## Data Entry, Data Import and Export

The scan() function is a useful method of inputting data quickly. You can use to quickly copy and paste values into the R environment.

It is best used in the manner as described in the following example. Create a variable “X” and use the scan() function to populate it with values.

Type in a value, and then press return.

Once you have entered all the values, press return again to return to normal operation.

|  |
| --- |
| **> X=scan()**  **1: 4**  **2: 5**  **3: 5**  **4: 6**  **5:**  **Read 4 items** |

**Spreadsheet Interface**

R provides a spreadsheet interface for editing the values of existing data sets.

We use the command data.entry() , and name of the data object as the argument.

|  |
| --- |
| **> data.entry(X) # Edit the data set and exit interface**  **> X** |

## Data Import

It is necessary to import outside data into R before you start analysing it

Quite often, the sample data is in Excel format, and needs to be imported into R prior to use. For this, we use the **read.xls()** function from the “gdata” package. It reads from an Excel spreadsheet and returns a data frame. The following shows how to load an Excel spreadsheet named "mydata.xls". As the package is not in the core R library, it has to be installed and loaded into the R workspace.

|  |
| --- |
| **> library(gdata) # load the gdata package**  **> help(read.xls) # documentation**  **> mydata = read.xls("mydata.xls") # read from first sheet** |

**Table File**

A data table can resides in a text file. The cells inside the table are separated by blank characters. Here is an example of a table with 4 rows and 3 columns.

|  |
| --- |
| **100 a1 b1**  **200 a2 b2**  **300 a3 b3**  **400 a4 b4** |

Now copy and paste the table above in a file named "mydata.txt" with a text editor. Then load the data into the workspace with the **read.table()** function.

|  |
| --- |
| **> mydata = read.table("mydata.txt") # read text file**  **> mydata # print data frame**  **V1 V2 V3**  **1 100 a1 b1**  **2 200 a2 b2**  **3 300 a3 b3**  **4 400 a4 b4**  **> help(read.table)** |

**Data export**

The basic tool to produce output files is **write.table()**.

The only required argument to **write.table()**is the name of a dataset or matrix; with just a single argument, the output will be printed on the console, making it easy to test that the file you’ll be creating is in the correct format.

Usually, the second argument, **file=** will be used to specify the destination as either a character string to represent a file, or a connection (i.e. database connectivity).

By default, character strings are surrounded by quotes by **write.table()**; use the **quote=FALSE** argument to suppress this feature. To suppress row names or column names from being written to the file, use the **row.names=FALSE** or **col.names=FALSE** arguments, respectively.

Note that **col.names=TRUE** (the default) produces the same sort of headers that are read using the **header=TRUE** argument of **read.table()**.

Finally, the **sep=** argument can be used to specify a separator other than a blank space. Using **sep=’,’** (comma separated) or **sep=’\t’** (tab-separated) are two common choices.

|  |
| --- |
| **write.table(CO2 ,file=’co2.txt’, row.names=FALSE, sep=’,’)** |

>

Similarly to read.csv and read.csv2, the functions write.csv and write.csv2 are provided as wrappers to read.table, with appropriate options set to produce comma- or semicolon-separated files.

**CSV File**

The sample data can also be in comma separated values (CSV) format. Each cell inside such data file is separated by a special character, which usually is a comma, although other characters can be used as well.

The first row of the data file should contain the column names instead of the actual data. Here is a sample of the expected format.

|  |
| --- |
| **Col1,Col2,Col3**  **100,a1,b1**  **200,a2,b2**  **300,a3,b3** |

After we copy and paste the data above in a file named "mydata.csv" with a text editor, we can read the data with the read.csv function.

|  |
| --- |
| **> mydata = read.csv("mydata.csv") # read csv file**  **> mydata # print data frame**  **Col1 Col2 Col3**  **1 100 a1 b1**  **2 200 a2 b2**  **3 300 a3 b3**  **> help(read.csv)** |

In various European locales, as the comma character serves as decimal point, the read.csv2 function should be used instead.