R_Programming

Nicola Davide D'Avanzo

09/01/2015

This document concerns the statistical programming environment R.

Getting started

R is a dialect of S language, an internal statistical analysis environment initiated in 1976 and rewritten in C in 1988. R war created in 1991 and was made a free software through the GNU General Public License in 1995. R syntax is very similiar to S, while semantics are superficially similiar. R functionality is divided into modular packages. Graphics capabilities are sophisticated and better than other platforms. R contains a powerful prgramming language useful for developing new tools. Ther is a strong and active commmunity.

R is free. This makes you able to:

- run the program for any purpose;
- study how the program works and adapt it for your needs;
- redistribute copies;
- improve the program and release your improvements to the public.

Drawbacks of R:

- little built in support for 3-D graphics;
- functionality based on consumer demand;
- objects stored in physical memory;
- not ideal for all possible situations.

R system is divided into 2 conceptual parts:

- Base R, tha contains the Base Package (which includes the sub-packages *utils*, *stats*, *datasets*, *graphics*, *grDevices*, *grid*, *methods*, *tools*, *parallel*, *compiler*, *splines*, *tcltk*, *stats*);
- Everything else (expecially "Recommeded" packages *boot*, *class*, *cluster*, *codetools*, *foreign*, *KernSmooth*, *lattice*, *mgcv*, *nlme*, *rpart*, *survival*, *MASS*, *spatial*, *nnet*, *Matrix*).

In order to find answers on R you can try by:

- searching the archives of the forum;
- searching the Web;
- reading the manual;
- reading a FAQ;
- inspection and experimentation;
- asking a skilled friend (describe the gol not the step, be explicit about your question, provide the minimun amount of information necessary);
- reading the source code.

When having a problem do the right questions:

- what steps will reproduce the problem?

- what is the expected output?
- what do you see instead?
- what version of the product are you using?
- what operating system?
- additional information

R has five basic or "atomic" classes of objects:

- character;
- numeric (real numbers);
- integer (if you explicit want an integer, you need to specify the L suffix);
- complex;
- logical (True/False).

The most basic object is a vector. A vector can only contain objects of the same class. Empty vectors can be created with the *vector()* function. The one exception is a list, which is represented as a vector but can contain object of different classes. When different objects are mixed in a vector, coercion occurs so that every element in a vector is of the same class. Objects can be explicity coerced from one class to another using the *as.* function.

There is a special number *Inf* that represents infinity.

The missing values are denoted by *NA* and *NaN*. The value *NaN* represents an undefined number ("not a number", i.e. an undefined mathematical operation).

R objects can have attributes (accessible using the *attributes()* function):

- names, dimnames;
- dimensions (e.g. matrices, arrays);
- class;
- length;
- other metadata.

Matrices can be constructed using:

- *matrix()* function;
- from vectors by adding the dimension attribute with *dim()*;
- by column-binding or row-binding with *cbind()* and *rbind()*.

Factors are used to represent categorical data. One can think of a factor as an integer vector where each integer has a label. The order of the labels can be ordered by using the level attribute of *factor()* function. Factors are treated specially for modeling functions e.g. *lm()* and *glm()*.

Unlike matrices, data frames can store different classes of objects in each column.

R object can also have names, which is very useful for writing readable code and self-describing objects. List can also have names with *names()*, and also matrix with *dimnames()*.

Data type summary:

- atomic classes;
- vectors, lists;
- factors;
- missing values;

- data frames;
- names.

Reading data:

- *read.table*, *read.csv* for reading tabular data;
- *readLines*, for reading lines of a text file;
- *source*, for reading in R code files (inverse of *dump*);
- *dget*, for reading in R code files (inverse of *dput*);
- *load*, for reading in saved workspace;
- *unserialize*, for reading single R object in binary form.

Writing data:

- *write.table*
- *writeLines*
- *dump* (textual format, potentially recoverable because of preserving metadata)
- *dput* (textual format, potentially recoverable because of preserving metadata)
- *save*
- *serialize*

Reading in larger datasets:

Specifying the option *colClasses*, instead of using the default, can make *read.table* much faster, often twice as fast. In order to use this option, you have to know the class of each column in you data frame. So you can set the option *nrows* in order to set the number of rows and help the memory usage. You can use the Unix tool *wc* to calculate the number of lines in a file. It's useful to know a few things about your system:

- How much mermory is available?
- What other applications are in use?
- Are there other users logged into the same system?
- What operating system?
- Is the OS 32 or 64 bit?

In order to calculate the memory requirements of a data frame with numeric data you need to moltiplicate the number of rows and columns and 8 bytes/numeric. Then you need to divide the previous amount to 2^{30} bytes/GB.

Interfaces to the outside world:

- *file*, opens a connection to a file;
- *gzfile*, opens a connection to a file compressed in gzip;
- *bzfile*, opens a connection to a file compressed in bzip2;
- *url*, opens a connection to a webpage.