Lectura y Escritura de datos

David García Callejas

La estructura de datos más apropiada para cargar datos en R son, generalmente, los dataframes o las matrices. Por supuesto, R acepta muchos tipos de archivos especiales (por ejemplo, archivos raster o vectoriales para funciones SIG), pero por ahora nos centraremos en cargar y escribir datos en formato tabla.

Antes de empezar, una recomendación: utilizad en la medida de lo posible formatos abiertos para guardar tablas de datos. Por ejemplo, csv o txt. Los archivos de programas propietarios (e.g. excel) dependen de compañías privadas, que pueden dejar de dar soporte a sus programas en cualquier momento.

De manera similar, intentad almacenar los metadatos en archivos txt separados de los datos en sí. Documentar los datos en una hoja de excel es enormemente incómodo para trabajar de manera eficiente. Por ejemplo, importar datos de áreas protegidas terrestres para trabajar con ellos en R es imposible sin una limpieza previa. Esto impide la transferabilidad y la reproducibilidad de los análisis. Comparad ese conjunto de datos con los datos en terremotos ("earthquake_data").

Comenzaremos por repasar las funciones básicas de lectura y escritura de datos en R.

```
?read.table
?write.table
```

read.table es la función general para leer archivos de texto en tablas. En la función, el primer argumento que se especifica es la ruta del archivo a leer. Otros argumentos importantes son "header", que indica si el archivo incluye nombre de columnas; "sep", que indica el carácter que sirve de separación entre valores, o "dec", que indica el carácter que sirve como punto decimal.

Esta y otras funciones asociadas devuelven un dataframe al leer un archivo. En la carpeta "data" hay dos archivos exactamente iguales con extensión .csv, uno separado por comas y otro separado por punto y coma, que nos pueden servir de primer ejemplo.

```
mi.archivo <- read.table(file = "../data/starwars_names.csv",
                         header = TRUE,
                         sep = ";")

head(mi.archivo)
str(mi.archivo)
```

Existen funciones pensadas para facilitar (un poco) la vida, que tienen opciones por defecto para dos tipos de caracteres de separación. Para leer directamente un archivo csv separado por comas, podemos usar la funcion read.csv

```
a2 <- read.csv(file = "../data/starwars_names_coma.csv")
```

y para leer csv separado por punto y coma, la función read.csv2

```
a3 <- read.csv2(file = "../data/starwars_names.csv")
```

Estas funciones también asignan valores por defecto al carácter de separación decimal, tened cuidado.

R también permite leer hojas de excel como dataframes. Para ello necesitamos cargar un *paquete* externo. Un paquete, como vimos, no es más que una colección de funciones que no están incluidas por defecto en

nuestra instalación de R. En este caso, por defecto R no trae funciones para leer archivos de excel, pero la comunidad ha desarrollado estas funciones y nosotros podemos cargarlas y usarlas.

```
# la orden "library" se usa para cargar un paquete determinado
library(readxl)
hoja1 <- read_excel(path = "../data/hojas_excel.xlsx",sheet = "parcela_1")
```

Existen funciones específicas que sirven para leer de manera más eficiente archivos csv muy grandes, de miles o millones de filas. Nosotros no trabajaremos con archivos tan grandes, pero estas funciones están disponibles en el paquete "readr", y son practicamente iguales a las funciones por defecto de R.

```
library(readr)
?read_csv2
```

¿Cómo sabemos la ruta hacia un archivo? Si os habéis fijado, yo he usado rutas relativas. Para entender esto necesitamos saber lo que es el directorio de trabajo en R.

```
getwd()
```

Una sesión de R siempre trabaja teniendo una carpeta (o directorio) como base. Esta carpeta es aquella en la que, si no especificamos otra cosa, se guardan o se leen los datos que nos interesen. Al abrir una sesión nueva de R/Rstudio, nuestro directorio de trabajo será el que R asigna por defecto. en Linux, el directorio base del usuario. ¿y en Windows? abrid una sesión nueva en Rstudio y llamad a la función getwd.

Este directorio de trabajo se puede modificar con la función setwd

```
?setwd
```

En todo caso, si conocemos nuestro directorio de trabajo, podemos usar rutas relativas para leer nuestros archivos. Los dos puntos de más arriba "../" suben un nivel en la estructura de carpetas. Desde ese nivel, dentro de la carpeta "data", ya encuentro, en mi equipo, los archivos que me interesan. Para ayudar a la búsqueda, igual que con variables/funciones, puedo usar el tabulador.

Las funciones de lectura tienen sus equivalentes funciones de escritura, write.table y las asociadas. En este caso tenemos que especificar 1) el dataframe/matriz que queremos escribir, y 2) la ruta y el nombre con el que queremos guardar el archivo.

```
?write.table

a2$name[1] <- "Lucas"

# guardar archivo csv separado por punto y coma
# write.table es la función general
# podemos especificar si queremos una columna que guarde
# los nombres de las filas de nuestro dataframe

# estas dos órdenes son equivalentes
# write.table(x = a2,file = "../data/archivo_nuevo.csv",
#             sep = ";",
#             row.names = FALSE)
# write.csv2(x = a2,file = "../data/archivo_nuevo.csv",
#            row.names = FALSE)

# y para guardar un archivo csv separado por comas, usamos
# write.csv(x = a2, file = "../data/archivo_nuevo_comas.csv",
#           row.names = FALSE)
```

Excel puede leer perfectamente los archivos .csv, y son mucho más fáciles de compartir entre sistemas operativos, programas de hojas de cálculo y otro software. Os recomiendo guardar *siempre* vuestras tablas en

formato .csv (o .txt). En todo caso, existen paquetes de R que permiten guardar archivos directamente en formato excel.

```
# install.packages("xlsx")
# library(xlsx)
# ?write_xlsx
```

Una opción interesante de las funciones de escritura por defecto en R es el argumento "append". Si este argumento está activado (TRUE), R busca el nombre del archivo y en vez de sobreescribirlo completo, añade los datos que queramos escribir al final del archivo ya existente.

Otra función, que nos permite leer archivos línea a línea es `readLines`

```
my.file <- "../data/sample500tuits_starwarsandaluz.txt"
conn <- file(my.file,open = "r")
lineas <-readLines(conn)
```

En este caso, el archivo se lee como un vector de caracteres, donde cada línea es un elemento.