Lección 6

Data frames

Como ya comentamos en la Lección 2, la manera más conveniente de guardar en R los resultados de varias observaciones sobre diferentes individuos es en forma de *data frames*; estos objetos implementan en R una estructura de datos muy típica en estadística y que recibe diversos nombres, como tabla de datos, hoja de datos (*data sheet*) o, en algunas aplicaciones, hoja de cálculo (*spreadsheet*). Estas estructuras consisten en pequeñas bases de datos donde se han anotado los valores de algunas variables para una serie de observaciones. En concreto, un *data frame* es una tabla de doble entrada, formada por variables en las columnas y observaciones de estas variables en las filas, de manera que cada fila contiene los valores de las variables para un mismo caso o un mismo individuo. En este sentido, un *data frame* tiene la apariencia de una matriz, pero con la diferencia de que cada columna de un *data frame* puede contener datos de un tipo diferente (una columna puede estar formada por números; otra, por palabras; otra, por valores lógicos; etc.), siempre que todos los datos de una misma columna sean del mismo tipo.

6.1. Estructura de un *data frame*

La instalación básica de R lleva predefinidos algunos objetos de datos. Podemos echarles un vistazo entrando la instrucción `data()`, que abrirá una ventana con la lista de los objetos de datos a los que tenemos acceso en la sesión actual de R (los que lleva la instalación básica de R y los que aportan los paquetes que tengamos cargados). Estos objetos no aparecen cuando hacemos `ls()` ni se borran con `rm(list=ls())`, y podemos pedir información sobre cada uno de ellos con la instrucción `help`. Al final de la lista que obtenemos con `data()` se nos indica que si entramos

```
data(package=.packages(all.available=TRUE)),
```

obtendremos la lista de *todos* los objetos de datos a los que tenemos acceso, incluyendo los de los paquetes que tengamos instalados, pero que no estén cargados en la sesión actual.

Una de las tablas de datos más populares que lleva R es el llamado *iris data set*, que contiene la longitud y la anchura de los pétalos y sépalos y la especie de 150 flores iris. El famoso estadístico Sir R. A. Fisher usó este conjunto de datos en su artículo «The Use of Multiple Measurements in Taxonomic Problems» (*Annals of Eugenics* 7 (1936), pp. 179–188).[1] En R, este conjunto de datos de flores iris está recogido en el *data frame* `iris`. Para más información sobre estos datos, podéis consultar su entrada en la *Wikipedia*,[2] y para más información específica sobre la tabla `iris` de R, podéis usar `help(iris)`.

En esta sección vamos a trabajar con esta tabla de datos, así que lo primero que haremos será copiarla en un nuevo *data frame* que llamaremos `d.f`; de este modo trabajaremos sobre la copia `d.f` y tendremos acceso a la tabla `iris` original si necesitamos volver a ella. De todas formas, si

[1] Se puede descargar de `http://digital.library.adelaide.edu.au/coll/special//fisher/138.pdf`. Si buscáis `"iris data set"` en *Google*, encontraréis casi 400 000 referencias a este conjunto de datos.

[2] `http://en.wikipedia.org/wiki/Iris_flower_data_set`.

trabajamos directamente sobre `iris` y la echamos a perder, la podemos recuperar ejecutando la instrucción `data(iris)`, con la que este *data frame* recuperará su contenido original.

```
> d.f=iris
```

Si pidiéramos ahora a `R` que nos mostrase el objeto `d.f`,

```
> d.f
```

obtendríamos el contenido del *data frame* en la consola: una larga lista de datos formada por las 150 filas de la tabla. No vamos a copiar aquí esta salida, pero sí que vamos a consultar las primeras filas del *data frame*, para así poder entender su estructura y conocer los nombres de sus variables; para hacerlo, hemos de aplicar la función `head` al *data frame* y al número de filas que queremos que muestre; su valor por defecto es 6.

```
> head(d.f,5)  #Las primeras 5 filas
  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
```

La función `tail`, con una estructura similar a `head`, nos muestra la «cola» de la tabla.

```
> tail(d.f,5)    #Las últimas 5 filas
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
146          6.7         3.0          5.2         2.3 virginica
147          6.3         2.5          5.0         1.9 virginica
148          6.5         3.0          5.2         2.0 virginica
149          6.2         3.4          5.4         2.3 virginica
150          5.9         3.0          5.1         1.8 virginica
```

Observad que las columnas del *data frame* tienen *nombres* (`R` los llama `names`) y las filas, que corresponden a individuos, tienen como *identificador* (`R` llama a estos identificadores `rownames`) un número natural correlativo que va del 1 al 150. Podemos ver también que, en cada fila, la columna `Species` contiene una palabra que describe la especie de la flor, mientras que las otras cuatro columnas contienen números que corresponden a medidas. Como veremos, la columna `Species` no es un vector, sino un factor (con niveles las tres especies de iris: *setosa*, *versicolor* y *virginica*), lo que nos facilita el uso de esta variable para clasificar las flores.

Si queremos conocer la estructura global de un *data frame*, podemos usar la función `str`.

```
> str(d.f)
'data.frame': 150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1
    1 1 1 1 1 1 ...
```

El resultado de esta instrucción nos muestra la estructura del objeto `d.f`. Nos dice que es un *data frame* formado por 150 observaciones (filas) de 5 variables (columnas), y de cada variable nos da su nombre (precedido de un signo `$`) y su tipo de datos: las cuatro primeras son vectores de números (`num`), y la quinta es un factor con (`w`, de *with*) 3 niveles. Además, nos muestra los primeros valores de cada variable: en el caso del factor, los unos significan «setosa», su primer nivel.

Las funciones siguientes nos permiten obtener los nombres de las variables, los identificadores de las filas y las dimensiones de un *data frame*:

- `names`: Produce un vector con los nombres de las columnas.
- `rownames`: Produce un vector con los identificadores de las filas.
- `dimnames`: Produce una `list` formada por dos vectores: el de los identificadores de las filas y el de los nombres de las columnas.
- `dim`: Produce un vector con el número de filas y el número de columnas.

Veamos sus valores sobre nuestro *data frame*:

```
> names(d.f)
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
[5] "Species"
> rownames(d.f)
  [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"
 [11] "11"  "12"  "13"  "14"  "15"  "16"  "17"  "18"  "19"  "20"
 [21] "21"  "22"  "23"  "24"  "25"  "26"  "27"  "28"  "29"  "30"
 [31] "31"  "32"  "33"  "34"  "35"  "36"  "37"  "38"  "39"  "40"
 [41] "41"  "42"  "43"  "44"  "45"  "46"  "47"  "48"  "49"  "50"
 [51] "51"  "52"  "53"  "54"  "55"  "56"  "57"  "58"  "59"  "60"
 [61] "61"  "62"  "63"  "64"  "65"  "66"  "67"  "68"  "69"  "70"
 [71] "71"  "72"  "73"  "74"  "75"  "76"  "77"  "78"  "79"  "80"
 [81] "81"  "82"  "83"  "84"  "85"  "86"  "87"  "88"  "89"  "90"
 [91] "91"  "92"  "93"  "94"  "95"  "96"  "97"  "98"  "99"  "100"
[101] "101" "102" "103" "104" "105" "106" "107" "108" "109" "110"
[111] "111" "112" "113" "114" "115" "116" "117" "118" "119" "120"
[121] "121" "122" "123" "124" "125" "126" "127" "128" "129" "130"
[131] "131" "132" "133" "134" "135" "136" "137" "138" "139" "140"
[141] "141" "142" "143" "144" "145" "146" "147" "148" "149" "150"
> dimnames(d.f)
[[1]]
  [1] "1"   "2"   "3"   "4"   "5"   "6"   "7"   "8"   "9"   "10"
 [11] "11"  "12"  "13"  "14"  "15"  "16"  "17"  "18"  "19"  "20"
 [21] "21"  "22"  "23"  "24"  "25"  "26"  "27"  "28"  "29"  "30"
 [31] "31"  "32"  "33"  "34"  "35"  "36"  "37"  "38"  "39"  "40"
 [41] "41"  "42"  "43"  "44"  "45"  "46"  "47"  "48"  "49"  "50"
 [51] "51"  "52"  "53"  "54"  "55"  "56"  "57"  "58"  "59"  "60"
 [61] "61"  "62"  "63"  "64"  "65"  "66"  "67"  "68"  "69"  "70"
 [71] "71"  "72"  "73"  "74"  "75"  "76"  "77"  "78"  "79"  "80"
 [81] "81"  "82"  "83"  "84"  "85"  "86"  "87"  "88"  "89"  "90"
 [91] "91"  "92"  "93"  "94"  "95"  "96"  "97"  "98"  "99"  "100"
```

```
[101] "101" "102" "103" "104" "105" "106" "107" "108" "109" "110"
[111] "111" "112" "113" "114" "115" "116" "117" "118" "119" "120"
[121] "121" "122" "123" "124" "125" "126" "127" "128" "129" "130"
[131] "131" "132" "133" "134" "135" "136" "137" "138" "139" "140"
[141] "141" "142" "143" "144" "145" "146" "147" "148" "149" "150"

[[2]]
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"
[5] "Species"

> dim(d.f)
[1] 150   5
```

Fijaos en que el resultado de `rownames` son los identificadores de las filas entre comillas, es decir, considerados como palabras; `R` entiende siempre que estos identificadores son palabras, aunque, como en este caso, sean números.

Recordaréis que se puede obtener el valor de una componente de una `list` añadiendo al nombre de esta última un sufijo formado por el símbolo `$` seguido del nombre de la componente. De manera similar, para obtener una columna concreta de un *data frame* basta añadir a su nombre un sufijo formado por el símbolo `$` seguido del nombre de la variable; el resultado será un vector o un factor, según cómo esté definida la columna dentro del *data frame*.

```
> d.f$Sepal.Length[1:30]
 [1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9 5.4 4.8 4.8 4.3 5.8
[16] 5.7 5.4 5.1 5.7 5.1 5.4 5.1 4.6 5.1 4.8 5.0 5.0 5.2 5.2 4.7
> d.f$Species[1:30]
 [1] setosa setosa setosa setosa setosa setosa setosa setosa setosa
[10] setosa setosa setosa setosa setosa setosa setosa setosa setosa
[19] setosa setosa setosa setosa setosa setosa setosa setosa setosa
[28] setosa setosa setosa
Levels: setosa versicolor virginica
```

Como también pasaba con las componentes de las `list`, las variables de un *data frame* son internas, no están definidas en el entorno global de trabajo de `R`.

```
> Sepal.Length
Error: object 'Sepal.Length' not found
```

En la Sección 1.8 explicaremos cómo podemos declarar las variables internas de un *data frame* como variables globales, y así poder usarlas directamente por su nombre, sin tener que añadirles delante el nombre del *data frame* y el `$`.

Los *data frames* comparten con las matrices el uso de los corchetes para extraer trozos por filas y columnas. Los resultados que se obtienen son de nuevo *data frames*, y tanto los nombres de las columnas como los identificadores de las filas se heredan del *data frame* original; podemos comprobarlo en los siguientes ejemplos:

```
> d.f[1:5, ] #La subtabla de las 5 primeras filas
  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
> d.f[1:5, 1:3]  #La subtabla de las 5 primeras filas y las 3
   primeras columnas
   Sepal.Length Sepal.Width Petal.Length
1           5.1         3.5          1.4
2           4.9         3.0          1.4
3           4.7         3.2          1.3
4           4.6         3.1          1.5
5           5.0         3.6          1.4
> d.f[d.f$Species=="virginica" & d.f$Sepal.Length>7, ]  #La
   subtabla de filas con Species=virginica y Sepal.Length>7
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
103          7.1         3.0          5.9         2.1 virginica
106          7.6         3.0          6.6         2.1 virginica
108          7.3         2.9          6.3         1.8 virginica
110          7.2         3.6          6.1         2.5 virginica
118          7.7         3.8          6.7         2.2 virginica
119          7.7         2.6          6.9         2.3 virginica
123          7.7         2.8          6.7         2.0 virginica
126          7.2         3.2          6.0         1.8 virginica
130          7.2         3.0          5.8         1.6 virginica
131          7.4         2.8          6.1         1.9 virginica
132          7.9         3.8          6.4         2.0 virginica
136          7.7         3.0          6.1         2.3 virginica
> d.f[d.f$Species=="virginica" & d.f$Sepal.Length>7, ][1:4,]  #La
   subtabla de las 4 primeras filas de la anterior
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
103          7.1         3.0          5.9         2.1 virginica
106          7.6         3.0          6.6         2.1 virginica
108          7.3         2.9          6.3         1.8 virginica
110          7.2         3.6          6.1         2.5 virginica
```

En todos los casos, obtenemos subtablas del *data frame* `d.f` con sus filas y columnas determinadas por las expresiones entre corchetes.

6.2. Cómo importar y exportar *data frames*

`R` dispone de varias instrucciones que permiten importar ficheros externos como objetos de datos; por ejemplo, ya vimos en la Lección 3 la función `scan`, que permitía definir un vector con los elementos de un fichero. En esta sección estudiaremos básicamente la función `read.table`, que sirve para definir un *data frame* a partir de una tabla de datos contenida en fichero; este fichero puede estar en nuestro ordenador, y en este caso lo más práctico es que la tengamos en el directorio de trabajo de `R`, o bien podemos conocer su *url*. El fichero a importar con `read.table` ha de ser de texto simple: nada de ficheros en formato *doc* o *xls*, cuya lectura requiere de funciones específicas.

Para crear un *data frame* a partir de un fichero de texto simple, tenemos que aplicar la función `read.table` a su nombre, o a su dirección, entre comillas. Esta función `read.table` tiene algunos parámetros que evitan errores en la creación del *data frame*:

- `sep`. Como ya ocurría con la función `scan`, la función `read.table` entiende por defecto que las separaciones entre columnas en el fichero están marcadas por espacios en blanco. Si esta separación está marcada por comas, signos de punto y coma, tabuladores o de cualquier otra manera, hay que especificarlo con este parámetro.

  Por ejemplo, tenemos que especificar `sep=","` para indicar que las separaciones entre columnas son comas, `sep=";"` para indicar que las separaciones entre columnas son signos de punto y coma, y `sep="\t"` para indicar que las separaciones entre columnas son tabuladores.

- `header`. Si la tabla que importamos tiene una primera fila con los nombres de las columnas, es conveniente especificarlo con el parámetro `header=TRUE`; en caso contrario, y según cómo esté formateado el fichero, puede que `R` entienda la fila de los nombres de las variables como la primera fila de observaciones. Si, en cambio, las columnas de la tabla que importamos no tienen nombre, no hace falta especificar el parámetro `header` (o usar `header=FALSE`, que es su valor por defecto), y después ya pondremos nombres a las variables con la función `names` (véase la Sección 1.4).

- `dec`. Como ya explicamos en la función `scan`, podemos usar el parámetro `dec` para especificar el signo que separa la parte entera de la decimal en los números, si no es un punto.

- `stringsAsFactors`. Por defecto, `read.table` transforma en factores las columnas de palabras de la tabla que importa. Para prohibir esta transformación, y que los vectores de palabras se importen como tales, podemos usar `stringsAsFactors=FALSE`.

Para otros parámetros, podéis consultar `help(read.table)`.

Vamos a ilustrar el uso de esta función. Descargad en vuestro directorio de trabajo de `R` los dos ficheros siguientes, manteniendo sus nombres y extensiones:

```
http://bioinfo.uib.es/~recerca/RMOOC/NotaHermanos.txt
http://bioinfo.uib.es/~recerca/RMOOC/NotaHermanosc.txt
```

Ambos ficheros son básicamente la misma tabla de datos, que recoge, para algunos estudiantes de primer curso de la UIB de hace unos años, el grado en el que estaban matriculados (Biología, «BL», o Bioquímica, «BQ»), su número de hermanos y la nota que obtuvieron en un determinado examen; la diferencia es que, en el primero, las columnas están separadas por espacios en blanco, y en el segundo, por comas. Ambos ficheros contienen una primera fila con los nombres de las variables.

Para crear un *data frame* llamado `NH1` a partir de la copia local del fichero `NotaHermanos.txt`, basta aplicar la función `read.table` a su nombre entre comillas y especificar `header=TRUE`.

```
> NH1=read.table("NotaHermanos.txt", header=TRUE)
```

Es una buena costumbre, una vez definido un *data frame* a partir de un fichero externo, comprobar que se ha importado bien. Como una tabla de datos puede tener muchas filas, verla entera en la consola puede ser poco práctico; lo mejor es usar las funciones `str` y `head`. Os aconsejamos que, siempre que vayáis a trabajar con un *data frame*, le peguéis antes un vistazo con estas funciones para comprobar su estructura, los nombres de sus variables, los tipos de sus datos, etc. Esto es especialmente importante cuando se trata de *data frames* importados,

porque si no especificamos los parámetros adecuados, puede que en el proceso de lectura y definición del *data frame* se pierdan los nombres de las variables o la estructura de columnas.

```
> head(NH1)
  Grado Hermanos Nota
1    BQ        1  9.3
2    BL        1  3.5
3    BL        1  5.5
4    BL        1  7.7
5    BL        2  8.1
6    BQ        0  8.6
> str(NH1)
'data.frame': 87 obs. of  3 variables:
 $ Grado   : Factor w/ 2 levels "BL","BQ": 2 1 1 1 1 2 2 1 1 1 ...
 $ Hermanos: int  1 1 1 1 2 0 1 1 1 1 ...
 $ Nota    : num  9.3 3.5 5.5 7.7 8.1 8.6 9.7 4 1.6 3.1 ...
```

Si no hubiéramos especificado `header=TRUE`, el formato del *data frame* resultante no habría sido el adecuado.

```
> NH1.mal=read.table("NotaHermanos.txt")
> str(NH1.mal)
'data.frame': 88 obs. of  3 variables:
 $ V1: Factor w/ 3 levels "BL","BQ","Grado": 3 2 1 1 1 1 2 2 1 1
   ...
 $ V2: Factor w/ 6 levels "0","1","2","3",..: 6 2 2 2 2 3 1 2 2 2
   ...
 $ V3: Factor w/ 50 levels "1.0","1.1","1.6",..: 50 48 13 26 38 41
   45 49 17 3 ...
```

Sin el parámetro `header=TRUE`, `R` ha entendido que `Grado`, `Hermanos` y `Nota` son elementos de sus columnas respectivas, y no sus nombres; en consecuencia, por un lado, ha llamado por defecto `V1`, `V2` y `V3` a las columnas del *data frame*, y por otro, como `Grado`, `Hermanos` y `Nota` son palabras y los datos de cada columna de un *data frame* han de ser del mismo tipo, ha considerado que todas las entradas de cada columna eran palabras. Finalmente, como, al crear un *data frame*, `R` interpreta los vectores de palabras como factores si no especificamos lo contrario, hemos obtenido un *data frame* formado por tres factores, y hemos perdido la información numérica que contenían las columnas con el número de hermanos y la nota.

Para importar el fichero `NotaHermanosc.txt`, donde las columnas están separadas por comas, hay que usar el parámetro `sep=","`.

```
> NH2=read.table("NotaHermanosc.txt", header=TRUE, sep=",")
> str(NH2)
'data.frame': 87 obs. of  3 variables:
 $ Grado   : Factor w/ 2 levels "BL","BQ": 2 1 1 1 1 2 2 1 1 1 ...
 $ Hermanos: int  1 1 1 1 2 0 1 1 1 1 ...
 $ Nota    : num  9.3 3.5 5.5 7.7 8.1 8.6 9.7 4 1.6 3.1 ...
```

Sin este parámetro, `R` hubiera entendido cada fila como una sola palabra.

```
> NH2.mal=read.table("NotaHermanosc.txt", header=TRUE)
```

```
> str(NH2.mal)
'data.frame': 87 obs. of  1 variable:
 $ Grado.Hermanos.Nota: Factor w/ 71 levels "BL,0,2.0","BL,0,4.0"
   ,..: 62 12 18 24 38 47 63 15 8 11 ...
```

Veamos el efecto de `stringsAsFactors=FALSE`:

```
> NH3=read.table("NotaHermanosc.txt", header=TRUE, sep=",",
   stringsAsFactors=FALSE)
> str(NH3)
'data.frame': 87 obs. of  3 variables:
 $ Grado   : chr  "BQ" "BL" "BL" "BL" ...
 $ Hermanos: int  1 1 1 1 2 0 1 1 1 1 ...
 $ Nota    : num  9.3 3.5 5.5 7.7 8.1 8.6 9.7 4 1.6 3.1 ...
```

La variable `Grado` se ha importado como un vector de palabras: lo indica el `chr`, de `character`, en la fila correspondiente del resultado de `str`.

Finalmente, para importar las versiones de Internet de estos ficheros, tendríamos que usar las mismas instrucciones, cambiando el nombre del fichero por su *url*, siempre entre comillas.

```
> NH4=read.table("http://bioinfo.uib.es/~recerca/RMOOC/NotaHermanos
   .txt", header=TRUE)
> str(NH4)
'data.frame': 87 obs. of  3 variables:
 $ Grado   : Factor w/ 2 levels "BL","BQ": 2 1 1 1 1 2 2 1 1 1 ...
 $ Hermanos: int  1 1 1 1 2 0 1 1 1 1 ...
 $ Nota    : int  93 35 55 77 81 86 97 40 16 31 ...
```

Naturalmente, para poder importar un fichero de Internet, hay que estar conectados a Internet, el servidor que aloja el fichero tiene que funcionar, y además no tiene que requerir una palabra clave para acceder.

Para importar un fichero de una página web segura (cuyo *url* empiece con `https`), no podemos entrar directamente la dirección en `read.table`; una solución es instalar y cargar el paquete `RCurl` y entonces usar la instrucción

`read.table(textConnection(getURL("`*url*`")),...)`.

Para mostrar su uso, hemos guardado una copia de `NotaHermanos.txt` en una carpeta pública de *Dropbox*. Su *url* es

`https://dl.dropboxusercontent.com/u/72911936/NotaHermanos.txt`

Si intentamos importar este fichero como lo hemos hecho hasta ahora, `R` nos dará un mensaje de error.

```
> NH5=read.table("https://dl.dropboxusercontent.com/u/72911936/
   NotaHermanos.txt", header=TRUE)
Error in file(file, "rt") : cannot open the connection
In addition: Warning message:
In file(file, "rt") : unsupported URL scheme
```

Veamos cómo importarla sin ningún problema:

```
> #Instalamos y cargamos el paquete RCurl
...
> NH5=read.table(textConnection(getURL("https://dl.
   dropboxusercontent.com/u/72911936/NotaHermanos.txt")), header=
   TRUE)
> str(NH5)
'data.frame': 87 obs. of  3 variables:
 $ Grado   : Factor w/ 2 levels "BL","BQ": 2 1 1 1 1 2 2 1 1 1 ...
 $ Hermanos: int  1 1 1 1 2 0 1 1 1 1 ...
 $ Nota    : int  93 35 55 77 81 86 97 40 16 31 ...
```

Además de `read.table`, que sólo sirve para importar tablas en formato texto simple, `R` dispone de otras instrucciones similares para importar otros tipos de ficheros. Las más útiles son:

- `read.csv`, para importar ficheros en formato CSV, un formato estándar para tablas de datos.
- `read.xls` y `read.xlsx`, del paquete `xlsx`, para importar hojas de cálculo tipo *Excel* u *OpenOffice* en formato XLS o XLSX, respectivamente. Estas funciones usan Java; si no lo tenéis instalado y no lo queréis instalar, lo mejor es que guardéis la hoja de cálculo en formato CSV (tanto *Excel* como *OpenOffice* o *Numbers* ofrecen esta posibilidad) y uséis `read.csv`.
- `read.mtb` y `read.spss`, del paquete `foreign`, para importar tablas de datos de Minitab y SPSS, respectivamente.

Si necesitáis otras funciones de este tipo, entrad `help.search("read")`, buscad la función que os convenga y pedid información sobre ella con la instrucción `help`.

Podemos exportar un *data frame* a un fichero usando la función `write.table`. Su sintaxis básica es

`write.table(`*data frame*`, file="`*nombre del fichero*`").`

Esta función crea un fichero, en el directorio de trabajo de `R`, que contiene el *data frame* que hemos especificado en el argumento, y lo llama el nombre que hemos especificado en `file`. Además, podemos usar el parámetro `sep` para indicar el símbolo de separación de columnas en el fichero que creemos y el parámetro `dec` para indicar la separación entre parte entera y parte decimal en los números; por ejemplo,

```
> A=iris[1:5, 1:4]
> A
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1          5.1         3.5          1.4         0.2
2          4.9         3.0          1.4         0.2
3          4.7         3.2          1.3         0.2
4          4.6         3.1          1.5         0.2
5          5.0         3.6          1.4         0.2
> write.table(A, file="trozoiris.txt")
```

guarda el *data frame A* en un fichero llamado `trozoiris.txt`. Ahora podemos volver a importar este fichero.

```
> B=read.table("trozoiris.txt", header=TRUE)   #Importamos la tabla
> B
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1          5.1         3.5          1.4         0.2
2          4.9         3.0          1.4         0.2
3          4.7         3.2          1.3         0.2
4          4.6         3.1          1.5         0.2
5          5.0         3.6          1.4         0.2
```

6.3. Cómo crear *data frames*

Además de trabajar con *data frames* importados o predefinidos, también podemos crearlos; para ello, organizaremos en forma de tabla algunos vectores, cada uno de los cuales contendrá las observaciones de una variable para un conjunto de individuos o casos, y de manera que los datos en todos estos vectores estén en el mismo orden de los individuos: es decir, que la primera entrada de cada vector corresponda a un mismo individuo, la segunda entrada de cada vector corresponda a otro individuo, y así sucesivamente.

Para construir un *data frame* a partir de unos vectores, se usa la instrucción `data.frame` aplicada a los vectores en el orden en el que queremos disponer las columnas de la tabla; de esta manera, las variables tomarán los nombres de los vectores. Estos nombres también se pueden especificar en el argumento de `data.frame`, entrando cada columna con una construcción de la forma

Nombre de la variable=*Vector con el contenido de la variable*.

Vamos a ilustrar esta función con un ejemplo sencillo, que arrastraremos durante buena parte de lo que queda de lección. Vamos a construir un *data frame* que contenga algunos datos sobre estudiantes; concretamente, este *data frame* tendrá tres variables: una primera columna con el sexo del estudiante, la segunda columna con su edad en años y la tercera con su número de hermanos.

```
> Sexo=c("Hombre","Hombre","Mujer","Hombre","Hombre",
   "Hombre", "Mujer")
> Edad=c(17,18,20,18,18,18,19)
> Hermanos=c(2,0,0,1,1,1,0)
> d.f1=data.frame(Sexo, Edad, Hermanos)
> d.f1
    Sexo Edad Hermanos
1 Hombre   17        2
2 Hombre   18        0
3  Mujer   20        0
4 Hombre   18        1
5 Hombre   18        1
6 Hombre   18        1
7  Mujer   19        0
```

Puesto que iremos modificando este *data frame*, vamos a guardar una copia de seguridad (un *backup*) en `d.f1bk` para poder recuperar su estado original si lo necesitamos.

```
> d.f1bk=d.f1
```

Es muy prudente y recomendable que guardéis siempre copias de seguridad de los *data frames* que generéis si vais a manipularlos, porque hay cambios en un *data frame* que son irreversibles y, por lo tanto, si os arrepentís de haber hecho algún cambio, bien podría ser que no lo pudierais deshacer.

Vamos a consultar algunas características del *data frame* `d.f1`.

```
> str(d.f1)
'data.frame': 7 obs. of  3 variables:
 $ Sexo    : Factor w/ 2 levels "Hombre","Mujer": 1 1 2 1 1 1 2
 $ Edad    : num  17 18 20 18 18 18 19
 $ Hermanos: num  2 0 0 1 1 1 0
> rownames(d.f1)
[1] "1" "2" "3" "4" "5" "6" "7"
```

Vemos que la primera variable es un factor con dos niveles, y las otras dos son vectores de números (`num`). Vemos también que `R` ha asignado como identificadores de las filas los números del 1 al 7, pero los considera palabras.

La función `data.frame` dispone de algunos parámetros que permiten ajustar a nuestras necesidades el *data frame* que creamos. Los más útiles son los siguientes:

- `row.names`. Sirve para especificar los identificadores de las filas.
- `stringsAsFactors`. Por defecto, `data.frame` convierte los vectores de palabras en factores. Con `stringsAsFactors=FALSE`, imponemos que los vectores de palabras se mantengan como tales en el *data frame*.

Veamos un ejemplo de aplicación de estos dos parámetros.

```
> d.f2=data.frame(Sexo, Edad, Hermanos, stringsAsFactors=FALSE,
  row.names=c("E1","E2","E3","E4","E5","E6","E7"))
> d.f2
     Sexo Edad Hermanos
E1 Hombre   17        2
E2 Hombre   18        0
E3  Mujer   20        0
E4 Hombre   18        1
E5 Hombre   18        1
E6 Hombre   18        1
E7  Mujer   19        0
> str(d.f2)
'data.frame': 7 obs. of  3 variables:
 $ Sexo    : chr  "Hombre" "Hombre" "Mujer" "Hombre" ...
 $ Edad    : num  17 18 20 18 18 18 19
 $ Hermanos: num  2 0 0 1 1 1 0
> d.f2$Sexo
```

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

Como vemos, el efecto de `stringsAsFactors=FALSE` ha sido que la variable `sexo` es ahora un vector de palabras, y el del parámetro `row.names` ha sido llamar `E1,E2,...,E7` a las filas.

Si hemos asignado identificadores a las filas, podemos usarlos para extraer subtablas del *data frame* (aunque también podemos seguir usando los números de las filas). Recordad que, si usáis los identificadores, como son palabras, hay que escribirlos entre comillas.

```
> d.f2[c("E1", "E2"), ] #Subtabla con las filas E1 y E2
     Sexo Edad Hermanos
E1 Hombre   17        2
E2 Hombre   18        0
```

Otra manera de crear un *data frame* con `R` es usando el editor de datos del que ya hemos hablado en la Lección 3.[3] Recordaréis que, para abrir un objeto de datos con este editor, se le aplica la función `fix`. `R` abre entonces el objeto en una nueva ventana de edición. Los cambios que realicemos en un objeto con el editor de datos se guardarán cuando cerremos esta ventana.

Para crear un *data frame* con el editor de datos, lo primero que hay que hacer es crear un *data frame* con la primera fila, y luego abrirlo con el editor para ir añadiendo filas (y columnas, si se desea). La apariencia exacta de esta ventana y la manera de editar el *data frame* dependen de la interficie de `R` que se use; por ejemplo, en el *RStudio* de Mac OS X, entraríamos

```
> d.f3=data.frame(Sexo=c("Hombre"), Edad=c(17), Hermanos=c(2))
> fix(d.f3)
```

y se abriría la ventana que muestra la Figura 1.1.

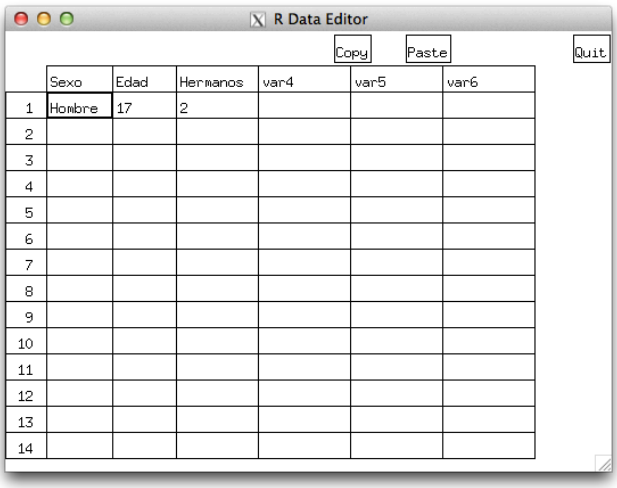

Figura 6.1. Editor de *data frames* del *RStudio* de Mac OS X.

[3] El uso del editor de *RStudio* para manipular *data frames* posiblemente requiera la instalación de algún programa auxiliar; por ejemplo, en el caso del Mac OS X, se tiene que tener instalada la última versión de *XQuartz*, que será el programa en el que se abrirá el editor. El instalador de *XQuartz* se puede descargar de `http://xquartz.macosforge.org`.

6.4. Cómo modificar un *data frame*

En esta sección veremos la manera de modificar un *data frame* una vez creado o importado.

Para cambiar los nombres de las variables, podemos usar la instrucción

`names(`*data frame*`)=`*vector con los nombres de las variables*.

Volvamos al `d.f1`. Tras recordar su estructura, cambiaremos los nombres de sus variables, sustituyéndolos por sus iniciales, y volveremos a consultar su estructura para ver cómo han cambiado estos nombres.

```
> str(d.f1)
'data.frame': 7 obs. of  3 variables:
 $ Sexo    : Factor w/ 2 levels "Hombre","Mujer": 1 1 2 1 1 1 2
 $ Edad    : num  17 18 20 18 18 18 19
 $ Hermanos: num  2 0 0 1 1 1 0
> names(d.f1)=c("S","E","H")
> str(d.f1)
'data.frame': 7 obs. of  3 variables:
 $ S: Factor w/ 2 levels "Hombre","Mujer": 1 1 2 1 1 1 2
 $ E: num  17 18 20 18 18 18 19
 $ H: num  2 0 0 1 1 1 0
```

Si sólo queremos cambiar el nombre de algunas variables, pero no de todas, basta redefinir el trozo correspondiente del vector `names`. Así, si en la nueva copia de `d.f1` queremos volver a cambiar el nombre de la variable `E` por `Edad`, podríamos usar una de las dos instrucciones siguientes:

`names(d.f1)[2]="Edad"` o `names(d.f1)[names(d.f1)=="E"]="Edad"`.

Con la primera, cambiaríamos el nombre de la segunda variable por `Edad`; con la segunda, cambiaríamos el nombre de la variable llamada `E` por `Edad`.

Para modificar los identificadores de las filas, podemos usar la instrucción

`rownames(`*data frame*`)=`*vector con los nombres de las filas*.

Como cada identificador ha de determinar el individuo al cual corresponde la fila, conviene que estos identificadores sean todos diferentes.

```
> rownames(d.f1)=paste("Alumno", 1:7, sep=" ")
> d.f1
              S  E H
Alumno 1 Hombre 17 2
Alumno 2 Hombre 18 0
Alumno 3  Mujer 20 0
Alumno 4 Hombre 18 1
Alumno 5 Hombre 18 1
Alumno 6 Hombre 18 1
Alumno 7  Mujer 19 0
```

Como podéis deducir de su efecto, la función `paste` *pega* vectores, entrada a entrada, usando como separador el que especificamos con `sep`; el separador por defecto es un espacio en blanco, por lo que no hubiera hecho falta especificarlo, lo hemos hecho sólo para mostrar el parámetro. Si en lugar de pegar un vector pegamos un elemento, éste se entiende como un vector constante formado por el número adecuado de copias.

Podemos modificar los nombres de las filas y de las columnas simultáneamente con la instrucción

> `dimnames(`*data frame*`)=list(`*vector con los nombres de las filas*, *vector con los nombres de las columnas*`)`

Por ejemplo, vamos a cambiar de golpe en `d.f1` los identificadores de las filas, para que ahora sean números romanos, y los nombres de las variables, para que pasen a ser `V1,V2,V3`:

```
> dimnames(d.f1)=list(c("I","II","III","IV","V","VI","VII"),
  c("V1","V2","V3"))
> d.f1
          V1 V2 V3
I    Hombre 17  2
II   Hombre 18  0
III   Mujer 20  0
IV   Hombre 18  1
V    Hombre 18  1
VI   Hombre 18  1
VII   Mujer 19  0
```

Para modificar entradas concretas de un *data frame*, podemos usar el editor de datos que se abre con `fix`, o podemos usar instrucciones similares a las que usábamos en los vectores y las matrices para modificar entradas.

```
> d.f1=d.f1bk #Volvemos a la copia guardada
> d.f1[1,2]="Joven" #Sustituimos la entrada (1,2) por la palabra
   "Joven"
> str(d.f1)
'data.frame': 7 obs. of  3 variables:
 $ Sexo    : Factor w/ 2 levels "Hombre","Mujer": 1 1 2 1 1 1 2
 $ Edad    : chr  "Joven" "18" "20" "18" ...
 $ Hermanos: num  2 0 0 1 1 1 0
```

¡Vaya! Al introducir un dato de tipo palabra en la variable `Edad`, toda la columna ha pasado a ser un vector de palabras: recordemos que `R` considera del mismo tipo de datos todas las entradas de una columna de un *data frame*; y ahora ya no lo podemos arreglar volviendo a poner un número:

```
> d.f1[1,2]=17
> str(d.f1)
'data.frame': 7 obs. of  3 variables:
 $ Sexo    : Factor w/ 2 levels "Hombre","Mujer": 1 1 2 1 1 1 2
 $ Edad    : chr  "17" "18" "20" "18" ...
 $ Hermanos: num  2 0 0 1 1 1 0
```

Parece que hemos estropeado definitivamente el *data frame* `d.f1`. Pero no es así: simplemente tenemos que volver a redefinir la variable `Edad` como un vector numérico.

Para modificar una columna entera, hay que igualar su nombre (que, recordemos, se especifica añadiendo al nombre del *data frame* el sufijo formado por el símbolo `$` seguido del nombre de la variable dentro del *data frame*) a su nuevo valor:

data frame`$`*variable*`=`*nuevo valor*.

Este nuevo valor puede ser en concreto el resultado de un cambio de tipo de datos aplicado a la variable. `R` dispone de una serie de funciones de la forma `as.`*tipo_de_objeto*, que convierten el objeto al tipo que expresa el nombre de la función. Ya hemos visto en la Sección 3.4 la función `as.factor`, que transforma un vector en un factor. Otras funciones de este estilo son `as.character`, `as.integer` o `as.numeric`, que transforman todos los datos de un objeto en palabras, números enteros o números reales, respectivamente.

Así, para transformar la variable `Edad` de la versión estropeada de `d.f1` en un vector numérico, basta entrar lo siguiente:

```
> d.f1$Edad=as.numeric(d.f1$Edad)
> str(d.f1)
'data.frame': 7 obs. of  3 variables:
 $ Sexo    : Factor w/ 2 levels "Hombre","Mujer": 1 1 2 1 1 1 2
 $ Edad    : num  17 18 20 18 18 18 19
 $ Hermanos: num  2 0 0 1 1 1 0
```

Un error típico al intentar modificar la variable `Edad` es entrar sólo `as.numeric(d.f1$Edad)`; con esta instrucción, obtenemos como resultado el vector $17, 18, 20, 18, 18, 18, 19$, pero no se modifica la variable `Edad` del *data frame*. Tampoco sirve entrar `Edad=as.numeric(d.f1$Edad)`, porque esto define un vector llamado `Edad` de entradas $17, 18, 20, 18, 18, 18, 19$, pero tampoco modifica la variable `Edad` del *data frame*. La manera correcta de hacerlo es mediante

data frame`$`*variable*`=`*nuevo valor*.

Veamos otros ejemplos. Vamos a convertir la variable `Sexo` en un vector de palabras:

```
> d.f1$Sexo=as.character(d.f1$Sexo)
> str(d.f1)
'data.frame': 7 obs. of  3 variables:
 $ Sexo    : chr  "Hombre" "Hombre" "Mujer" "Hombre" ...
 $ Edad    : num  17 18 20 18 18 18 19
 $ Hermanos: num  2 0 0 1 1 1 0
```

Y ahora vamos a convertir la variable `Hermanos` en un factor ordenado:

```
> d.f1$Hermanos=ordered(d.f1$Hermanos, levels=c(0,1,2))
> str(d.f1)
'data.frame': 7 obs. of  3 variables:
 $ Sexo    : chr  "Hombre" "Hombre" "Mujer" "Hombre" ...
 $ Edad    : num  17 18 20 18 18 18 19
 $ Hermanos: Ord.factor w/ 3 levels "0"<"1"<"2": 3 1 1 2 2 2 1
```

Naturalmente, podemos cambiar el contenido de toda una variable de otras maneras que no sea sólo modificar su tipo de datos: basta igualarla a su nuevo valor.

```
> d.f1$Edad="Joven"
> d.f1
    Sexo  Edad Hermanos
1 Hombre Joven        2
2 Hombre Joven        0
3  Mujer Joven        0
4 Hombre Joven        1
5 Hombre Joven        1
6 Hombre Joven        1
7  Mujer Joven        0
```

Al igualar la variable `Edad` a la palabra «Joven», la ha substituido por el correspondiente vector constante.

6.5. Cómo añadir filas y columnas a un *data frame*

Podemos añadir filas a un *data frame* definiéndolas con *data frame*`[`*fila*`, ]=c(...)`; pero esta construcción no es muy recomendable, porque si no vamos con cuidado puede provocar resultados no deseados:

- Las filas que añadimos de esta manera son vectores, y por lo tanto sus entradas han de ser todas del mismo tipo. Por consiguiente, esta opción sólo se puede usar en *data frames* cuyas variables contengan todas el mismo tipo de datos.
- Si no añadimos las filas inmediatamente siguientes a la última del *data frame*, los valores entre su última fila y las que añadimos quedarán no definidos, y aparecerán como `NA`; esto puede ser un problema a la hora de aplicar funciones al *data frame* y además estropea los factores.

La mejor manera de añadir filas a un *data frame* es organizándolas en un nuevo *data frame* con los mismos nombres de las variables, y a continuación concatenarlas al *data frame* usando la función `rbind` que ya usábamos para matrices.

```
> d.f1=d.f1bk  #Volvemos a la copia original
> d.f1
    Sexo Edad Hermanos
1 Hombre   17        2
2 Hombre   18        0
3  Mujer   20        0
4 Hombre   18        1
5 Hombre   18        1
6 Hombre   18        1
7  Mujer   19        0
> nuevas.filas=data.frame(Sexo=c("Hombre","Hombre"), Edad=c(18,18),
  Hermanos=c(1,2))
> d.f1=rbind(d.f1, nuevas.filas)
> d.f1
    Sexo Edad Hermanos
1 Hombre   17        2
```

```
2 Hombre   18        0
3  Mujer   20        0
4 Hombre   18        1
5 Hombre   18        1
6 Hombre   18        1
7  Mujer   19        0
8 Hombre   18        1
9 Hombre   18        2
> str(d.f1)   #Los tipos de las variables no han cambiado
'data.frame': 9 obs. of  3 variables:
 $ Sexo    : Factor w/ 2 levels "Hombre","Mujer": 1 1 2 1 1 1 2 1 1
 $ Edad    : num  17 18 20 18 18 18 19 18 18
 $ Hermanos: num  2 0 0 1 1 1 0 1 2
```

Para añadir una variable, basta especificar el valor de la columna correspondiente; por ejemplo, vamos a añadir a `d.f1` una nueva variable llamada `Nueva` con el producto de la edad por el número de hermanos:

```
> d.f1$Nueva=d.f1$Edad*d.f1$Hermanos
> d.f1
    Sexo Edad Hermanos Nueva
1 Hombre   17        2    34
2 Hombre   18        0     0
3  Mujer   20        0     0
4 Hombre   18        1    18
5 Hombre   18        1    18
6 Hombre   18        1    18
7  Mujer   19        0     0
8 Hombre   18        1    18
9 Hombre   18        2    36
```

También podemos concatenar columnas a un *data frame* usando la función `cbind`; en este caso, se puede añadir directamente la columna, sin necesidad de convertirla previamente en un *data frame*. La variable añadida ha de tener la misma longitud que las variables del *data frame* original; en caso contrario, se añadirán valores `NA` a las variables del *data frame* original o a la variable que añadimos hasta completar la misma longitud.

```
> d.f1=cbind(d.f1, Grado=rep("Biología",9))
> d.f1
     Sexo Edad Hermanos Nueva    Grado
1 Hombre   17        2    34 Biología
2  Mujer   18        0     0 Biología
3  Mujer   20        0     0 Biología
4  Mujer   18        1    18 Biología
5 Hombre   18        1    18 Biología
6 Hombre   18        1    18 Biología
7  Mujer   19        0     0 Biología
8 Hombre   18        1    18 Biología
9 Hombre   18        2    36 Biología
> str(d.f1)
'data.frame': 9 obs. of  5 variables:
```

```
 $ Sexo    : Factor w/ 2 levels "Hombre","Mujer": 1 1 2 1 1 1 2 1 1
 $ Edad    : num  17 18 20 18 18 18 19 18 18
 $ Hermanos: num  2 0 0 1 1 1 0 1 2
 $ Nueva   : num  34 0 0 18 18 18 0 18 36
 $ Grado   : Factor w/ 1 level "Biología": 1 1 1 1 1 1 1 1 1
```

Finalmente, hay que recordar que también podemos usar el editor de datos para añadir (o eliminar) filas o columnas a un *data frame*.

6.6. Cómo seleccionar trozos de un *data frame*

Veamos ahora con más detalle cómo podemos extraer trozos de un *data frame*; el método básico es similar al que usábamos en las matrices. De hecho, para seleccionar un trozo de un *data frame*, podemos hacerlo exactamente como en las matrices, especificando los índices de las filas y columnas que nos interesen dentro de corchetes `[ , ]`. Pero como los individuos y las variables tienen nombres, también podemos usarlos para especificar filas y columnas.

Por ejemplo, supongamos que queremos crear un nuevo *data frame* formado por las dos primeras filas de `d.f1`; podríamos hacerlo, al menos, de las tres maneras siguientes:

```
> d.f1=d.f1bk  #Volvemos a la copia original
> d.f1[1:2, ]   #Con los índices de las filas
    Sexo Edad Hermanos
1 Hombre   17        2
2 Hombre   18        0
> d.f1[c("1","2"), ]   #Con los identificadores de las filas
    Sexo Edad Hermanos
1 Hombre   17        2
2 Hombre   18        0
> d.f1[rownames(d.f1)[1:2], ]   #Con un subvector del vector de
   identificadores de las filas
    Sexo Edad Hermanos
1 Hombre   17        2
2 Hombre   18        0
```

Podemos usar este tipo de instrucciones para obtener las filas en un orden diferente del que presentan en el *data frame* original.

```
> d.f1[c(2,3,1), ]
    Sexo Edad Hermanos
2 Hombre   18        0
3  Mujer   20        0
1 Hombre   17        2
```

También podemos seleccionar filas de un *data frame* mediante una condición lógica; en este caso, nos quedaremos sólo con los individuos que satisfagan esta condición. Por ejemplo, si queremos seleccionar los estudiantes de `d.f1` de menos de 19 años, podemos usar la construcción siguiente:

```
> d.f1[d.f1$Edad<19, ]
    Sexo Edad Hermanos
```

```
1 Hombre   17         2
2 Hombre   18         0
4 Hombre   18         1
5 Hombre   18         1
6 Hombre   18         1
```

Esta condición lógica puede involucrar más de una variable.

```
> d.f1[d.f1$Edad<19 & d.f1$Hermanos==1, ]
    Sexo Edad Hermanos
4 Hombre   18         1
5 Hombre   18         1
6 Hombre   18         1
```

En cada caso, hemos obtenido un *data frame*. Vamos a llamar `d.f.18` al primero.

```
> d.f.18=d.f1[d.f1$Edad<19, ]
> d.f.18
    Sexo Edad Hermanos
1 Hombre   17         2
2 Hombre   18         0
4 Hombre   18         1
5 Hombre   18         1
6 Hombre   18         1
> str(d.f.18)
'data.frame': 5 obs. of  3 variables:
 $ Sexo    : Factor w/ 2 levels "Hombre","Mujer": 1 1 1 1 1
 $ Edad    : num  17 18 18 18 18
 $ Hermanos: num  2 0 1 1 1
```

Como vemos, las columnas que son factores heredan en estos sub*data frames* todos los niveles del factor original, aunque no aparezcan en el trozo que hemos extraído. Podemos borrar los niveles sobrantes de todos los factores redefiniendo el *data frame* como el resultado de aplicarle la función `droplevels`.

```
> d.f.18=droplevels(d.f.18)
> str(d.f.18)
'data.frame': 5 obs. of  3 variables:
 $ Sexo    : Factor w/ 1 level "Hombre": 1 1 1 1 1
 $ Edad    : num  17 18 18 18 18
 $ Hermanos: num  2 0 1 1 1
```

Naturalmente, el mismo tipo de construcción se puede utilizar para definir un *data frame* formado sólo por algunas variables de la tabla original, o incluso por sólo algunas filas y columnas. Si sólo queremos definir la subtabla quedándonos con algunas variables, basta aplicar el nombre del *data frame* al vector de variables (sin necesidad de dejar el espacio en blanco seguido de una coma que serviría para indicar que nos referimos a columnas y no a filas); por ejemplo, con las cuatro instrucciones siguientes obtenemos cada vez el mismo resultado, un *data frame* formado por las dos primeras variables de `d.f1`:

```
> d.f1[ , c(1,2)]  #Con la coma
```

```
    Sexo Edad
1 Hombre   17
2 Hombre   18
3  Mujer   20
4 Hombre   18
5 Hombre   18
6 Hombre   18
7  Mujer   19
> d.f1[c(1,2)]   #Sin la coma
    Sexo Edad
1 Hombre   17
2 Hombre   18
3  Mujer   20
4 Hombre   18
5 Hombre   18
6 Hombre   18
7  Mujer   19
> d.f1[-3]   #Eliminamos la tercera columna
    Sexo Edad
1 Hombre   17
2 Hombre   18
3  Mujer   20
4 Hombre   18
5 Hombre   18
6 Hombre   18
7  Mujer   19
> d.f1[c("Sexo","Edad")] #Especificamos los nombres
    Sexo Edad
1 Hombre   17
2 Hombre   18
3  Mujer   20
4 Hombre   18
5 Hombre   18
6 Hombre   18
7  Mujer   19
```

Esta construcción se puede usar también para reordenar las columnas de un *data frame*; por ejemplo:

```
> d.f1=d.f1[c("Edad","Hermanos","Sexo")]
> d.f1
  Edad Hermanos   Sexo
1   17        2 Hombre
2   18        0 Hombre
3   20        0  Mujer
4   18        1 Hombre
5   18        1 Hombre
6   18        1 Hombre
7   19        0  Mujer
```

El paquete `dplyr` incluye la función `select` que amplía las posibilidades para especificar las variables que queremos extraer de un *data frame*; por ejemplo:

- `select(`*data frame*`, starts_with("`x`"))` extrae del *data frame* las variables cuyo nombre empieza con la palabra x.
- `select(`*data frame*`, ends_with("`x`"))` extrae del *data frame* las variables cuyo nombre termina con la palabra x.
- `select(`*data frame*`, contains("`x`"))` extrae del *data frame* las variables cuyo nombre contiene en algún sitio la palabra x.

```
> #Instalamos y cargamos el paquete dplyr
...
> iris_Petal=select(iris, starts_with("Petal"))
> head(iris_Petal, 4)
  Petal.Length Petal.Width
1          1.4         0.2
2          1.4         0.2
3          1.3         0.2
4          1.5         0.2
> iris_Length=select(iris, ends_with("Length"))
> head(iris_Length, 4)
  Sepal.Length Petal.Length
1          5.1          1.4
2          4.9          1.4
3          4.7          1.3
4          4.6          1.5
```

Para más información sobre otras habilidades de la función `select`, podéis consultar su `help`.

Como podéis constatar, hay muchas maneras de extraer una misma subtabla de un *data frame* dado; veamos una última posibilidad, usando la función `subset`. La instrucción

`subset(`*data frame*`,` *condición*`, select=`*columnas*`)`

extrae del *data frame* las filas que cumplen la `condición` y las columnas especificadas en el `select`; si queremos todas las filas, no hay que especificar ninguna condición, y si queremos todas las columnas, no hace falta especificar el parámetro `select`. Así, si del *data frame* `iris` queremos extraer un *data frame* formado sólo por las plantas de especie `virginica`, podemos usar la instrucción siguiente:

```
> iris.vir=subset(iris, Species=="virginica")
> head(iris.vir, 5)
    Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
101          6.3         3.3          6.0         2.5 virginica
102          5.8         2.7          5.1         1.9 virginica
103          7.1         3.0          5.9         2.1 virginica
104          6.3         2.9          5.6         1.8 virginica
105          6.5         3.0          5.8         2.2 virginica
```

Fijaos en que las variables en la *condición* se especifican con su nombre, sin añadir antes el nombre del *data frame*.

En este *data frame*, la variable `Species` es redundante, porque todos los individuos toman el mismo valor; por lo tanto, podríamos haberla eliminado al construirlo.

```
> iris.vir=subset(iris, Species=="virginica", select=1:4)
> head(iris.vir, 5)
    Sepal.Length Sepal.Width Petal.Length Petal.Width
101          6.3         3.3          6.0         2.5
102          5.8         2.7          5.1         1.9
103          7.1         3.0          5.9         2.1
104          6.3         2.9          5.6         1.8
105          6.5         3.0          5.8         2.2
```

Las filas de este *data frame* han heredado los identificadores del *data frame* `iris`. Si esto nos molesta, los podemos cambiar.

```
> rownames(iris.vir)=1:length(rownames(iris.vir))
> head(iris.vir, 5)
  Sepal.Length Sepal.Width Petal.Length Petal.Width
1          6.3         3.3          6.0         2.5
2          5.8         2.7          5.1         1.9
3          7.1         3.0          5.9         2.1
4          6.3         2.9          5.6         1.8
5          6.5         3.0          5.8         2.2
```

6.7. Cómo aplicar una función a las variables de un *data frame*

La mejor manera de aplicar una función a todas las columnas de un *data frame* en un solo paso es por medio de la instrucción

`sapply(`*data frame*`,` *función*`).`

A modo de ejemplo, vamos a aplicar algunas funciones a las columnas numéricas del *data frame* `iris`:

```
> str(iris)
'data.frame': 150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1
    1 1 1 1 1 1 1 ...
> sapply(iris[ ,1:4], mean) #Medias de las variables numéricas
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
    5.843333     3.057333     3.758000     1.199333
> sapply(iris[ ,1:4], sum) #Sumas de las variables numéricas
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
       876.5        458.6        563.7        179.9
> f=function(x){sqrt(sum(x^2))}
```

```
> sapply(iris[ ,1:4], f) #Normas euclídeas de las variables numé
   ricas
Sepal.Length  Sepal.Width Petal.Length  Petal.Width
    72.27621     37.82063     50.82037     17.38764
```

Es conveniente añadir en el argumento de `sapply` el parámetro `na.rm=TRUE`, sin el cual el valor que devolverá la función para las columnas que contengan algún `NA` será `NA`.

```
> D=data.frame(V1=c(1,2,NA,3), V2=c(2,5,2,NA))
> D
  V1 V2
1  1  2
2  2  5
3 NA  2
4  3 NA
> sapply(D, mean)
V1 V2
NA NA
> sapply(D, mean, na.rm=TRUE)
V1 V2
 2  3
```

A menudo querremos aplicar una función a variables de un *data frame* clasificadas por los niveles de un, o más de un, factor.; esto se puede hacer con la instrucción `aggregate`, cuya sintaxis básica es

`aggregate(`*variable(s)*`~`*factor(es)*`, data=`*data frame*`, FUN=`*función*`)`.

El resultado será un *data frame* con las variables y factores usados.

Por ejemplo, si queremos calcular las medias de las longitudes de los pétalos de las flores de cada una de las tres especies representadas en la tabla `iris`, podemos entrar la instrucción siguiente:

```
> aggregate(Petal.Length~Species, data=iris, FUN=mean, na.rm=TRUE)
     Species Petal.Length
1     setosa        1.462
2 versicolor        4.260
3  virginica        5.552
```

Hemos añadido `na.rm=TRUE` dentro del `aggregate` para que no tenga en cuenta los `NA` al calcular la media, por si acaso. Observad que el resultado es un *data frame* con variables `Species` y `Petal.Length`.

Si queremos aplicar la función a más de una variable, tenemos que agruparlas a la izquierda de la tilde con `cbind`.

```
> aggregate(cbind(Petal.Length, Petal.Width)~Species, data=iris,
   FUN=mean)
     Species Petal.Length  Petal.Width
1     setosa        1.462        0.246
2 versicolor        4.260        1.326
3  virginica        5.552        2.026
```

Si queremos separar las variables mediante más de un factor, tenemos que agruparlos a la derecha de la tilde con signos +: *factor1+factor2+...*; veamos un ejemplo.

El paquete `alr4` contiene la tabla de datos `Rateprof`, con los resultados globales de la evaluación de un grupo de profesores universitarios por parte de sus estudiantes. Algunas de sus variables son: `gender`, el sexo del profesor; `pepper`, que indica si en las encuestas se le ha considerado mayoritariamente atractivo o no; y `clarity` y `easiness`, que valoran, entre 1 y 5, la claridad de exposición y la accesibilidad del profesor, respectivamente. Vamos a calcular las medias de estas dos últimas variables agrupándolas por sexo y atractivo.

```
> #Instalamos y cargamos el paquete alr4
...
> aggregate(cbind(clarity,easiness)~gender+pepper, data=Rateprof,
   FUN=mean)
  gender pepper  clarity easiness
1 female     no 3.341391 3.147606
2   male     no 3.451456 2.999056
3 female    yes 4.345082 3.599128
4   male    yes 4.371824 3.689741
```

Observamos que tanto la claridad como la accesibilidad medias de los profesores atractivos de ambos sexos son considerablemente mayores que las de sus colegas considerados no atractivos. Además, se considera a los profesores no atractivos menos accesibles que a las profesoras no atractivas, mientras que a los profesores atractivos se les considera *más* accesibles que a las profesoras atractivas.

6.8. Cómo añadir las variables de un *data frame* al entorno global

Hasta ahora, cada vez que queríamos referirnos a una variable de un *data frame*, teníamos que escribir el nombre del *data frame* seguido de `$` y el nombre de la variable. Aplicando `attach` a un *data frame*, hacemos que `R` entienda sus variables como globales y que las podamos usar por su nombre, sin necesidad de añadir delante el nombre del *data frame* y el símbolo `$`; esto puede ser útil para no tener que escribir mucho.

A modo de ejemplo, vamos a añadir las variables del *data frame* `iris` al entorno global de `R`:

```
> Petal.Length
Error: object 'Petal.Length' not found
> attach(iris)
> Petal.Length[1:30]
 [1] 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 1.5 1.6 1.4 1.1 1.2
[16] 1.5 1.3 1.4 1.7 1.5 1.7 1.5 1.0 1.7 1.9 1.6 1.6 1.5 1.4 1.6
```

Si ya hubiera existido una variable definida con el mismo nombre que una variable del *data frame* al que aplicamos `attach`, hubiéramos obtenido un mensaje de error al ejecutar esta función, y no se hubiera reescrito la variable global original.

La función `detach` devuelve la situación original, eliminando del entorno global las variables del *data frame*.

```
> detach(iris)
> Petal.Length
Error: object 'Petal.Length' not found
```

6.9. Guía rápida

- `data()` produce una lista con los objetos de datos a los que tenemos acceso.
- `head` aplicado a un *data frame* y un número n, nos muestra las primeras n filas del *data frame* (por defecto, 6).
- `tail` aplicado a un *data frame* y un número n, nos muestra las n últimas filas del *data frame* (por defecto, 6).
- `str` da la estructura global de un objeto de datos.
- `names` sirve para obtener un vector con los nombres de las columnas de un *data frame*, y también para modificar estos nombres.
- `rownames` sirve para obtener un vector con los identificadores de las filas de un *data frame*, y también para modificar estos identificadores.
- `dimnames` sirve para obtener una `list` formada por el vector de los identificadores de las filas y el vector de los nombres de las columnas de un *data frame*, y también para modificar estos vectores simultáneamente.
- `dim` da el número de filas y el número de columnas de un *data frame*.
- *data frame*`$`*variable* indica la *variable* del *data frame*.
- `read.table` permite importar un fichero externo en formato simple en un *data frame*. Algunos parámetros importantes:
  - `sep`: sirve para indicar los separadores de las columnas.
  - `header`: sirve para indicar si el fichero a importar tiene una primera fila con los nombres de las variables o no.
  - `dec`: sirve para especificar el signo que separa la parte entera de la decimal en los números.
  - `as.is=stringsAsFactors`: sirve para prohibir que las columnas de palabras se transformen en factores.
- `read.csv`, `read.xls`, `read.xlsx` (ambas del paquete `xlsx`), `read.mtb` y `read.spss` (ambas del paquete `foreign`) permiten importar en un *data frame* una tabla de datos en formato CSV, XLS, XLSX, Minitab o SPSS, respectivamente.
- `write.table(`*data frame*`,file="`*fichero*`")` exporta el *data frame* al *fichero* externo.
- `data.frame` crea un *data frame* con los vectores a los que se aplica. Algunos parámetros importantes:

    - `rownames`: sirve para especificar los identificadores de las filas.
    - `stringsAsFactors`: igualado a `FALSE`, impone que los vectores de palabras se mantengan como tales, y no como factores.

- `fix` abre un objeto de datos en el editor de datos.
- `paste(`x`,`y`,,sep="...")` pega los vectores x e y, entrada a entrada, usando como separador el valor del parámetro `sep`.
- `as.character`, `as.integer` y `as.numeric` transforman todos los datos de un objeto en palabras, números enteros o números reales, respectivamente.
- `droplevels` borra todos los niveles sobrantes de todos los factores de un *data frame*.
- `subset(`*data frame*`,`*condición*`,select=`*columnas*`)` define un *dataframe* con las filas del *data frame* que cumplen la `condición` y las columnas especificadas en el parámetro `select`.
- `sapply(`*data frame*`,`*función*`)` aplica la *función* a las columnas de un *data frame*.
- `aggregate` sirve para aplicar una función a una o varias variables de un *data frame* agrupando sus entradas por los niveles de uno o varios factores.
- `attach` añade las variables de un *data frame* al entorno global de `R`.
- `detach` deshace el efecto de `attach`.

6.10. Ejercicio

La tabla de datos `pulse.txt` que encontraréis en `http://bioinfo.uib.es/~recerca/RMOOC/pulse.txt` recoge una serie de informaciones de tipo general (altura, peso, sexo, si corren, si fuman y su nivel de actividad física: 1 si es bajo, 2 si es moderado y 3 si es alto) sobre algunos estudiantes matriculados en un curso de estadística de la Universidad Estatal de Pensilvania hace unos años. A estos estudiantes se les pidió que lanzasen una moneda al aire: a los que sacaron cara, se les hizo correr un minuto sin moverse del sitio, y los que sacaron cruz, descansaron un minuto. Todos los estudiantes (tanto los que corrieron como los que no) midieron sus pulsaciones por minuto antes y después de este minuto de ejercicio o descanso, y estas medidas también aparecen en esta tabla (en las variables `PuBefor` y `PuAfter`, respectivamente).

(a) ¿Cuántos estudiantes tomaron parte en este estudio? ¿Cuántos son hombres y cuántas mujeres?

(b) Calculad el porcentaje medio de variación en el número de pulsaciones tras el minuto de ejercicio o descanso de los estudiantes que corrieron (se indica con el valor `yes` en la variable `ran?`) y de los que no. ¿Hay mucha diferencia?

(c) Calculad el porcentaje medio de incremento en el número de pulsaciones tras el minuto de ejercicio sólo para los estudiantes que corrieron, pero ahora distinguiendo los hombres de las mujeres. ¿Cuál de los dos incrementos medios es mayor?

(d) Calculad el porcentaje medio de incremento en el número de pulsaciones tras el minuto de ejercicio para los estudiantes que corrieron, pero ahora distinguiendo los estudiantes que fuman de los que no. ¿Cuál de los dos incrementos medios es mayor?

(e) Calculad el número medio de pulsaciones antes del minuto de ejercicio o descanso de todos los estudiantes, separados según su nivel de actividad física. ¿Se observa alguna diferencia significativa?