Data Science with R
Reading Data into R

Graham.Williams@togaware.com

9th June 2014

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

In this module we explore options for reading data into R. We start by reading a dataset from a CSV file and store it into an R *data frame*. We also consider other sources of data and illustrate how they can be read into R.

The required packages for this module include:

```
library(RCurl)
library(foreign)
```

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 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] "dvdtrans.csv"   "heart.csv"      "ozdata.csv"     "stroke.csv"
## [5] "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
....

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
....

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...
....
```

2 Reading from CSV—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
....

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
....

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
....
```

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.

3 Reading from CSV—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
....

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...
....
```

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:  invalid argument type
```

4 Reading from CSV—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
....

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
....

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...
....
```

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, ...)
....
```

5 Reading from CSV—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)
```

6 Viewing the Data

```
View(stroke)
```

```
library(RGtk2Extras)
dfedit(stroke)
```

```
library(Deducer)
date.viewer()
```

7 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
....
```

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
....

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
....

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...
....
```

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(".", "?", " "))
```

8 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"
....
```

9 Writing to CSV

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

10 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.

11 Data from Spreadsheets

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

12 Loading tab/txt Files

Exercise: Illustrate loading a tab delimited txt file.

13 Loading Fixed Width Files

Exercies: Illustrate reading a fixed width data file.

```
read.fwf()
```

14 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 1.9000000000000001    0     1.54    2
....

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
....

str(tolerance)

## 'data.frame': 80 obs. of  6 variables:
##  $ id       : int  9 9 9 9 9 45 45 45 45 45 ...
##  $ age      : int  11 12 13 14 15 11 12 13 14 15 ...
##  $ tolerance: Factor w/ 17 levels "1","1.11","1.12",..: 13 9 10 12 15 3 6 ...
....

summary(tolerance)

##        id             age       tolerance       male      
##  Min.   :   9   Min.   :11   1.12   :10   Min.   :0.000  
##  1st Qu.: 410   1st Qu.:12   1.22   :10   1st Qu.:0.000  
##  Median : 674   Median :13   1.99   : 9   Median :0.000  
....
```

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"))
```

15 Data from Google Drive

Exercise: This is an older way but Google has updated how this works. Investigate and demonstrate how to directly read a Google Drive document.

We can load data into R from a spreadsheet stored on Google Docs. To access Internet based data we use RCurl (Temple Lang, 2013). To access the contents of a Google Docs document we use example code that was publicly provided by Tony Hirst.

A helper function is defined to load data from a shared public Google Spreadsheet:

```
library(RCurl)
gsqAPI <- function(key, query, gid=0)
{
  return(read.csv(paste('http://spreadsheets.google.com/tq?',
                        'tqx=out:csv',
                        '&tq=', curlEscape(query),
                        '&key=', key,
                        '&gid=', gid, sep="")))
}
```

We identify the spreadsheet key. The data is from the McLaren F1 Live Dashboard collected during a race and it is Copyright (c) McLaren Marketing Ltd 2010 An SQL-like query over the spreadsheet can be used to extract the data. We send the query to Google Docs to retrieve the data, and display the result to ensure all looks okay.

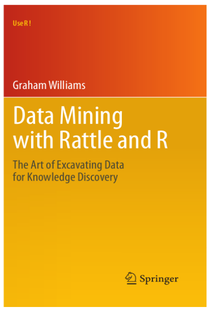
```
key='0AmbQbL4Lrd61dER5Qnl3bHo4MkVNRlZ1OVdicnZnTHc'
q <- "select *"
f1 <- gsqAPI(key, q)
dim(f1)
head(f1)
```

We save the dataset for future reference:

```
save(f1, file="f1.RData")
```

16 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.


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.

foreign (R Core Team, 2014a) provides access to various format datasets.

17 References

R Core Team (2014a). *foreign: Read Data Stored by Minitab, S, SAS, SPSS, Stata, Systat, Weka, dBase, ...* R package version 0.8-61, URL http://CRAN.R-project.org/package=foreign.

R Core Team (2014b). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Temple Lang D (2013). *RCurl: General network (HTTP/FTP/...) client interface for R*. R package version 1.95-4.1, URL http://CRAN.R-project.org/package=RCurl.

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 ReadO.Rnw revision 419, was processed by KnitR version 1.6 of 2014-05-24 and took 2.4 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 2014-06-09 10:27:52.