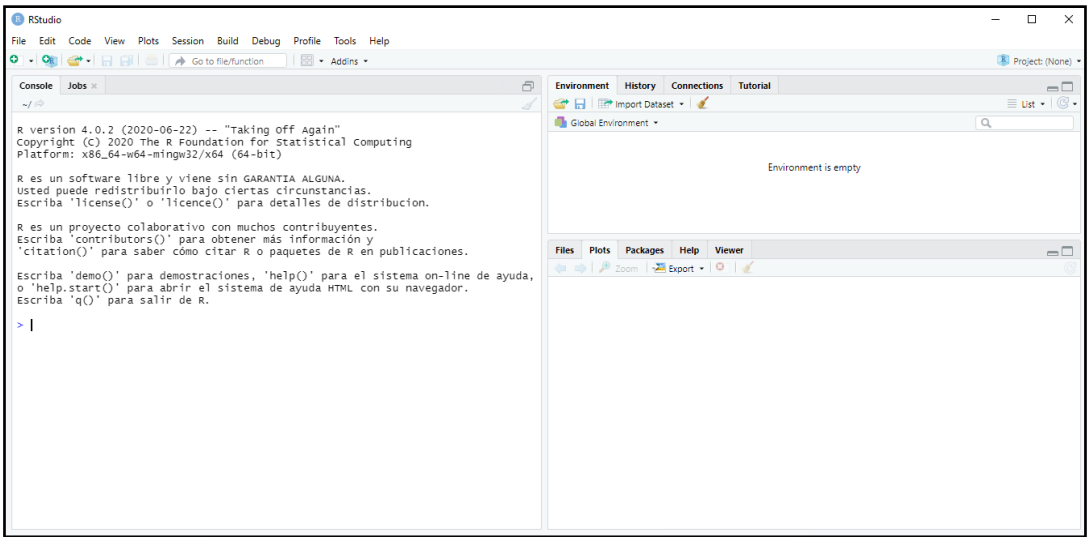
Introducción a R y R-Studio

R y R-Studio. Comienzo: Proyectos y Scripts.

Abriremos R-Studio pulsando en el icono correspondiente. Aparecerá la siguiente ventana:



La parte izquierda de la ventana es la "consola". En la consola es donde podemos manejar R mediante la introducción de código. Por ejemplo, podemos escribir 2+2 después del cursor (signo ">"), y pulsar Enter. La propia consola nos devolverá el valor 4.

Pero la forma más eficiente de trabajar es mediante "proyectos" y "scripts"

Un **proyecto** básicamente viene asociado a la carpeta donde R trabajará, buscando los datos que sean sus "inputs", y, en su caso, enviando sus resultados u "outputs". Para crear un nuevo proyecto, seguiremos la instrucción File → New Project, luego se nos preguntará si se crea el proyecto en una nueva carpeta o en una ya existente. Vamos a crearlo, por ejemplo, en el disco extraíble D, carpeta R, subcarpeta "explora", que ya

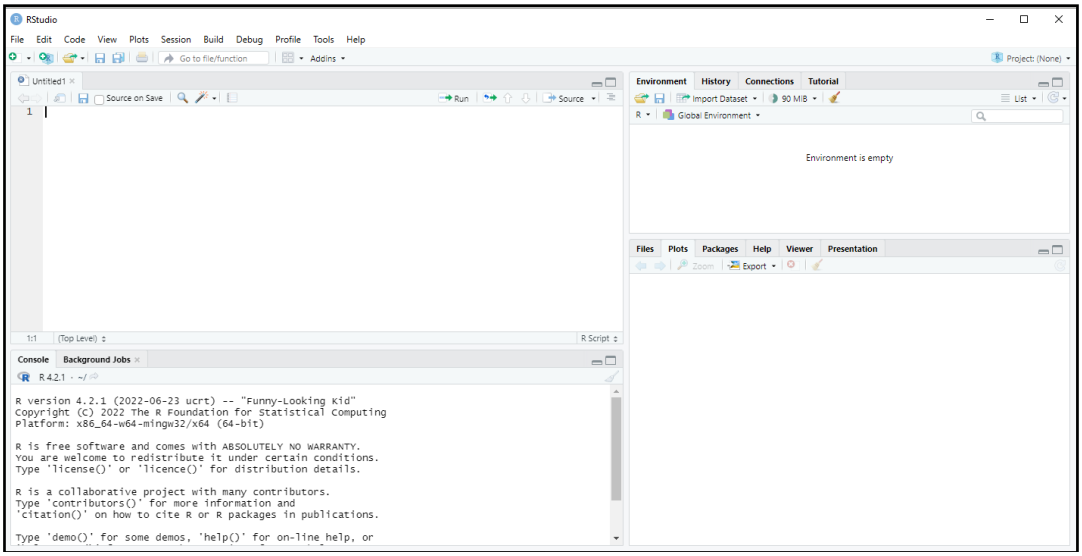
está creada. Nos saldrá una ventana para buscar la carpeta y, cuando la encontremos, pulsaremos Open y Create Project. Ya tendremos creado nuestro proyecto. Si nos vamos al explorador de Windows®, y buscamos la carpeta "explora", encontraremos que en tal carpeta aparece un archivo de nombre "explora", con un icono de un cubo con una "R". Ese archivo lo que está haciendo es actuar como un "faro" que le dice a R que, cuando trabajemos en el proyecto "explora", todos los archivos de datos necesarios estarán en esa carpeta (también llamada "explora", porque el proyecto adopta el nombre de la carpeta donde lo localizamos). Y que, si nuestro trabajo aporta algún fichero de "output", también se depositará en esa carpeta del proyecto.

En futuras sesiones, si queremos trabajar en el mismo proyecto, en lugar de seguir la ruta File → New Project, tendremos que hacer File → Open Project.

En cuanto a los **scripts**, son programas o rutinas donde varias instrucciones se ejecutan secuencialmente. Para crear un script, se seguirá la ruta File → New File → R Script. Y si el script lo guardamos, ¿dónde lo hará? Pues en la carpeta "explora", que es la del proyecto en el que estamos trabajando.

Informáticamente, un script es un archivo de texto plano, simplemente. Se puede modificar con cualquier editor de texto. Afortunadamente, para no estar entrando y saliendo de R-Studio, esta interfaz incorpora un editor de scripts, lo cual es muy cómodo.

Vemos como ahora, a la izquierda de R-Studio, ha aparecido, en la parte superior, una nueva ventana, pasando la consola a ocupar la parte inferior. Es la ventana del "editor".



Igual que con los proyectos, podemos crear desde R-Studio un script nuevo, o abrir uno preexistente; y modificarlo, ejecutarlo, o volverlo a guardar.

Vamos a comenzar a escribir nuestro script. Si queremos hacer un comentario que no ejecute ninguna instrucción, éste irá precedido del símbolo almohadilla o *hashtag* "#". Luego, vamos a ordenar a R que haga la operación de suma: 2+2. Escribimos, por tanto, en el editor:

```
#Ejemplo de Script
2+2  #este script hace una simple suma
```

Si pulsamos **Control + Mayúsculas + ENTER** o al desplegable de Source → Source with Echo, se ejecutará el script (para ejecutar solo la línea donde está el cursor, pulsaremos **Control + ENTER** o el botón de Run; y para ejecutar varias líneas, hemos de sombrearlas y pulsar **Control + ENTER** o el botón de Run). En la consola aparecerá:

```
> #Ejemplo de Script
> 2+2  #este script hace una simple suma
[1] 4
```

Podemos guardar el script con File → Save As… ¿Dónde se guardará por defecto? Pues en la carpeta "explora", que es la de nuestro proyecto. Una vez nuestro script ya tiene nombre, podemos ir guardándolo de vez en cuando pulsando simplemente en el botón del "disquete" del editor. Vamos a llamarlo, por ejemplo, "explorando". Si vamos, en el explorador de Windows®, a nuestra carpeta de proyecto, veremos que hay un archivo de texto llamado "explorando" con extensión ".R" (explorando.R). Este script lo podremos ejecutar cuantas veces queramos sin tener que escribir nada,

o reescribirlo si vemos que no funciona o que necesitamos hacer modificaciones. Esa es la ventaja de trabajar con scripts.

Para recuperar un script en una nueva sesión de trabajo simplemente tenemos que seguir las instrucciones File → Open File… y seleccionarlo.

Objetos. Datos.

Como acabamos de ver al ejecutar un sencillo script (o al escribir instrucciones directamente desde la consola), R es interactivo: responde a las entradas que recibe. Las **entradas** son **expresiones** que básicamente pueden ser:

- Expresiones aritméticas.
- Expresiones lógicas.
- Llamadas a funciones.
- Asignaciones**.**

Las expresiones realizan acciones sobre ***objetos*** de R. Los objetos en R son entes que tienen ciertas características, *metadatos*, llamados **atributos***.* No todos los objetos tienen los mismos atributos y, ni tan siquiera, todos los objetos tienen atributos que los caractericen.

Los objetos más importantes en R son ciertas estructuras o contenedores para **almacenar elementos**:

- Vectores.
- Matrices.
- Listas.
- *Data frames.*
- Factores.

Los elementos almacenados en los objetos se dividen en ***clases****.* Entre las diferentes clases, destacan las clases referidas a **datos**, que pueden ser de diferentes ***modos***: *logical* (verdadero/falso), *numeric* (números) o *character* (cadena de texto). El modo *numeric* puede ser, a la vez, de **tipo** *integer* (número entero) o *double* (número real). En el caso de *logical* y *carácter*, modo y tipo coinciden.

En cuanto a los **vectores**, son conjuntos de elementos **de la misma clase**. Vamos a definir por ejemplo el vector x = (1,3,5,8). Para ello, vamos a escribir en nuestro script:

```
x <- c(1,3,5,8)
```

Ejecutamos la línea (situando el cursor en algún lugar de ella, dentro del script; y pulsando a la vez las teclas **Control + Enter** o pinchando con el ratón en el botón Run del editor). Ya tenemos nuestro primer objeto de tipo *vector* en memoria. Por cierto, lo que hemos hecho es una **asignación**, que se escribe con una flecha creada mediante los signos "<" y "-". Hemos asignado a un vector llamado "x" los elementos 1, 3, 5 y 8.

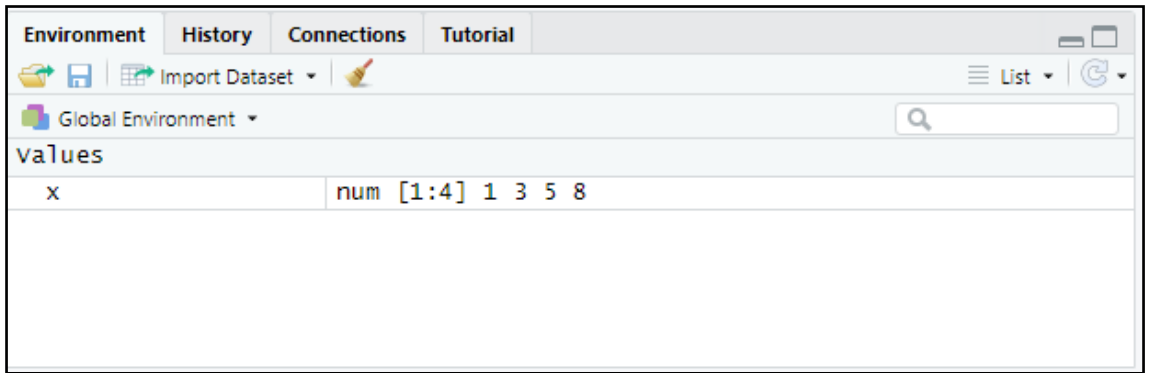
Para ver el vector simplemente escribimos en la consola (o en el script), ejecutando a continuación:

```
x
```

Y obtendremos:

```
[1] 1 3 5 8
```

Además, si miramos en la ventana superior-derecha de R-Studio, veremos que en el *Global Environment* sale nuestro vector, y además nos dice que es de modo *numérico*. Este *Global Environment* nos informa de los objetos que R tiene en memoria.



Si queremos obtener un vector de números consecutivos del 2 al 6, basta con escribir en la "consola" (o en el script), y ejecutar:

```
y <- c(2:6)
```

Al escribir el nombre del vector en la "consola" obtendremos:

```
[1] 2 3 4 5 6
```

Si queremos saber la longitud de un vector, usaremos la *función* length(). Por ejemplo, `length(y)` nos devolverá el valor 5. Escribamos en el script y ejecutemos.

Un vector puede incluir, además de números, por ejemplo, caracteres; siempre entrecomillados (lo fundamental es que sean elementos de la misma clase). Por ejemplo, el vector *genero* (¡no pongamos tildes o podemos tener problemas!). Así, si ejecutamos estas dos líneas:

```
genero<-c ("Mujer","Hombre")
genero
```

Dará como resultado:

```
[1] "Mujer"  "Hombre"
```

Podemos obtener la clase de los elementos almacenados en nuestro vector con la función class():

```
class(genero)
```

Al ejecutar la línea obtendremos como resultado:

```
[1] "character"
```

Si falta un dato en el vector, habrá que escribir “NA” (not available). Por ejemplo, si falta el tercer dato de este vector “z”, escribiremos:

```
Z <-c (1,2,NA,2,8)
```

Para **seleccionar un elemento** concreto del vector, indicaremos la posición en la que se encuentra entre corchetes. Por ejemplo, refiriéndonos al vector “x”, si ponemos en una línea del script:

```
x[3]
```

Y la ejecutamos, obtendremos:

```
[1] 5
```

Si queremos que se nos muestre los elementos del vector x del 2º al 4º:

```
x[2:4]
```

Y al ejecutar obtendremos:

```
[1] 3 5 8
```

Por último, si queremos sacar en pantalla los elementos 1º y 4º, tendremos que incluir una "c" seguida de un paréntesis que recoja el orden de los elementos que queremos seleccionar:

```
x[c(1,4)]
```

Y obtendremos:

```
[1] 1 8
```

Por otro lado, las **matrices** son vectores, pero con dos atributos adicionales: número de filas y número de columnas. Se definen mediante la función matrix(). Por ejemplo, para definir la matriz "a":

$$\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

Tendremos que escribir (bien en la consola, bien en el script):

```
a<-matrix(c(1,2,3,4,5,6,7,8,9),nrow=3,ncol=3)
```

Como vemos, por defecto, R va "cortando" el vector por columnas (si lo preferimos, lo puede hacer también por filas, añadiendo a la función matrix() el argumento by row = true; pero, en nuestro ejemplo, obtendríamos la matriz traspuesta a la que queremos almacenar).

Las dimensiones (número de filas y de columnas) de la matriz pueden obtenerse mediante la función dim(). Por ejemplo, `dim(a)` proporcionaría el resultado `3 3` (número de filas, número de columnas).

Si queremos seleccionar elementos concretos de una matriz, lo haremos utilizando corchetes para indicar filas y columnas. Por ejemplo, `a[2,3]` devolverá el valor `8`, mientras que `a[1:2,2:3]` devolverá:

```
     [,1] [,2]
[1,]    4    7
[2,]    5    8
```

`a[,c(1,3)]` devolverá:

```
     [,1] [,2]
[1,]    1    7
[2,]    2    8
[3,]    3    9
```

Por su parte, `a[c(1,3),]` devolverá:

```
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    3    6    9
```

Tanto para vectores como para matrices, funcionan las operaciones suma y diferencia sin más complicaciones. En el caso del producto, sin embargo, hay que tener en cuenta que, por ejemplo, `a*a` devuelve la multiplicación elemento a elemento, es decir:

```
     [,1] [,2] [,3]
[1,]    1   16   49
[2,]    4   25   64
[3,]    9   36   81
```

mientras que `a%*%a` sí devuelve el producto matricial:

```
     [,1] [,2] [,3]
[1,]   30   66  102
[2,]   36   81  126
[3,]   42   96  150
```

Los ***data frame*** son objetos que almacenan datos organizados mediante la clase data.frame. Esta organización consiste en que, por filas, se disponen los diferentes casos o sujetos; mientras que por columnas se posicionan las variables. Un *data frame:*

- Es similar a una matriz en el sentido de que tiene dos dimensiones. Podemos acceder a sus elementos con corchetes, tenemos nombres de filas y columnas, y podemos operar con ellas.
- Cada columna tiene un nombre y podemos acceder a una columna concreta con el símbolo **$**. Todas las columnas (variables) son vectores con la misma longitud.
- Cada columna es un vector, que puede ser numérico, factor, carácter o lógico.

Por ejemplo, vamos a crear el *data frame* "datos", con tres variables: "peso", "altura", y "color de ojos", llamadas "Peso", "Altura" y "Cl.ojos", respectivamente; para 3 individuos o casos. Una opción es crear primero las tres variables como vectores:

```
Peso<-c(68,75,88)
Altura<-c(1.6,1.8,1.9)
Cl.ojos<-c("azules","marrones","marrones")
```

Y luego asignar estos vectores a nuestro *data frame*, con la función data.frame():

```
datos<-data.frame(Peso,Altura,Cl.ojos)
```

Si ahora ejecutamos una línea con el nombre de nuestro *data frame*:

```
datos
```

Obtendremos la salida:

```
  Peso Altura  Cl.ojos
1   68    1.6   azules
2   75    1.8 marrones
3   88    1.9 marrones
```

Para obtener los nombres de las variables (es decir, el nombre de cada columna) teclearemos la función:

```
names(datos)
```

Obteniéndose:

```
[1] "Peso"       "Altura"     "Cl.ojos"
```

Para obtener solo los datos de la columna (variable) color de ojos teclearemos `datos$Cl.ojos`:

```
[1] "azules"   "marrones" "marrones"
```

Y para obtener los datos de peso, `datos$Peso`:

```
[1] 68 75 88
```

Para saber el número de filas y de columnas de una hoja de datos utilizaremos las funciones nrow() y ncol(). Por ejemplo, `ncol(datos)` es `3`.

Para seleccionar elementos de un *data frame*, se pueden seguir las mismas reglas que para la selección de elementos de una matriz (con el número de cada fila, que es cada individuo; y el número de cada columna, que es cada variable. Para elegir una variable, no obstante, ya hemos visto que es posible usar su nombre; aunque precedido del nombre del *data frame* y el signo "$". Por ejemplo, si ejecutamos:

```
datos[,2]
```

Y ejecutamos:

```
datos$Altura
```

Tendremos el mismo resultado:

```
[1] 1.6 1.8 1.9
```

¿Qué ocurre si no estamos seguros de qué estructura tiene un objeto? Podemos usar funciones como is.vector(), is.matrix() e is.data.frame(). Así, por ejemplo, `is.data.frame(a)` devolverá `FALSE` (recordemos que era una matriz); en cambio `is.data.frame(datos)` devolverá `TRUE`.

Para borrar un objeto tenemos la instrucción rm(), por ejemplo `rm("Peso")` borrará de la memoria de R el vector Peso (aunque ¡cuidado!: la variable "Peso" la seguimos teniendo; aunque contenida en el *data frame* "datos"). Si lo que queremos es borrar todos los objetos que tenemos en memoria, la línea a ejecutar será:

```
rm(list=ls(all=TRUE))
```

Importando y explorando nuestros datos.

Lo más frecuente es que no tecleemos los datos, como hemos hecho hasta ahora; sino que los importemos a R desde algún contenedor externo (archivo de texto, hoja de cálculo, base de datos…). Nosotros vamos a importar nuestros datos desde Microsoft ® Excel ®.

Vamos a cerrar el script que hemos estado construyendo (para conservarlo hay que guardarlo antes), aunque vamos a seguir trabajando en el proyecto "explora". Iremos a la carpeta del proyecto y vamos a guardar en ella los dos archivos de esta práctica: un archivo de Microsoft® Excel® llamado "eolica_20.xlsx" y un script que ya tenemos hecho, y que se llama "explora_eolica.R". Si abrimos el archivo de Microsoft® Excel® comprobaremos que se compone de tres hojas. La primera muestra el criterio de búsqueda de casos en la base de datos Sabi®; la segunda recoge la descripción de las variables consideradas; y la tercera (hoja "**Top 20**") guarda los datos que debemos importar desde R-Studio. Estos datos se corresponden con diferentes variables económico-financieras de las 20 empresas productoras de electricidad mediante generación eólica con mayor volumen de activo total.

Luego vamos a cerrar el archivo de Microsoft® Excel® y volveremos a R-Studio. Vamos a abrir nuestro script "explora_eolica.R" con File → Open File… Este script contiene el programa que vamos a ir ejecutando en la práctica.

La primera línea / instrucción en los scripts suele ser:

```
rm(list = ls())
```

La instrucción tiene como objeto limpiar el *Environment* (memoria) de objetos de anteriores sesiones de trabajo.

Para importar los datos, hay dos modos. Uno, utilizar las facilidades de importación de RStudio. Para ello, nos situaremos en la ventana superior derecha, pestaña *Environment*, y desplegaremos el menú "Import Dataset", seleccionando "From Excel…" Aparecerá una ventana emergente, donde, con "Browse", seleccionaremos nuestro archivo de Microsoft® Excel®. Nos aseguraremos de tener la "Hoja" o pestaña adecuada y las otras opciones de importación correctas.

Observaremos que hay un espacio en esta ventana, abajo a la derecha, donde aparece el **código** que realizaría, desde un script, el mismo proceso de importación. Podemos **copiar este código** e integrarlo en nuestro script para que, en futuras sesiones de trabajo, no tengamos que recurrir a las facilidades de importación de R-Studio. Pulsamos "Import".

El código que equivale a las acciones anteriores, y que también importaría los datos desde el archivo de Microsoft® Excel®, es, por tanto:

```
# DATOS

library(readxl)
eolica_20 <- read_excel("eolica_20.xlsx", sheet = "Top 20")
```

¡Atención! Si nunca se ha utilizado la librería readxl (que contiene el código necesario para importar datos de un archivo de Microsoft® Excel®), cuando la intentemos activar con la función library() nos dará un error o nos dirá que previamente hay que importarla. En ese caso, iremos a la ventana inferior-derecha y pulsaremos la pestaña "Packages", pulsaremos en "Install", y emergerá una ventana donde dejaremos el "repositorio" que viene por defecto y, en el campo "Packages", escribiremos el nombre del "paquete" que contiene la librería que nos hace falta (normalmente coincide con el nombre de la propia librería, en nuestro caso readxl). Una vez descargado el "paquete", podremos ejecutar el código anterior sin problemas.

Podemos observar como en el *Environment* ya aparece un objeto. Este objeto es una estructura de datos tipo *data frame*, se llama "eolica_20" (si importamos mediante las facilidades de R-Studio, el *data frame*, por defecto, adopta el nombre del archivo del que proceden los datos, aunque podemos cambiar el nombre si queremos) y contiene 12 columnas, una por cada una de las variables almacenadas en el archivo de Microsoft® Excel®. De estas variables, tres son de tipo cualitativo, formadas por cadenas de caracteres: el nombre de la empresa ("NOMBRE") y el nombre del grupo empresarial matriz al que pertenece ("MATRIZ"). Puede explorarse el contenido del *data frame* y los principales estadísticos con la función summary():

```
summary (eolica_20)
```

Veremos cómo aparecen las 11 variables con algunos estadísticos básicos.

R ha considerado la primera columna como una variable de tipo cualitativo. En realidad, no es una variable, sino el nombre de los individuos. Para evitar que R tome los nombres de los individuos como una variable, podemos redefinir nuestro *data frame* diciéndole que tome esa primera columna como los *nombres de los individuos*:

```
eolica_20 <- data.frame(eolica_20, row.names = 1)
```

En la línea anterior hemos asignado al *data frame* "eolica_20" los propios datos de "eolica_20"; pero indicando que la primera columna de datos no es una variable; sino el nombre de los casos. Si hacemos ahora el summary():

```
summary (eolica_20)
```

Vemos que ya no aparece "NOMBRE" como variable, y en el *Environment* ya aparece el *data frame* "eolica_20" con 20 observaciones (casos) pero con 11 variables (una menos).

Updated: 19/09/2022