Data Science with R
Dealing with Dates and Time

Graham.Williams@togaware.com

19th August 2014

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

In this chapter we review the functionality provided by R to deal with dates and times.

The required packages for this module include:

```
library(lubridate)          # Simplified date/time handling.
library(ggplot2)            # Visualise data.
library(tidyr)              # Tidy the dataset.
library(rattle)             # The weatherAUS dataset.
library(scales)             # Rescaling axes in ggplot2.
library(WDI)                # World bank data.
library(countrycode)
library(plyr)               # Transform data.
library(gridExtra)          # Multiple plots on a grid.
```

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.

1 Reading Dates

Here we load a dataset using `read.csv()`. The source dataset actually uses semicolons rather than commas to separate the fields, and missing values are represented as a period.

```
stroke <- read.csv(file.path("data", "stroke.csv"), sep=";", na.strings=".")
dds <- stroke
head(dds$DIED)

## [1] 7.01.1991  <NA>       2.06.1991  13.01.1991 23.01.1996 13.01.1991
## 414 Levels: 10.02.1993 10.03.1992 10.03.1993 10.04.1995 ... 9.11.1993

head(dds$DSTR)

## [1] 2.01.1991  3.01.1991  8.01.1991  11.01.1991 13.01.1991 13.01.1991
## 575 Levels: 10.01.1993 10.02.1991 10.02.1992 10.03.1991 ... 9.12.1991
```

We notice that there are two variables that look like dates: *DIED* and *DSTR*. Let's check:

```
class(dds$DIED)

## [1] "factor"

class(dds$DSTR)

## [1] "factor"
```

They have been read in as factors. We can use `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=`.

```
class(dds$DIED <- as.Date(dds$DIED, format="%d.%m.%Y"))

## [1] "Date"

class(dds$DSTR <- as.Date(dds$DSTR, format="%d.%m.%Y"))

## [1] "Date"
```

Notice now that the dates are printed in a standard ISO format (`%Y-%m-%d`).

```
head(dds$DIED)

## [1] "1991-01-07" NA           "1991-06-02" "1991-01-13" "1996-01-23"
## [6] "1991-01-13"

head(dds$DSTR)

## [1] "1991-01-02" "1991-01-03" "1991-01-08" "1991-01-11" "1991-01-13"
## [6] "1991-01-13"
```

2 Lubridate: Simplified Dates and Times

We will now load the dataset again using `read.csv()`, but this time we will use lubridate (Grolemund and Wickham, 2013) to perform the conversion, which is somewhat simpler.

```
ds <- stroke
```

We can convert the dates this time using `dmy()` from lubridate. We choose `dmy()` since that clearly corresponds to our date format.

```
library(lubridate)
class(ds$DIED <- dmy(ds$DIED))

## [1] "POSIXct" "POSIXt"

class(ds$DSTR <- dmy(ds$DSTR))

## [1] "POSIXct" "POSIXt"
```

The data types are now `POSIXct` standard date and time data types. In fact, we notice that the date fields now are members of two classes, `POSIXct` and `POSIXt`.

```
head(ds$DIED)

## [1] "1991-01-07 UTC" NA               "1991-06-02 UTC" "1991-01-13 UTC"
## [5] "1996-01-23 UTC" "1991-01-13 UTC"

head(ds$DSTR)

## [1] "1991-01-02 UTC" "1991-01-03 UTC" "1991-01-08 UTC" "1991-01-11 UTC"
## [5] "1991-01-13 UTC" "1991-01-13 UTC"
```

3 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] "2014-08-19 20:18:23 EST"

class(ct)

## [1] "POSIXct" "POSIXt"

str(ct)

##  POSIXct[1:1], format: "2014-08-19 20:18:23"

unclass(ct)

## [1] 1.408e+09
```

POSIXlt (local time) represents the date and time as a named list of vectors.

```
(ct <- as.POSIXlt(ct))

## [1] "2014-08-19 20:18:23 EST"

class(ct)

## [1] "POSIXlt" "POSIXt"

str(ct)

##  POSIXlt[1:1], format: "2014-08-19 20:18:23"

unclass(ct)

## $sec
## [1] 23.1
##
## $min
## [1] 18
##
## $hour
## [1] 20
##
## $mday
## [1] 19
##
....
```

4 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] "Tue 19 Aug 2014 20:18:23 EST"
```

The table below illustrates many of the available options.

%c	date and time	Tue 19 Aug 2014 20:18:23 EST
%x	date	19/08/14
%F	ISO 8601	2014-08-19
%d/%m/%Y	day/month/year	19/08/2014
%a %e %m %Y	day month year	Tue 19 Aug 2014
%A %d %B %Y	day month year	Tuesday 19 August 2014
Day %j and Week %U of %Y	day/week of the year	Day 231 and Week 33 of 2014
%A: Day %w of Week %U	day of week	Tuesday: Day 2 of Week 33
%y%m%d	two digit date stamp	140819
%X	time	20:18:23
%r	time	08:18:23 PM
%k.%M %p	24 hour time	20.18 PM
%l.%M %p	12 hour time	8.18 PM
%H%M%S	timestamp	201823
%I:%M:%S %p	time 12 hour clock	08:18:23 PM
%H:%M:%S %z	time and UTC offset	20:18:23 +1000
%H:%M:%S %Z	time and timezone	20:18:23 EST

There are more! See the help page for `strptime()` for details.

5 Computing on Dates and Times: difftime

R Dates can be used in computations quite naturally.

```
dds$LIVED <- dds$DIED - dds$DSTR
head(dds$LIVED)

## Time differences in days
## [1]    5   NA  145    2 1836    0

class(dds$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 secs
## [1]    432000        NA  12528000    172800 158630400         0

class(ds$LIVED)

## [1] "difftime"
```

Notice that 432000 seconds is 5 days:

```
as.integer(ds$LIVED[1])/60/60/24

## [1] 5
```

We can change the default displayed units if desired.

```
units(ds$LIVED)

## [1] "secs"

units(ds$LIVED) <- "days"
units(ds$LIVED)

## [1] "days"

head(ds$LIVED)

## Time differences in days
## [1]    5   NA  145    2 1836    0
```

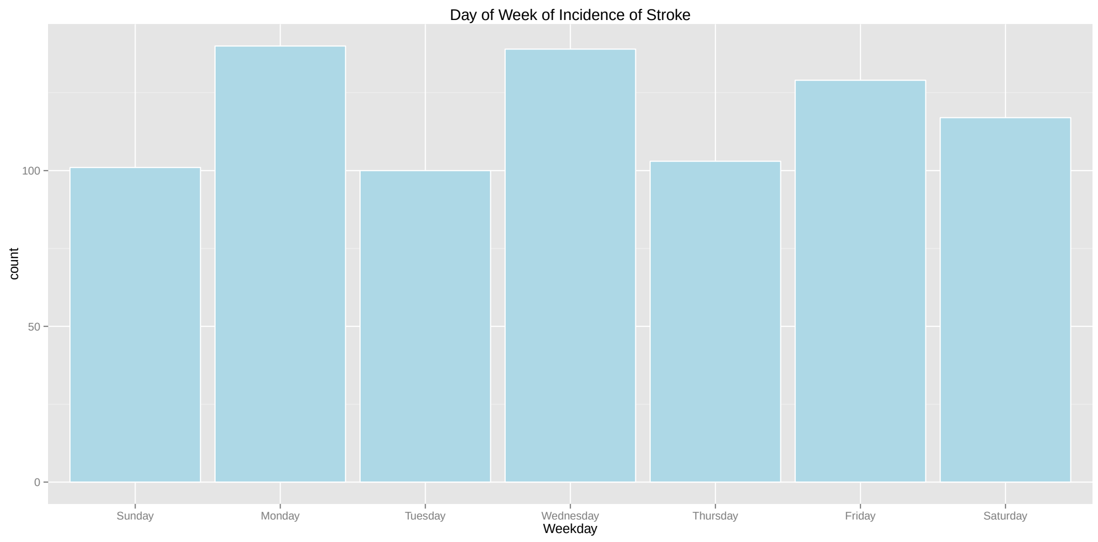
6 Lubridate Intervals

```
ds$INTERVAL <- with(ds, interval(DSTR, DIED))
head(ds$INTERVAL)

## [1] 1991-01-02 UTC--1991-01-07 UTC 1991-01-03 UTC--NA
## [3] 1991-01-08 UTC--1991-06-02 UTC 1991-01-11 UTC--1991-01-13 UTC
## [5] 1991-01-13 UTC--1996-01-23 UTC 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(duration(ds$INTERVAL))

## [1] "432000s (432000 seconds)"       NA
## [3] "12528000s (12528000 seconds)"   "172800s (172800 seconds)"
## [5] "158630400s (158630400 seconds)" "0s (0 seconds)"
```

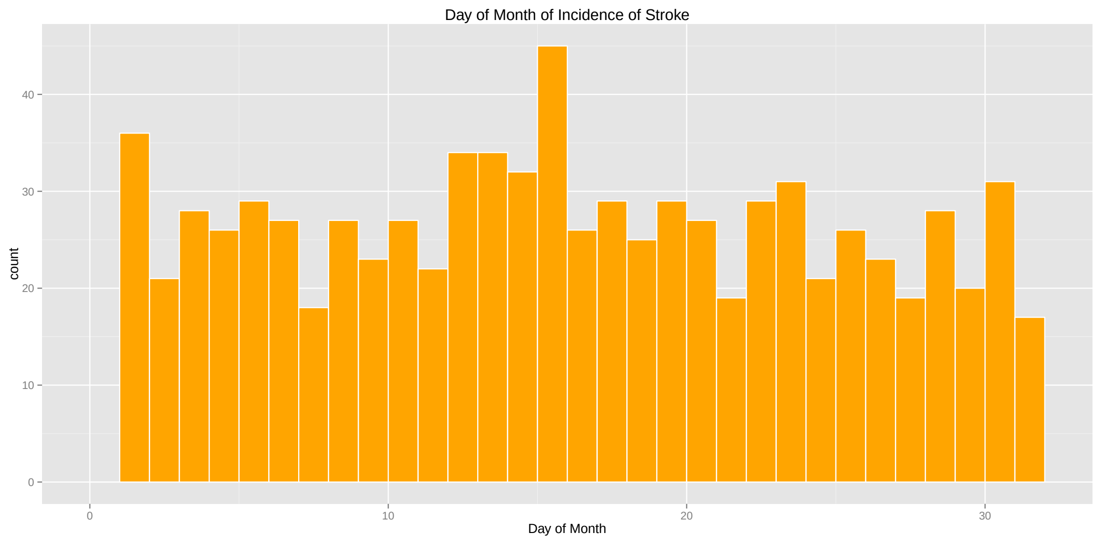
7 Plot Day of Week Frequencies

```
ds <- stroke
ds$DSTR <- dmy(ds$DSTR)
g <- ggplot(data=ds, aes(wday(DSTR, label=TRUE, abbr=FALSE)))
g <- g + geom_histogram(colour="white", fill="lightblue")
g <- g + ggtitle("Day of Week of Incidence of Stroke")
g <- g + xlab("Weekday")
print(g)
```



8 Plot Day of Month Frequencies

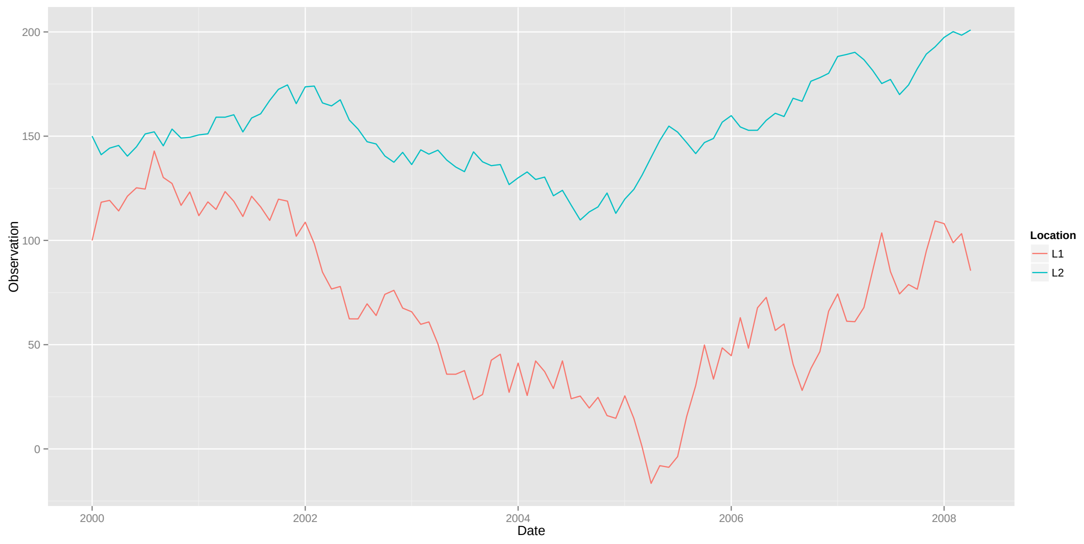
```
g <- ggplot(data=ds, aes(mday(DSTR)))
g <- g + geom_histogram(binwidth=1, colour="white", fill="orange")
g <- g + ggtitle("Day of Month of Incidence of Stroke")
g <- g + xlab("Day of Month")
print(g)
```



9 Plot Daily Observations

Using reshape2 (?) and ggplot2 (Wickham and Chang, 2014).

```
library("reshape2") # melt()
library("ggplot2")  # ggplot()

ds <- 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))

dsm <- melt(ds, id="Date")

g <- ggplot(data=dsm, aes(x=Date, y=value, colour=variable))
g <- g + geom_line()
g <- g + ylab("Observation")
g <- g + labs(colour="Location")
print(g)
```



10 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, 2013). We also use countrycode (Arel-Bundock, 2014) to map the country codes.

```
library(WDI)
library(ggplot2)
library(countrycode)
```

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.ADO.TFRT"
## [2,] "SP.DYN.TFRT.IN"
## [3,] "SP.DYN.WFRT"
....

(indicators <- meta.data[1:2, 1])

## [1] "SP.ADO.TFRT"    "SP.DYN.TFRT.IN"

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))

##    iso2c       country year SP.ADO.TFRT SP.DYN.TFRT.IN
## 1     AU     Australia 2001      17.229          1.739
## 2     AU     Australia 2002      16.724          1.756
## 3     AU     Australia 2003      16.584          1.748
....
```

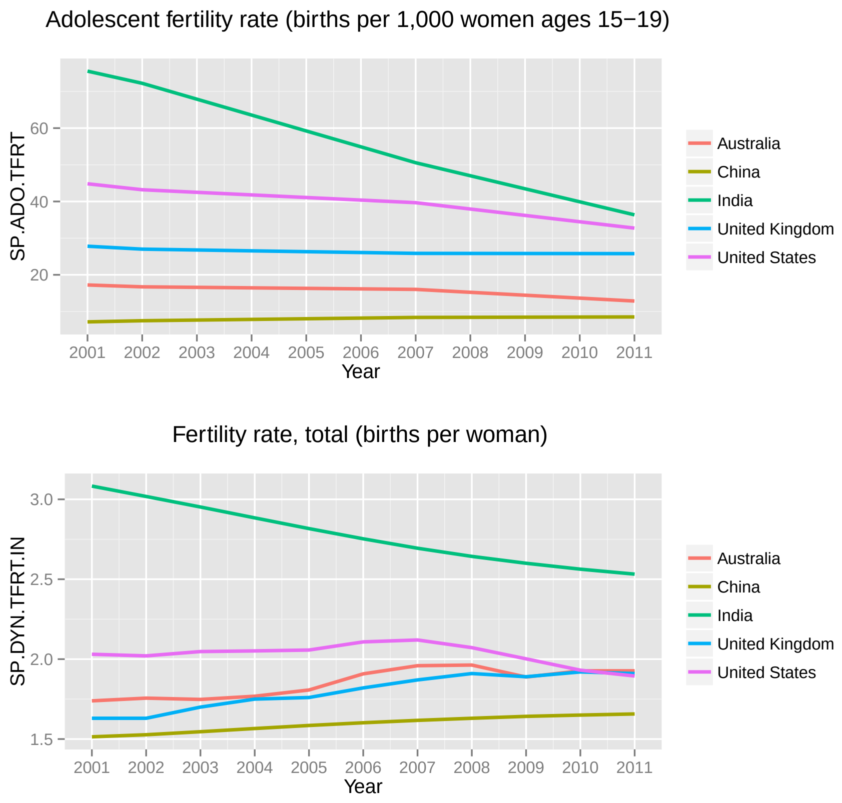
11 Plot World Bank Data: Multiple Plots

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 + ggtitle(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)
```



12 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]
```

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
```

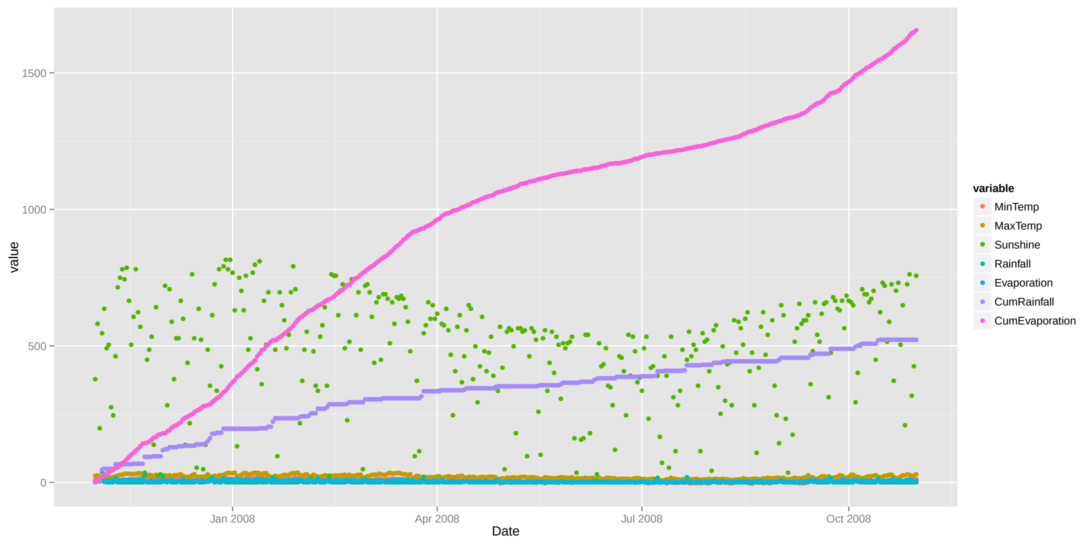
We will also accumulate the amount of rainfall and the amount of evaporation over the period:

```
ds$CumRainfall <- cumsum(ds$Rainfall)
ds$CumEvaporation <- cumsum(ds$Evaporation)
```

We now also melt the dataset into a form that will facilitate plotting all of the variables.

```
dsm <- melt(ds, id="Date")
```

```
g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))
g <- g + geom_point()
print(g)

## Warning: Removed 3 rows containing missing values (geom_point).
```

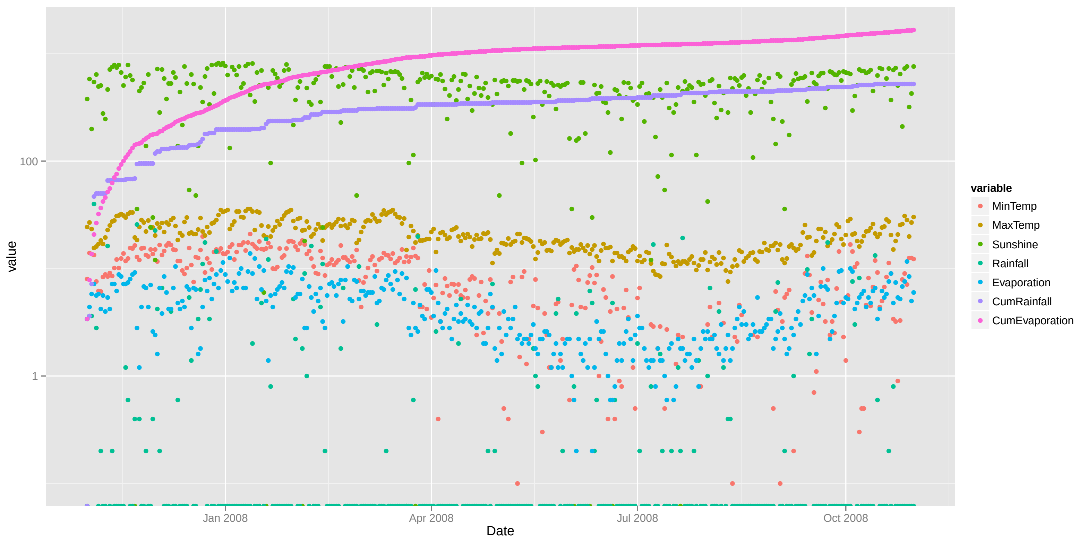


That's a start, but not real good. The very large numbers swamp the rest. Notice also the warning regarding observations with missing values. We'll ignore that (and turn the warning off for the following plots).

13 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))
g <- g + geom_point()
g <- g + scale_y_log10()
print(g)

## Warning:  NaNs produced
## Warning:  Removed 54 rows containing missing values (geom_point).
```

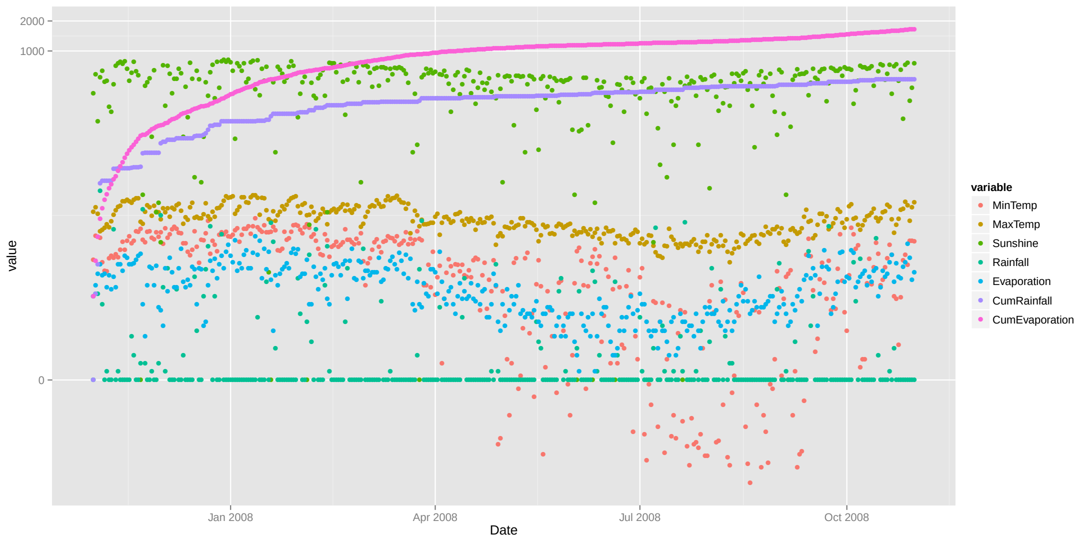


So that is a little better but note the warnings. We can not take the log of numbers less than or equal to zero. These data are ignored in plotting. That is not really what we wanted to do.

14 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))
g <- g + geom_point()
g <- g + scale_y_continuous(trans="asinh")
print(g)
```

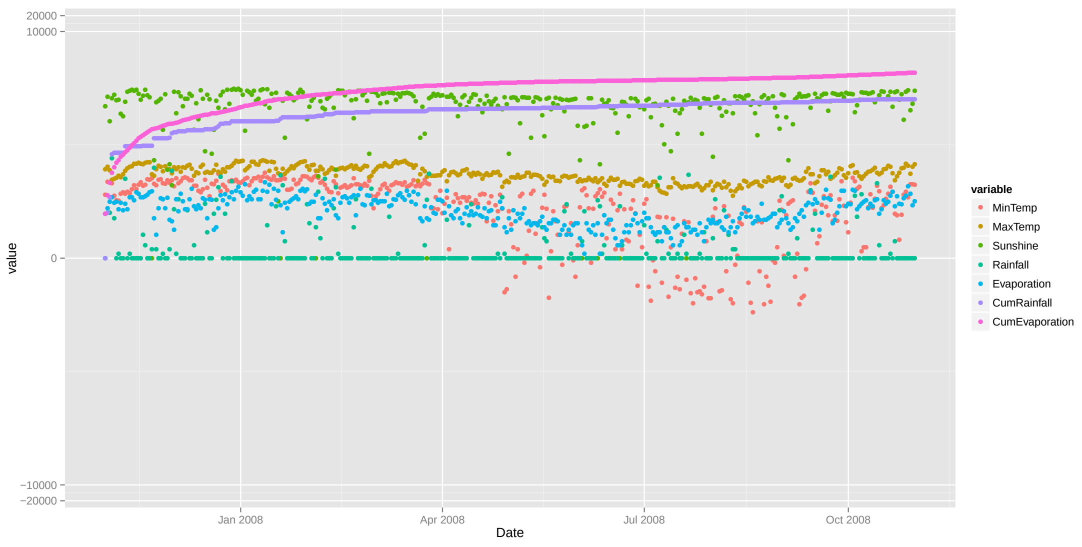


We now get the negatives and zeros into the picture.

15 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))
g <- g + geom_point()
g <- g + scale_y_continuous(trans="asinh",
                            limits=c(-1e4, 1e4))
print(g)
```

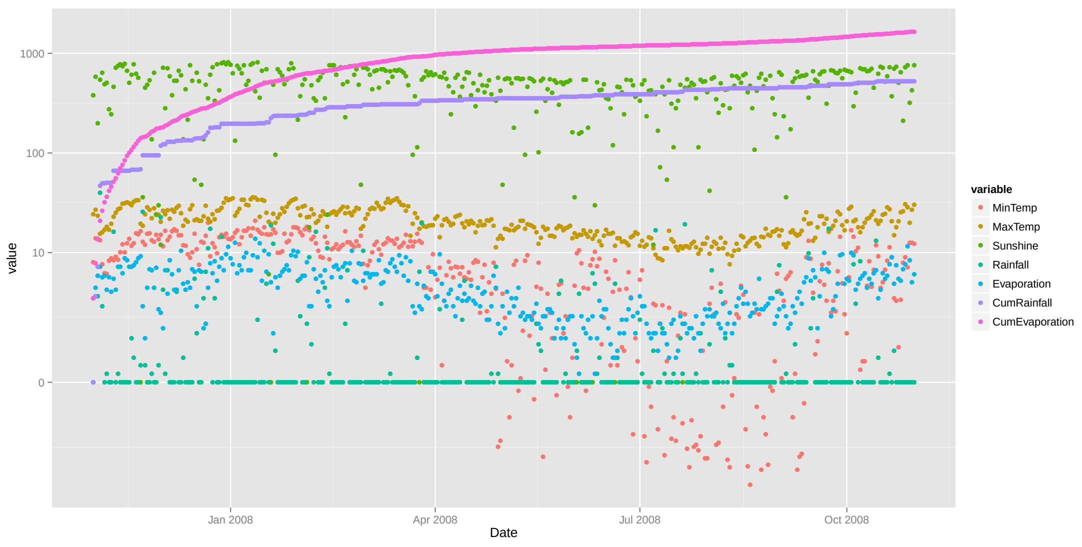


Actually though, there's quite a bit of wasted space now, so we'll drop the limits for the following plots. There is no point really in taking up precious real estate for no particular purpose.

16 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))
g <- g + geom_point()
g <- g + scale_y_continuous(trans="asinh",
                            breaks=c(-10, 0, 10, 1e2, 1e3))
print(g)
```

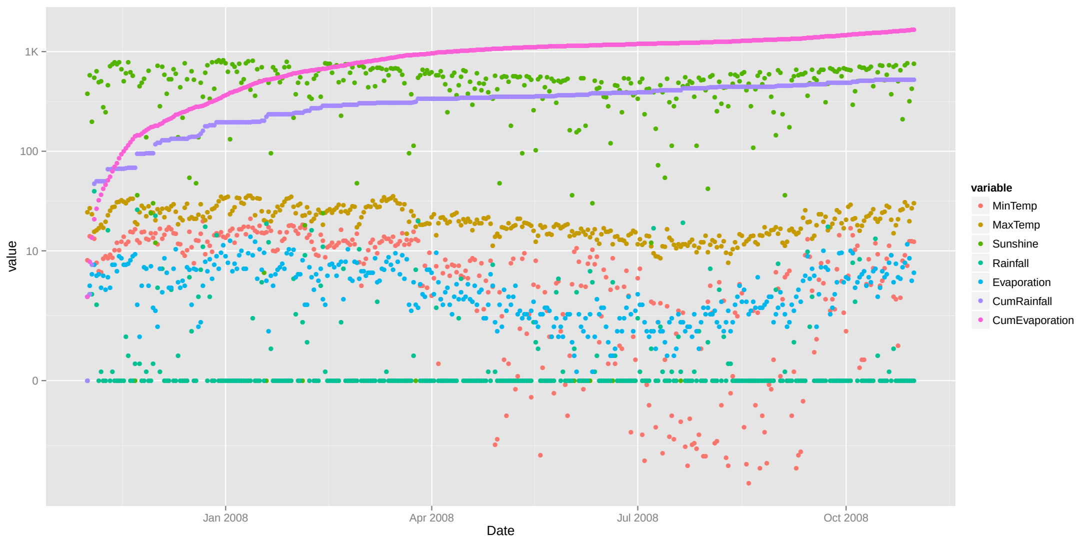


This does add value to the plot. The actual gradation of points along the y axis is now much easier to perceive.

17 Scale Options: Label the Breaks

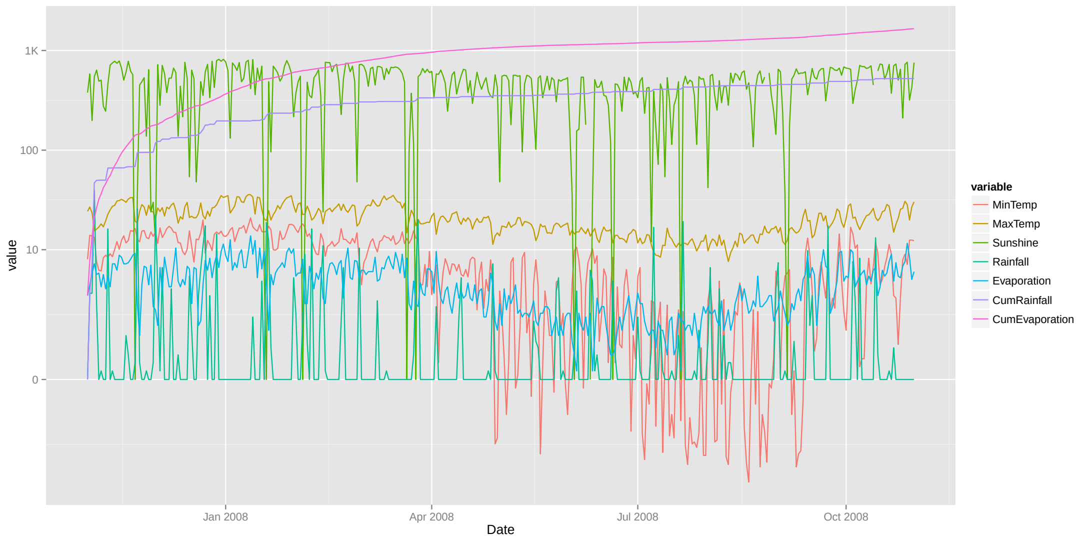
As well as specifying the breaks we can also specify how they are to be labelled. This could be useful when we want to abbreviate the labels in some standard way, if that improves the readability.

```
g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))
g <- g + geom_point()
g <- g + scale_y_continuous(trans="asinh",
                            breaks=c(-10, 0, 10, 1e2, 1e3),
                            labels=c("-10", "0", "10", "100", "1K"))
print(g)
```



18 Plot Lines instead of Points

```
g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))
g <- g + geom_line()
g <- g + scale_y_continuous(trans="asinh",
                            breaks=c(-10, 0, 10, 1e2, 1e3),
                            labels=c("-10", "0", "10", "100", "1K"))
print(g)
```

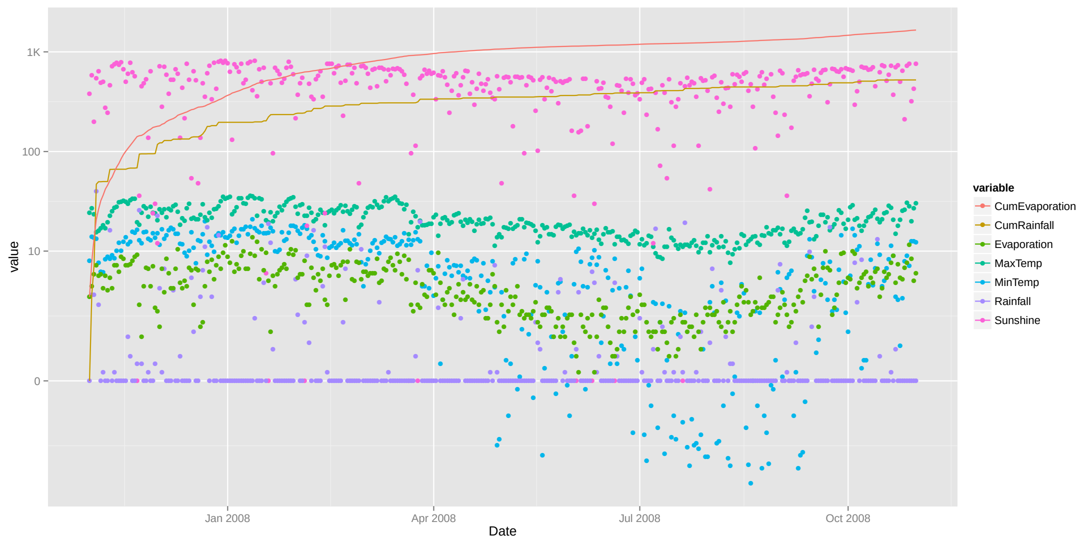


That is pretty messy looking and the story is hard to tell.

19 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.

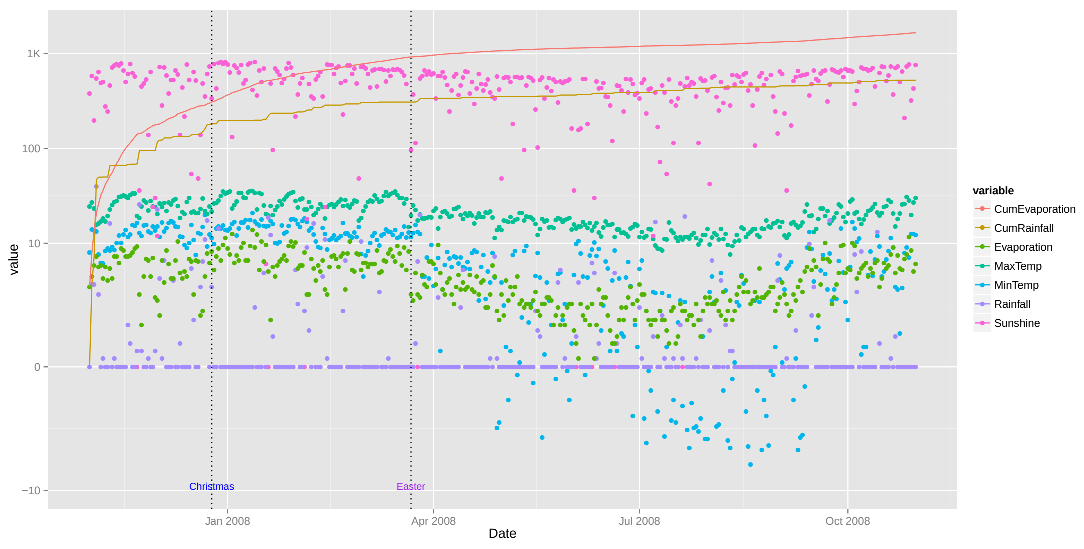
```
draw.lines <- c("CumRainfall", "CumEvaporation")

g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))
g <- g + geom_point(data=subset(dsm, !variable %in% draw.lines))
g <- g + geom_line(data=subset(dsm, variable %in% draw.lines))
g <- g + scale_y_continuous(trans="asinh",
                            breaks=c(-10, 0, 10, 1e2, 1e3),
                            labels=c("-10", "0", "10", "100", "1K"))
print(g)
```



20 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.

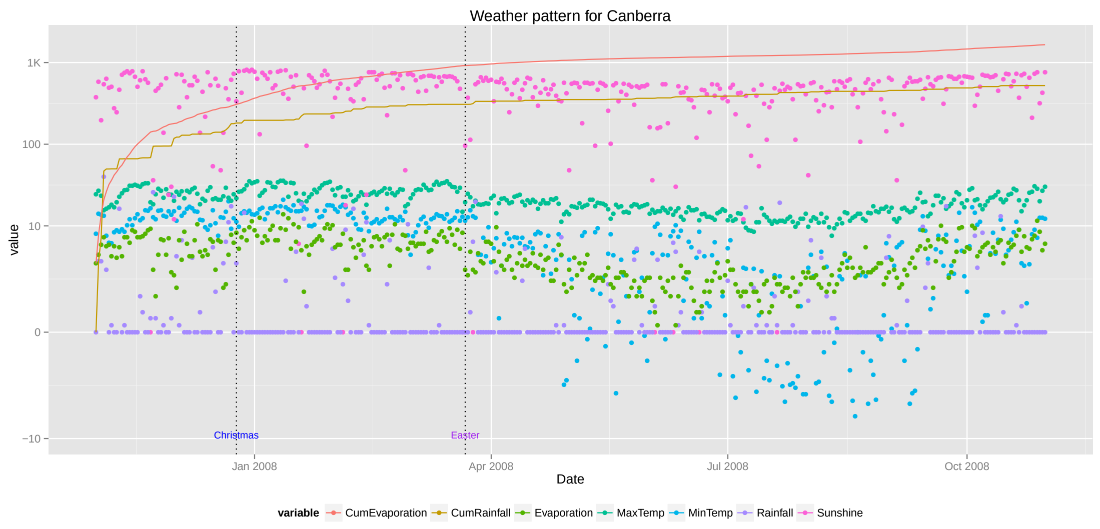
```
events <- as.Date(c("2007-12-25", "2008-03-22"))

g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))
g <- g + geom_point(data=subset(dsm, !variable %in% draw.lines))
g <- g + geom_line(data=subset(dsm, variable %in% draw.lines))
g <- g + scale_y_continuous(trans="asinh",
                            breaks=c(-10, 0, 10, 1e2, 1e3),
                            labels=c("-10", "0", "10", "100", "1K"))
g <- g + geom_vline(xintercept=as.numeric(events), linetype=3)
g <- g + annotate("text", events[1], -9, label="Christmas", size=3, colour="blue")
g <- g + annotate("text", events[2], -9, label="Easter", size=3, colour="purple")
print(g)
```



21 Finishing Touches

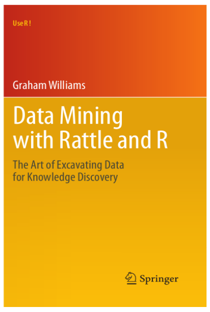
Add a title. Place the legend at the bottom.

```
g <- ggplot(dsm, aes(x=Date, y=value, colour=variable))
g <- g + geom_point(data=subset(dsm, !variable %in% draw.lines))
g <- g + geom_line(data=subset(dsm, variable %in% draw.lines))
g <- g + scale_y_continuous(trans="asinh",
                            breaks=c(-10, 0, 10, 1e2, 1e3),
                            labels=c("-10", "0", "10", "100", "1K"))
g <- g + geom_vline(xintercept=as.numeric(events), linetype=3)
g <- g + annotate("text", events[1], -9, label="Christmas", size=3, colour="blue")
g <- g + annotate("text", events[2], -9, label="Easter", size=3, colour="purple")
g <- g + ggtitle(sprintf("Weather pattern for %s", weather$Location[1]))
g <- g + theme(legend.direction="horizontal", legend.position="bottom")
print(g)
```



22 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.


Use R!
Graham Williams
Data Mining with Rattle and R
The Art of Excavating Data for Knowledge Discovery
Springer

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:

- 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.

23 References

Arel-Bundock V (2013). *WDI: World Development Indicators (World Bank).* R package version 2.4, URL http://CRAN.R-project.org/package=WDI.

Arel-Bundock V (2014). *countrycode: Convert country names and country codes.* R package version 0.17, URL http://CRAN.R-project.org/package=countrycode.

Grolemund G, Wickham H (2013). *lubridate: Make dealing with dates a little easier.* R package version 1.3.3, URL http://CRAN.R-project.org/package=lubridate.

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 3.1.4, URL http://rattle.togaware.com/.

This document, sourced from DateTimeO.Rnw revision 500, was processed by KnitR version 1.6 of 2014-05-24 and took 9.1 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 2014-08-19 20:18:31.