# Data Science with R
# Dealing with Big Data

Graham.Williams@togaware.com

9th July 2014

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

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(sqldf)
```

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 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
##   1.307   0.016   1.338

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
##   0.148   0.001   0.149

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
##   0.042   0.000   0.041

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
##   0.687   0.007   0.695
```

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         1.5569
....
```

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         1.5569
....
```

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.

## 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 (?).

## 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 (?).

## 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 (?).

## 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 (?).

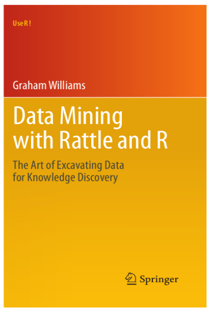
## 9 Working with Data Tables

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.


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

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:

- A good overview of `data.table` is available through the useR!2014 Tutorial.

## 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 1.9.2, 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: 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 BigDataO.Rnw revision 455, was processed by KnitR version 1.6 of 2014-05-24 and took 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 2014-07-09 21:44:39.*