```
write.table(Orange, "Data\\orange.txt", sep
= "", row.names = FALSE)
```

to give the following output:

```
 "Tree" "age" "circumference"
"1" 118 30
"1" 484 58
...
```

These are some of the ways that text files can be used to import or export data to or from the R system.

GMR-2009-001: Working with Text Files in R
©2009 GM-RAM Limited
This leaflet is part of a series covering Statistical Analysis using the R Statistical Software.
http://www.gm-ram.com
http://www.wekaleamstudios.co.uk

Working with Text Files in R

Ralph Mansson

Introduction

The data used during any statistical analysis is often recorded outside of the software that is used for the analysis. The first stage of the investigation is concerned with loading the data into the statistical software, validating the data and undertaking the required analysis.

The task of reading data into a statistical software package is not always a straight forward task and there are many varied file formats that are in use by different software systems. Text files are popular for sharing small or medium sized data sets, while full blown relational databases are more appropriate for larger data sets. The R Environment has functions that handle importing data that is stored in text format and it is also possible to interact with external database systems.

The most common data format is a delimited text file and functions used to import and export data to and from the R system are described in this document.

Anatomy of a Text Data File

When working with text files for loading and saving data sets we need to consider various issues with regard to the format of these files. The questions that we need to ask include:

- Does the first row of the text file contain information about the variable (field) names?
- What symbol is used to separate fields in the text file?
- Does the text file contain row or column names?

- What symbol is used to indicate missing data values?

Once this information has been collected we can make use of functions within R to simplify the data import/export process.

Importing Data from Text Files

When working with text files for storing data there are a number of common issues that need to be considered. A special character is used to distinguish between the columns of data, e.g. a comma or a tab. There is an optional first line that provides the names of the columns (variables) or the column names can be specified explicitly when importing the data. Missing values often cause problems when handling data and a special character can be specified to indicate missing data.

Comma Separated Variable Format

The comma separated variable text file format is straightforward to handle using R with the function `read.csv` where we specify the file name as our source of data. A simple example of using this function would be:

```
read.csv("Data\\exampledata1.csv")
```

The data is loaded and assuming there are no errors converted into a data frame that can be saved and subsequently analysed. To save the data as an object we could have run this code:

```
data1 = read.csv("Data\\exampledata1.csv")
```

This function makes use of the more general purpose function read.table which accepts a wider range of options to define the delimited text file that is imported into the R environment.

Other Delimited Text Formats

The first argument supplied to this function is also the file name and, if no other options are specified, the default is assumption is that a tab separates data in different columns and that the first line of the text file does not contain column name information. The header argument to this function can be set to `TRUE` to use the information in the first row as column names. An example of specifying this option is:

```
read.table("Data\\exampledata2.txt", header
= TRUE)
```

If the data file does not have a row of column headings then the `col.names` argument can be used to specify the names that should be given to these columns. An example of using this option:

```
read.table("Data\\exampledata3.txt",
col.names = c("Weight", "Group"))
```

The special character used to separate the data on each row can also be specified by the user via the `sep` argument to the function. An example of importing a data file where a semi-colon is used is shown here:

```
read.table("Data\\exampledata4.txt", header
= TRUE, sep = ";")
```

Exporting Data to Text Files

Exporting small or medium sized data sets from the R environment to text files is a straightforward task. The two functions that are most useful for this operation are `write.csv` and `write.table` which export data to comma separate variable format or a text format with a different character used to indicate separate columns of data.

Comma Separated Variable Format

If we had a data frame in R, for example with a name `sales.data`, then the basic function call to export this data to a `csv` file would be:

```
write.csv(sales.data, "SalesData.csv")
```

Data in R can have separate row names and the default option for `write.csv` is to include row names. If there are no row names then we end up with a redundant column of sequential numbers. As an example we might have:

```
"","Month","Year","Units Sold"
"1","Jan","2009",12500
"2","Feb","2009",11750
...
```

The function can be instructed to ignore the row names by supplying an additional argument, so the function call would have been:

```
write.csv(sales.data, "SalesData.csv",
row.names = FALSE)
```

The text file would now read:

```
"Month","Year","Units Sold"
"Jan","2009",12500
"Feb","2009",11750
...
```

Other Delimited Text Formats

The comma separated variable format is not the only text file format that can be created using R and the function `write.table` can be used if a variation is required. The function call is very similar to the one used above and if we wanted to export the Anscombe data set that is available within R then this code could be used:

```
write.table(anscombe, "Data\\anscombe.txt",
row.names = FALSE)
```

This would create a text file like this:

```
"x1" "x2" "x3" "x4" "y1" "y2" "y3" "y4"
10 10 10 8 8.04 9.14 7.46 6.58
8 8 8 8 6.95 8.14 6.77 5.76
...
```

so we can see that the default option is to use a space to separate the columns of data in the output file. The character used to separate the columns can be specified with the `sep` argument to this function. An example would be: