Ficheros

**Introducción**

El directorio de trabajo es la ubicación en tu sistema donde R buscará o guardará archivos.

> getwd()

[1] "C:/Users/azugadiz/Documents"

Para cambiar el directorio de trabajos existen dos formas:

* La función **setwd**(“nueva\_ruta\_del\_directorio\_de\_trabajo”)
* Session🡪 Set Working Directory 🡪 Choose Directory

Archivos de texto (read.table):

**Sintaxis básica**

**read.table**(file, # Archivo de datos TXT indicado como string o ruta com-pleta al archivo

 header = **FALSE**, # Si se muestra el encabezado (TRUE) o no (FALSE)

 sep = "", # Separador de las columnas del archivo

 dec = "." # Caracter utilizado para separar decimales de los números en el archivo

 )

Considerar, como ejemplo, que tienes un archivo llamado mi\_archivo.txt y que lo has guardado en tu directorio de trabajo. Puedes leerlo con el siguiente código, si quieres mostrar también el encabezado (nombres de las columnas).

dataframe <- **read.table**(file = **"mi\_archivo.txt"**, header = TRUE)

**head**(dataframe)

En caso de que tengas el archivo en **otro directorio** que no sea su directorio de trabajo, deberás especificar la **ruta completa** donde se encuentra el archivo de datos.



**Saltar filas con el argumento skip**

A veces, los archivos TXT que estás leyendo contienen algunas líneas de texto antes del conjunto de datos. Para ese propósito, puedes usar el argumento **skip, que por defecto está establecido en 0**.

Como ejemplo, en caso de que haya 5 líneas de texto antes de tus datos, puedes leer el archivo de la siguiente manera:

**read.table**(file = **"mi\_archivo.txt"**, skip = 5)

**Valores NULOS**

Si tu archivo de datos TXT contiene valores NULL, puedes establecer el argumento **skipnul** como TRUE para **obviarlos**.

**read.table**(file = **"C:\\My\_path\\mi\_archivo.txt"**, skipnul = TRUE)

**Importar TXT desde una URL**

En caso de que tengas un archivo TXT alojado en algún sitio web, puedes abrirlo sin descargarlo. Solo necesita pasar la URL como cadena al primer argumento de la función.

url <- **"http://courses.washington.edu/b517/Datasets/string.txt"**

datos <- **read.table**(url, header = TRUE)

**Descargar TXT en R**

Ahora que ya sabes cómo leer un TXT en R, debes tener en cuenta que puedes descargar directamente un archivo TXT a tu directorio de trabajo con la función **download.file**, pasando como primer argumento el enlace y como segundo el nombre que le quieras poner a los datos de tipo .txt.

**getwd**() *# Directorio donde se guardará el archivo*

url <- **"http://courses.washington.edu/b517/Datasets/string.txt"**

**download.file**(url, **"mi\_archivo.txt"**)

Si no quieres que el archivo se guarde en el directorio de trabajo actual, puedes especificar la ruta donde quieras que se descargue el archivo.

**download.file**(url, **"C:\\carpeta\\mi\_archivo.txt"**)

Probar con:

<https://ozonewatch.gsfc.nasa.gov/data/omps/Y2012/OMPS-NPP_NMTO3-L3-DAILY-Ozone-ASCII_v2.1_2012m0126_2017m0227t060849.txt>

Archivos csv (read.csv y read.csv2)

**Sintaxis básica**

*# Por defecto coma (,) como separador y punto (.) como separador decimal*

**read.csv**(file, *# Nombre del archivo o ruta completa del archivo*

header = TRUE, *# Leer el encabezado (TRUE) o no (FALSE)*

sep = **","**, *# Separador de los valores*

quote = **"\""**, *# Caracter de citaciones*

dec = **"."**, *# Punto decimal*

fill = TRUE, *# Rellenar celdas vacías (TRUE) o no (FALSE)*

comment.char = **""**, *# Carácter de los comentarios o cadenas vacías*

encoding = **"unknown"**, *# Codificación del archivo*

...) *# Argumentos adicionales*

*# Por defecto punto y coma (;) como separador y coma (,) como separador decimal*

**read.csv2**(file,

header = TRUE,

sep = **";"**,

quote = **"\""**,

dec = **","**,

fill = TRUE,

comment.char = "",

encoding = **"unknown",** ...)

Es posible que hayas notado que la única **diferencia** entre las funciones es el **separador de valores y el separador decimal,** debido a que en algunos países usan comas como separador decimal.

En este **segundo** caso, para crear archivos CSV se necesita el **punto y coma** y, por tanto, para importar archivos con decimales, es necesario cambiar los argumentos por defecto de la función read.csv, o usar directamente la función read.csv2.

Para cargar un archivo CSV en R con los argumentos predeterminados puedes pasar el archivo como cadena de caracteres a la función correspondiente. **La salida será de clase data.frame**

**read.csv**(**"mi\_archivo.csv"**)

Si tan solo ejecutas el código anterior imprimirás el data frame, pero **no se almacenará en memoria**, ya que no lo estás asignado a ninguna variable. Si lo guardas, por ejemplo, en un variable llamada mi\_archivo, podrás acceder a las variables o a los datos que quieras.

mi\_archivo <- **read.csv**(**"mi\_archivo.csv"**)

**El argumento na.strings**

Algunas veces los archivos contienen alguna cadena de caracteres que representan los **valores faltantes u omitidos**. Para resolver este problema, puedes **convertirlos a valores NA** con el argumento na.strings, especificando la cadena de caracteres que representa el valor faltante.

Si, por ejemplo, en nuestro archivo los valores -9999 representan valor omitidos o faltantes podemos escribir:

**read.csv**(**"mi\_archivo.csv"**, na.strings = **"-9999"**)

Además, en caso de que el archivo contenga múltiples na.strings, puedes especificar todo dentro de un vector.

**read.csv**(**"mi\_archivo.csv"**, na.strings = **c**(**"-9999"** , **"Na"** ))

Sin embargo, si necesitas eliminar los valores NA después de abrir el CSV, deberás usar la función que corresponda según tus datos. La función más habitual es **na.omit**.

**El argumento stringsAsFactors**

El argumento **stringsAsFactors** transformará las columnas de tipo carácter del **conjunto de datos en factores**.

**read.csv**(**"mi\_archivo.csv"**, stringsAsFactos = TRUE)

**Leer múltiples CSV en R**

Para finalizar, cabe destacar que es posible **importar múltiples archivos CSV** al mismo tiempo en lugar de cargarlos en R uno por uno. Para ese propósito, puedes usar la función **list.files** para buscar todos los archivos CSV y luego leerlos aplicando la función read.csv (o read.csv2) con la función sapply.

archivos <- **list.files**(pattern = **"\*.csv"**)

multiples\_csv <- **sapply**(archivos, read.csv)

Archivos EXCEL

Para trabajar con datos de Excel en R deberás usar un paquete desarrollado para tal fin. Hay varias opciones, pero los mejores paquetes para leer archivos de Excel podrían ser **openxlsx** y **readxl**

**Importar datos de Excel en RStudio desde el menú**

Si estás usando RStudio puedes ir a **File → Import Dataset → From Excel....** Luego, selecciona tu archivo de Excel dándole a Browse... y personaliza la salida (el nombre de la variable, la hoja, el rango de celdas, …). También puedes ver una vista previa del código que se ejecutará de fondo y de los datos que se cargarán:





**Ten en cuenta que para utilizar este procedimiento necesitas tener instalado el paquete readxl.**

Leer XLSX sin JAVA en R: readxl y openxlsx

**El paquete readxl**

El paquete **readxl** forma parte del paquete **tidyverse**, creado por Hadley Wickham (científico jefe en RStudio) y su equipo.

El paquete proporciona algunos archivos Excel (XLS y XLSX) de muestra almacenados en la carpeta de instalación del paquete, por lo que con el objetivo de ofrecer un ejemplo reproducible, en los siguientes ejemplos vamos a utilizar el archivo clippy.xlsx, cuya primera hoja es la siguiente:

Para cargar la ruta del archivo Excel de muestra, puedes utilizar la función **readxl\_example**. Una vez cargado, o una vez que tengas la ruta de tu propio archivo Excel, puedes usar la función **excel\_sheets** para obtener los nombres de las hojas del archivo Excel, en caso de que lo necesites.

*# install.packages("readxl")*

**library**(readxl)

*# Obtener la ruta de un archivo XLSX de ejemplo del paquete*

ruta\_archivo <- **readxl\_example**(**"clippy.xlsx"**)

*# Comprobar los nombres de las hojas del archivo*

**excel\_sheets**(ruta\_archivo) *# "list-column" "two-row-header"*

La función genérica del paquete para leer archivos de Excel en R es la función **read\_excel**, que ‘adivina’ el tipo de archivo (XLS o XLSX) según la extensión del archivo y el archivo en sí, en ese orden.

**read\_excel**(ruta\_archivo)

**Output**

*# A tibble: 4 x 2*

name value

<chr> <chr>

1 Name Clippy

2 Species paperclip

3 Approx date of death 39083

4 Weight **in** grams 0.9 7

El argumento sheet permite **especificar la hoja que quieres cargar**, pasando su nombre o el número corres-pondiente de la pestaña. Ten en cuenta que, por defecto, la función carga la primera hoja de Excel.

*# Seleccionamos la otra hoja del Excel*

**read\_excel**(ruta\_archivo, sheet = **"two-row-header"**)

**read\_excel**(ruta\_archivo, sheet = 2) *# Equivalente*

**Output**

*# A tibble: 2 x 4*

name species death weight

<chr> <chr> <chr> <chr>

1 (at birth) (office supply type) (date is approximate) (**in** grams)

2 Clippy paperclip 39083 0.9

También puedes saltar filas con el argumento skip de la función:

*# Saltar la primera fila*

**read\_excel**(ruta\_archivo, skip = 1)

**Output**

*# A tibble: 3 x 2*

Name Clippy

<chr> <chr>

1 Species paperclip

2 Approx date of death 39083

3 Weight **in** grams 0.9

Ten en cuenta que también puedes especificar un rango de celdas con el argumento **range**. En este caso, el argumento **skip** no se tendrá en cuenta si lo especificas.

**read\_excel**(ruta\_archivo, range = **"B1:B5"**)

**Output**

*# A tibble: 4 x 1*

value

<chr>

1 Clippy

2 paperclip

3 39083

4 0.9

Además, si quieres evitar leer los nombres de las columnas, puedes establecer el argumento **col\_names** como FALSE:

**read\_excel**(ruta\_archivo, col\_names = FALSE)

**Output**

New names:

\* `` -> ...1

\* `` -> ...2

...1 ...2

1 name value

2 Name Clippy

3 Species paperclip

4 Approx date of death 39083

5 Weight **in** grams 0.9

Sin embargo, es posible que hayas notado que la salida es de clase **tibble** (un tipo moderno de data frame). Si quieres que la salida sea de clase **data.frame**, deberás usar la función as.data.frame de la siguiente manera:

data <- **read\_excel**(ruta\_archivo, skip = 1)

**as.data.frame**(data)

**Output**

Name Clippy

1 Species paperclip

2 Approx date of death 39083

3 Weight **in** grams 0.9

Recuerda que la función read\_excel ‘adivina’ la extensión de archivo. No obstante, si conoces la extensión del archivo que vas a leer, puedes usar la función correspondiente de las siguientes para evitar ese proceso de ‘adivinación’:

*# Si conoces la extensión del archivo*

*# usa una de estas dos funciones*

*# Para archivos XLS*

**read\_xls**()

*# Para archivos XLSX*

**read\_xlsx**()

**El paquete openxlsx**

El paquete **openxlsx** usa Rcpp y, como no depende de JAVA, es una alternativa interesante al pa-quete readxl para leer un archivo Excel en R. Las diferencias respecto al paquete anterior son que la **salida es de clase** data.frame por defecto en lugar de tibble y que su uso principal no es solo la importación de archi-vos de Excel, ya que **también proporciona una amplia variedad de funciones para escribir, diseñar y editar archivos de Excel**.

La función para leer archivos XLSX se llama read.xlsx:

*# install.packages("openxlsx")*

**library**(openxlsx)

**read.xlsx**(ruta\_archivo)

**Output**

name value

1 Name Clippy

2 Species paperclip

3 Approx date of death 39083

4 Weight **in** grams 0.9

Como en la función del paquete anterior, existen varios argumentos que puedes personalizar, como sheet, skip o colNames. Si quieres seleccionar celdas específicas puedes usar los argumentos rows y cols. Recuerda escribir ?read.xlsx o help(read.xlsx) para obtener información adicional.

**read.xlsx**(ruta\_archivo, cols = 1:2, rows = 2:3)

**Output**

Name Clippy

1 Species paperclip

**El paquete xlsx**

Aunque este paquete **requiere que JAVA esté instalado en tu ordenador**, es muy popular. Las funciones principales para importar archivos de Excel son read.xlsx y read.xlsx2. El segundo tiene ligeras diferencias en los argumentos predeterminados y hace más trabajo en JAVA, logrando un mejor rendimiento.

*# install.packages("xlsx")*

**library**(xlsx)

**read.xlsx**(ruta\_archivo)

**read.xlsx2**(ruta\_archivo)

Puedes personalizar diversos argumentos, como sheetIndex, sheetName, header, rowIndex, colIndex, entre otros. Ejecuta ?read.xlsx o help(read.xlsx) para obtener detalles adicionales.

**El paquete XLConnect**

Una alternativa al paquete xlsx es **XLConnect**, que permite escribir, leer y dar formato a archivos de Excel. Para leer un Excel en R puedes usar la función **readWorksheetFromFile** como sigue

**Importar una hoja una sola vez**

*# install.packages("XLConnect")*

**library**(XLConnect)

data <- **readWorksheetFromFile**(ruta\_archivo, sheet = **"list-column"**,

startRow = 1, endRow = 5,

startCol = 1, endCol = 2)

En caso de que quieras cargar varias hojas, se recomienda utilizar la función **loadWorkbook** y luego cargar cada hoja con la función **readWorksheet:**

**Leer varias hojas de Excel**

load <- **loadWorkbook**(ruta\_archivo)

data <- **readWorksheet**(load, sheet = **"list-column"**,

startRow = 1, endRow = 5,

startCol = 1, endCol = 2)

data2 <- **readWorksheet**(load, sheet = **"two-row-header"**,

startRow = 1, endRow = 3,

startCol = 1, endCol = 4)

Además, este paquete proporciona una función para cargar regiones con nombre de Excel. Análogo al ejemplo anterior, puedes importar solo una región con **readNamedRegionFromFile**, especificando el nombre del archivo (si el archivo está en tu directorio de trabajo) o la ruta del archivo y el nombre de la región en otro caso.

**Importar una región con nombre una vez**

data <- **readNamedRegionFromFile**(file, *# Ruta del archivo*

name, *# Nombre de la región*

...) *# Argumentos de readNamedRegion()*

Si quieres cargar varias regiones con nombre, puedes cargar el libro de Excel con la función **loadWorkbook** y luego importar cada región con la función **readNamedRegion.**

**Leer múltiples regiones con nombres**

cargar <- **loadWorkbook**(ruta\_archivo)

data <- **readNamedRegion**(cargar, name\_Region\_1, ...)

data2 <- **readNamedRegion**(cargar, name\_Region\_2, ...)

Cabe mencionar que si tienes problemas con los paquetes que requieren JAVA, puedes obtener y establecer la ruta de JAVA en R con los siguientes códigos:

*# Imprime la ruta de JAVA en R*

**Sys.getenv**(**"JAVA\_HOME"**)

*# Establece la ruta de JAVA*

**Sys.setenv**(JAVA\_HOME = **"path\_to\_jre\_java\_folder"**)

Ten en cuenta que deberás especificar la ruta a la carpeta jre dentro de la carpeta Java de tu ordenador, que por lo general se encontrará en Archivos de Programa.

**Convertir archivos XLSX a CSV en R**

Por último, también podrías convertir tus archivos de Excel a formato CSV y leer el archivo CSV en R. Para este propósito, puedes usar la función convert del paquete rio. Una alternativa sería guardar directamente el archivo de Excel como CSV con el menú de Microsoft Excel.

*# install.packages("rio")*

**library**(rio)

**convert**(ruta\_archivo, **"file.csv"**)