# Input and output data. Keyboard, Monitor, CSV, XLS, XLSX, Matlab and SPSS

Alberto Gil

## Contents



# 1. Introduction

Para conseguir la persistencia del trabajo realizado en programas de R es imprescindible escribir nuestros datos en almacenamiento persistente, habitualmente el disco duro, y habitualmente en forma de fichero, la forma que tienen los sistemas operativos de almacenar la información. De forma análoga, también es imprescindible la lectura de datos de fichero para cargar datos de otros programas o equipos.

La lectura y escritura de datos en ficheros nos permitirá también compartir nuestro trabajo con otros programas o equipos. Para trabajar con datos guardados en ficheros es clave conocer la estructura de los datos para guardarlos y/o cargarlos.

## 1.1 Using data from built-int R

Esta funcionalidad se introdujo en el tema 1 para descargar o utilizar datos de paquetes:

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
data(iris)
```

### 1.2 Downloading data from the network

Vale la pena señalar que podemos descargar archivos de internet usando R con la función **download.file(url, destfile, method, quiet = FALSE, mode = "w", cacheOK = TRUE, extra = getOption("download.file.extra"), headers = NULL, ...)**

Por ejemplo, si queremos utilizar el fichero iris.data del tema 1, podríamos cargarla directamente con esta función como sigue:

```
# data(iris) # El conjunto de datos descargado es el mismo que el conjunto iris de ejemplo de R.ç

download.file(
  url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data",
  destfile = "iris.data"
  )
```

Esto guardaría el conjunto iris.data en el directorio donde lo ejecutemos con el nombre iris.data que le hemos especificado en el parámetro destfile.

## 1.3 Exercises

Descarga el siguiente conjunto de datos en formato CSV: https://www.stats.govt.nz/assets/Uploads/Injury-statistics/Injury-statistics-work-related-claims-2018/Download-data/injury-statistics-work-related-claims-2018-csv.csv

```
download.file(
  url = "https://www.stats.govt.nz/assets/Uploads/Serious-injury-outcome-indicators/Serious-injury-outc
  destfile = "injury-statistics.csv"
  )
```

# 2. Importing data from a file to R

**Reminder: the working directory**

El directorio de trabajo en el que ejecutamos nuestro programa es clave para leer o escribir ficheros especificados de forma relativa.

```
getwd() # conocer el path de ejecución del programa
```

```
## [1] "/home/alberto/repos/R_course_ceindo/4_InputOutput"
```

```
#setwd("C:\\") # to set the new working Directory to the root in windows.
#setwd("/home/alberto") # to set the new working Directory to the root in windows.
```

## 2.1 Generic data import

### Graphical import

R Studio proporciona una interfaz gráfica para importar conjuntos de datos. Veámos esta interfaz con varios ejemplos.

### Command line import

R proporciona una función para leer cualquier tipo de archivo que contenga una estructura de tipo tabla: read.table()

El único requisito que tiene esta función es que los datos estén separados por un salto de línea y que en cada línea los datos estén separados por algún caracter o grupo de caracteres concreto que indican las columnas. Estos separadores varían en función del idioma del sistema operativo por defecto en excel. Los más habituales son ",", ";", ":", "\t también conocido como tabulador,"|".

Los siguientes datos representan un fichero csv separado por "," que se encuentran en example.csv:

1, 2, 3, 4 5, 6, 7, 8 9, 10, 11, 12 13, 14, 15, 16

La función genérica **read.table()** acepta un número de argumentos muy alto, pero el único obligatorio es el parámetro file que indica la ruta del fichero. Como siempre, podéis ver los parámetros:

```
?read.table()
```

Podemos descargar el siguiente fichero sobre estadísticas de cáncer de mama de la universidad de Wisconsin: https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer-wisconsin.data

```
download.file(
  url = "https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer
  destfile = "breast-cancer.data"
  )
```

Podemos utilizar la función read.table() para leer este fichero y guardarlo en una variable bcancer:

```
bcancer = read.table(file = "breast-cancer.data")
head(bcancer) #Veamos las primeras 6 líneas
```

```
##                              V1
## 1    1000025,5,1,1,1,2,1,3,1,1,2
## 2   1002945,5,4,4,5,7,10,3,2,1,2
## 3    1015425,3,1,1,1,2,2,3,1,1,2
## 4    1016277,6,8,8,1,3,4,3,7,1,2
## 5    1017023,4,1,1,3,2,1,3,1,1,2
## 6 1017122,8,10,10,8,7,10,9,7,1,4
```

Nuestros datos no están muy bien formateados. No hay datos de encabezados (header), y no hemos especificado el separador de columnas, por lo que ha cargado los datos en una única columna. Repitamos la acción para tener diferentes columnas:

```
bcancer = read.table(file = "breast-cancer.data", header = FALSE, sep = ",")
head(bcancer) #Veamos las primeras 6 líneas
```

```
##        V1 V2 V3 V4 V5 V6 V7 V8 V9 V10 V11
## 1 1000025  5  1  1  1  2  1  3  1   1   2
## 2 1002945  5  4  4  5  7 10  3  2   1   2
## 3 1015425  3  1  1  1  2  2  3  1   1   2
## 4 1016277  6  8  8  1  3  4  3  7   1   2
## 5 1017023  4  1  1  3  2  1  3  1   1   2
```

```
## 6 1017122  8 10 10  8  7 10  9  7   1   4
```

Este nombre de columnas V1, V2, . . . , Vn no parece el más descriptivo. Por suerte, tenemos un fichero que nos facilita los nombres de las columnas.

```
download.file(
  url = "https://archive.ics.uci.edu/ml/machine-learning-databases/breast-cancer-wisconsin/breast-cancer
  destfile = "breast-cancer.names"
  )
```

De donde podemos obtener las abreviaturas de los nombres de cada columna:

```
header_names = c("id", "clump_t", "u_csize", "u_cshape", "m_adh", "spcs", "b_nuc",
             "b_chr", "n_nuc", "mit", "class")
```

Y podemos utilizar el vector para el nombre la columna:

```
bcancer = read.table(file = "breast-cancer.data", header = FALSE, sep = ",", col.names = header_names)
head(bcancer) #Veamos las primeras 6 líneas
```

```
##        id clump_t u_csize u_cshape m_adh spcs b_nuc b_chr n_nuc mit class
## 1 1000025       5       1        1     1    2     1     3     1   1     2
## 2 1002945       5       4        4     5    7    10     3     2   1     2
## 3 1015425       3       1        1     1    2     2     3     1   1     2
## 4 1016277       6       8        8     1    3     4     3     7   1     2
## 5 1017023       4       1        1     3    2     1     3     1   1     2
## 6 1017122       8      10       10     8    7    10     9     7   1     4
```

```
class(bcancer) #Veamos el tipo de dato importado
```

```
## [1] "data.frame"
```

La función read.table() siempre devuelve un tipo de dato dataframe.

Si no sabemos qué tipo de dato tenemos, podemos utilizar la función readLines para obtener las n primeras líneas del fichero:

```
dont_know_what_is = readLines("breast-cancer.data", n = 6)
dont_know_what_is # AQUI PODEMOS VER LAS 6 PRIMERAS lÍNEAS COMO STRING Y VER QUE EL SEPARADOR ES UNA ",
```

```
## [1] "1000025,5,1,1,1,2,1,3,1,1,2"    "1002945,5,4,4,5,7,10,3,2,1,2"
## [3] "1015425,3,1,1,1,2,2,3,1,1,2"    "1016277,6,8,8,1,3,4,3,7,1,2"
## [5] "1017023,4,1,1,3,2,1,3,1,1,2"    "1017122,8,10,10,8,7,10,9,7,1,4"
```

## 2.2 CSV data import

Un caso particular de tablas es el formato estándar con extensión **.csv** (comma separated values). Este tipo de ficheros se utiliza muy a menudo, pues permite guardar en formato texto (no binario) datos y ser compartidos por diferentes programas, facilitando la comptabilidad de los datos, y ocupando poco tamaño. Este tipo de archivos también puede ser leído con la función read.table, pero R facilita dos funciones específicas para los ficheros **.csv**: **read.csv() y read.csv2()**. Estas dos funciones son casi iguales, pero tienen un valor por defecto para el parámetro **sep** y **dec** diferente. El parámetro **sep** indica el separador de columnas, mientras que el parámetro **dec** indica el separador de números decimales, que en formato europeo es la "," y en formato anglosajón es el ".". Podemos resumir de forma práctica que se puede utilizar **read.csv()** para leer ficheros csv en formato anglosajón, y **read.csv2()** para leer datos en formato europeo.

Ambas funciones son una optimización de la función read.table para formatos **.csv()**

```
header_names = c("id", "clump_t", "u_csize", "u_cshape", "m_adh", "spcs", "b_nuc",
             "b_chr", "n_nuc", "mit", "class")
```

```
bcancer = read.csv(file = "breast-cancer.data", header = FALSE, sep = ",", col.names = header_names) #
head(bcancer) #Veamos las primeras 6 líneas
```

```
##        id clump_t u_csize u_cshape m_adh spcs b_nuc b_chr n_nuc mit class
## 1 1000025       5       1        1     1    2     1     3     1   1     2
## 2 1002945       5       4        4     5    7    10     3     2   1     2
## 3 1015425       3       1        1     1    2     2     3     1   1     2
## 4 1016277       6       8        8     1    3     4     3     7   1     2
## 5 1017023       4       1        1     3    2     1     3     1   1     2
## 6 1017122       8      10       10     8    7    10     9     7   1     4
```

```
class(bcancer) #Veamos el tipo de dato importado
```

```
## [1] "data.frame"
```

Esta cabecera ahora puede añadirse al fichero, podéis descargarlo de los archivos de Teams y se llama "breast-cancer_with_header.data"

```
# Los datos de la cabecera, la primera línea, formarán el conjunto de nombres de las columnas
bcancer = read.csv(file = "breast-cancer_with_header.data", sep = ",") # Ya podemos dejar el valor por
head(bcancer) #Veamos las primeras 6 líneas
```

```
##        id clump_t u_csize u_cshape m_adh spcs b_nuc b_chr n_nuc mit class
## 1 1000025       5       1        1     1    2     1     3     1   1     2
## 2 1002945       5       4        4     5    7    10     3     2   1     2
## 3 1015425       3       1        1     1    2     2     3     1   1     2
## 4 1016277       6       8        8     1    3     4     3     7   1     2
## 5 1017023       4       1        1     3    2     1     3     1   1     2
## 6 1017122       8      10       10     8    7    10     9     7   1     4
```

```
class(bcancer) #Veamos el tipo de dato importado
```

```
## [1] "data.frame"
```

## 2.3 Excel data import

Existen diferentes paquetes de R que permiten leer ficheros con formato xls o xlsx (excel) directamente, sin hacer un cambio de formato a CSV. Uno de los más conocidos es xlsx, así que vamos a proceder a instalarlo y leer el mismo fichero csv en formato xls y xlsx.

```
# UNCOMMENT AND EXECUTE THIS LINE ONLY IF YOU ARE WORKING ON WINDOWS!
# install.packages("rJava", type="win.binary") # install the sources from binary for Windows
install.packages("xlsx") # install the package from CRAN
```

```
## Installing package into '/home/alberto/R/x86_64-pc-linux-gnu-library/4.3'
## (as 'lib' is unspecified)
```

```
library("xlsx") # Load the package from CRAN
```

Este paquete tiene dos funciones principales para cargar archivos de excel: **read.xlsx(file, sheetIndex, header = TRUE)** y **read.xlsx2(file, sheetIndex, header = TRUE)**. La segunda de ellas es más rápida para lectura de ficheros más grandes. Ambas tienen parámetros como la fila de inicio, la fila final, el tipo de datos de cada columna, o la posibilidad de incluir o no cabeceras. Se recomienda revisar la documentación como siempre con ?read.xlsx Leamos el mismo fichero con datos de cáncer de mama de la universidad de Wisconsin:

```
bcancer = read.xlsx(file="breast-cancer.xlsx", sheetIndex=1, header=FALSE) #SIN CABECERA
head(bcancer)
```

```
##        X1 X2 X3 X4 X5 X6 X7 X8 X9 X10 X11
## 1 1000025  5  1  1  1  2  1  3  1   1   2
## 2 1002945  5  4  4  5  7 10  3  2   1   2
## 3 1015425  3  1  1  1  2  2  3  1   1   2
## 4 1016277  6  8  8  1  3  4  3  7   1   2
## 5 1017023  4  1  1  3  2  1  3  1   1   2
## 6 1017122  8 10 10  8  7 10  9  7   1   4
```

```
class(bcancer)
```

```
## [1] "data.frame"
```

Ahora, leamos el mismo fichero con cabecera

```
bcancer = read.xlsx(file="breast-cancer_with_header.xlsx", sheetIndex=1, header=TRUE) #CON CABECERA
head(bcancer)
```

```
##        id clump_t u_csize u_cshape m_adh spcs b_nuc b_chr n_nuc mit class
## 1 1000025       5       1        1     1    2     1     3     1   1     2
## 2 1002945       5       4        4     5    7    10     3     2   1     2
## 3 1015425       3       1        1     1    2     2     3     1   1     2
## 4 1016277       6       8        8     1    3     4     3     7   1     2
## 5 1017023       4       1        1     3    2     1     3     1   1     2
## 6 1017122       8      10       10     8    7    10     9     7   1     4
```

```
class(bcancer)
```

```
## [1] "data.frame"
```

Si intentamos leer las primeras líneas del fichero, vemos que no es un fichero de texto y no tiene forma rectangular.

```
#readLines(con = "breast-cancer.xlsx", n = 5)
```

Y, una vez leído el fichero xls o xlsx, ya tenemos el data frame correspondiente para trabajar con el.

## 2.4 SPSS data import

Para la lectura de datos procedentes de programas de estadística tenemos que recurrir a librerías de R. Una de las más utilizadas es SPSS, y para leer datos procedentes de SPSS se puede utilizar el paquete haven

```
install.packages("haven")
```

```
## Installing package into '/home/alberto/R/x86_64-pc-linux-gnu-library/4.3'
## (as 'lib' is unspecified)
```

```
library(haven)
```

Tras esto, ya podemos leer ficheros provenientes de spss con la función **read_sav()** o **read_por()** si se utiliza el formato antiguo de SPSS. Veamos un ejemplo de un dataset de coches.

```
library(haven)
data_frame_cars = read_sav("mtcars.sav")
head(data_frame_cars)
```

```
## # A tibble: 6 x 11
##     mpg   cyl  disp    hp  drat    wt  qsec    vs    am  gear  carb
##   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
## 1  21       6   160   110  3.9   2.62  16.5     0     1     4     4
## 2  21       6   160   110  3.9   2.88  17.0     0     1     4     4
## 3  22.8     4   108    93  3.85  2.32  18.6     1     1     4     1
```

```
## 4  21.4     6   258   110  3.08  3.22  19.4     1     0     3     1
## 5  18.7     8   360   175  3.15  3.44  17.0     0     0     3     2
## 6  18.1     6   225   105  2.76  3.46  20.2     1     0     3     1
```

```
class(data_frame_cars)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

Y, una vez leído el fichero proveniente de spss (.sav o el viejo formato llamado .por), ya tenemos el data frame correspondiente para trabajar con él.

## 2.5 Matlab data import

De forma análoga a la lectura de datos en otros formatos, existe un paquete de R que nos permite

```
install.packages("R.matlab")
```

```
## Installing package into '/home/alberto/R/x86_64-pc-linux-gnu-library/4.3'
## (as 'lib' is unspecified)
```

```
library(R.matlab)
```

```
## R.matlab v3.7.0 (2022-08-25 21:52:34 UTC) successfully loaded. See ?R.matlab for help.
```

```
##
## Attaching package: 'R.matlab'
```

```
## The following objects are masked from 'package:base':
##
##     getOption, isOpen
```

Y este contiene una función readMat() para leer ficheros en formato matlab

```
library(R.matlab)
data_frame_cars2 = readMat("mtcars.mat")
head(data_frame_cars2)
```

```
## $data.frame.cars
## , , 1
##
##      [,1]
## mpg  numeric,32
## cyl  numeric,32
## disp numeric,32
## hp   numeric,32
## drat numeric,32
## wt   numeric,32
## qsec numeric,32
## vs   numeric,32
## am   numeric,32
## gear numeric,32
## carb numeric,32
```

```
class(data_frame_cars2)
```

```
## [1] "list"
```

# 3. Exporting data to a file

Una de las estructuras más comunes para el análisis de datos es la estructura en filas y columnas, comúnmente conocido como tabla. R cuenta con diversas funciones para escribir en este tipo de archivos y, como podéis esperar, vamos a utilizar los mismos paquetes de lectura de cada tipo de dato para realizar la escritura correspondiente. También se pueden escribir tipos de datos diferentes como listas u objetos, tal y como veremos en la última sección.

En esta sección, vamos a proceder a escribir el mismo dataset de datos en diferentes formatos, que posteriormente procederemos a leer en la sección de ejercicios. El conjunto de datos elegido es iris, con el que ya os habéis familiarizado en los temas anteriores.

## 3.1 Generic data export

Si nuestros datos se encuentran en una estructura de datos rectangular, como pueden ser las **matrices o los data frames** podemos utilizar la versión análoga a la función genérica para leer archivos de texto **read.table()**, que se llama **write.table()**

Los parámetros más usados de esta función son:

- x (obligatorio): nombre de la matriz o data frame a exportar.
- file: (opcional, default: " " o consola): nombre del fichero donde se van a guardar los datos.
- sep: caracter de separación de columnas.
- row.names: si queremos guardar los nombres de filas. Habitualmente no se guardan y el parámetro toma el valor FALSE
- row.columns: si queremos guardar los nombres de columnas. Habitualmente se recomienda fijarlo a TRUE para tener esta información.

```
write.table(x = iris, file = "iris.csv", sep = ",",
            row.names = FALSE, col.names = TRUE) # Elijamos como separador la ,
```

## 3.2 CSV Data export

Podemos utilizar la función específica para guardar ficheros .csv.

```
write.csv(x = iris, file = "iris2.csv", row.names = FALSE) #Por defecto toma el nombre de las columnas
```

## 3.3 XLS/XLSX Data export

Y la función análoga para guardar ficheros en formato xls o xlsx.

```
library(readxl)
write.xlsx(x = iris, file = "iris.xlsx", row.names = FALSE)
```

## 3.4 SPSS Data export

Ahora lo guardamos en formato SPSS

```
library(haven)
write_sav(iris, "iris.sav")
```

## 3.5 MatLab Data export

Y, por último, en formato MatLab:

```
library(R.matlab)
writeMat("iris.mat", iris = iris) # el parámetro se debe llamar igual al nombre del objeto para guardar
```

### 3.6 Exercises

Lee en cada uno de los formatos correspondientes los ficheros escritos del dataset Iris en data frames diferentes (**CSV, XLSX, SAV Y MAT**). Utiliza como nombre de variable iris_ donde TIPODEFICHEROLEIDO debe modificarse por cada tipo de datos.

Posteriormente, vamos a aplicar un filtro sobre el conjunto de datos iris y escribirlo en los diferentes formatos. El filtro a utilizar será que la especie sea versicolor y que Petal.Width esté entre 1.0 y 1.3 incluidos estos valores o que la especia sea setosa y la longitud del pétalo (Petal.Length) esté entre 1.5 y 2.0 excluidos estos valores. Se recomienda crear un único subconjunto para luego escribirlo en todos los formatos correspondientes.

Recuerda cargar el paquete **library(dplyr)** para aplicar filtros. El resultado deben ser 41 filas.

```
iris_csv = read.csv(file = "iris.csv", sep = ",", )
head(iris_csv) #Veamos las primeras 6 líneas
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
iris_xlsx = read.xlsx(file = "iris.xlsx", sheetIndex=1, header=TRUE )
head(iris_xlsx) #Veamos las primeras 6 líneas
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
iris_sav = read_sav(file = "iris.sav")
head(iris_sav) #Veamos las primeras 6 líneas
```

```
## # A tibble: 6 x 5
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##          <dbl>       <dbl>        <dbl>       <dbl> <dbl+lbl>
## 1          5.1         3.5          1.4         0.2 1 [setosa]
## 2          4.9         3            1.4         0.2 1 [setosa]
## 3          4.7         3.2          1.3         0.2 1 [setosa]
## 4          4.6         3.1          1.5         0.2 1 [setosa]
## 5          5           3.6          1.4         0.2 1 [setosa]
## 6          5.4         3.9          1.7         0.4 1 [setosa]
```

```
iris_mat = readMat("iris.mat")
head(iris_mat) #Veamos las primeras 6 líneas
```

```
## $iris
## , , 1
##
##              [,1]
## Sepal.Length numeric,150
## Sepal.Width  numeric,150
## Petal.Length numeric,150
```

```
## Petal.Width  numeric,150
## Species      list,150
```

**data subset generation**

```
iris_filtered = filter(iris_csv, (Species == "versicolor" & Petal.Width >= 1.0 & Petal.Width <= 1.3) |
#  & Petal.Width >= 1.0 & Petal.Width <= 1.3) | (Species == "setosa" & Petal.Length > 1.5 & Petal.Lengt
```

**file writing**

```
write.csv(x = iris_filtered, file = "iris_subset.csv", row.names = FALSE)
```

```
write.xlsx(x = iris_filtered, file = "iris_subset.xlsx", row.names = FALSE)
```

```
write_sav(iris_filtered, "iris_subset.sav")
```

```
writeMat("iris_subset.mat", iris_filtered = iris_filtered)
```