the function is executing.

```
myFun <- function(x)
{
  y <- 1
  print(environment())
  print(ls())
  print(parent.env(environment()))
}

environment()
## <environment: R_GlobalEnv>

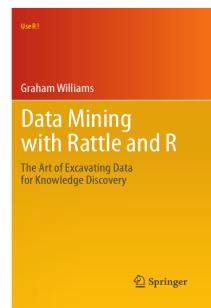
myFun()
## <environment: 0x1603c50>
## [1] "x" "y"
## <environment: R_GlobalEnv>
```

This is useful to ensure that what we do within a function stays within the function and objects created within the function are removed at the termination of the function, and do not overwrite objects of the same name outside of the function.

7 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 module is one of many OnePageR modules available from <http://onepager.togaware.com>. In particular follow the links on the website with a * which indicates the generally more developed OnePageR modules.



8 References

- 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/>.
- 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.
- 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 EnvironmentsO.Rnw revision 460, was processed by KnitR version 1.6 of 2014-05-24 and took 1.5 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-07-19 10:27:24.

Hands-On Data Science with R

Text Mining

Graham.Williams@togaware.com

10th January 2016

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

Text Mining (or Text Analytics) applies analytic tools to learn from collections of text data, like social media, books, newspapers, emails, etc. The goal can be considered to be similar to humans learning by reading such material. However, using automated algorithms we can learn from massive amounts of text, very much more than a human can. The material could consist of millions of newspaper articles to perhaps summarise the main themes and to identify those that are of most interest to particular people. Or we might be monitoring twitter feeds to identify emerging topics that we might need to act upon, as it emerges.

The required packages for this chapter include:

```
library(tm)          # Framework for text mining.  
library(qdap)        # Quantitative discourse analysis of transcripts.  
library(qdapDictionaries)  
library(dplyr)        # Data wrangling, pipe operator %>%().  
library(RColorBrewer)  # Generate palette of colours for plots.  
library(ggplot2)       # Plot word frequencies.  
library(scales)        # Include commas in numbers.  
library(Rgraphviz)     # Correlation plots.
```

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 © 2013-2015 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.



Draft Only

1 Getting Started: The Corpus

The primary package for text mining, `tm` (Feinerer and Hornik, 2015), provides a framework within which we perform our text mining. A collection of other standard R packages add value to the data processing and visualizations for text mining.

The basic concept is that of a **corpus**. This is a collection of texts, usually stored electronically, and from which we perform our analysis. A corpus might be a collection of news articles from Reuters or the published works of Shakespeare. Within each corpus we will have separate documents, which might be articles, stories, or book volumes. Each document is treated as a separate entity or record.

Documents which we wish to analyse come in many different formats. Quite a few formats are supported by `tm` (Feinerer and Hornik, 2015), the package we will illustrate text mining with in this module. The supported formats include text, PDF, Microsoft Word, and XML.

A number of open source tools are also available to convert most document formats to text files. For our corpus used initially in this module, a collection of PDF documents were converted to text using `pdftotext` from the `xpdf` application which is available for GNU/Linux and MS/Windows and others. On GNU/Linux we can convert a folder of PDF documents to text with:

```
system("for f in *.pdf; do pdftotext -enc ASCII7 -nopgbrk $f; done")
```

The `-enc ASCII7` ensures the text is converted to ASCII since otherwise we may end up with binary characters in our text documents.

We can also convert Word documents to text using `antiword`, which is another application available for GNU/Linux.

```
system("for f in *.doc; do antiword $f; done")
```

Draft Only

1.1 Corpus Sources and Readers

There are a variety of sources supported by tm. We can use `getSources()` to list them.

```
getSources()  
## [1] "DataframeSource" "DirSource"          "URISource"        "VectorSource"  
## [5] "XMLSource"       "ZipSource"
```

In addition to different kinds of sources of documents, our documents for text analysis will come in many different formats. A variety are supported by tm:

```
getReaders()  
## [1] "readDOC"           "readPDF"  
## [3] "readPlain"          "readRCV1"  
## [5] "readRCV1asPlain"    "readReut21578XML"  
## [7] "readReut21578XMLasPlain" "readTabular"  
## [9] "readTagged"          "readXML"
```

Draft Only

1.2 Text Documents

We load a sample corpus of text documents. Our corpus consists of a collection of research papers all stored in the folder we identify below. To work along with us in this module, you can create your own folder called `corpus/txt` and place into that folder a collection of text documents. It does not need to be as many as we use here but a reasonable number makes it more interesting.

```
cname <- file.path(".", "corpus", "txt")
cname
## [1] "./corpus/txt"
```

We can list some of the file names.

```
length(dir(cname))
## [1] 46
dir(cname)
## [1] "acnn96.txt"
## [2] "adm02.txt"
## [3] "ai02.txt"
## [4] "ai03.txt"
## [5] "ai97.txt"
## [6] "atobmars.txt"
....
```

There are 46 documents in this particular corpus.

After loading the `tm` (Feinerer and Hornik, 2015) package into the R library we are ready to load the files from the directory as the source of the files making up the corpus, using `DirSource()`. The source object is passed on to `Corpus()` which loads the documents. We save the resulting collection of documents in memory, stored in a variable called `docs`.

```
library(tm)
docs <- Corpus(DirSource(cname))
docs
## <<VCorpus>>
## Metadata: corpus specific: 0, document level (indexed): 0
## Content: documents: 46
class(docs)
## [1] "VCorpus" "Corpus"
class(docs[[1]])
## [1] "PlainTextDocument" "TextDocument"
summary(docs)
##                                     Length Class          Mode
## acnn96.txt                      2 PlainTextDocument list
## adm02.txt                      2 PlainTextDocument list
```

Draft Only

Data Science with R

Hands-On

Text Mining

```
## ai02.txt          2      PlainTextDocument list
## ai03.txt          2      PlainTextDocument list
## ai97.txt          2      PlainTextDocument list
....
```

Draft Only

1.3 PDF Documents

If instead of text documents we have a corpus of PDF documents then we can use the `readPDF()` reader function to convert PDF into text and have that loaded as our Corpus.

```
docs <- Corpus(DirSource(cname), readerControl=list(reader=readPDF))
```

This will use, by default, the `pdftotext` command from `xpdf` to convert the PDF into text format. The `xpdf` application needs to be installed for `readPDF()` to work.

Draft Only

1.4 Word Documents

A simple open source tool to convert Microsoft Word documents into text is `antiword`. The separate `antiword` application needs to be installed, but once it is available it is used by `tm` to convert Word documents into text for loading into R.

To load a corpus of Word documents we use the `readDOC()` reader function:

```
docs <- Corpus(DirSource(cname), readerControl=list(reader=readDOC))
```

Once we have loaded our corpus the remainder of the processing of the corpus within R is then as follows.

The `antiword` program takes some useful command line arguments. We can pass these through to the program from `readDOC()` by specifying them as the character string argument:

```
docs <- Corpus(DirSource(cname), readerControl=list(reader=readDOC("-r -s")))
```

Here, `-r` requests that removed text be included in the output, and `-s` requests that text hidden by Word be included.

Draft Only

2 Exploring the Corpus

We can (and should) inspect the documents using `inspect()`. This will assure us that data has been loaded properly and as we expect.

```
inspect(docs[16])  
## <<VCorpus>>  
## Metadata: corpus specific: 0, document level (indexed): 0  
## Content: documents: 1  
##  
## [[1]]  
## <<PlainTextDocument>>  
## Metadata: 7  
## Content: chars: 44776  
  
viewDocs <- function(d, n) {d %>% extract2(n) %>% as.character() %>% writeLines()}  
viewDocs(docs, 16)  
  
## Hybrid weighted random forests for  
## classifying very high-dimensional data  
## Baoxun Xu1 , Joshua Zhexue Huang2 , Graham Williams2 and  
## Yunming Ye1  
## 1  
##  
....
```

Draft Only

3 Preparing the Corpus

We generally need to perform some pre-processing of the text data to prepare for the text analysis. Example transformations include converting the text to lower case, removing numbers and punctuation, removing stop words, stemming and identifying synonyms. The basic transforms are all available within `tm`.

```
getTransformations()  
## [1] "removeNumbers"      "removePunctuation" "removeWords"  
## [4] "stemDocument"       "stripWhitespace"
```

The function `tm_map()` is used to apply one of these transformations across all documents within a corpus. Other transformations can be implemented using R functions and wrapped within `content_transformer()` to create a function that can be passed through to `tm_map()`. We will see an example of that in the next section.

In the following sections we will apply each of the transformations, one-by-one, to remove unwanted characters from the text.

Draft Only

3.1 Simple Transforms

We start with some manual special transforms we may want to do. For example, we might want to replace “/”, used sometimes to separate alternative words, with a space. This will avoid the two words being run into one string of characters through the transformations. We might also replace “@” and “|” with a space, for the same reason.

To create a custom transformation we make use of `content_transformer()` to create a function to achieve the transformation, and then apply it to the corpus using `tm_map()`.

```
toSpace <- content_transformer(function(x, pattern) gsub(pattern, " ", x))
docs <- tm_map(docs, toSpace, "/")
docs <- tm_map(docs, toSpace, "@")
docs <- tm_map(docs, toSpace, "\\|")
```

This can be done with a single call:

```
docs <- tm_map(docs, "/|@|\\|")
```

Check the email address in the following.

```
inspect(docs[16])
## <<VCorpus>>
## Metadata: corpus specific: 0, document level (indexed): 0
## Content: documents: 1
##
## [[1]]
## <<PlainTextDocument>>
## Metadata: 7
## Content: chars: 44776
```

Draft Only

3.2 Conversion to Lower Case

```
docs <- tm_map(docs, content_transformer(tolower))

inspect(docs[16])
## <>VCorpus>
## Metadata: corpus specific: 0, document level (indexed): 0
## Content: documents: 1
##
## [[1]]
## <>PlainTextDocument>
## Metadata: 7
## Content: chars: 44776
```

General character processing functions in R can be used to transform our corpus. A common requirement is to map the documents to lower case, using `tolower()`. As above, we need to wrap such functions with a `content_transformer()`:

Draft Only

3.3 Remove Numbers

```
docs <- tm_map(docs, removeNumbers)

viewDocs(docs, 16)

## hybrid weighted random forests for
## classifying very high-dimensional data
## baoxun xu , joshua zhixue huang , graham williams and
## yunming ye
##
##
## department of computer science, harbin institute of technology shenzhen gr...
## school, shenzhen , china
##
## shenzhen institutes of advanced technology, chinese academy of sciences, s...
## , china
## email: amusing@gmail.com
## random forests are a popular classification method based on an ensemble of a
## single type of decision trees from subspaces of data. in the literature, t...
## are many different types of decision tree algorithms, including c., cart, and
## chaid. each type of decision tree algorithm may capture different information
## and structure. this paper proposes a hybrid weighted random forest algorithm,
## simultaneously using a feature weighting method and a hybrid forest method to
## classify very high dimensional data. the hybrid weighted random forest alg...
## can effectively reduce subspace size and improve classification performance
## without increasing the error bound. we conduct a series of experiments on ...
## high dimensional datasets to compare our method with traditional random fo...
## methods and other classification methods. the results show that our method
## consistently outperforms these traditional methods.
## keywords: random forests; hybrid weighted random forest; classification; d...
##
....
```

Numbers may or may not be relevant to our analyses. This transform can remove numbers simply.

Draft Only

3.4 Remove Punctuation

```
docs <- tm_map(docs, removePunctuation)

viewDocs(docs, 16)

## hybrid weighted random forests for
## classifying very highdimensional data
## baoxun xu joshua zhexue huang graham williams and
## yunming ye
##
##
## department of computer science harbin institute of technology shenzhen gra...
## school shenzhen china
##
## shenzhen institutes of advanced technology chinese academy of sciences she...
## china
## email amusing gmailcom
## random forests are a popular classification method based on an ensemble of a
## single type of decision trees from subspaces of data in the literature there
## are many different types of decision tree algorithms including c cart and
## chaid each type of decision tree algorithm may capture different information
## and structure this paper proposes a hybrid weighted random forest algorithm
## simultaneously using a feature weighting method and a hybrid forest method to
## classify very high dimensional data the hybrid weighted random forest algo...
## can effectively reduce subspace size and improve classification performance
## without increasing the error bound we conduct a series of experiments on e...
## high dimensional datasets to compare our method with traditional random fo...
## methods and other classification methods the results show that our method
## consistently outperforms these traditional methods
## keywords random forests hybrid weighted random forest classification decis...
##
....
```

Punctuation can provide grammatical context which supports understanding. Often for initial analyses we ignore the punctuation. Later we will use punctuation to support the extraction of meaning.

Draft Only

3.5 Remove English Stop Words

```
docs <- tm_map(docs, removeWords, stopwords("english"))

inspect(docs[16])

## <<VCorpus>>
## Metadata: corpus specific: 0, document level (indexed): 0
## Content: documents: 1
##
## [[1]]
## <<PlainTextDocument>>
## Metadata: 7
## Content: chars: 32234
```

Stop words are common words found in a language. Words like *for*, *very*, *and*, *of*, *are*, etc, are common stop words. Notice they have been removed from the above text.

We can list the stop words:

```
length(stopwords("english"))

## [1] 174

stopwords("english")

## [1] "i"          "me"         "my"         "myself"      "we"
## [6] "our"        "ours"       "ourselves"   "you"        "your"
## [11] "yours"      "yourself"    "yourselves"  "he"         "him"
## [16] "his"        "himself"    "she"        "her"        "hers"
## [21] "herself"    "it"         "its"        "itself"     "they"
## [26] "them"       "their"      "theirs"      "themselves" "what"
....
```

Draft Only

3.6 Remove Own Stop Words

```
docs <- tm_map(docs, removeWords, c("department", "email"))

viewDocs(docs, 16)

## hybrid weighted random forests
## classifying highdimensional data
## baoxun xu joshua zhexue huang graham williams
## yunming ye
##
##
## computer science harbin institute technology shenzhen graduate
## school shenzhen china
##
## shenzhen institutes advanced technology chinese academy sciences shenzhen
## china
## amusing gmailcom
## random forests popular classification method based ensemble
## single type decision trees subspaces data literature
## many different types decision tree algorithms including c cart
## chaid type decision tree algorithm may capture different information
## structure paper proposes hybrid weighted random forest algorithm
## simultaneously using feature weighting method hybrid forest method
## classify high dimensional data hybrid weighted random forest algorithm
## can effectively reduce subspace size improve classification performance
## without increasing error bound conduct series experiments eight
## high dimensional datasets compare method traditional random forest
## methods classification methods results show method
## consistently outperforms traditional methods
## keywords random forests hybrid weighted random forest classification decis...
##
....
```

Previously we used the English stopwords provided by `tm`. We could instead or in addition remove our own stop words as we have done above. We have chosen here two words, simply for illustration. The choice might depend on the domain of discourse, and might not become apparent until we've done some analysis.

Draft Only

3.7 Strip Whitespace

```
docs <- tm_map(docs, stripWhitespace)

viewDocs(docs, 16)

## hybrid weighted random forests
## classifying highdimensional data
## baoxun xu joshua zhuxue huang graham williams
## yunming ye
##
##
## computer science harbin institute technology shenzhen graduate
## school shenzhen china
##
## shenzhen institutes advanced technology chinese academy sciences shenzhen
## china
## amusing gmailcom
## random forests popular classification method based ensemble
## single type decision trees subspaces data literature
## many different types decision tree algorithms including c cart
## chaid type decision tree algorithm may capture different information
## structure paper proposes hybrid weighted random forest algorithm
## simultaneously using feature weighting method hybrid forest method
## classify high dimensional data hybrid weighted random forest algorithm
## can effectively reduce subspace size improve classification performance
## without increasing error bound conduct series experiments eight
## high dimensional datasets compare method traditional random forest
## methods classification methods results show method
## consistently outperforms traditional methods
## keywords random forests hybrid weighted random forest classification decis...
##
....
```

Draft Only

3.8 Specific Transformations

We might also have some specific transformations we would like to perform. The examples here may or may not be useful, depending on how we want to analyse the documents. This is really for illustration using the part of the document we are looking at here, rather than suggesting this specific transform adds value.

```
toString <- content_transformer(function(x, from, to) gsub(from, to, x))
docs <- tm_map(docs, toString, "harbin institute technology", "HIT")
docs <- tm_map(docs, toString, "shenzhen institutes advanced technology", "SIAT")
docs <- tm_map(docs, toString, "chinese academy sciences", "CAS")

inspect(docs[16])

## <<VCorpus>>
## Metadata: corpus specific: 0, document level (indexed): 0
## Content: documents: 1
##
## [[1]]
## <<PlainTextDocument>>
## Metadata: 7
## Content: chars: 30117
```

Draft Only

3.9 Stemming

```
docs <- tm_map(docs, stemDocument)

viewDocs(docs, 16)

## hybrid weight random forest
## classifi highdimension data
## baoxun xu joshua zhexu huang graham william
## yunm ye
##
##
## comput scienc HIT shenzhen graduat
## school shenzhen china
##
## SIAT CAS shenzhen
## china
## amus gmailcom
## random forest popular classif method base ensembl
## singl type decis tree subspac data literatur
## mani differ type decis tree algorithm includ c cart
## chaid type decis tree algorithm may captur differ inform
## structur paper propos hybrid weight random forest algorithm
## simultan use featur weight method hybrid forest method
## classifi high dimension data hybrid weight random forest algorithm
## can effect reduc subspac size improv classif perform
## without increas error bound conduct seri experi eight
## high dimension dataset compar method tradit random forest
## method classif method result show method
## consist outperform tradit method
## keyword random forest hybrid weight random forest classif decis tree
##
....
```

Stemming uses an algorithm that removes common word endings for English words, such as “es”, “ed” and “s”.

Draft Only

4 Creating a Document Term Matrix

A document term matrix is simply a matrix with documents as the rows and terms as the columns and a count of the frequency of words as the cells of the matrix. We use `DocumentTermMatrix()` to create the matrix:

```
dtm <- DocumentTermMatrix(docs)

dtm

## <<DocumentTermMatrix (documents: 46, terms: 6508)>>
## Non-/sparse entries: 30061/269307
## Sparsity           : 90%
## Maximal term length: 56
## Weighting          : term frequency (tf)
```

We can inspect the document term matrix using `inspect()`. Here, to avoid too much output, we select a subset of inspect.

```
inspect(dtm[1:5, 1000:1005])

## <<DocumentTermMatrix (documents: 5, terms: 6)>>
## Non-/sparse entries: 7/23
## Sparsity           : 77%
## Maximal term length: 9
## Weighting          : term frequency (tf)
##
....
```

The document term matrix is in fact quite sparse (that is, mostly empty) and so it is actually stored in a much more compact representation internally. We can still get the row and column counts.

```
class(dtm)

## [1] "DocumentTermMatrix"    "simple_triplet_matrix"

dim(dtm)

## [1] 46 6508
```

The transpose is created using `TermDocumentMatrix()`:

```
tdm <- TermDocumentMatrix(docs)
tdm

## <<TermDocumentMatrix (terms: 6508, documents: 46)>>
## Non-/sparse entries: 30061/269307
## Sparsity           : 90%
## Maximal term length: 56
## Weighting          : term frequency (tf)
```

We will use the document term matrix for the remainder of the chapter.

Draft Only

5 Exploring the Document Term Matrix

We can obtain the term frequencies as a vector by converting the document term matrix into a matrix and summing the column counts:

```
freq <- colSums(as.matrix(dtm))
length(freq)
## [1] 6508
```

By ordering the frequencies we can list the most frequent terms and the least frequent terms:

```
ord <- order(freq)

# Least frequent terms.
freq[head(ord)]

## aaaaaaaaaaaaaeaceeeaeaaeiiaiaiaciaiicaiaeaeaoeneiacaeeeeooooo
##                                     1
##                                     aab
##                                     1
##                                     aadrbltn
##                                     1
##                                     aaddrhtmliv
##                                     1
##                                     aai
##                                     1
...
....
```

Notice these terms appear just once and are probably not really terms that are of interest to us. Indeed they are likely to be spurious terms introduced through the translation of the original document from PDF to text.

```
# Most frequent terms.
freq[tail(ord)]

##      can dataset pattern      use     mine     data
##    709     776     887    1366    1446    3101
```

These terms are much more likely to be of interest to us. Not surprising, given the choice of documents in the corpus, the most frequent terms are: data, mine, use, pattern, dataset, can.

Draft Only

6 Distribution of Term Frequencies

```
# Frequency of frequencies.  
head(table(freq), 15)  
  
## freq  
##   1    2    3    4    5    6    7    8    9    10   11   12   13   14   15  
## 2381 1030 503 311 210 188 134 130 82 83 65 61 54 52 51  
  
tail(table(freq), 15)  
  
## freq  
## 483 544 547 555 578 609 611 616 703 709 776 887 1366 1446 3101  
##   1    1    1    1    1    1    1    1    1    1    1    1    1    1    1    1
```

So we can see here that there are 2381 terms that occur just once.

Draft Only

7 Conversion to Matrix and Save to CSV

We can convert the document term matrix to a simple matrix for writing to a CSV file, for example, for loading the data into other software if we need to do so. To write to CSV we first convert the data structure into a simple matrix:

```
m <- as.matrix(dtm)
dim(m)

## [1] 46 6508
```

For very large corpus the size of the matrix can exceed R's calculation limits. This will manifest itself as a integer overflow error with a message like:

```
## Error in vector(typeof(x$v), nr * nc) : vector size cannot be NA
## In addition: Warning message:
## In nr * nc : NAs produced by integer overflow
```

If this occurs, then consider removing sparse terms from the document term matrix, as we discuss shortly.

Once converted into a standard matrix the usual `write.csv()` can be used to write the data to file.

```
write.csv(m, file="dtm.csv")
```

Draft Only

8 Removing Sparse Terms

We are often not interested in infrequent terms in our documents. Such “sparse” terms can be removed from the document term matrix quite easily using `removeSparseTerms()`:

```
dim(dtm)
## [1] 46 6508
dtms <- removeSparseTerms(dtm, 0.1)
dim(dtms)
## [1] 46 6
```

This has removed most terms!

```
inspect(dtms)
## <<DocumentTermMatrix (documents: 46, terms: 6)>>
## Non-/sparse entries: 257/19
## Sparsity           : 7%
## Maximal term length: 7
## Weighting          : term frequency (tf)
##
....
```

We can see the effect by looking at the terms we have left:

```
freq <- colSums(as.matrix(dtms))
freq
##      data   graham   inform     time      use william
##      3101      108      467      483     1366      236
table(freq)
## freq
## 108 236 467 483 1366 3101
##    1    1    1    1    1    1
```

Draft Only

9 Identifying Frequent Items and Associations

One thing we often do first is to get an idea of the most frequent terms in the corpus. We use `findFreqTerms()` to do this. Here we limit the output to those terms that occur at least 1,000 times:

```
findFreqTerms(dtm, lowfreq=1000)  
## [1] "data" "mine" "use"
```

So that only lists a few. We can get more of them by reducing the threshold:

```
findFreqTerms(dtm, lowfreq=100)  
## [1] "accuraci"      "acsi"        "adr"         "advers"       "age"  
## [6] "algorithm"     "allow"       "also"        "analysi"      "angioedema"  
## [11] "appli"         "applic"      "approach"    "area"        "associ"  
## [16] "attribut"      "australia"    "australian"   "avail"        "averag"  
## [21] "base"          "build"       "call"        "can"         "care"  
## [26] "case"          "chang"       "claim"       "class"       "classif"  
....
```

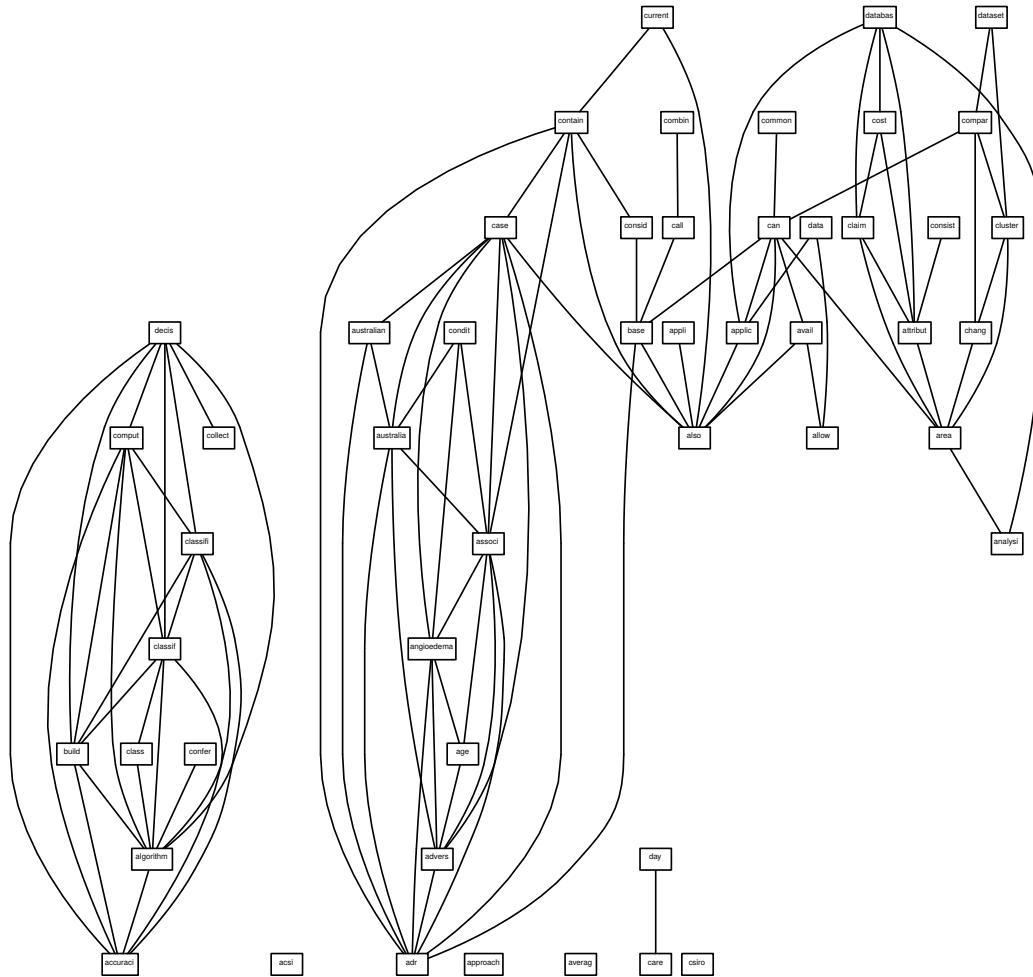
We can also find associations with a word, specifying a correlation limit.

```
findAssocs(dtm, "data", corlimit=0.6)  
## $data  
##      mine      induct      challeng      know      answer  
##      0.90      0.72      0.70      0.65      0.64  
##      need statistician      foundat      general      boost  
##      0.63      0.63      0.62      0.62      0.61  
##      major      mani      come  
....
```

If two words always appear together then the correlation would be 1.0 and if they never appear together the correlation would be 0.0. Thus the correlation is a measure of how closely associated the words are in the corpus.

Draft Only

10 Correlations Plots



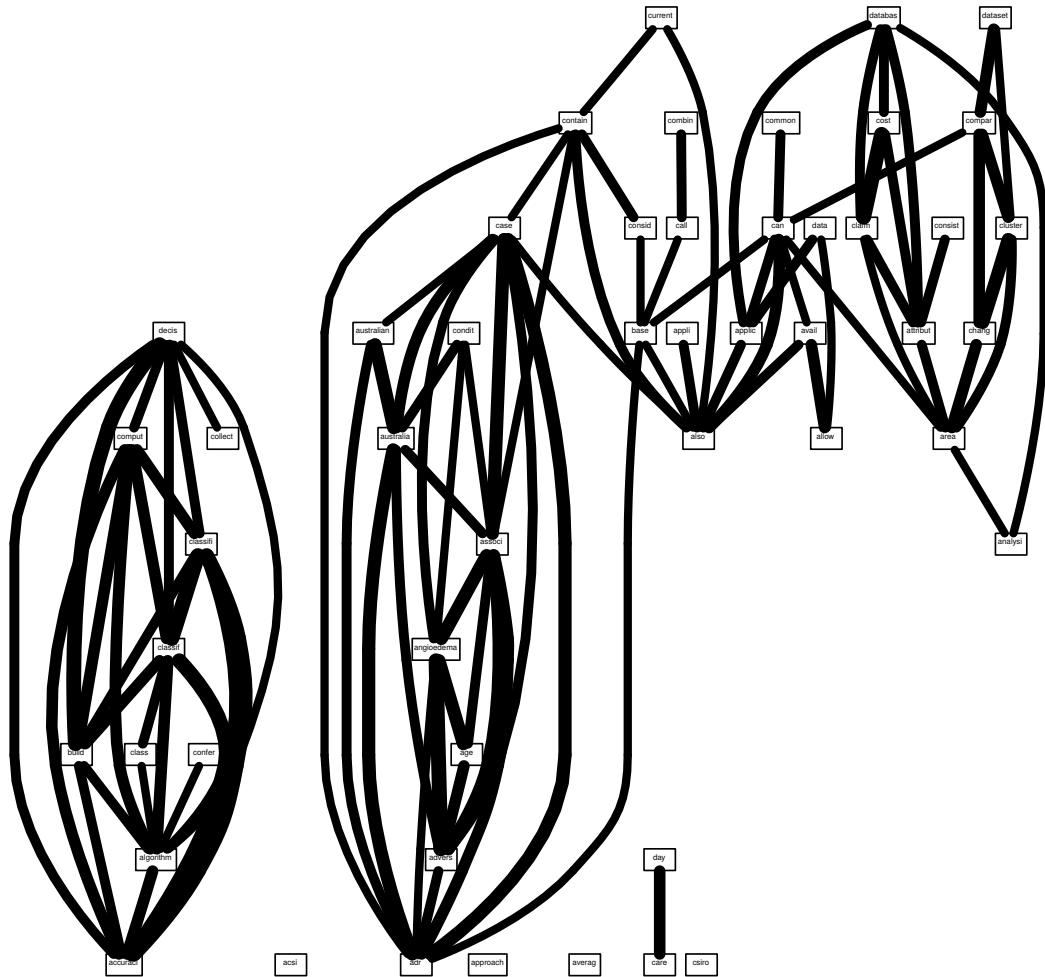
```
plot(dtm,
      terms=findFreqTerms(dtm, lowfreq=100)[1:50],
      corThreshold=0.5)
```

Rgraphviz (Hansen *et al.*, 2016) from the BioConductor repository for R (bioconductor.org) is used to plot the network graph that displays the correlation between chosen words in the corpus. Here we choose 50 of the more frequent words as the nodes and include links between words when they have at least a correlation of 0.5.

By default (without providing terms and a correlation threshold) the plot function chooses a random 20 terms with a threshold of 0.7.

Draft Only

11 Correlations Plot—Options



```
plot(dtm,
  terms=findFreqTerms(dtm, lowfreq=100) [1:50] ,
  corThreshold=0.5)
```

Draft Only

12 Plotting Word Frequencies

We can generate the frequency count of all words in a corpus:

```
freq <- sort(colSums(as.matrix(dtm)), decreasing=TRUE)
head(freq, 14)

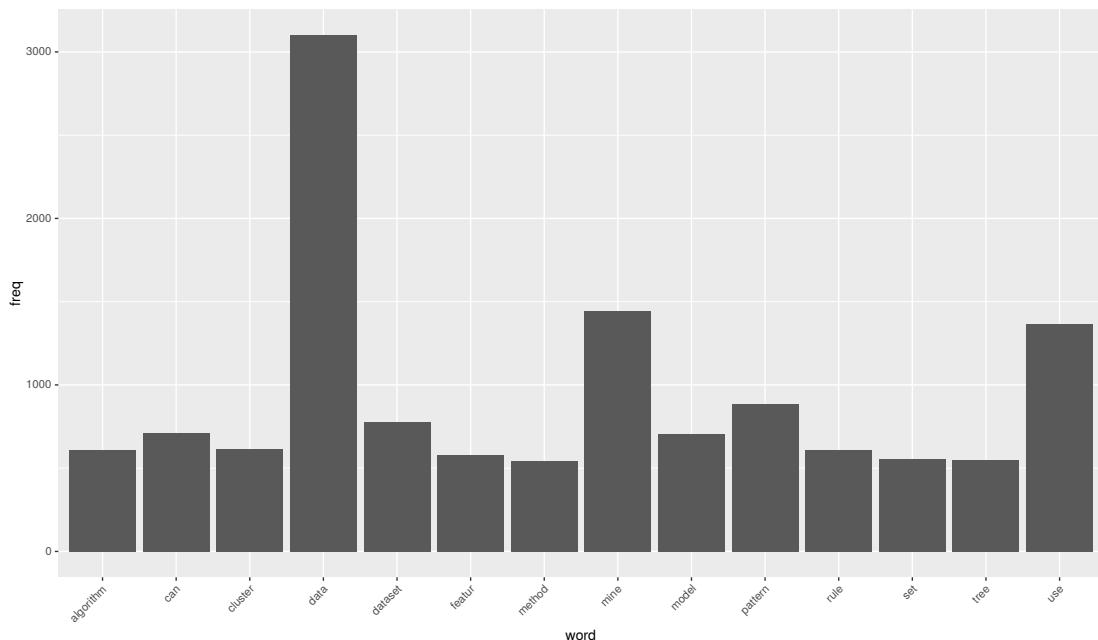
##      data      mine      use  pattern dataset      can      model
##      3101     1446    1366     887    776    709     703
## cluster algorithm      rule   featur   set      tree   method
##      616      611     609     578    555    547     544

wf   <- data.frame(word=names(freq), freq=freq)
head(wf)

##           word freq
## data       data 3101
## mine      mine 1446
## use       use 1366
## pattern  pattern 887
## dataset dataset 776
## ...
```

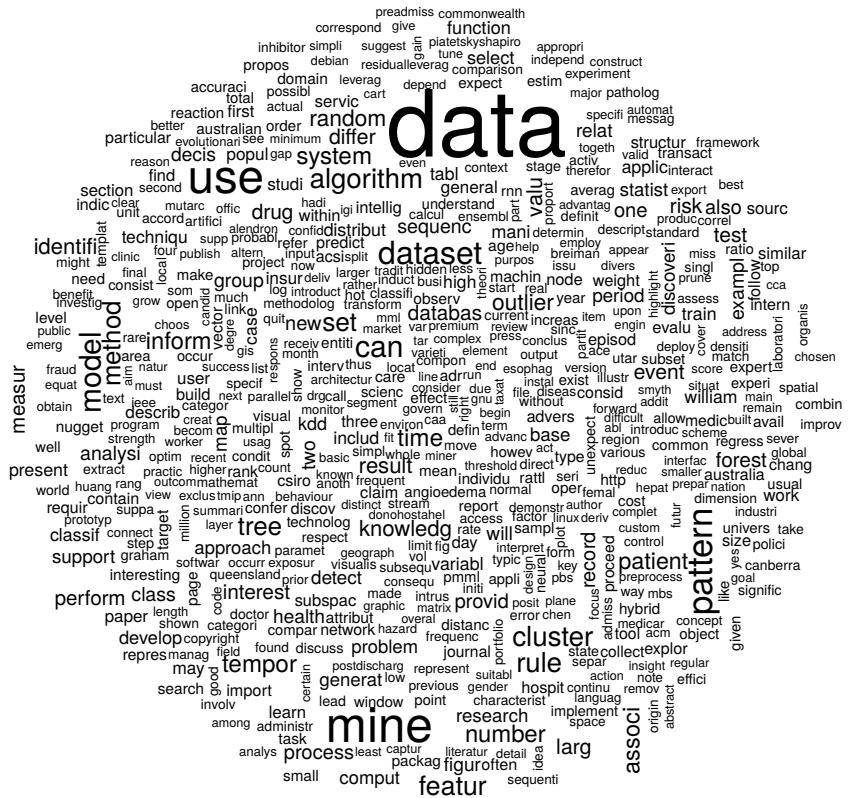
We can then plot the frequency of those words that occur at least 500 times in the corpus:

```
library(ggplot2)
subset(wf, freq>500) %>%
  ggplot(aes(word, freq)) +
  geom_bar(stat="identity") +
  theme(axis.text.x=element_text(angle=45, hjust=1))
```



Draft Only

13 Word Clouds



We can generate a word cloud as an effective alternative to providing a quick visual overview of the frequency of words in a corpus.

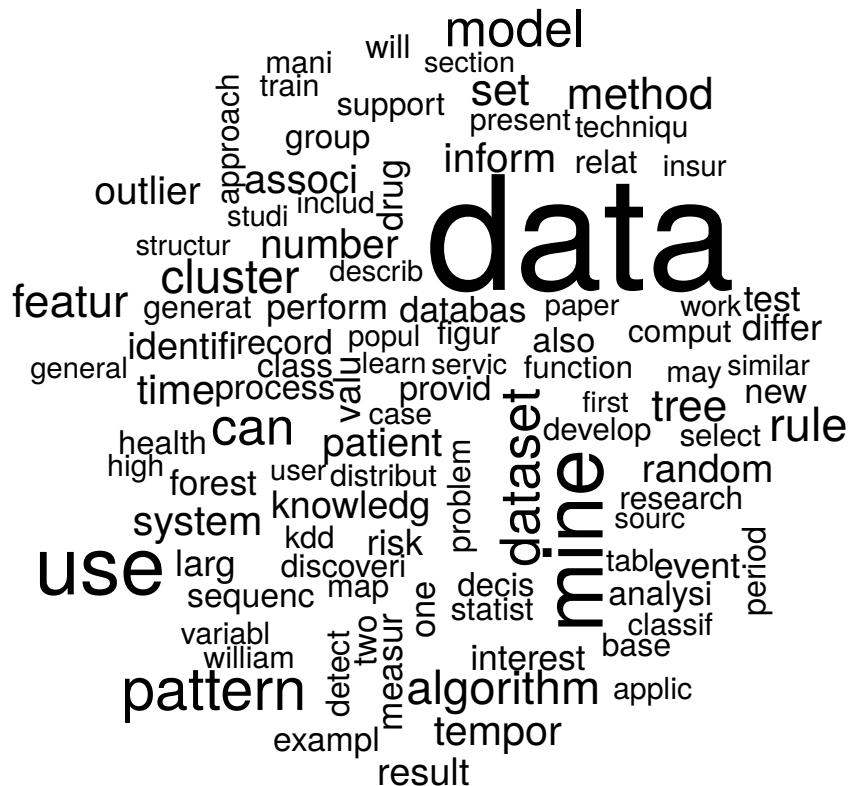
The `wordcloud` (?) package provides the required function.

```
library(wordcloud)
set.seed(123)
wordcloud(names(freq), freq, min.freq=40)
```

Notice the use of `set.seed()` only so that we can obtain the same layout each time—otherwise a random layout is chosen, which is not usually an issue.

Draft Only

13.1 Reducing Clutter With Max Words

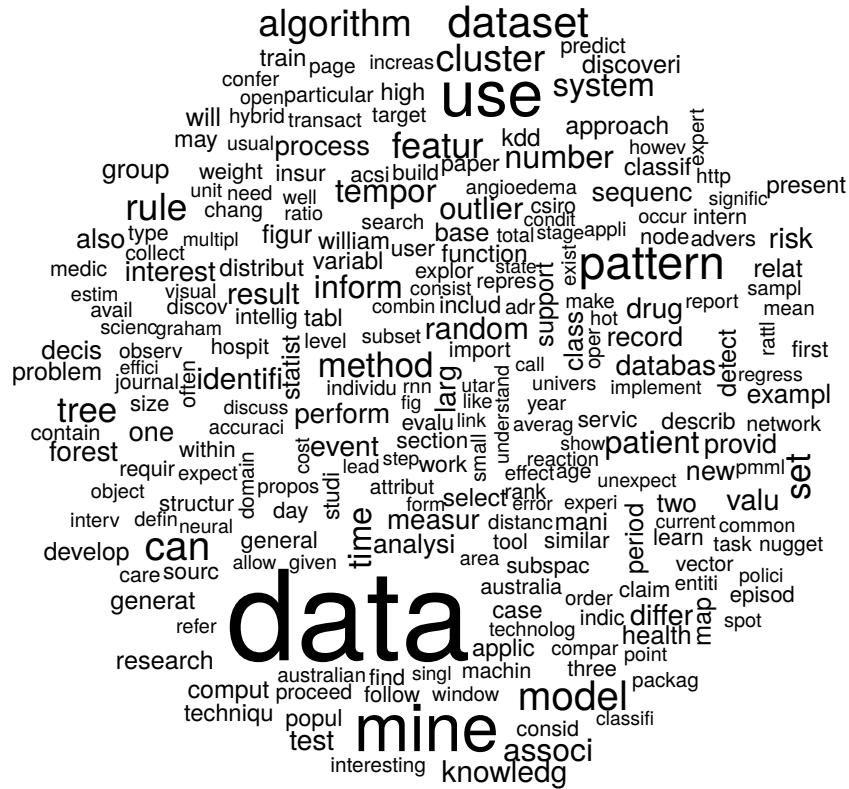


To increase or reduce the number of words displayed we can tune the value of `max.words`=. Here we have limited the display to the 100 most frequent words.

```
set.seed(142)  
wordcloud(names(freq), freq, max.words=100)
```

Draft Only

13.2 Reducing Clutter With Min Freq

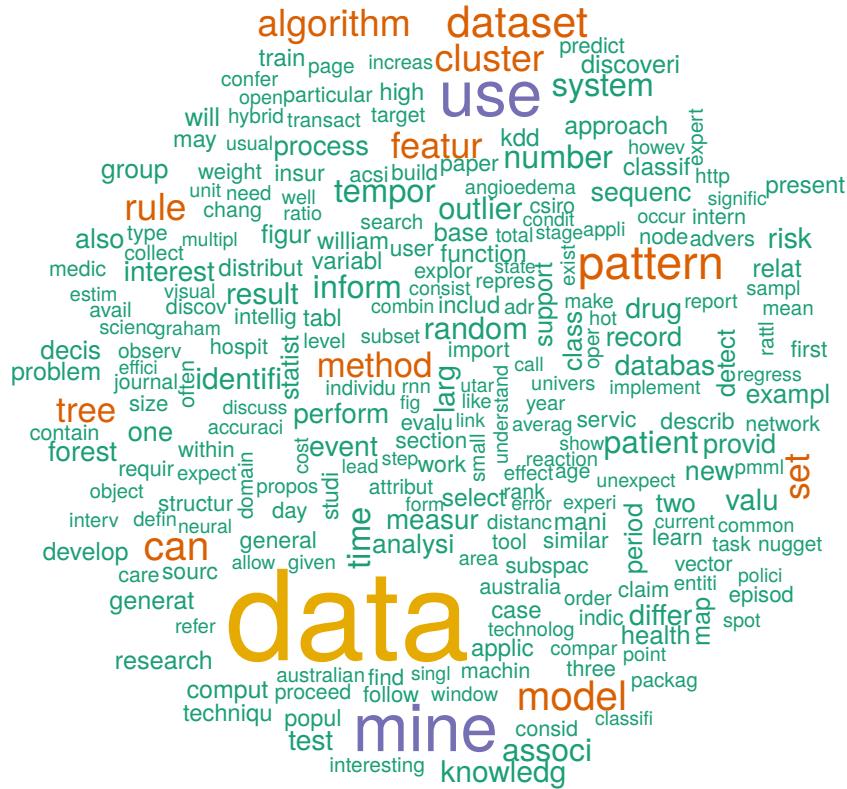


A more common approach to increase or reduce the number of words displayed is by tuning the value of `min.freq=`. Here we have limited the display to those words that occur at least 100 times.

```
set.seed(142)
wordcloud(names(freq), freq, min.freq=100)
```

Draft Only

13.3 Adding Some Colour

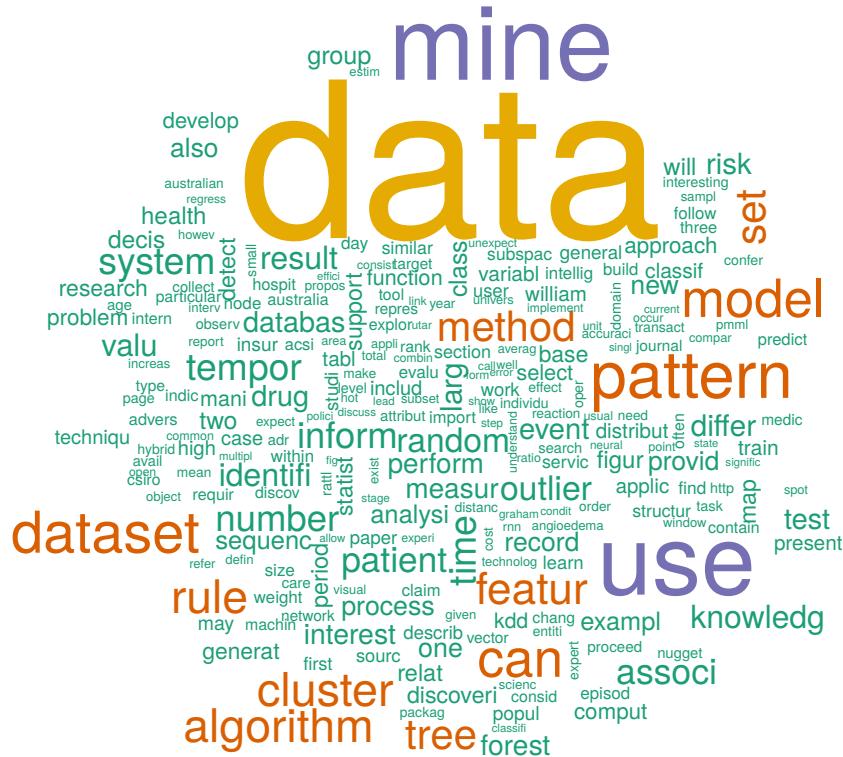


We can also add some colour to the display. Here we make use of `brewer.pal()` from RColorBrewer (Neuwirth, 2014) to generate a palette of colours to use.

```
set.seed(142)
wordcloud(names(freq), freq, min.freq=100, colors=brewer.pal(6, "Dark2"))
```

Draft Only

13.4 Varying the Scaling

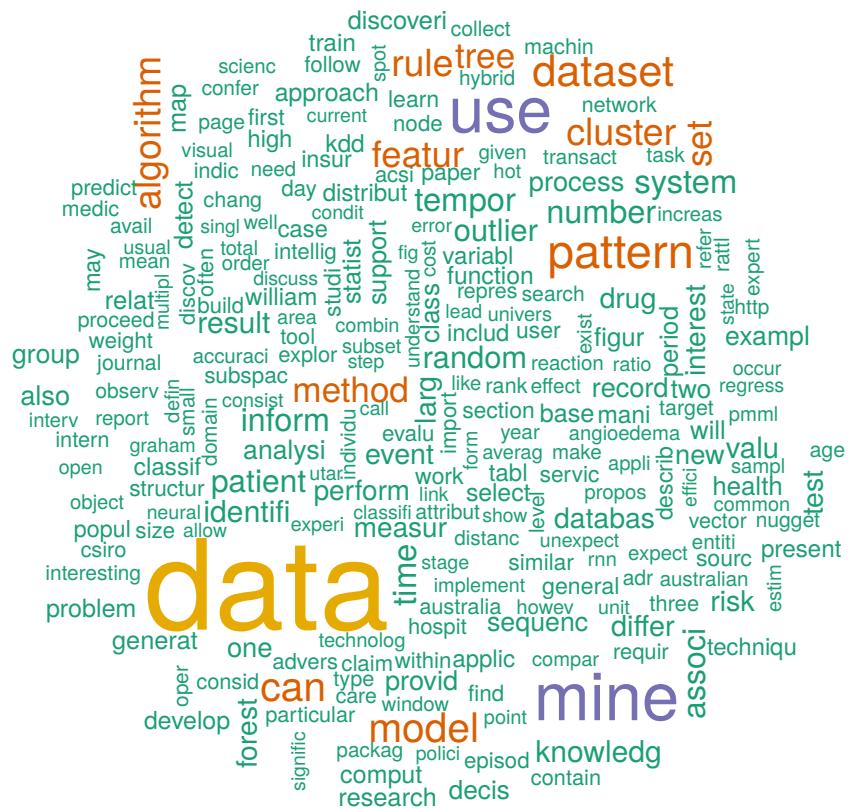


We can change the range of font sizes used in the plot using the `scale=` option. By default the most frequent words have a scale of 4 and the least have a scale of 0.5. Here we illustrate the effect of increasing the scale range.

```
set.seed(142)
wordcloud(names(freq), freq, min.freq=100, scale=c(5, .1), colors=brewer.pal(6, "Dark2"))
```

Draft Only

13.5 Rotating Words



We can change the proportion of words that are rotated by 90 degrees from the default 10% to, say, 20% using `rot.per=0.2`.

```
set.seed(142)
dark2 <- brewer.pal(6, "Dark2")
wordcloud(names(freq), freq, min.freq=100, rot.per=0.2, colors=dark2)
```

Draft Only

14 Quantitative Analysis of Text

The `qdap` (Rinker, 2015) package provides an extensive suite of functions to support the quantitative analysis of text.

We can obtain simple summaries of a list of words, and to do so we will illustrate with the terms from our Term Document Matrix `tdm`. We first extract the shorter terms from each of our documents into one long word list. To do so we convert `tdm` into a matrix, extract the column names (the terms) and retain those shorter than 20 characters.

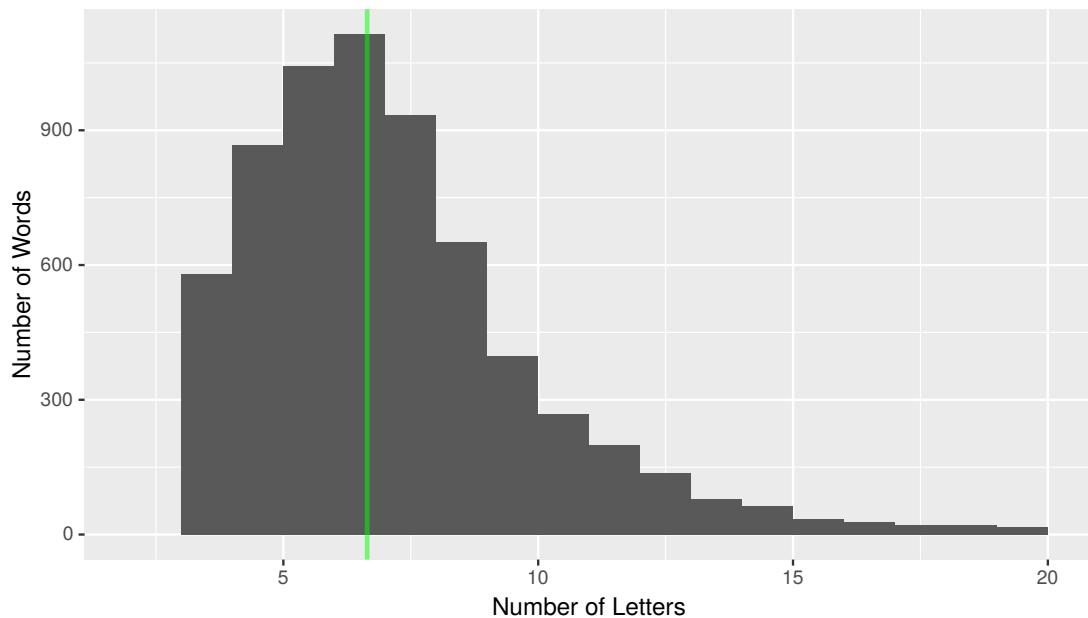
```
words <- dtm %>%  
  as.matrix %>%  
  colnames %>%  
  (function(x) x[nchar(x) < 20])
```

We can then summarise the word list. Notice, in particular, the use of `dist_tab()` from `qdap` to generate frequencies and percentages.

```
length(words)  
## [1] 6456  
  
head(words, 15)  
## [1] "aaai"      "aab"       "aad"       "aadrbltn"  "aadrbltn"  
## [6] "aadrhtmliv" "aai"       "aam"       "aba"       "abbrev"  
## [11] "abbrevi"    "abc"       "abcd"     "abdul"    "abel"  
  
summary(nchar(words))  
##   Min. 1st Qu. Median  Mean 3rd Qu.  Max.  
## 3.000 5.000 6.000 6.644 8.000 19.000  
  
table(nchar(words))  
##  
##   3    4    5    6    7    8    9    10   11   12   13   14   15   16   17  
## 579  867 1044 1114  935  651  397  268  200  138   79   63   34   28   22  
##   18   19  
##   21   16  
  
dist_tab(nchar(words))  
##   interval freq cum.freq percent cum.percent  
## 1          3 579      579    8.97      8.97  
## 2          4 867     1446   13.43     22.40  
## 3          5 1044    2490   16.17     38.57  
## 4          6 1114    3604   17.26     55.82  
## 5          7  935    4539   14.48     70.31  
## 6          8  651    5190   10.08     80.39  
## 7          9  397    5587    6.15     86.54  
## 8         10  268    5855    4.15     90.69  
## 9         11  200    6055    3.10     93.79  
## 10        12  138    6193    2.14     95.93  
....
```

Draft Only

14.1 Word Length Counts

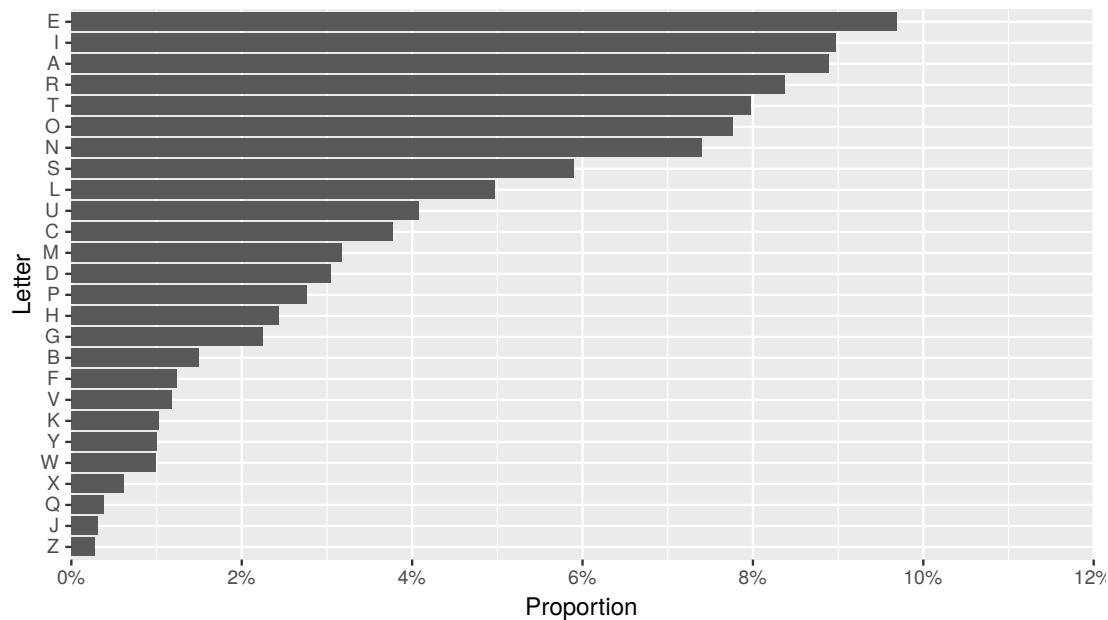


A simple plot is then effective in showing the distribution of the word lengths. Here we create a single column data frame that is passed on to `ggplot()` to generate a histogram, with a vertical line to show the mean length of words.

```
data.frame(nletters=nchar(words)) %>%
  ggplot(aes(x=nletters)) +
  geom_histogram(binwidth=1) +
  geom_vline(xintercept=mean(nchar(words)),
             colour="green", size=1, alpha=.5) +
  labs(x="Number of Letters", y="Number of Words")
```

Draft Only

14.2 Letter Frequency



Next we want to review the frequency of letters across all of the words in the discourse. Some data preparation will transform the vector of words into a list of letters, which we then construct a frequency count for, and pass this on to be plotted.

We again use a pipeline to string together the operations on the data. Starting from the vector of words stored in `word` we split the words into characters using `str_split()` from `stringr` (Wickham, 2015), removing the first string (an empty string) from each of the results (using `sapply()`). Reducing the result into a simple vector, using `unlist()`, we then generate a data frame recording the letter frequencies, using `dist_tab()` from `qdap`. We can then plot the letter proportions.

```
library(dplyr)
library(stringr)

words %>%
  str_split("") %>%
  sapply(function(x) x[-1]) %>%
  unlist %>%
  dist_tab %>%
  mutate(Letter=factor(toupper(interval),
                       levels=toupper(interval[order(freq)]))) %>%
  ggplot(aes(Letter, weight=percent)) +
  geom_bar() +
  coord_flip() +
  labs(y="Proportion") +
  scale_y_continuous(breaks=seq(0, 12, 2),
                     label=function(x) paste0(x, "%"))
```

Draft Only

Data Science with R

Hands-On

Text Mining

```
expand=c(0,0), limits=c(0,12))
```

Draft Only

14.3 Letter and Position Heatmap



The `qheat()` function from `qdap` provides an effective visualisation of tabular data. Here we transform the list of words into a position count of each letter, and constructing a table of the proportions that is passed on to `qheat()` to do the plotting.

```
words %>%
  lapply(function(x) sapply(letters, gregexpr, x, fixed=TRUE)) %>%
  unlist %>%
  (function(x) x[x!= -1]) %>%
  (function(x) setNames(x, gsub("\\d", "", names(x)))) %>%
  (function(x) apply(table(data.frame(letter=toupper(names(x))),
                           position=unname(x))), %>%
    1, function(y) y/length(x))) %>%
  qheat(high="green", low="yellow", by.column=NULL,
        values=TRUE, digits=3, plot=FALSE) +
  labs(y="Letter", x="Position") +
  theme(axis.text.x=element_text(angle=0)) +
  guides(fill=guide_legend(title="Proportion")) +
```

Draft Only

14.4 Miscellaneous Functions

We can generate gender from a name list, using the `genderdata (?)` package

```
devtools::install_github("lmullen/gender-data-pkg")  
  
name2sex(qcv(graham, frank, leslie, james, jacqui, jack, kerry, kerrie))  
## The genderdata package needs to be installed.  
## Error in install.genderdata.package(): Failed to install the genderdata package.  
## Please try installing the package for yourself using the following command:  
##     install.packages("genderdata", repos = "http://packages.ropensci.org", type  
= "source")
```

Draft Only

15 Word Distances

Continuous bag of words (CBOW). Word2Vec associates each word in a vocabulary with a unique vector of real numbers of length d. Words that have a similar syntactic context appear closer together within the vector space. The syntactic context is based on a set of words within a specific window size.

```
install.packages("tmcn.word2vec", repos="http://R-Forge.R-project.org")
## Installing package into '/home/gjw/R/x86_64-pc-linux-gnu-library/3.2'
## (as 'lib' is unspecified)

##
## The downloaded source packages are in
## '/tmp/Rtmpt1u3GR/downloaded_packages'

library(tmcn.word2vec)
model <- word2vec(system.file("examples", "rfaq.txt", package = "tmcn.word2vec"))

## The model was generated in '/home/gjw/R/x86_64-pc-linux-gnu-library/3.2/tm...
distance(model$model_file, "the")

##      Word    CosDist
## 1      a 0.8694174
## 2      is 0.8063422
## 3      and 0.7908007
## 4      an 0.7738196
## 5  please 0.7595193
....
```

Draft Only

16 Review—Preparing the Corpus

Here in one sequence is collected the code to perform a text mining project. Notice that we would not necessarily do all of these steps so pick and choose as is appropriate to your situation.

```
# Required packages

library(tm)
library(wordcloud)

# Locate and load the Corpus.

cname <- file.path(".", "corpus", "txt")
docs <- Corpus(DirSource(cname))

docs
summary(docs)
inspect(docs[1])

# Transforms

toSpace <- content_transformer(function(x, pattern) gsub(pattern, " ", x))
docs <- tm_map(docs, "/|@|\\")

docs <- tm_map(docs, content_transformer(tolower))
docs <- tm_map(docs, removeNumbers)
docs <- tm_map(docs, removePunctuation)
docs <- tm_map(docs, removeWords, stopwords("english"))
docs <- tm_map(docs, removeWords, c("own", "stop", "words"))
docs <- tm_map(docs, stripWhitespace)

toString <- content_transformer(function(x, from, to) gsub(from, to, x))
docs <- tm_map(docs, toString, "specific transform", "ST")
docs <- tm_map(docs, toString, "other specific transform", "OST")

docs <- tm_map(docs, stemDocument)
```

Draft Only

17 Review—Analysing the Corpus

```
# Document term matrix.

dtm <- DocumentTermMatrix(docs)
inspect(dtm[1:5, 1000:1005])

# Explore the corpus.

findFreqTerms(dtm, lowfreq=100)
findAssocs(dtm, "data", corlimit=0.6)

freq <- sort(colSums(as.matrix(dtm)), decreasing=TRUE)
wf   <- data.frame(word=names(freq), freq=freq)

library(ggplot2)

p <- ggplot(subset(wf, freq>500), aes(word, freq))
p <- p + geom_bar(stat="identity")
p <- p + theme(axis.text.x=element_text(angle=45, hjust=1))

# Generate a word cloud

library(wordcloud)
wordcloud(names(freq), freq, min.freq=100, colors=brewer.pal(6, "Dark2"))
```

Draft Only

18 LDA

Topic Models such as Latent Dirichlet Allocation has been popular for text mining in last 15 years. Applied with varying degrees of success. Text is fed into LDA to extract the topics underlying the text document. Examples are the AP corpus and the Science Corpus 1880-2002 (Blei and Lafferty 2009). PERHAPS USEFUL IN BOOK?

When is LDA applicable - it will fail on some data and need to choose number of topics to find and how many documents are needed. How do we know the topics learned are correct topics.

Two fundamental papers - independently discovered: Blei, Ng, Jordan - NIPS 2001 with 11k citations. Other paper is Pritchard, Stephens, and Donnelly in Genetics June 2001 14K citations - models are exactly the same except for minor differences: except topics versus population structures.

No theoretic analysis as such. How to guarantee correct topics and how efficient is the learning procedure?

Observations:

LDA won't work on many short tweets or very few long documents.

We should not liberally over-fit the LDA with too many redundant topics...

Limiting factors:

We should use as many documents as we can and short documents less than 10 words won't work even if there are many of them. Need sufficiently long documents.

Small Dirichlet parameter helps especially if we overfit. See Long Nguen's keynote at PAKDD 2015 in Vietnam.

number of documents the most important factor

document length plays a useful role too

avoid overfitting as you get too many topics and don't really learn anything as the human needs to cull the topics.

New work detects new topics as they emerge.

```
library(lda)
## Error in library(lda): there is no package called 'lda'

# From demo(lda)
library("ggplot2")
library("reshape2")
data(cora.documents)

## Warning in data(cora.documents): data set 'cora.documents' not found
data(cora.vocab)

## Warning in data(cora.vocab): data set 'cora.vocab' not found
```

Draft Only

```
theme_set(theme_bw())
set.seed(8675309)
K <- 10 ## Num clusters
result <- lda.collapsed.gibbs.sampler(cora.documents,
                                         K,    ## Num clusters
                                         cora.vocab,
                                         25,   ## Num iterations
                                         0.1,
                                         0.1,
                                         compute.log.likelihood=TRUE)

## Error in eval(expr, envir, enclos): could not find function "lda.collapsed.gibbs.sampler"

## Get the top words in the cluster
top.words <- top.topic.words(result$topics, 5, by.score=TRUE)

## Error in eval(expr, envir, enclos): could not find function "top.topic.words"

## Number of documents to display
N <- 10

topic.proportions <- t(result$document_sums) / colSums(result$document_sums)

## Error in t(result$document_sums): object 'result' not found

topic.proportions <-
  topic.proportions[sample(1:dim(topic.proportions)[1], N),]

## Error in eval(expr, envir, enclos): object 'topic.proportions' not found

topic.proportions[is.na(topic.proportions)] <- 1 / K

## Error in topic.proportions[is.na(topic.proportions)] <- 1/K: object 'topic.proportions' not found

colnames(topic.proportions) <- apply(top.words, 2, paste, collapse=" ")
## Error in apply(top.words, 2, paste, collapse = " "): object 'top.words' not found

topic.proportions.df <- melt(cbind(data.frame(topic.proportions),
                                      document=factor(1:N)),
                               variable.name="topic",
                               id.vars = "document")

## Error in data.frame(topic.proportions): object 'topic.proportions' not found

ggplot(topic.proportions.df, aes(x=topic, y=value, fill=topic)) +
  geom_bar(stat="identity") +
  theme(axis.text.x = element_text(angle=45, hjust=1, size=7),
        legend.position="none") +
  coord_flip() +
  facet_wrap(~ document, ncol=5)

## Error in ggplot(topic.proportions.df, aes(x = topic, y = value, fill = topic)): object 'topic.proportions.df' not found
```

Draft Only

Data Science with R

Hands-On

Text Mining

19 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](#).

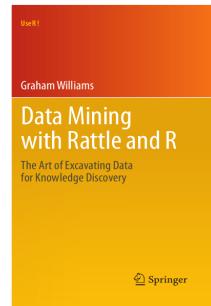
This chapter is one of many chapters available from <http://HandsOnDataScience.com>. In particular follow the links on the website with a * which indicates the generally more developed chapters.

Other resources include:

- The Journal of Statistical Software article, *Text Mining Infrastructure in R* is a good start <http://www.jstatsoft.org/v25/i05/paper>
- [Bilisoly \(2008\)](#) presents methods and algorithms for text mining using Perl.

Thanks also to Tony Nolan for suggestions of some of the examples used in this chapter.

Some of the `qdap` examples were motivated by <http://trinkerrstuff.wordpress.com/2014/10/31/exploration-of-letter-make-up-of-english-words/>.



Draft Only

20 References

- Bilisoly R (2008). *Practical Text Mining with Perl*. Wiley Series on Methods and Applications in Data Mining. Wiley. ISBN 9780470382851. URL <http://books.google.com.au/books?id=YkMFVbsrdzkC>.
- Feinerer I, Hornik K (2015). *tm: Text Mining Package*. R package version 0.6-2, URL <https://CRAN.R-project.org/package=tm>.
- Hansen KD, Gentry J, Long L, Gentleman R, Falcon S, Hahne F, Sarkar D (2016). *Rgraphviz: Provides plotting capabilities for R graph objects*. R package version 2.12.0.
- Neuwirth E (2014). *RColorBrewer: ColorBrewer Palettes*. R package version 1.1-2, URL <https://CRAN.R-project.org/package=RColorBrewer>.
- R Core Team (2015). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- Rinker T (2015). *qdap: Bridging the Gap Between Qualitative Data and Quantitative Analysis*. R package version 2.2.4, URL <https://CRAN.R-project.org/package=qdap>.
- Wickham H (2015). *stringr: Simple, Consistent Wrappers for Common String Operations*. R package version 1.0.0, URL <https://CRAN.R-project.org/package=stringr>.
- 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.

This document, sourced from TextMiningO.Rnw bitbucket revision 76, was processed by KnitR version 1.12 of 2016-01-06 and took 41.3 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04.3 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 8 cores and 12.3GB of RAM. It completed the processing 2016-01-10 09:58:57.

Draft Only

Generated 2016-01-10 10:00:58+11:00

Social Network Analysis

Chapter pending / not available

Genetic Programming

Chapter pending / not available

Time Series Analysis

Chapter pending / not available

One Page R Data Science

Coding with Style

Graham.Williams@togaware.com

3rd June 2018

Visit <https://essentials.togaware.com/onepagers> for more Essentials.

Data scientists write programs to ingest, manage, wrangle, visualise, analyse and model data in many ways. It is an art to be able to communicate our explorations and understandings through a language, albeit a programming language. Of course our programs must be executable by computers but computers care little about our programs except that they be syntactically correct. Our focus should be on engaging others to read and understand the narratives we present through our programs.

In this chapter we present simple stylistic guidelines for programming in R that support the transparency of our programs. We should aim to write programs that clearly and effectively communicate the story of our data to others. Our programming style aims to ensure consistency and ease our understanding whilst of course also encouraging correct programs for execution by computer.

Through this guide new R commands will be introduced. The reader is encouraged to review the command's documentation and understand what the command does. Help is obtained using the `? command` as in:

```
?read.csv
```

Documentation on a particular package can be obtained using the `help=` option of `library()`:

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

This chapter is intended to be hands on. To learn effectively the reader is encouraged to run R locally (e.g., RStudio or Emacs with ESS mode) and to replicate all commands as they appear here. Check that output is the same and it is clear how it is generated. Try some variations. Explore.

Copyright © 2000-2018 Graham Williams. This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](#) allowing this work to be copied, distributed, or adapted, with attribution and provided under the same license.



1 Why We Should Care

Programming is an art and a way to express ourselves. Often that expression is unique to us individually. Just as we can often ascertain who the author is of a play or the artist of a painting from their style we can often tell the programmer from the program coding structures and styles.

As we write programs we should keep in mind that something like 90% of a programmers' time (at least in business and government) is spent reading and modifying and extending other programmers' code. We need to facilitate the task—so that others can quickly come to a clear understanding of the narrative.

As data scientists we also practice this art of programming and indeed even more so to share the narrative of what we discover through our living and breathing of data. Writing our programs so that others understand why and how we analysed our data is crucial. Data science is so much more than simply building black box models—we should be seeking to expose and share the process and the knowledge that is discovered from the data.

Data scientists rarely begin a new project with an empty coding sheet. Regularly we take our own or other's code as a starting point and begin from that. We find code on Stack Overflow or elsewhere on the Internet and modify it to suit our needs. We collect templates from other data scientists and build from there, tuning the templates for our specific needs and datasets.

In being comfortable to share our code and narratives with others we often develop a style. Our style is personal to us as we innovate and express ourselves and we need consistency in how we do that. Often a style guide helps us as we journey through a new language and gives us a foundation for developing, over time, our own style.

A style guide is useful for sharing our tips and tricks for communicating clearly through our programs—our expression of how to solve a problem or actually how we model the world. We express this in the form of a language—a language that also happens to be executable by a computer. In this language we follow precisely specified syntax/grammar to develop sentences, paragraphs, and whole stories. Whilst there is infinite leeway in how we express ourselves and we each express ourselves differently, we share a common set of principles as our style guide.

The style guide here has evolved from over 30 years of programming and data experience. Nonetheless we note that style changes over time. Change can be motivated by changes in the technology itself and we should allow variation as we mature and learn and change our views.

Irrespective of whether the specific style suggestions here suit you or not, when coding do aim to communicate to other readers in the first instance. When we write programs we *write for others to easily read and to learn from and to build upon*.

2 Naming Files

1. Files containing R code use the uppercase .R extension. This aligns with the fact that the language is unambiguously called “R” and not “r.”

Preferred

```
power_analysis.R
```

Discouraged

```
power_analysis.r
```

2. Some files may contain support functions that we have written to help us repeat tasks more easily. Name the file to match the name of the function defined within the file. For example, if the support function we’ve defined in the file is `myFancyPlot()` then name the file as below. This clearly differentiates support function filenames from analysis scripts and we have a ready record of the support functions we might have developed simply by listing the folder contents.

Preferred

```
myFancyPlot.R
```

Discouraged

```
utility_functions.R  
MyFancyPlot.R  
my_fancy_plot.R  
my.fancy.plot.R  
my_fancy_plot.r
```

3. R binary data filenames end in “.RData”. This is descriptive of the file containing data for R and conforms to a capitalised naming scheme.

Preferred

```
weather.RData
```

Discouraged

```
weather.rdata  
weather.Rdata  
weather.rData
```

4. Standard file names use lowercase where there is a choice.

Preferred

```
weather.csv
```

Discouraged

```
weather.CSV
```

3 Multiple File Scripts

5. For multiple scripts associated with a project that have a processing order associated with them use a simple two digit number prefix scheme. Separating by 10's allows additional script files to be added into the sequence later.

Sometimes this can become a burden. Users find themselves reverting to a single script file for their code. It requires some judgement and discipline to modularise your code in this way, and maybe some assistance too from the integrated development environment being used.

Suggested

```
00_setup.R  
10_ingest.R  
20_observe.R  
30_process.R  
40_meta.R  
50_save.R  
60_classification.R  
62_rpart.R  
64_randomForest.R  
66_xgboost.R  
68_h2o.R  
70_regression.R  
72_lm.R  
74_rpart.R  
76_mxnet.R  
80_evaluate.R  
90_deploy.R  
99_all.R
```

4 Naming Objects

6. **Function names** begin lowercase with capitalised *verbs*. A common alternative is to use underscore to separate words but we use this specifically for variables.

Preferred

```
displayPlotAgain()
```

Discouraged

```
DisplayPlotAgain()  
displayplotagain()  
display.plot.again()  
display_plot_again()
```

7. **Variable names** use underscore separated *nouns*. A very common alternative is to use a period in place of the underscore. However, the period is often used to identify class hierarchies in R and the period has specific meanings in many database systems which presents an issue when importing from and exporting to databases.

Preferred

```
num_frames <- 10
```

Discouraged

```
num.frames <- 10  
numframes <- 10  
numFrames <- 10
```

5 Functions

8. **Function argument names** use period separated *nouns*. Function argument names do not risk being confused with class hierarchies and the style is useful in differentiating the argument name from the argument value. Within the body of the function it is also useful to be reminded of which variables are function arguments and which are local variables.

Preferred

```
buildCyc(num.frames=10)
buildCyc(num.frames=num_frames)
```

Discouraged

```
buildCyc(num_frames=10)
buildCyc(numframes=10)
buildCyc(numFrames=10)
```

9. **Keep variable and function names shorter** but self explanatory. A long variable or function name is problematic with layout and similar names are hard to tell apart. Single letter names like `x` and `y` are often used within functions and facilitate understanding, particularly for mathematically oriented functions but should otherwise be avoided.

Preferred

```
# Perform addition.

addSquares <- function(x, y)
{
  return(x^2 + y^2)
}
```

Discouraged

```
# Perform addition.

addSquares <- function(first_argument, second_argument)
{
  return(first_argument^2 + second_argument^2)
}
```

6 Comments

10. Use a single `#` to introduce ordinary comments and separate comments from code with a single empty line before and after the comment. Comments should be full sentences beginning with a capital and ending with a full stop.

Preferred

```
# How many locations are represented in the dataset.

ds$location %>%
  unique() %>%
  length()

# Identify variables that have a single value.

ds[vars] %>%
  sapply(function(x) all(x == x[1L])) %>%
  which() %>%
  names() %T>%
  print() ->
constants
```

11. Sections might begin with all uppercase titles and subsections with initial capital titles. The last four dashes at the end of the comment are a section marker supported by RStudio. Other conventions are available for structuring a document and different environments support different conventions.

Preferred

```
# DATA WRANGLING ----

# Normalise Variable Names ----

# Review the names of the dataset columns.

names(ds)

# Normalise variable names and confirm they are as expected.

names(ds) %<>% rattle::normVarNames() %T>% print()

# Specifically Wrangle weatherAUS ----

# Convert the character variable 'date' to a Date data type.

class(ds$date)
ds$date %<>%
  lubridate::ymd() %>%
  as.Date() %T>%
  {class(.); print()}
```

7 Layout

12. Keep lines to less than 80 characters for easier reading and fitting on a printed page.
13. Align curly braces so that an opening curly brace is on a line by itself. This is at odds with many style guides. My motivation is that the open and close curly braces belong to each other more so than the closing curly brace belonging to the keyword (`while` in the example). The extra white space helps to reduce code clutter. This style also makes it easier to comment out, for example, just the line containing the `while` and still have valid syntax. We tend not to need to focus so much any more on reducing the number of lines in our code so we can now avoid [Egyptian brackets](#).

Preferred

```
while (blueSky())
{
  openTheWindows()
  doSomeResearch()
}
retireForTheDay()
```

Alternative

```
while (blueSky()) {
  openTheWindows()
  doSomeResearch()
}
retireForTheDay()
```

14. If a code block contains a single statement, then curly braces remain useful to emphasise the limit of the code block; however, some prefer to drop them.

Preferred

```
while (blueSky())
{
  doSomeResearch()
}
retireForTheDay()
```

Alternatives

```
while (blueSky())
  doSomeResearch()
retireForTheDay()
```

```
while (blueSky()) doSomeResearch()
retireForTheDay()
```

8 If-Else Issue

15. R is an interpretive language and encourages interactive development of code within the R console. Consider typing the following code into the R console.

```
if (TRUE)
{
  seed <- 42
}
else
{
  seed <- 666
}
```

After the first closing brace the interpreter identifies and executes a syntactically valid statement (`if` with no `else`). The following `else` is then a syntactic error.

```
Error: unexpected 'else' in "else"

> source("examples.R")
Error in source("examples.R") : tmp.R:5:1: unexpected 'else'
4: }
5: else
^
```

This is not an issue when embedding the `if` statement inside a block of code as within curly braces since the text we enter is not parsed until we hit the final closing brace.

```
{ 
  if (TRUE)
  {
    seed <- 42
  }
  else
  {
    seed <- 666
  }
}
```

Another solution is to move the `else` to the line with the closing braces to inform the interpreter that we have more to come:

```
if (TRUE)
{
  seed <- 42
} else
{
  seed <- 666
}
```

9 Indentation

16. Use a consistent indentation. I personally prefer 2 spaces within both Emacs ESS and RStudio with a good font (e.g., Courier font in RStudio works well but Courier 10pt is too compressed). Some argue that 2 spaces is not enough to show the structure when using smaller fonts. If it is an issue, then try 4 or choose a different font. We still often have limited lengths on lines on some forms of displays where we might want to share our code and about 80 characters seems about right. Indenting 8 characters is probably too much because it makes it difficult to read through the code with such large leaps for our eyes to follow to the right. Nonetheless, there are plenty of tools to reindent to a different level as we choose.

Preferred

```
window_delete <- function(action, window)
{
  if (action %in% c("quit", "ask"))
  {
    ans <- TRUE
    msg <- "Terminate?"
    if (! dialog(msg))
      ans <- TRUE
    else
      if (action == "quit")
        quit(save="no")
    else
      ans <- FALSE
  }
  return(ans)
}
```

Not Ideal

```
window_delete <- function(action, window)
{
  if (action %in% c("quit", "ask"))
  {
    ans <- TRUE
    msg <- "Terminate?"
    if (! dialog(msg))
      ans <- TRUE
    else
      if (action == "quit")
        quit(save="no")
    else
      ans <- FALSE
  }
  return(ans)
}
```

17. Always use spaces rather than the invisible tab character.

10 Alignment

18. Align the assignment operator for blocks of assignments. The rationale for this style suggestion is that it is easier for us to read the assignments in a tabular form than it is when it is jagged. This is akin to reading data in tables—such data is much easier to read when it is aligned. Space is used to enhance readability.

Preferred

```
a      <- 42
another <- 666
b      <- mean(x)
brother <- sum(x)/length(x)
```

Default

```
a <- 42
another <- 666
b <- mean(x)
brother <- sum(x)/length(x)
```

19. In the same vein we might think to align the `stringr::%>%` operator in pipelines and the `base:::+` operator for `ggplot2` (Wickham and Chang, 2016) layers. This provides a visual symmetry and avoids the operators being lost amongst the text. Such alignment though requires extra work and is not supported by editors. Also, there is a risk the operator too far to the right is overlooked on an inspection of the code.

Preferred

```
ds      <- weatherAUS
names(ds) <- rattle::normVarNames(names(ds))
ds %>%
  group_by(location) %>%
  mutate(rainfall=cumsum(risk_mm)) %>%
  ggplot(aes(date, rainfall)) +
  geom_line() +
  facet_wrap(~location) +
  theme(axis.text.x=element_text(angle=90))
```

Alternative

```
ds      <- weatherAUS
names(ds) <- rattle::normVarNames(names(ds))
ds           %>%
  group_by(location)      %>%
  mutate(rainfall=cumsum(risk_mm))    %>%
  ggplot(aes(date, rainfall))        +
  geom_line()                      +
  facet_wrap(~location)           +
  theme(axis.text.x=element_text(angle=90))
```

11 Sub-Block Alignment

20. An interesting variation on the alignment of pipelines including graphics layering is to indent the graphics layering and include it within a code block (surrounded by curly braces). This highlights the graphics layering as a different type of concept to the data pipeline and ensures the graphics layering stands out as a separate stanza to the pipeline narrative. Note that a period is then required in the `ggplot2::ggplot()` call to access the pipelined dataset. The pipeline can of course continue on from this expression block. Here we show it being piped into a `dimRed::print()` to have the plot displayed and then saved into a variable for later processing. This style was suggested by Michael Thompson.

Preferred

```
ds      <- weatherAUS
names(ds) <- rattle::normVarNames(names(ds))
ds %>%
  group_by(location) %>%
  mutate(rainfall=cumsum(risk_mm)) %>%
{
  ggplot(., aes(date, rainfall)) +
    geom_line() +
    facet_wrap(~location) +
    theme(axis.text.x=element_text(angle=90))
} %T>%
  print() ->
plot_cum_rainfall_by_location
```

12 Functions

21. Functions should be no longer than a screen or a page. Long functions generally suggest the opportunity to consider more modular design. Take the opportunity to split the larger function into smaller functions.
22. When referring to a function in text include the empty round brackets to make it clear it is a function reference as in rpart().
23. Generally prefer a single `base::return()` from a function. Understanding a function with multiple and nested returns can be difficult. Sometimes though, particularly for simple functions as in the alternative below, multiple returns work just fine.

Preferred

```
factorial <- function(x)
{
  if (x==1)
  {
    result <- 1
  }
  else
  {
    result <- x * factorial(x-1)
  }

  return(result)
}
```

Alternative

```
factorial <- function(x)
{
  if (x==1)
  {
    return(1)
  }
  else
  {
    return(x * factorial(x-1))
  }
}
```

13 Function Definition Layout

24. Align function arguments in a function definition one per line. Aligning the = is also recommended to make it easier to view what is going on by presenting the assignments as a table.

Preferred

```
showDialPlot <- function(label      = "User!",
                           value       = 78,
                           dial.radius = 1,
                           label.cex   = 3,
                           label.color = "black")
{
  ...
}
```

Alternative

```
showDialPlot <- function(label="User!",
                           value=78,
                           dial.radius=1,
                           label.cex=3,
                           label.color="black")
{
  ...
}
```

Discouraged

```
showDialPlot <- function(label="User!", value=78,
                           dial.radius=1, label.cex=3,
                           label.color="black")
{
  ...
}

showDialPlot <- function(label="User!",
                           value=78,
                           dial.radius=1,
                           label.cex=3,
                           label.color="black")
```

Alternative

```
showDialPlot <- function(
  label="User!",
  value=78,
  dial.radius=1,
  label.cex=3,
  label.color="black"
)
```

14 Function Call Layout

25. Don't add spaces around the = for named arguments in parameter lists. Visually this ties the named arguments together and highlights this as a parameter list. This style is at odds with the default R printing style and is the only situation where I tightly couple a binary operator. In all other situations there should be a space around the operator.

Preferred

```
readr::read_csv(file="data.csv", skip=1e5, na=". ", progress=FALSE)
```

Discouraged

```
read.csv(file = "data.csv", skip =
          1e5, na = ". ", progress
          = FALSE)
```

26. For long lists of parameters improve readability using a table format by aligning on the =.

Preferred

```
readr::read_csv(file      = "data.csv",
                skip       = 1e5,
                na        = ". ",
                progress   = FALSE)
```

27. All but the final argument to a function call can be easily commented out. However, the latter arguments are often optional and whilst exploring them we will likely comment them out. An idiosyncratic alternative places the comma at the beginning of the line so that we can easily comment out specific arguments except for the first one, which is usually the most important argument and often non-optional. This is quite a common style amongst SQL programmers and can be useful for R programming too.

Usual

```
dialPlot(value      = 78,
          label       = "UseR!",
          dial.radius = 1,
          label.cex   = 3,
          label.color = "black")
```

Alternative

```
dialPlot(value      = 78
          , label       = "UseR!"
          , dial.radius = 1
          , label.cex   = 3
          , label.color = "black"
          )
```

Discouraged

```
dialPlot( value=78, label="UseR!", dial.radius=1,
          label.cex=3, label.color="black")
```

15 Functions from Packages

28. R has a mechanism (called namespaces) for identifying the names of functions and variables from specific packages. There is no rule that says a package provided by one author can not use a function name already used by another package or by base R. Thus, functions from one package might overwrite the definition of a function with the same name from another package or from base R itself. A mechanism to ensure we are using the correct function is to prefix the function call with the name of the package providing the function, just like `dplyr::mutate()`.

Generally in commentary we will use this notation to clearly identify the package which provides the function. In our interactive R usage and in scripts we tend not to use the namespace notation. It can clutter the code and arguably reduce its readability even though there is the benefit of clearly identifying where the function comes from.

For common packages we tend not to use namespaces but for less well-known packages a namespace at least on first usage provides valuable information. Also, when a package provides a function that has the same name as a function in another namespace, it is useful to explicitly supply the namespace prefix.

Preferred

```
library(dplyr)      # Data wrangling, mutate().
library(lubridate) # Dates and time, ymd_hm().
library(ggplot2)    # Visualize data.

ds <- get(dsname) %>%
  mutate(timestamp=ymd_hm(paste(date, time))) %>%
  ggplot(aes(timestamp, measure)) +
  geom_line() +
  geom_smooth()
```

Alternative

The use of the namespace prefix increases the verbosity of the presentation and that has a negative impact on the readability of the code. However it makes it very clear where each function comes from.

```
ds <- get(dsname) %>%
  dplyr::mutate(timestamp=
    lubridate::ymd_hm(paste(date, time))) %>%
  ggplot2::ggplot(ggplot2::aes(timestamp, measure)) +
  ggplot2::geom_line() +
  ggplot2::geom_smooth()
```

16 Assignment

29. Avoid using `base:::=` for assignment. It was introduced in S-Plus in the late 1990s as a convenience but is ambiguous (named arguments in functions, mathematical concept of equality). The traditional backward assignment operator `base::<-` implies a flow of data and for readability is explicit about the intention.

Preferred

```
a <- 42
b <- mean(x)
```

Discouraged

```
a = 42
b = mean(x)
```

30. The forward assignment `base::->` should generally be avoided. A single use case justifies it in pipelines where logically we do an assignment at the end of a long sequence of operations. As a side effect operator it is vitally important to highlight the assigned variable whenever possible and so out-denting the variable after the forward assignment to highlight it is recommended.

Preferred

```
ds[vars] %>%
  sapply(function(x) all(x == x[1L])) %>%
  which() %>%
  names() %T>%
  print() ->
constants
```

Traditional

```
constants <-
  ds[vars] %>%
  sapply(function(x) all(x == x[1L])) %>%
  which() %>%
  names() %T>%
  print()
```

Discouraged

```
ds[vars] %>%
  sapply(function(x) all(x == x[1L])) %>%
  which() %>%
  names() %T>%
  print() ->
constants
```

17 Miscellaneous

31. Do not use the semicolon to terminate a statement unless it makes a lot of sense to have multiple statements on one line. Line breaks in R make the semicolon optional.

Preferred

```
threshold <- 0.7
maximum   <- 1.5
minimum   <- 0.1
```

Alternative

```
threshold <- 0.7; maximum <- 1.5; minimum <- 0.1
```

Discouraged

```
threshold <- 0.7;
maximum   <- 1.5;
minimum   <- 0.1;
```

32. Do not abbreviate TRUE and FALSE to T and F.

Preferred

```
is_windows  <- FALSE
open_source <- TRUE
```

Discouraged

```
is_windows  <- F
open_source <- T
```

33. Separate parameters in a function call with a comma followed by a space.

Preferred

```
dialPlot(value=78, label="UseR!", dial.radius=1)
```

Discouraged

```
dialPlot(value=78,label="UseR!",dial.radius=1)
```

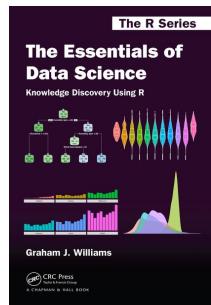
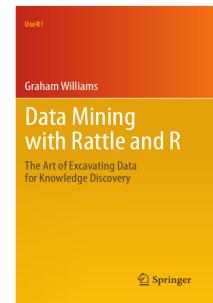
34. Ensure that files are under version control such as with github to allow recovery of old versions of the file and to support multiple people working on the same files.

18 Good Practise

35. Ensure that files are under version control such as with github or bitbucket. This allows recovery of old versions of the file and multiple people working on the same repository. It also facilitates sharing of the material. If the material is not to be shared then bitbucket is a good option for private repositories.

19 Further Reading

The [Rattle](#) book ([Williams, 2011](#)), published by Springer, provides a comprehensive introduction to data mining and analytics using Rattle and R. It is available from [Amazon](#). Rattle provides a graphical user interface through which the user is able to load, explore, visualise, and transform data, and to build, evaluate, and export models. Through its Log tab it specifically aims to provide an R template which can be exported and serve as the starting point for further programming with data in R.



The [Essentials of Data Science](#) book ([Williams, 2017a](#)), published by CRC Press, provides a comprehensive introduction to data science through programming with data using R. It is available from [Amazon](#). The book provides a template based approach to doing data science and knowledge discovery. Templates are provided for data wrangling and model building. These serve as generic starting points for programming with data, and are designed to require minimal effort to get started. Visit <https://essentials.togaware.com> for further guides and templates.

There are many style guides available and the guidelines here are generally consistent and overlap considerably with many others. I try to capture the motivation for each choice. My style choices are based on my experience over 30 years of programming in very many different languages and it should be recognised that some elements of style are personal preference and others have very solid foundations. Unfortunately in reading some style guides the choices made are not always explained and without the motivation we do not really have a basis to choose or to debate.

The guidelines at [Google](#) and from [Hadley Wickham](#) and [Colin Gillespie](#) are similar but I have some of my own idiosyncrasies. Also see [Wikipedia](#) for an excellent summary of many styles.

Rasmus Bååth, in [The State of Naming Conventions in R](#), reviews naming conventions used in R, finding that the initial lower case capitalised word scheme for functions was the most popular, and dot separated names for arguments similarly. We are however seeing a migration away from the dot in variable names as it is also used as a class separator for object oriented coding. Using the underscore is now preferred.

20 References

- R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.
- Wickham H, Chang W (2016). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 2.2.1, URL <https://CRAN.R-project.org/package=ggplot2>.
- 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.
- Williams GJ (2017a). *The Essentials of Data Science: Knowledge discovery using R*. The R Series. CRC Press.
- Williams GJ (2017b). *rattle: Graphical User Interface for Data Science in R*. R package version 5.1.0, URL <https://CRAN.R-project.org/package=rattle>.

This document, sourced from StyleO.Rnw bitbucket revision 241, was processed by Knitr version 1.20 of 2018-02-20 10:11:46 UTC and took 4 seconds to process. It was generated by gjw on Ubuntu 18.04 LTS.

Hands-On Data Science with R

Pulling it Together into a Package

Graham.Williams@togaware.com

1st June 2016

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

In this module we review the `package.skeleton()` function of R `R-base` (?).

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 © 2013-2015 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.



Draft Only

1 Functions to Package

We will create a package containing the following functions. The Functions module covered the development of these functions and more. The aim is to get started with a package rather than to produce a fully developed package. Thus we include two quite simple functions to illustrate the process.

```
varWeights <- function(formula, data)
{
  vars <- as.character(attr(terms(model.frame(formula, data)),
                           "variables"))[-1L]
  target <- vars[[1]]
  inputs <- vars[-1]
  abscor <- function(x)
    abs(suppressWarnings(cor(as.numeric(data[[x]]),
                               as.numeric(data[[target]]),
                               use="pairwise.complete.obs")))
  correlation <- sapply(inputs, abscor)

  correlation[is.na(correlation)] <- 0
  correlation[correlation == 1] <- 0

  return(correlation/sum(correlation))
}

selectVars <- function(formula, data, n)
{
  pvar <- varWeights(formula, data)
  draws = sample(names(pvar), size=n, replace=FALSE, prob=pvar)
  return(draws)
}
```

Draft Only

2 Package Skeleton

Having defined the above functions in an R session we can then begin creating the package to include just these two functions, to start with. We'll call the package `vw`.

```
package.skeleton(name = "vw", list = c("varWeights", "selectVars"))

## Creating directories ...
## Creating DESCRIPTION ...
## Creating NAMESPACE ...
## Creating Read-and-delete-me ...
## Saving functions and data ...
## Making help files ...
## Done.

## Further steps are described in './vw/Read-and-delete-me'.
```

This creates a folder with the name `vw`. The folder contains three files. One is called `DESCRIPTION` and should be edited to provide details of the package. Another is called `NAMESPACE` and will not need to be changed for now. The third is called `Read-me-and-delete` which, as the name suggests, is worth having a read of, but then deleting.

Two folders are also created, named `R` and `man`. The `R` folder contains several `.R` files, one for each of the functions forming the package, containing the actual function definitions. The `man` folder contains documentation (manual pages), again a single file for each function, plus an additional file for documenting the package in general. Each file has the `.Rd` extension, for R documentation.

Draft Only

3 Check Package

After editing each of the .Rd files to describe the functionality, we can check that the package has integrity using:

```
$ R CMD check vw
```

Draft Only

4 Build Package

If all is okay, we can then build the package:

```
$ R CMD build vw
```

Draft Only

5 Install Package

Finally, install the package:

```
$ R CMD INSTALL vw_1.0.tar.gz
```

The package file `vw_1.0.tar.gz` is the source code and documentation, stored in a `.tar` archive format, compressed using `gzip` to provide a `.gz` file for distribution. This is a so-called source package. It can be installed on GNU/Linux systems such as Ubuntu. Extra work is required to build a binary package for MS/Windows or a package for Mac/OSX.

Once the package is installed we can load it into R:

```
library(vw)
```

Draft Only

Data Science with R

Hands-On

Pulling it Together into a Package

6 Cleanup

Draft Only

7 Submitting to CRAN

The CRAN team are volunteers dedicated to maintaining the quality of R and the suite of packages available to R. Kurt Hornik and Uwe Ligges, in particular, work tirelessly managing the uploading of packages across the different platforms. Thus the onus needs to be on us, the package contributors, to minimise their effort by ensuring our packages have been thoroughly checked before an upload to CRAN.

Draft Only

Data Science with R

Hands-On

Pulling it Together into a Package

8 Further Reading

Draft Only

9 References

R Core Team (2016). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL <https://www.R-project.org/>.

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.

This document, sourced from PackageO.Rnw bitbucket revision 148, was processed by KnitR version 1.12.3 of 2016-01-22 and took 1.2 seconds to process. It was generated by gjw on theano running Ubuntu 14.04.4 LTS with Intel(R) Core(TM) i7-3517U CPU @ 1.90GHz having 4 cores and 3.9GB of RAM. It completed the processing 2016-06-01 19:43:15.

Draft Only

Generated 2016-06-01 19:43:19+08:00