

Introducción al análisis y manejo de datos con el programa

Isaac Subirana & Joan Vila

`isubirana@imim.es & jvila@imim.es`

RICAD

Research on Inflammatory and Cardiovascular Disorders Program
(IMIM-Parc de Salut Mar)

Junio 1011

En página principal de “R” <http://www.r-project.org> se puede leer:
*R is a **language and environment for statistical computing and graphics**. It is a GNU project which is similar to the S language and environment which was developed at Bell Laboratories (formerly AT&T, now Lucent Technologies) by John Chambers and colleagues. R can be considered as a different implementation of S. There are some important differences, but much code written for S runs unaltered under R.*

*R provides a wide variety of **statistical** (linear and nonlinear modelling, classical statistical tests, time-series analysis, classification, clustering, ...) **and graphical techniques** , and is highly extensible. The *S* language is often the vehicle of choice for research in statistical methodology, and R provides an Open Source route to participation in that activity.*

*One of R's strengths is the ease with which **well-designed publication-quality plots** can be produced, including mathematical symbols and formulae where needed. Great care has been taken over the defaults for the minor design choices in graphics, but the user retains full control.*

*R is available as **Free Software** under the terms of the Free Software Foundation's GNU General Public License in source code form. It compiles and runs on a wide variety of UNIX platforms and similar systems (including FreeBSD and Linux), Windows and MacOS.*

En **Wikipedia** añade:

*R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is now developed by the R Development Core Team. It is named partly after the first names of the first two R authors (Robert Gentleman and Ross Ihaka), and partly as a play on the name of S. The R language **has become a de facto standard among statisticians for the development of statistical software.***

Sólo añadir a modo introductorio que es una programa orientado a objetos

a

Devuelve: *Error: objeto "a" no encontrado*

```
> a <- 3
> a
```

```
[1] 3
```

Ahora a es un *objeto* que contiene el valor 3

```
> age <- c(24, 29, 53, 45, 32, 28)
> sex <- c(0, 1, 1, 0, 0, 1)
> age
```

```
[1] 24 29 53 45 32 28
```

```
> sex
```

```
[1] 0 1 1 0 0 1
```

Ahora **age** es un *objeto* que contiene una colección de **seis valores de edad** y **sex** es un *objeto* que contiene una colección de **seis valores de sexo**

Es un programa que se basa en **funciones** por ejemplo la función *t.test*

```
> resu <- t.test(age ~ sex)
> resu
```

```
Welch Two Sample t-test
```

```
data: age by sex
t = -0.2939, df = 3.706, p-value = 0.7846
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -32.25237  26.25237
sample estimates:
mean in group 0 mean in group 1
 33.66667      36.66667
```

```
> round(resu$p.value, 3)
```

```
[1] 0.785
```

Varias funciones se almacenan en **libraries** como se verá a lo largo del curso.

Antes de cargar nuevos datos es mejor eliminar todos los posibles objetos del espacio de trabajo que hayan podido quedar de sesiones anteriores. La instrucción es:

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

Si se quiere cargar unos datos que están en formato SPSS primero hay que cargar el paquete que permite cargar datos de SPSS.


```
> library(foreign)
```

Especificar la carpeta dónde se va a trabajar. Obsérvese que el separador es la barra y no la contrabarra.

```
> setwd("C:/cursoR/data")
```


Para cargar la base de datos “partoFin.sav” con formato SPSS y almacenarla en un *objeto* que se llamará “datos”:

```
> datos <- read.spss("partoFin.sav", use.value.labels = FALSE,  
+   to.data.frame = TRUE)
```

El  es mayúscula sensible, así que para evitar errores es mejor convertir el nombre de todas las variables a minúsculas

```
> names(datos) <- tolower(names(datos))
```

Para guardar la base de datos en formato :

```
> save(datos, file = "c:/CursoR/data/datos.Rdata")
```

Para recuperarla, si primero eliminamos todos los objetos que haya en memoria, observaremos que no nos queda ninguno:

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

```
> objects()
```

```
character(0)
```

Y si ahora la cargamos:

```
> load(file = "c:/CursoR/data/datos.Rdata")
```

```
> objects()
```

```
[1] "datos"
```

Para saber el nombre de las variables:

```
> names(datos)
```

```
[1] "id"      "ini"      "dia_nac"  "dia_entr" "ulti_lac" "tx"  
[7] "edad"    "peso"     "sexo"     "tip_par"  "hermanos" "fuma_an"  
[13] "fuma_de" "horas_an" "horas_de" "naci_ca"  "masde12"  "sem_lac"
```

Cuántas filas (registros) tiene la base de datos:

```
> nrow(datos)
```

```
[1] 28
```

Cuántas columnas (variables) tiene la base de datos:

```
> ncol(datos)
```

```
[1] 18
```

Para saber las filas y columnas:

```
> dim(datos)
```

```
[1] 28 18
```

Observe que el resultado de `dim(datos)` nos devuelve dos *componentes* (dos números). El primero por ejemplo nos informa del número de registros que tiene nuestra base de datos:

```
> n <- dim(datos)[1]
> n

[1] 28
```

Para ver todos los datos. Obsérvese que las fechas se han importado tal como las tiene almacenadas internamente el SPSS, es decir, los segundos que han pasado desde el inicio del año Gregoriano (0:0 horas del 14-Oct-1582) hasta la fecha en cuestión.

> datos

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo	tip_par
1	1	GADI	13211856000	13212288000	13215484800	2	24	3.38	2	1
2	2	CAEL	13211942400	13212460800	13233110400	2	27	2.50	1	2
3	3	COMO	13212028800	13213324800	13212633600	1	44	3.15	2	2
4	4	VIMU	13212201600	13212633600	13227926400	2	25	2.74	1	1
5	5	PAVI	13212288000	13212806400	13212892800	1	27	3.60	2	2
6	6	PASA	13212374400	13213324800	13212979200	1	36	2.65	2	1
7	7	VERI	13212374400	13213238400	13219632000	2	35	2.97	1	1
8	8	ADJU	13212460800	13212806400	13219718400	2	23	3.20	1	2
9	9	BEMI	13212547200	13213670400	13218595200	1	40	2.40	2	2
10	10	JUNA	13212633600	13213411200	13221100800	2	32	2.10	2	2
11	11	LOKO	13212892800	13213411200	13217731200	1	26	3.45	1	2
12	12	FRFU	13212979200	13213584000	13234752000	2	29	3.45	2	2
13	13	FUFU	13213756800	13214707200	13214361600	1	36	3.40	2	2
14	14	POCA	13214361600	13215312000	13224643200	2	36	3.05	1	2
15	15	LOLO	13214361600	13214448000	13214966400	1	17	3.60	1	2
16	16	BOPE	13214448000	13215571200	13230777600	1	40	3.40	1	2
17	17	ANZO	13214793600	13215312000	13226889600	2	27	3.15	1	2
18	18	MEVE	13214793600	13215571200	13236566400	2	32	3.32	2	2
19	19	TOPO	13214880000	13215484800	13222137600	1	29	2.65	2	2
20	20	PUPI	13214966400	13215225600	13222224000	2	21	4.46	2	2
21	21	ROPA	13214966400	13215830400	13217385600	1	35	3.15	2	2
22	22	LOMA	13215052800	13215571200	13234406400	2	27	3.70	1	2
23	23	CEMA	13215139200	13215571200	13216953600	1	24	3.79	1	2
24	24	CAGI	13215225600	13215398400	13215830400	2	18	3.75	2	2
25	25	GRSE	13215312000	13216176000	13216521600	1	34	2.95	2	2

Para ver sólo los datos de los 7 primeros individuos

```
> datos[1:7, ]
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo	tip_par
1	1	GADI	13211856000	13212288000	13215484800	2	24	3.38	2	1
2	2	CAEL	13211942400	13212460800	13233110400	2	27	2.50	1	2
3	3	COMO	13212028800	13213324800	13212633600	1	44	3.15	2	2
4	4	VIMU	13212201600	13212633600	13227926400	2	25	2.74	1	1
5	5	PAVI	13212288000	13212806400	13212892800	1	27	3.60	2	2
6	6	PASA	13212374400	13213324800	13212979200	1	36	2.65	2	1
7	7	VERI	13212374400	13213238400	13219632000	2	35	2.97	1	1

	hermanos	fuma_an	fuma_de	horas_an	horas_de	naci_ca	masde12	sem_lac
1	2	1	0	6	2	3	0	6
2	1	0	0	2	2	1	1	35
3	1	0	1	3	0	1	0	1
4	1	1	1	11	6	2	1	26
5	1	1	0	10	22	1	0	1
6	2	0	0	9	9	1	0	1
7	2	0	0	8	8	1	0	12

Para ver sólo los datos de las 5 primeras variables

```
> datos[, 1:5]
```

	id	ini	dia_nac	dia_entr	ulti_lac
1	1 GADI	13211856000	13212288000	13215484800	
2	2 CAEL	13211942400	13212460800	13233110400	
3	3 COMO	13212028800	13213324800	13212633600	
4	4 VIMU	13212201600	13212633600	13227926400	
5	5 PAVI	13212288000	13212806400	13212892800	
6	6 PASA	13212374400	13213324800	13212979200	
7	7 VERI	13212374400	13213238400	13219632000	
8	8 ADJU	13212460800	13212806400	13219718400	
9	9 BEMI	13212547200	13213670400	13218595200	
10	10 JUNA	13212633600	13213411200	13221100800	
11	11 LOKO	13212892800	13213411200	13217731200	
12	12 FRFU	13212979200	13213584000	13234752000	
13	13 FUFU	13213756800	13214707200	13214361600	
14	14 POCA	13214361600	13215312000	13224643200	
15	15 LOLO	13214361600	13214448000	13214966400	
16	16 BOPE	13214448000	13215571200	13230777600	
17	17 ANZO	13214793600	13215312000	13226889600	
18	18 MEVE	13214793600	13215571200	13236566400	
19	19 TOPO	13214880000	13215484800	13222137600	
20	20 PUPI	13214966400	13215225600	13222224000	
21	21 ROPA	13214966400	13215830400	13217385600	
22	22 LOMA	13215052800	13215571200	13234406400	
23	23 CEMA	13215139200	13215571200	13216953600	
24	24 CAGI	13215225600	13215398400	13215830400	
25	25 GRSE	13215312000	13216176000	13216521600	
26	26 GUMA	13215398400	13215916800	13227494400	
27	27 PERI	13215398400	13215830400	13230518400	
28	28 MAPE	13215398400	13215830400	13225075200	

Para ver sólo la información sobre edad y sexo de los 10 primeros individuos

```
> datos[1:10, c("sexo", "edad")]
```

	sexo	edad
1	2	24
2	1	27
3	2	44
4	1	25
5	2	27
6	2	36
7	1	35
8	1	23
9	2	40
10	2	32

Las funciones `head` y `tail` seleccionan respectivamente los primeros y los últimos seis individuos.

```
> head(datos[, c("sexo", "edad")])
```

	sexo	edad
1	2	24
2	1	27
3	2	44
4	1	25
5	2	27
6	2	36

```
> tail(datos[, c("sexo", "edad")])
```

	sexo	edad
23	1	24
24	2	18
25	2	34
26	2	27
27	1	25
28	2	24

Ordenar por edad de forma ascendente

```
> datos[order(datos$edad), ]
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo	tip_par
15	15	LOLO	13214361600	13214448000	13214966400	1	17	3.60	1	2
24	24	CAGI	13215225600	13215398400	13215830400	2	18	3.75	2	2
20	20	PUPI	13214966400	13215225600	13222224000	2	21	4.46	2	2
8	8	ADJU	13212460800	13212806400	13219718400	2	23	3.20	1	2
1	1	GADI	13211856000	13212288000	13215484800	2	24	3.38	2	1
23	23	CEMA	13215139200	13215571200	13216953600	1	24	3.79	1	2
28	28	MAPE	13215398400	13215830400	13225075200	1	24	3.53	2	2
4	4	VIMU	13212201600	13212633600	13227926400	2	25	2.74	1	1
27	27	PERI	13215398400	13215830400	13230518400	2	25	3.44	1	2
11	11	LOKO	13212892800	13213411200	13217731200	1	26	3.45	1	2
2	2	CAEL	13211942400	13212460800	13233110400	2	27	2.50	1	2
5	5	PAVI	13212288000	13212806400	13212892800	1	27	3.60	2	2
17	17	ANZO	13214793600	13215312000	13226889600	2	27	3.15	1	2
22	22	LOMA	13215052800	13215571200	13234406400	2	27	3.70	1	2
26	26	GUMA	13215398400	13215916800	13227494400	2	27	2.90	2	1
12	12	FRFU	13212979200	13213584000	13234752000	2	29	3.45	2	2
19	19	TOPO	13214880000	13215484800	13222137600	1	29	2.65	2	2
10	10	JUNA	13212633600	13213411200	13221100800	2	32	2.10	2	2
18	18	MEVE	13214793600	13215571200	13236566400	2	32	3.32	2	2
25	25	GRSE	13215312000	13216176000	13216521600	1	34	2.95	2	2
7	7	VERI	13212374400	13213238400	13219632000	2	35	2.97	1	1
21	21	ROPA	13214966400	13215830400	13217385600	1	35	3.15	2	2
6	6	PASA	13212374400	13213324800	13212979200	1	36	2.65	2	1
13	13	FUFE	13213756800	13214707200	13214361600	1	36	3.40	2	2
14	14	POCA	13214361600	13215312000	13224643200	2	36	3.05	1	2
9	9	BEMI	13212547200	13213670400	13218595200	1	40	2.40	2	2
16	16	BOPE	13214448000	13215571200	13230777600	1	40	3.40	1	2
3	3	COMO	13212028800	13213324800	13212633600	1	44	3.15	2	2

	hermanos	fuma_an	fuma_de	horas_an	horas_de	naci_ca	masde12	sem_lac
15	2	0	0	10	0	3	0	1
24	2	1	0	11	7	2	0	1
20	2	1	1	7	0	2	0	12

Ordenar por edad de forma descendente

```
> datos[order(-datos$edad), ]
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo	tip_par
3	3	COMO	13212028800	13213324800	13212633600	1	44	3.15	2	2
9	9	BEMI	13212547200	13213670400	13218595200	1	40	2.40	2	2
16	16	BOPE	13214448000	13215571200	13230777600	1	40	3.40	1	2
6	6	PASA	13212374400	13213324800	13212979200	1	36	2.65	2	1
13	13	FUFE	13213756800	13214707200	13214361600	1	36	3.40	2	2
14	14	POCA	13214361600	13215312000	13224643200	2	36	3.05	1	2
7	7	VERI	13212374400	13213238400	13219632000	2	35	2.97	1	1
21	21	ROPA	13214966400	13215830400	13217385600	1	35	3.15	2	2
25	25	GRSE	13215312000	13216176000	13216521600	1	34	2.95	2	2
10	10	JUNA	13212633600	13213411200	13221100800	2	32	2.10	2	2
18	18	MEVE	13214793600	13215571200	13236566400	2	32	3.32	2	2
12	12	FRFU	13212979200	13213584000	13234752000	2	29	3.45	2	2
19	19	TOPO	13214880000	13215484800	13222137600	1	29	2.65	2	2
2	2	CAEL	13211942400	13212460800	13233110400	2	27	2.50	1	2
5	5	PAVI	13212288000	13212806400	13212892800	1	27	3.60	2	2
17	17	ANZO	13214793600	13215312000	13226889600	2	27	3.15	1	2
22	22	LOMA	13215052800	13215571200	13234406400	2	27	3.70	1	2
26	26	GUMA	13215398400	13215916800	13227494400	2	27	2.90	2	1
11	11	LOKO	13212892800	13213411200	13217731200	1	26	3.45	1	2
4	4	VIMU	13212201600	13212633600	13227926400	2	25	2.74	1	1
27	27	PERI	13215398400	13215830400	13230518400	2	25	3.44	1	2
1	1	GADI	13211856000	13212288000	13215484800	2	24	3.38	2	1
23	23	CEMA	13215139200	13215571200	13216953600	1	24	3.79	1	2
28	28	MAPE	13215398400	13215830400	13225075200	1	24	3.53	2	2
8	8	ADJU	13212460800	13212806400	13219718400	2	23	3.20	1	2
20	20	PUPI	13214966400	13215225600	13222224000	2	21	4.46	2	2
24	24	CAGI	13215225600	13215398400	13215830400	2	18	3.75	2	2
15	15	LOLO	13214361600	13214448000	13214966400	1	17	3.60	1	2

```
hermanos fuma_an fuma_de horas_an horas_de naci_ca masde12 sem_lac
```

```
3 1 0 1 3 0 1 0 1
```

```
9 2 1 1 12 10 1 0 10
```

```
16 2 0 0 5 0 1 1 27
```

Ordenar por sexo y dentro de sexo por edad

```
> datos[order(datos$sexo, datos$edad), ]
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo	tip_par
15	15	LOLO	13214361600	13214448000	13214966400	1	17	3.60	1	2
8	8	ADJU	13212460800	13212806400	13219718400	2	23	3.20	1	2
23	23	CEMA	13215139200	13215571200	13216953600	1	24	3.79	1	2
4	4	VIMU	13212201600	13212633600	13227926400	2	25	2.74	1	1
27	27	PERI	13215398400	13215830400	13230518400	2	25	3.44	1	2
11	11	LOKO	13212892800	13213411200	13217731200	1	26	3.45	1	2
2	2	CAEL	13211942400	13212460800	13233110400	2	27	2.50	1	2
17	17	ANZO	13214793600	13215312000	13226889600	2	27	3.15	1	2
22	22	LOMA	13215052800	13215571200	13234406400	2	27	3.70	1	2
7	7	VERI	13212374400	13213238400	13219632000	2	35	2.97	1	1
14	14	POCA	13214361600	13215312000	13224643200	2	36	3.05	1	2
16	16	BOPE	13214448000	13215571200	13230777600	1	40	3.40	1	2
24	24	CAGI	13215225600	13215398400	13215830400	2	18	3.75	2	2
20	20	PUPI	13214966400	13215225600	13222224000	2	21	4.46	2	2
1	1	GADI	13211856000	13212288000	13215484800	2	24	3.38	2	1
28	28	MAPE	13215398400	13215830400	13225075200	1	24	3.53	2	2
5	5	PAVI	13212288000	13212806400	13212892800	1	27	3.60	2	2
26	26	GUMA	13215398400	13215916800	13227494400	2	27	2.90	2	1
12	12	FRFU	13212979200	13213584000	13234752000	2	29	3.45	2	2
19	19	TOPO	13214880000	13215484800	13222137600	1	29	2.65	2	2
10	10	JUNA	13212633600	13213411200	13221100800	2	32	2.10	2	2
18	18	MEVE	13214793600	13215571200	13236566400	2	32	3.32	2	2
25	25	GRSE	13215312000	13216176000	13216521600	1	34	2.95	2	2
21	21	ROPA	13214966400	13215830400	13217385600	1	35	3.15	2	2
6	6	PASA	13212374400	13213324800	13212979200	1	36	2.65	2	1
13	13	FUFE	13213756800	13214707200	13214361600	1	36	3.40	2	2
9	9	BEMI	13212547200	13213670400	13218595200	1	40	2.40	2	2
3	3	COMO	13212028800	13213324800	13212633600	1	44	3.15	2	2

	hermanos	fuma_an	fuma_de	horas_an	horas_de	naci_ca	masde12	sem_lac
15	2	0	0	10	0	3	0	1
8	2	0	0	5	2	2	0	12
23	1	1	1	4	10	2	0	2

Ordenar por sexo y dentro de sexo por edad, y en una base de datos (“muestra”) se almacenan los datos de sólo los 6 primeros individuos y sólo las variables id, edad, peso y sexo.

```
> muestra <- datos[order(datos$sexo, datos$edad), c("id", "edad",  
+ "peso", "sexo")]  
> muestra <- muestra[1:6, ]  
> muestra
```

	id	edad	peso	sexo
15	15	17	3.60	1
8	8	23	3.20	1
23	23	24	3.79	1
4	4	25	2.74	1
27	27	25	3.44	1
11	11	26	3.45	1

Para saber los atributos de la base de datos

```
> attributes(datos)
```

```
$names
```

```
[1] "id"      "ini"      "dia_nac"  "dia_entr" "ulti_lac" "tx"
[7] "edad"    "peso"     "sexo"     "tip_par"  "hermanos" "fuma_an"
[13] "fuma_de" "horas_an" "horas_de" "naci_ca"  "masde12"  "sem_lac"
```

```
$row.names
```

```
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
[26] 26 27 28
```

```
$class
```

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

```
$variable.labels
```

```
id
""
ini
  "Iniciales del niño"
dia_nac
  "Dia nacimiento"
dia_entr
  "Dia entrada en el estudio"
ulti_lac
  "Ultimo dia lactancia"
tx
  "Regimen visitas asignado"
edad
  "Edad de la madre"
peso
  "peso del niño"
sexo
  "sexo de la criatura"
tip_par
  "Tipo de parto"
```

Para referirse a un determinado atributo (las etiquetas de las variables, pe.)

```
> attr(datos, "variable.labels")
```

```
      id  
      ""  
      ini  
      "Iniciales del niño"  
      dia_nac  
      "Dia nacimiento"  
      dia_entr  
      "Dia entrada en el estudio"  
      ulti_lac  
      "Ultimo dia lactancia"  
      tx  
      "Regimen visitas asignado"  
      edad  
      "Edad de la madre"  
      peso  
      "peso del niño"  
      sexo  
      "sexo de la criatura"  
      tip_par  
      "Tipo de parto"  
      hermanos  
      "Tiene hermanos "  
      fuma_an  
      "Fuma antes embarazo"  
      fuma_de  
      "Fuma despues embarazo"  
      horas_an  
      "Horas ejercicio semanal antes embarazo"  
      horas_de  
      "Horas ejercicio semanal despues embarazo"
```

Para ver el atributo *variable.labels* en columna

```
> cbind(attr(datos, "variable.labels"))
```

```
      [,1]  
id      ""  
ini      "Iniciales del niño"  
dia_nac  "Dia nacimiento"  
dia_entr "Dia entrada en el estudio"  
ulti_lac "Ultimo dia lactancia"  
tx        "Regimen visitas asignado"  
edad      "Edad de la madre"  
peso      "peso del niño"  
sexo      "sexo de la criatura"  
tip_par   "Tipo de parto"  
hermanos  "Tiene hermanos "  
fuma_an   "Fuma antes embarazo"  
fuma_de   "Fuma despues embarazo"  
horas_an  "Horas ejercicio semanal antes embarazo"  
horas_de  "Horas ejercicio semanal despues embarazo"  
naci_ca   "nacionalidad"  
masde12   "Lactancia mas de 12 semanas"  
sem_lac   "Semanas de lactancia"
```


Para ver los atributos de una variable

```
> attributes(datos$sexo)
```

```
$value.labels  
niña niño  
  2    1
```

Para ver sólo el atributo *value.labels*

```
> attr(datos$sexo, "value.labels")
```

```
niña niño  
  2    1
```

La función `summary` devuelve una información general sobre la variable: Mínimo, primer Cuartil, Mediana, Media, tercer Cuartil, Máximo y número de “missings” (si hay)

```
> summary(datos$edad)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
17.00	24.75	27.00	29.29	35.00	44.00

Para obtener la Desviación Estándar (`na.rm=TRUE` elimina los missing en el cálculo):

```
> sd(datos$edad, na.rm = TRUE)
```

```
[1] 6.743211
```

Para obtener la media:

```
> mean(datos$edad, na.rm = TRUE)
```

```
[1] 29.28571
```

Para obtener la mediana:

```
> median(datos$edad, na.rm = TRUE)
```

```
[1] 27
```

Para obtener los valores mínimo y máximo:

```
> range(datos$edad, na.rm = TRUE)
```

```
[1] 17 44
```

Para obtener el mínimo:

```
> min(datos$edad, na.rm = TRUE)
```

```
[1] 17
```

Para obtener el máximo:


```
> max(datos$edad, na.rm = TRUE)
```

```
[1] 44
```

Para obtener los percentiles 5, 10, 25, 50 (la mediana), 75, 90 y 95:

```
> quantile(datos$edad, prob = c(0.05, 0.1, 0.25, 0.5, 0.75, 0.9,  
+ 0.95), na.rm = TRUE)
```

```
 5%  10%  25%  50%  75%  90%  95%  
19.05 22.40 24.75 27.00 35.00 37.20 40.00
```

Nota: En  se pueden calcular los percentiles de 9 maneras distintas. Para obtener los mismos resultados que en SPSS hay que añadir el tipo:

```
> quantile(datos$edad, prob = c(0.05, 0.95), na.rm = TRUE, type = 6)
```

```
 5%  95%  
17.45 42.20
```

Más información ejecutando:

```
> ?quantile
```

Para obtener los valores de una variable categórica:

```
> table(datos$sexo)
```

```
 1  2  
12 16
```

Este resultado se puede almacenar en un objeto, que en este caso será un *vector* de dos elementos.

```
> tabla <- table(datos$sexo)
```


```
> tabla
```

```
 1  2  
12 16
```

De este objeto se puede saber qué proporción hay de cada categoría:

```
> prop.table(tabla)
```

```
      1      2  
0.4285714 0.5714286
```

En  los valores perdidos se muestran como **NA** . Por ejemplo si se quiere declarar como missing los valores de edad menores a 20:

```
> datos$edad <- with(datos, ifelse(edad < 20, NA, edad))  
> with(datos, sum(is.na(edad)))
```

```
[1] 2
```

```
> with(datos, sum(!is.na(edad)))
```

```
[1] 26
```

Para evitar errores se van a reemplazar los valores que en la diapositiva anterior se han convertido a **NA**

```
> datos$edad
```

```
[1] 24 27 44 25 27 36 35 23 40 32 26 29 36 36 NA 40 27 32 29 21 35 27 24 NA 34  
[26] 27 25 24
```

```
> datos$edad[15] <- 17
```

```
> datos$edad[24] <- 18
```

```
> with(datos, sum(is.na(edad)))
```

```
[1] 0
```

Para obtener el Intervalo de Confianza de una proporción (además de un test de Hipótesis):

```
> binom.test(12, 28, conf.level = 0.99)
```

```
Exact binomial test
```

```
data: 12 and 28
number of successes = 12, number of trials = 28, p-value = 0.5716
alternative hypothesis: true probability of success is not equal to 0.5
99 percent confidence interval:
 0.2001911 0.6813584
sample estimates:
probability of success
 0.4285714
```

Se obtienen los mismos resultados si el objeto *tabla* sólo contiene dos valores y el primer valor se corresponde con el número de éxitos:

```
> binom.test(tabla, conf.level = 0.99)
```

```
Exact binomial test
```

```
data: tabla
number of successes = 12, number of trials = 28, p-value = 0.5716
alternative hypothesis: true probability of success is not equal to 0.5
99 percent confidence interval:
 0.2001911 0.6813584
sample estimates:
probability of success
 0.4285714
```


El resultado de *binom.test* se puede almacenar, (p.e. en “bt”). Lo que se almacena es una *lista* (una colección) de sub-objetos, y se puede obtener la “lista” de lo que contiene “bt”:

```
> bt <- binom.test(tabla, conf.level = 0.99)
> str(bt)
```

```
List of 9
```

```
$ statistic : Named num 12
..- attr(*, "names")= chr "number of successes"
$ parameter : Named num 28
..- attr(*, "names")= chr "number of trials"
$ p.value    : Named num 0.572
..- attr(*, "names")= chr "1"
$ conf.int   : atomic [1:2] 0.2 0.681
..- attr(*, "conf.level")= num 0.99
$ estimate   : Named num 0.429
..- attr(*, "names")= chr "probability of success"
$ null.value : Named num 0.5
..- attr(*, "names")= chr "probability of success"
$ alternative: chr "two.sided"
$ method     : chr "Exact binomial test"
$ data.name  : chr "tabla"
- attr(*, "class")= chr "htest"
```

El objeto “bt” se puede desestructurar en cada una de sus partes:

```
> unclass(bt)

$statistic
number of successes
      12

$parameter
number of trials
      28

$p.value
      1
0.5715882

$conf.int
[1] 0.2001911 0.6813584
attr(,"conf.level")
[1] 0.99

$estimate
probability of success
      0.4285714

$null.value
probability of success
      0.5

$alternative
[1] "two.sided"

$method
[1] "Exact binomial test"

$data.name
[1] "table"
```

Y se puede obtener cada una de sus partes. Por ejemplo:

```
> bt$conf.int
```

```
[1] 0.2001911 0.6813584  
attr(,"conf.level")  
[1] 0.99
```

```
> bt$conf.int[1]
```

```
[1] 0.2001911
```

```
> bt$estimate
```

```
probability of success  
0.4285714
```

Para obtener el Intervalo de Confianza de una Media (además de un test de Hipótesis):

```
> t.test(datos$edad, conf.level = 0.99)
```

```
One Sample t-test
```

```
data: datos$edad  
t = 22.981, df = 27, p-value < 2.2e-16  
alternative hypothesis: true mean is not equal to 0  
99 percent confidence interval:  
 25.75490 32.81653  
sample estimates:  
mean of x  
 29.28571
```

Se puede almacenar en un objeto (p.e. *tt*):

```
> tt <- t.test(datos$edad, conf.level = 0.99)
```

Con el objeto “tt” se puede realizar todo lo explicado en el intervalo de confianza de una proporción:

```
> tt  
> str(tt)  
> unclass(tt)  
> tt$conf.int  
> tt$conf.int[1]  
> tt$estimate
```

Se puede calcular el intervalo de confianza *manualmente*. Se almacena la media en “m”, la desviación estándar en “ds” el número de individuos en “n”:

```
> m <- mean(datos$edad, na.rm = TRUE)
> ds <- sd(datos$edad, na.rm = TRUE)
> n <- sum(!is.na(datos$edad))
```

Se calcula el valor “t” de la distribución t-Student para grados de libertad = (n-1) y confianza = 95 % (i.e. $=1-0.05/2$)

```
> tv <- qt(1 - 0.05/2, n - 1)
> tv
```

```
[1] 2.051831
```

Y se obtiene el límite inferior y superior:

```
> m - tv * ds/sqrt(n)
```

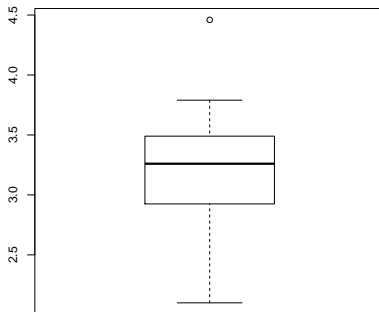
```
[1] 26.67097
```

```
> m + tv * ds/sqrt(n)
```

```
[1] 31.90046
```

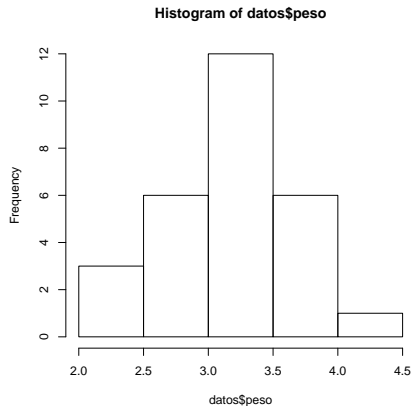
El gráfico de caja y bigotes *boxplot* es una excelente forma de visualizar una variable continua.

```
> boxplot(datos$peso)
```



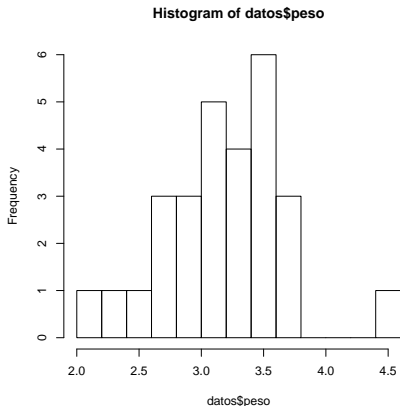
El histograma es otra forma de ver la distribución de una variable continua .

```
> hist(datos$peso)
```



Se puede modificar el número de de “rectángulos” del histograma:

```
> hist(datos$peso, breaks = 10)
```



Cuando la muestra está formada por pocos individuos (p.e. < 60) se puede realizar un test, *Shapiro-Wilk*, para obtener la probabilidad de que los datos se hayan obtenido de una población que sigue una distribución **Normal**

```
> shapiro.test(datos$peso)
```

```
Shapiro-Wilk normality test
```

```
data: datos$peso
```

```
W = 0.977, p-value = 0.7723
```

Pero la mejor forma para valorar la asunción de normalidad es a través del gráfico Cuantil-Cuantil (*Normal QQ-plot*):

```
> qqnorm(datos$peso)  
> qqline(datos$peso)
```

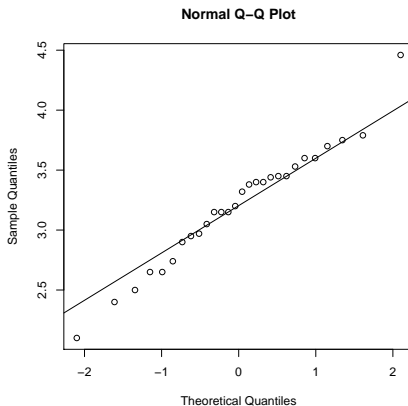


Tabla de frecuencias entre dos variables.

```
> library(foreign)
> datos <- read.spss("c:/CursoR/data/partoFin.sav", use.value.labels = FALSE,
+   to.data.frame = TRUE)
> with(datos, table(tx, masde12))
```

```
      masde12
tx    0    1
  1  11    2
  2   5   10
```

El resultado se almacena en un objeto, p.e. *tb*

```
> tb <- with(datos, table(tx, masde12))
```

El objeto “tb” es una **matriz** de 2 filas y 2 columnas.

Más adelante se utilizará la función `table` con 3 variables y lo que se almacenará será un **array**.

Un **array** es una matriz con más de dos dimensiones

Proporción de cada celda respecto al total:

```
> prop.table(tb)
```

```
masde12
tx      0      1
1 0.39285714 0.07142857
2 0.17857143 0.35714286
```

Proporción por filas y por columnas:

```
> prop.table(tb, margin = 1)
```

```
masde12
tx      0      1
1 0.8461538 0.1538462
2 0.3333333 0.6666667
```

```
> prop.table(tb, margin = 2)
```

```
masde12
tx      0      1
1 0.6875000 0.1666667
2 0.3125000 0.8333333
```

Ejemplo con 3 variables. Se almacena tb3var como un *array*.

```
> tb3var <- with(datos, table(fuma_an, fuma_de, tip_par))
```

```
> tb3var
```

```
, , tip_par = 1
```

```
      fuma_de  
fuma_an 0 1  
      0 3 0  
      1 1 1
```

```
, , tip_par = 2
```

```
      fuma_de  
fuma_an 0 1  
      0 9 2  
      1 5 7
```

Proporciones de cada celda respecto al total de individuos.

```
> prop.table(tb3var)
```

```
, , tip_par = 1
```

	fuma_de	
fuma_an	0	1
0	0.10714286	0.00000000
1	0.03571429	0.03571429

```
, , tip_par = 2
```

	fuma_de	
fuma_an	0	1
0	0.32142857	0.07142857
1	0.17857143	0.25000000

La subinstrucción **margin** = permite *fijar* dimensiones del array.
Por ejemplo `c(1,3)` devuelve el resultado por filas (primera dimensión) para cada grupo de la tercera variable (tercera dimensión).

```
> prop.table(tb3var, margin = c(1, 3))
```

```
, , tip_par = 1
```

	fuma_de	
fuma_an	0	1
0	1.0000000	0.0000000
1	0.5000000	0.5000000

```
, , tip_par = 2
```

	fuma_de	
fuma_an	0	1
0	0.8181818	0.1818182
1	0.4166667	0.5833333

$c(2,3)$ devuelve el resultado por columnas (segunda dimensión) para cada grupo de la tercera variable (tercera dimensión).

```
> prop.table(tb3var, margin = c(2, 3))
```

```
, , tip_par = 1
```

	fuma_de	
fuma_an	0	1
0	0.7500000	0.0000000
1	0.2500000	1.0000000

```
, , tip_par = 2
```

	fuma_de	
fuma_an	0	1
0	0.6428571	0.2222222
1	0.3571429	0.7777778

La prueba de Ji al cuadrado de Pearson asintótica.

```
> tabla <- with(datos, table(tx, masde12))  
> chisq.test(tabla)
```

```
Pearson's Chi-squared test with Yates' continuity correction
```

```
data:  tabla  
X-squared = 5.5312, df = 1, p-value = 0.01868
```

La prueba de Ji al cuadrado de Pearson con **B** simulaciones de Montecarlo.

```
> chisq.test(tabla, simulate.p.value = TRUE, B = 10000)
```

```
Pearson's Chi-squared test with simulated p-value (based on 10000  
replicates)
```

```
data:  tabla  
X-squared = 7.4786, df = NA, p-value = 0.008499
```

Para obtener los valores esperados bajo la hipótesis nula.

```
> res <- chisq.test(tabla)
```

```
> names(res)
```

```
[1] "statistic" "parameter" "p.value"    "method"    "data.name" "observed"
```

```
[7] "expected"  "residuals" "stdres"
```

```
> res$expected
```

```
masde12  
tx      0      1  
1 7.428571 5.571429  
2 8.571429 6.428571
```

La prueba exacta de Fisher.

```
> fisher.test(tabla)
```

```
Fisher's Exact Test for Count Data
```

```
data: tabla  
p-value = 0.009324  
alternative hypothesis: true odds ratio is not equal to 1  
95 percent confidence interval:  
 1.387231 127.034012  
sample estimates:  
odds ratio  
 9.94178
```

Intervalo de confianza de la diferencia de dos proporciones.
Primero hay que preparar una tabla de forma que en la primera columna se encuentre el “outcome” de interés:

```
> datos$gr12 <- car::recode(datos$masde12, "0=2;1=1;else=NA")  
> tabla <- with(datos, table(tx, gr12))  
> tabla
```

```
      gr12  
tx    1  2  
  1  2 11  
  2 10  5
```

```
> prop.test(tabla, correct = FALSE)
```

```
2-sample test for equality of proportions without continuity  
correction
```

```
data:  tabla  
X-squared = 7.4786, df = 1, p-value = 0.006244  
alternative hypothesis: two.sided  
95 percent confidence interval:  
 -0.8216531 -0.2039880  
sample estimates:  
   prop 1    prop 2  
0.1538462 0.6666667
```

Para calcular el intervalo de confianza de un OR se utiliza la siguiente fórmula:

	Casos	Controles	TOTAL
Expuestos	a	b	a+b
No Expuestos	c	d	c+d
TOTAL	a+c	b+d	

$$OR = \frac{a/c}{b/d}$$

$$IC(OR) = e^{\{ \ln(OR) \pm Z_{\alpha/2} * \sqrt{\frac{1}{a} + \frac{1}{b} + \frac{1}{c} + \frac{1}{d}} \}}$$

Donde $Z_{\alpha/2}$ es el valor de la distribución normal que deja a su *derecha* el $(\alpha/2 * 100) \%$ del área acumulada.

EJEMPLO

	Casos	Controles	TOTAL
Fumadores	196	46	242
No Fumadores	206	296	502
TOTAL	402	342	

$$OR = \frac{196/206}{46/296} = 6,1$$

Si el intervalo de confianza que se quiere calcular es el del 95 %

$$IC\ 95\ \%(OR) = e^{\{(\ln(6,1) \pm 1,96 * \sqrt{\frac{1}{196} + \frac{1}{46} + \frac{1}{206} + \frac{1}{296}})\}} = \{4,2; 8,8\}$$

Para calcular el intervalo de confianza de un RR se utiliza la siguiente fórmula:

	<i>Outcome</i>	No outcome	TOTAL
Expuestos	a	b	a+b
No Expuestos	c	d	c+d
TOTAL	a+c	b+d	

$$RR = \frac{a/(a+b)}{c/(c+d)}$$

$$IC(RR) = e^{\{ \ln(RR) \pm Z_{\alpha/2} * \sqrt{\frac{b/a}{a+b} + \frac{d/c}{c+d}} \}}$$

Donde $Z_{\alpha/2}$ es el valor de la distribución normal que deja a su *derecha* el $(\alpha/2 * 100) \%$ del área acumulada.


EJEMPLO

	Outcome	No outcome	TOTAL
Fumadores	196	46	242
No Fumadores	206	296	502
TOTAL	402	342	

$$RR = \frac{196/242}{206/502} = 1,97$$

Si el intervalo de confianza que se quiere calcular es el del 95 %

$$IC\ 95\ \%(RR) = e^{\{(\ln(1,97) \pm 1,96 * \sqrt{\frac{46/196}{242} + \frac{296/206}{502}})\}} = \{1,75; 2,23\}$$

Más adelante se enseñara como hacer funciones con , pero una primera aproximación que permite calcular OR, RR y sus correspondientes intervalos de confianza, podría ser:

```
> a <- 196  
> b <- 46  
> c <- 206  
> d <- 296  
> conf <- 0.95  
> z <- qnorm((1 - (1 - conf)/2))  
> nexp <- a + b  
> nnoexp <- c + d
```

Para calcular OR y el correspondiente intervalo de confianza:

```
> or <- (a/b)/(c/d)
> varilogor <- sqrt(1/a + 1/b + 1/c + 1/d)
> orsup <- exp(log(or) + z * varilogor)
> orinf <- exp(log(or) - z * varilogor)
```

Para calcular RR y el correspondiente intervalo de confianza:

```
> rr <- (a/nexp)/(c/mnoexp)
> varilogrr <- sqrt((b/a)/nexp + (d/c)/mnoexp)
> rrsup <- exp(log(rr) + z * varilogrr)
> rrinf <- exp(log(rr) - z * varilogrr)
```

```
> or
[1] 6.122415
```

```
> orinf
[1] 4.241402
```

```
> orsup
[1] 8.837634
```

```
> rr
[1] 1.973682
```

```
> rrinf
[1] 1.748172
```

```
> rrsup
[1] 2.228283
```

Para calcular el OR y el RR en el ejemplo de la lactancia materna:

```
> with(datos, table(tx, masde12))
```

```
      masde12  
tx      0    1  
  1  11    2  
  2   5   10
```

Asignamos cada valor donde corresponde:

```
> a <- 10  
> b <- 5  
> c <- 2  
> d <- 11  
> conf <- 0.95
```

El resultado de ejecutar el esbozo de “función” explicado anteriormente para calcular OR, RR y los intervalos de confianza:

```
> or
```

```
[1] 11
```

```
> orinf
```

```
[1] 1.72966
```

```
> orsup
```

```
[1] 69.95595
```

```
> rr
```

```
[1] 4.333333
```

```
> rrinf
```

```
[1] 1.152832
```

```
> rrsup
```

```
[1] 16.28839
```

La *library("epitools")* permite obtener Odds Ratio, Riesgos Relativos y sus correspondientes intervalos de confianza, además de calcular tasas estandarizadas y otros cálculos epidemiológicos.

Estos cálculos se realizan a partir de los datos entrados en formato tabla. Obsérvese el resultado de las siguientes instrucciones:

```
> library("epitools")
> tb <- with(datos, table(tx, masde12))
> tb
```

```
      masde12
tx  0  1
  1 11  2
  2  5 10
```


```
> oddsratio(tb, method = "wald")
```

```
$data
      masde12
tx      0  1 Total
  1      11  2   13
  2       5 10   15
Total 16 12   28
```


```
$measure
      odds ratio with 95% C.I.
tx estimate lower upper
  1         1      NA   NA
  2        11 1.72966 69.95595
```


```
$p.value
      two-sided
tx midp.exact fisher.exact chi.square
```

Como en la mayoría (si no todos) de los software de estadística, para que se calcule correctamente el Odds Ratio (OR) o el Riesgo relativo (RR), los valores de cada casilla deben estar en una posición concreta.

Observe la correspondencia entre dónde tienen que estar **a**, **b**, **c** y **d** tal como se ha explicado anteriormente (situación estándar) y como deben estar situados en :

	Estándar	
	Disease	No disease
Exposed	a	b
No exposed	c	d

		
	No disease	Disease
No exposed	d	c
Exposed	b	a

Con la función `rev` se puede modificar la situación de las filas y casillas hasta situarlas en el lugar adecuado para que el  calcule correctamente el OR y RR.

Observe el resultado de aplicar las instrucciones que hay a continuación y recuerde que el $OR = 11$ y el $RR = 4.33$.

```
> oddsratio(tb, method = "wald")
> oddsratio(tb, rev = "rows", method = "wald")
> oddsratio(tb, rev = "columns", method = "wald")
> oddsratio(tb, rev = "both", method = "wald")
> riskratio(tb, method = "wald")
> riskratio(tb, rev = "rows", method = "wald")
> riskratio(tb, rev = "columns", method = "wald")
> riskratio(tb, rev = "both", method = "wald")
```

El índice de concordancia Kappa

Se ha de tener cargado el paquete irr

```
> library(irr)  
> kappa2(datos[, c("fuma_an", "fuma_de")])
```

Cohen's Kappa for 2 Raters (Weights: unweighted)

```
Subjects = 28  
Raters = 2  
Kappa = 0.429
```

```
z = 2.37  
p-value = 0.018
```

La prueba *t-Student* para comparar dos medias independientes

Por defecto asume que las variancias no son iguales (aproximación de Welch).

```
> t.test(sem_lac ~ tx, data = datos)
```

```
Welch Two Sample t-test
```

```
data: sem_lac by tx
t = -3.7597, df = 25.083, p-value = 0.0009121
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -21.009082 -6.139636
sample estimates:
mean in group 1 mean in group 2
 6.692308      20.266667
```

Descriptiva

```
> with(datos, by(sem_lac, tx, function(x) c(mean(x, na.rm = TRUE),
+      sd(x, na.rm = TRUE))))
```

```
tx: 1
[1] 6.692308 7.878077
```

```
tx: 2
[1] 20.26667 11.13211
```

```
> with(datos, by(sem_lac, tx, function(x) c(quantile(x, na.rm = TRUE,
+      type = 6))))
```

```
tx: 1
  0%  25%  50%  75% 100%
  1    1    3   11   27
```

```
tx: 2
  0%  25%  50%  75% 100%
  1   12   20   32   36
```

La prueba *U* de Mann-Whitney para comparar "rangos" de dos grupos

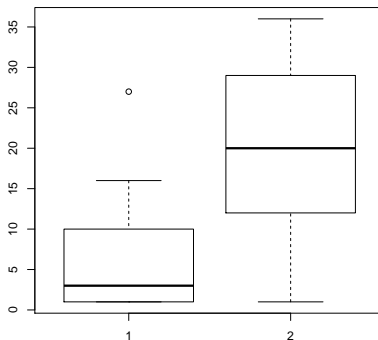
```
> kruskal.test(sem_lac ~ tx, data = datos)

Kruskal-Wallis rank sum test

data:  sem_lac by tx
Kruskal-Wallis chi-squared = 9.7944, df = 1, p-value = 0.00175
```

Boxplot para 2 o más categorías

```
> with(datos, boxplot(sem_lac ~ tx))
```



Análisis de la varianza (ANOVA)

Obsérvese como se especifica que "naci_ca" es un factor.

```
> table(datos$naci_ca)
```

```
 1  2  3
14  7  7
```

```
> attr(datos$naci_ca, "value.labels")
```

```
Sudamérica      Otras      Española
           3             2             1
```

```
> fit <- aov(edad ~ as.factor(naci_ca), data = datos)
```

```
> summary(fit)
```

	Df	Sum Sq	Mean Sq	F value	Pr(>F)
as.factor(naci_ca)	2	567.07	283.536	10.729	0.0004324 ***
Residuals	25	660.64	26.426		

```
---
```

```
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Prueba no paramétrica de *Kruskal-Wallis*

```
> kruskal.test(edad ~ naci_ca, data = datos)
```

```
Kruskal-Wallis rank sum test
```

```
data: edad by naci_ca
```

```
Kruskal-Wallis chi-squared = 14.1511, df = 2, p-value = 0.0008455
```

Si se utiliza la prueba no paramétrica y se quiere realizar comparaciones 2 a 2:

```
> kruskal.test(edad ~ naci_ca, data = subset(datos, naci_ca %in%  
+ c(1, 2)))
```

```
Kruskal-Wallis rank sum test
```

```
data: edad by naci_ca
```

```
Kruskal-Wallis chi-squared = 9.9934, df = 1, p-value = 0.001571
```


No paramétrica, comparaciones 2 a 2, continuación

```
> kruskal.test(edad ~ naci_ca, data = subset(datos, naci_ca %in%  
+ c(1, 3)))
```

```
Kruskal-Wallis rank sum test
```

```
data: edad by naci_ca
```

```
Kruskal-Wallis chi-squared = 8.7868, df = 1, p-value = 0.003034
```

```
> kruskal.test(edad ~ naci_ca, data = subset(datos, naci_ca %in%  
+ c(2, 3)))
```

```
Kruskal-Wallis rank sum test
```

```
data: edad by naci_ca
```

```
Kruskal-Wallis chi-squared = 0.0165, df = 1, p-value = 0.8977
```

No paramétrica, comparaciones 2 a 2, continuación

Hay tres pares de comparaciones posibles (1 vs 2; 1 vs 3 y 2 vs 3), es decir **k=3** por lo aplicando la corrección de *Bonferroni* sólo se considerarán significativos a nivel de $p \leq 0,05$ los valores de $p \leq \alpha$:

$$p\text{-valor}_c = 1 - (1 - 0,05)^{\frac{1}{3}} = 0,017$$

Una alternativa es:

```
> library(pgirmess)
> with(datos, kruskalmc(edad, naci_ca, probs = 0.05))
```

Multiple comparison test after Kruskal-Wallis

p.value: 0.05

Comparisons

	obs.dif	critical.dif	difference
1-2	11.67857143	9.116003	TRUE
1-3	11.60714286	9.116003	TRUE
2-3	0.07142857	10.526254	FALSE

Corrección por comparaciones múltiples cuando la variable continua cumple el supuesto de normalidad

```
> fit <- aov(edad ~ as.factor(naci_ca), data = datos)
> TukeyHSD(fit, "as.factor(naci_ca)")
```

```
Tukey multiple comparisons of means
 95% family-wise confidence level
```

```
Fit: aov(formula = edad ~ as.factor(naci_ca), data = datos)
```

```
$`as.factor(naci_ca)`
      diff      lwr      upr      p adj
2-1 -8.9285714 -14.855833 -3.001310 0.0025956
3-1 -9.0714286 -14.998690 -3.144167 0.0022331
3-2 -0.1428571 -6.987069  6.701355 0.9985110
```

Otros métodos de corrección se encuentran en *p.adjust* del paquete **stats**, *LDuncan* del paquete **laercio**, etc.

La prueba de *Mc Nemar*

```
> tabla <- with(datos, table(fuma_an, fuma_de))  
> tabla
```

```
      fuma_de  
fuma_an 0  1  
      0 12  2  
      1  6  8
```

```
> mcnemar.test(tabla)
```

```
McNemar's Chi-squared test with continuity correction
```

```
data:  tabla
```

```
McNemar's chi-squared = 1.125, df = 1, p-value = 0.2888
```

La prueba de *t-Student* para datos apareados

```
> with(datos, t.test(horas_an, horas_de, paired = TRUE))
```

```
Paired t-test
```

```
data: horas_an and horas_de
```

```
t = 0.8866, df = 27, p-value = 0.3831
```

```
alternative hypothesis: true difference in means is not equal to 0
```

```
95 percent confidence interval:
```

```
-1.173414  2.959128
```

```
sample estimates:
```

```
mean of the differences
```

```
0.8928571
```

La prueba no paramétrica de *Wilcoxon* para datos apareados

```
> with(datos, wilcox.test(horas_an, horas_de, paired = TRUE))
```

```
Wilcoxon signed rank test with continuity correction
```

```
data: horas_an and horas_de
```

```
V = 185.5, p-value = 0.3159
```

```
alternative hypothesis: true location shift is not equal to 0
```

Por defecto calcula el p-valor *exacto*, pero nos muestra un “warning” porque puede no ser correcto cuando hay empates.

Se puede relajar la opción, pidiendo que no se calcule el test exacto.

```
> with(datos, wilcox.test(horas_an, horas_de, paired = TRUE, exact = FALSE))
```

```
Wilcoxon signed rank test with continuity correction
```

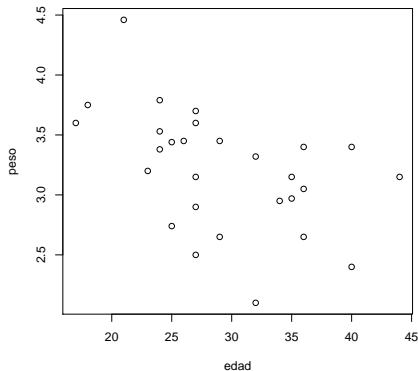
```
data: horas_an and horas_de
```

```
V = 185.5, p-value = 0.3159
```

```
alternative hypothesis: true location shift is not equal to 0
```

La siguiente figura parece indicar que, a pesar de la dispersión, a mayor edad de la madre menor es el peso de niño.

```
> with(datos, plot(edad, peso))
```



El índice de correlación de Pearson

```
> with(datos, cor.test(edad, peso, method = "pearson"))
```

```
Pearson's product-moment correlation
```

```
data: edad and peso
```

```
t = -2.7502, df = 26, p-value = 0.01069
```

```
alternative hypothesis: true correlation is not equal to 0
```

```
95 percent confidence interval:
```

```
-0.7202342 -0.1235120
```

```
sample estimates:
```

```
cor
```

```
-0.4747143
```

Índice de correlación no paramétrico de Spearman

```
> with(datos, cor.test(edad, peso, method = "spearman", exact = FALSE))  
  
Spearman's rank correlation rho  
  
data: edad and peso  
S = 5678.872, p-value = 0.002215  
alternative hypothesis: true rho is not equal to 0  
sample estimates:  
rho  
-0.5541522
```

Por defecto (si no se especifica **exact=FALSE**) calcula el índice de Kendall, pero es incorrecto cuando hay empates. Si hay empates, la función directamente calcula Spearman y muestra un “Warning”.

Regresión lineal simple

```
> lm(peso ~ edad, data = datos)
```

Call:

```
lm(formula = peso ~ edad, data = datos)
```

Coefficients:

```
(Intercept)      edad  
  4.22882      -0.03485
```

Por defecto sólo devuelve la constante y la pendiente.

Pero si se almacena el resultado en un “objeto” (p.e. *fit*). . . .

```
> fit <- lm(peso ~ edad, data = datos)
```

Regresión lineal simple: summary

```
> summary(fit)
```

Call:

```
lm(formula = peso ~ edad, data = datos)
```

Residuals:

Min	1Q	Median	3Q	Max
-1.0136	-0.2515	0.0791	0.2519	0.9630

Coefficients:

	Estimate	Std. Error	t value	Pr(> t)
(Intercept)	4.22882	0.38047	11.12	2.25e-11 ***
edad	-0.03485	0.01267	-2.75	0.0107 *

Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.444 on 26 degrees of freedom

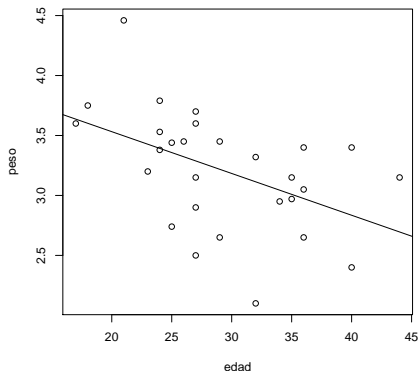
Multiple R-squared: 0.2254, Adjusted R-squared: 0.1956

F-statistic: 7.564 on 1 and 26 DF, p-value: 0.01069

Gráfico de la regresión lineal simple

Para realizar un gráfico que incluya la recta de regresión:

```
> with(datos, plot(edad, peso))  
> with(datos, abline(lm(peso ~ edad)))
```



Regresión lineal simple: residuales

```
> observ <- cbind(datos$peso)
> predicho <- cbind(predict(fit))
> residual <- cbind(resid(fit))
> stanresid <- cbind(rstandard(fit))
> studresid <- cbind(rstudent(fit))
> predicciones <- cbind(observ, predicho, residual, stanresid,
+   studresid)
> colnames(predicciones) <- c("observados", "predichos", "residuales",
+   "resi. standa.", "resi.studen.")
> predicciones[1:5, ]
```

	observados	predichos	residuales	resi. standa.	resi.studen.
1	3.38	3.392421	-0.01242088	-0.02883052	-0.0282711
2	2.50	3.287871	-0.78787119	-1.81104781	-1.8997425
3	3.15	2.695423	0.45457703	1.15339981	1.1610962
4	2.74	3.357571	-0.61757098	-1.42756717	-1.4581579
5	3.60	3.287871	0.31212881	0.71747794	0.7106149

Regresión lineal simple: residuales

Se pueden obtener varios gráficos que permitan analizar los residuales. En cada gráfico esperará que se confirme el cambio de página. El primero muestra los predichos vs. residuales.

```
> plot(fit, which = 1)
```

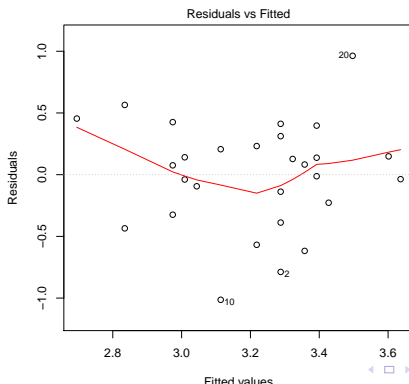
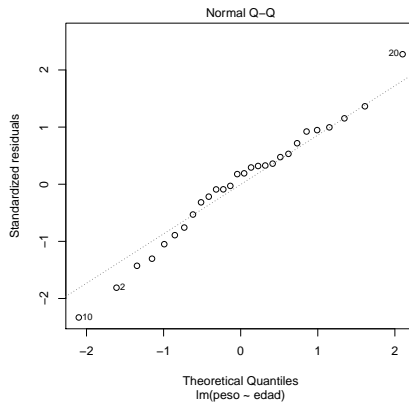


Gráfico para valorar el supuesto de normalidad de los residuales

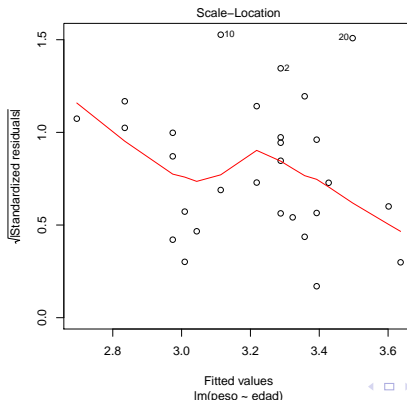
```
> plot(fit, which = 2)
```



Predichos vs. raíz cuadrada de los valores absolutos de los residuales estandarizados

Señala los valores que se encuentran más allá de $\sqrt{1,6}$.

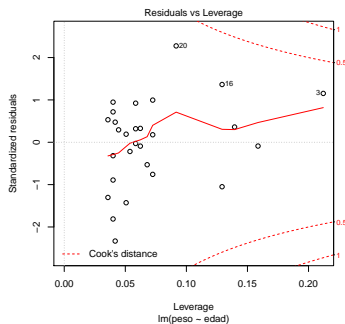
```
> plot(fit, which = 3)
```



Leverage (medida de ausencia de "vecinos") vs. los valores predichos

Asimismo marca la distancia de Cook para detectar casos influyentes (los valores más allá de 1 deberían revisarse).

```
> plot(fit, which = 5)
```



Empezamos cargando los paquetes necesarios y fijando la carpeta de trabajo

```
> rm(list = ls())  
> library(foreign)  
> library(chron)  
> library(RODBC)  
> library(xlsReadWrite)  
> setwd("C:/cursoR/data")
```

- La `library(foreign)` permite exportar/importar de SPSS, STATA, SAS, etc.
- La `library(chron)` permite manejar fechas
- La `library(RODBC)` permite el acceso a bases de datos basadas en SQL
- La `library(xlsReadWrite)` es específica para manejar ficheros de EXCEL

Para ver los formatos posibles de importación y exportación:

```
> help(package = "foreign")
```

Es importante entender el concepto de la “clase” **factor** de una variable. Ejecute las siguientes instrucciones:

```
> x1 <- c("maria", "joan", "maria", "joan", "pep")
> x2 <- c("maria", "joan", "maria", "joan", "pep")
> xxx <- as.data.frame(cbind(x1, x2))
> xxx$x1
> xxx$x2
> xxx$x2 <- as.character(xxx$x2)
> xxx$x1[5] <- NA
> xxx$x2[5] <- NA
> table(xxx$x1)
> table(xxx$x2)
> as.numeric(xxx$x1)
> xxx
```

Cuando es un factor:

- “recuerda” la información de todos los “niveles” de los factores, aunque ningún individuo presente aquel nivel
- internamente está codificado como números consecutivos por orden alfabético
- aparentemente, y sólo aparentemente, los factores parecen “cadenas”

```
joan maria    pep  
  2      2      0
```

```
joan maria  
  2      2
```

```
[1]  2  1  2  1 NA
```

```
      x1    x2  
1 maria maria  
2 joan  joan  
3 maria maria  
4 joan  joan  
5 <NA> <NA>
```

- Para leer variables separadas por un carácter especial (el tabulador es: `\t`)
- Notése que la primera línea del fichero 'parto.dat' **no** contiene el nombre de las variables. Para ello, hay que especificar la opción `header=FALSE`.
- La opción `as.is=TRUE` sirve para que las variables cadena **no** se conviertan en factores.

```
> datos <- read.table("parto2.dat", sep = ";", as.is = TRUE, header = FALSE)
> datos
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
1	1	GADI	14-JUN-2001	19-JUN-2001	26-JUL-2001	2	24	3.38	2	1	2
2	2	CAEL	15-JUN-2001	21-JUN-2001	15-FEB-2002	2	27	2.50	1	2	1
3	3	COMO	16-JUN-2001	01-JUL-2001	23-JUN-2001	1	44	3.15	2	2	1
4	4	VIMU	18-JUN-2001	23-JUN-2001	17-DEC-2001	2	25	2.74	1	1	1
5	5	PAVI	19-JUN-2001	25-JUN-2001	26-JUN-2001	1	27	3.60	2	2	1
6	6	PASA	20-JUN-2001	01-JUL-2001	27-JUN-2001	1	36	2.65	2	1	2
7	7	VERI	20-JUN-2001	30-JUN-2001	12-SEP-2001	2	35	2.97	1	1	2
8	8	ADJU	21-JUN-2001	25-JUN-2001	13-SEP-2001	2	23	3.20	1	2	2
9	9	BEMI	22-JUN-2001	05-JUL-2001	31-AUG-2001	1	40	2.40	2	2	2
10	10	JUNA	23-JUN-2001	02-JUL-2001	29-SEP-2001	2	32	2.10	2	2	2
11	11	LOKO	26-JUN-2001	02-JUL-2001	21-AUG-2001	1	26	3.45	1	2	2
12	12	FRFU	27-JUN-2001	04-JUL-2001	06-MAR-2002	2	29	3.45	2	2	1
13	13	FUFE	06-JUL-2001	17-JUL-2001	13-JUL-2001	1	36	3.40	2	2	2
14	14	POCA	13-JUL-2001	24-JUL-2001	09-NOV-2001	2	36	3.05	1	2	2
15	15	LOLO	13-JUL-2001	14-JUL-2001	20-JUL-2001	1	17	3.60	1	2	2
16	16	BOPE	14-JUL-2001	27-JUL-2001	19-JAN-2002	1	40	3.40	1	2	2
17	17	ANZO	18-JUL-2001	24-JUL-2001	05-DEC-2001	2	27	3.15	1	2	1
18	18	MEVE	19-JUL-2001	27-JUL-2001	07-MAR-2002	2	28	2.88	2	2	2

Para nombrar las variables:

```
> names(datos) <- c("id", "ini", "dia_nac", "dia_entr", "ulti_lac",  
+ "tx", "edad", "peso", "sexo", "tip_par", "hermanos")
```

Para ver las características de las variables:

```
> str(datos$id)
```

```
int [1:28] 1 2 3 4 5 6 7 8 9 10 ...
```

```
> str(datos$ini)
```

```
chr [1:28] "GADI" "CAEL" "COMO" "VIMU" "PAVI" "PASA" "VERI" ...
```

```
> str(datos$dia_nac)
```

```
chr [1:28] "14-JUN-2001" "15-JUN-2001" "16-JUN-2001" "18-JUN-2001" ...
```

```
> str(datos$tx)
```

```
int [1:28] 2 2 1 2 1 1 2 2 1 2 ...
```

```
> str(datos$peso)
```

```
num [1:28] 3.38 2.5 3.15 2.74 3.6 2.65 2.97 3.2 2.4 2.1 ...
```

Observe qué ocurre si se especifica: `as.is=FALSE`

```
> datos <- read.table("parto2.dat", sep = ";", as.is = FALSE, header = FALSE)
> names(datos) <- c("id", "ini", "dia_nac", "dia_entr", "ulti_lac",
+   "tx", "edad", "peso", "sexo", "tip_par", "hermanos")
> str(datos$id)

int [1:28] 1 2 3 4 5 6 7 8 9 10 ...

> str(datos$ini)

Factor w/ 28 levels "ADJU","ANZO",...: 11 5 8 28 21 20 27 1 3 14 ...

> str(datos$dia_nac)

Factor w/ 22 levels "06-JUL-2001",...: 4 5 6 8 10 12 12 14 16 18 ...

> str(datos$tx)

int [1:28] 2 2 1 2 1 1 2 2 1 2 ...

> str(datos$peso)

num [1:28] 3.38 2.5 3.15 2.74 3.6 2.65 2.97 3.2 2.4 2.1 ...
```


- Para leer ficheros Excell (.xls) usamos la función `read.xls` del package `xlsReadWrite`. ¡IMPORTANTE! Este paquete sólo está disponible para Windows 32bits.
- Para evitar posibles confusiones con otra función del mismo nombre del package `gdata`, se pone el nombre del package delante del nombre de la función con '::'.
- Hay que especificar las clases de cada variable, para que se garantice la lectura correcta del tipo de cada variable. Ello se especifica en el argumento `colClasses` con un vector en el mismo orden en que aparecen las variables en la base de datos.

```
> datos <- xlsReadWrite::read.xls(file = "C:/cursoR/data/parto2.xls",
+   sheet = 1, colClasses = c("character", "isodate", "isodate",
+   "isodate", "character", "integer", "character", "character",
+   "character", "character"))
```

```
> datos
```

	Iniciales	Dia.de.nacimiento	Dia.reclutamiento	Dia.última.lactancia
1	GADI	2001-06-14	2001-06-19	2001-07-26
2	CAEL	2001-06-15	2001-06-21	2002-02-15
3	COMO	2001-06-16	2001-07-01	2001-06-23
4	VIMU	2001-06-18	2001-06-23	2001-12-17
5	PAVI	2001-06-19	2001-06-25	2001-06-26
6	PASA	2001-06-20	2001-07-01	2001-06-27
7	VERI	2001-06-20	2001-06-30	2001-09-12

Obsérvese qué ocurre si no se especifica colClasses

```
> datos <- xlsReadWrite::read.xls(file = "C:/cursoR/data/parto2.xls",  
+   sheet = 1)  
> str(datos)
```

```
'data.frame':      28 obs. of  10 variables:  
 $ Iniciales      : Factor w/ 28 levels "ADJU","ANZO",...: 11 5 8 28 21 20 27 1 3 14 ...  
 $ Dia.de.nacimiento : num  37056 37057 37058 37060 37061 ...  
 $ Dia.reclutamiento : num  37061 37063 37073 37065 37067 ...  
 $ Dia.última.lactancia: num  37098 37302 37065 37242 37068 ...  
 $ Tratamiento     : Factor w/ 2 levels "Estandar","Intesivo": 2 2 1 2 1 1 2 2 1 2 ...  
 $ EDAD            : num  24 27 44 25 27 36 35 23 40 32 ...  
 $ PESO            : Factor w/ 22 levels "2,1","2,4","2,5",...: 13 3 10 5 18 4 8 11 2 1 ...  
 $ SEXO            : Factor w/ 2 levels "Niña","Niño": 1 2 1 2 1 1 2 2 1 1 ...  
 $ Tipo.de.parto   : Factor w/ 2 levels "Instrumen","No instrumen.": 1 2 2 1 2 1 1 2 2 2 ...  
 $ hermanos.anteriores : Factor w/ 2 levels "No","Si": 1 2 2 2 2 1 1 1 1 1 ...
```

Excel Office 2007

- El 'package' `xlsReadWrite` permite leer/escribir ficheros Excel de **Office 2003**.
- Para ficheros de Excel de **Office 2007 (.xlsx)** hay que usar otros 'packages' como el `xlsx`.
- Por ejemplo, la función para leer archivos `.xlsx` es:

```
> read.xlsx(file, sheetIndex, sheetName = NULL, rowIndex = NULL,  
+          colIndex = NULL, as.data.frame = TRUE, header = TRUE, colClasses = NA,  
+          keepFormulas = FALSE, ...)
```

Lectura de ficheros de ACCESS (.mdb)

```
> canal <- odbcConnectAccess("spss.mdb")
> query <- "SELECT * FROM parto2"
> datos <- sqlQuery(canal, query)
> odbcClose(canal)
> datos
```

	ID	INI	DIA_NAC	DIA_ENTR	ULTI_LAC	TX	EDAD	PESO	SEXO	TIP_PAR	HERMANOS
1	1	GADI	2001-06-14	2001-06-19	2001-07-26	2	24	3.38	2	1	2
2	2	CAEL	2001-06-15	2001-06-21	2002-02-15	2	27	2.50	1	2	1
3	3	COMO	2001-06-16	2001-07-01	2001-06-23	1	44	3.15	2	2	1
4	4	VIMU	2001-06-18	2001-06-23	2001-12-17	2	25	2.74	1	1	1
5	5	PAVI	2001-06-19	2001-06-25	2001-06-26	1	27	3.60	2	2	1
6	6	PASA	2001-06-20	2001-07-01	2001-06-27	1	36	2.65	2	1	2
7	7	VERI	2001-06-20	2001-06-30	2001-09-12	2	35	2.97	1	1	2
8	8	ADJU	2001-06-21	2001-06-25	2001-09-13	2	23	3.20	1	2	2
9	9	BEMI	2001-06-22	2001-07-05	2001-08-31	1	40	2.40	2	2	2
10	10	JUNA	2001-06-23	2001-07-02	2001-09-29	2	32	2.10	2	2	2
11	11	LOKO	2001-06-26	2001-07-02	2001-08-21	1	26	3.45	1	2	2
12	12	FRFU	2001-06-27	2001-07-04	2002-03-06	2	29	3.45	2	2	1
13	13	FUFE	2001-07-06	2001-07-17	2001-07-13	1	36	3.40	2	2	2
14	14	POCA	2001-07-13	2001-07-24	2001-11-09	2	36	3.05	1	2	2
15	15	LOLO	2001-07-13	2001-07-14	2001-07-20	1	17	3.60	1	2	2
16	16	BOPE	2001-07-14	2001-07-27	2002-01-19	1	40	3.40	1	2	2
17	17	ANZO	2001-07-18	2001-07-24	2001-12-05	2	27	3.15	1	2	1
18	18	MEVE	2001-07-18	2001-07-27	2002-03-27	2	32	3.32	2	2	2
19	19	TOPO	2001-07-19	2001-07-26	2001-10-11	1	29	2.65	2	2	2
20	20	PUPI	2001-07-20	2001-07-23	2001-10-12	2	21	4.46	2	2	2
21	21	ROPA	2001-07-20	2001-07-30	2001-08-17	1	35	3.15	2	2	2
22	22	LOMA	2001-07-21	2001-07-27	2002-03-02	2	27	3.70	1	2	1
23	23	CEMA	2001-07-22	2001-07-27	2001-08-12	1	24	3.79	1	2	1
24	24	CAGI	2001-07-23	2001-07-25	2001-07-30	2	18	3.75	2	2	2
25	25	GRSE	2001-07-24	2001-08-03	2001-08-07	1	34	2.95	2	2	1
26	26	GUMA	2001-07-25	2001-07-31	2001-12-12	2	27	2.90	2	1	1

Para arreglar las variables de formato fecha para ponerlo en formato d/m/a

```
> datos$DIA_NAC <- format(datos$DIA_NAC, "%d/%m/%Y")
> datos$DIA_ENTR <- format(datos$DIA_ENTR, "%d/%m/%Y")
> datos$ULTI_LAC <- format(datos$ULTI_LAC, "%d/%m/%Y")
> datos
```

	ID	INI	DIA_NAC	DIA_ENTR	ULTI_LAC	TX	EDAD	PESO	SEXO	TIP_PAR	HERMANOS
1	1	GADI	14/06/2001	19/06/2001	26/07/2001	2	24	3.38	2	1	2
2	2	CAEL	15/06/2001	21/06/2001	15/02/2002	2	27	2.50	1	2	1
3	3	COMO	16/06/2001	01/07/2001	23/06/2001	1	44	3.15	2	2	1
4	4	VIMU	18/06/2001	23/06/2001	17/12/2001	2	25	2.74	1	1	1
5	5	PAVI	19/06/2001	25/06/2001	26/06/2001	1	27	3.60	2	2	1
6	6	PASA	20/06/2001	01/07/2001	27/06/2001	1	36	2.65	2	1	2
7	7	VERI	20/06/2001	30/06/2001	12/09/2001	2	35	2.97	1	1	2
8	8	ADJU	21/06/2001	25/06/2001	13/09/2001	2	23	3.20	1	2	2
9	9	BEMI	22/06/2001	05/07/2001	31/08/2001	1	40	2.40	2	2	2
10	10	JUNA	23/06/2001	02/07/2001	29/09/2001	2	32	2.10	2	2	2
11	11	LOKO	26/06/2001	02/07/2001	21/08/2001	1	26	3.45	1	2	2
12	12	FRFU	27/06/2001	04/07/2001	06/03/2002	2	29	3.45	2	2	1
13	13	FUFE	06/07/2001	17/07/2001	13/07/2001	1	36	3.40	2	2	2
14	14	POCA	13/07/2001	24/07/2001	09/11/2001	2	36	3.05	1	2	2
15	15	LOLO	13/07/2001	14/07/2001	20/07/2001	1	17	3.60	1	2	2
16	16	BOPE	14/07/2001	27/07/2001	19/01/2002	1	40	3.40	1	2	2
17	17	ANZO	18/07/2001	24/07/2001	05/12/2001	2	27	3.15	1	2	1
18	18	MEVE	18/07/2001	27/07/2001	27/03/2002	2	32	3.32	2	2	2
19	19	TOPO	19/07/2001	26/07/2001	11/10/2001	1	29	2.65	2	2	2
20	20	PUPI	20/07/2001	23/07/2001	12/10/2001	2	21	4.46	2	2	2
21	21	ROPA	20/07/2001	30/07/2001	17/08/2001	1	35	3.15	2	2	2
22	22	LOMA	21/07/2001	27/07/2001	02/03/2002	2	27	3.70	1	2	1
23	23	CEMA	22/07/2001	27/07/2001	12/08/2001	1	24	3.79	1	2	1
24	24	CAGI	23/07/2001	25/07/2001	30/07/2001	2	18	3.75	2	2	2
25	25	GRSE	24/07/2001	03/08/2001	07/08/2001	1	34	2.95	2	2	1

Lectura de ficheros de SPSS(.sav)

```
> datos <- read.spss("parto2.sav", use.value.labels = FALSE, to.data.frame = TRUE)
```

```
> datos
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo	tip_par
1	10	JUNA	13212633600	13213411200	13221100800	2	32	2.10	2	2
2	9	BEMI	13212547200	13213670400	13218595200	1	40	2.40	2	2
3	2	CAEL	13211942400	13212460800	13233110400	2	27	2.50	1	2
4	6	PASA	13212374400	13213324800	13212979200	1	36	2.65	2	1
5	19	TOPO	13214880000	13215484800	13222137600	1	29	2.65	2	2
6	4	VIMU	13212201600	13212633600	13227926400	2	25	2.74	1	1
7	26	GUMA	13215398400	13215916800	13227494400	2	27	2.90	2	1
8	25	GRSE	13215312000	13216176000	13216521600	1	34	2.95	2	2
9	7	VERI	13212374400	13213238400	13219632000	2	35	2.97	1	1
10	14	POCA	13214361600	13215312000	13224643200	2	36	3.05	1	2
11	3	COMO	13212028800	13213324800	13212633600	1	44	3.15	2	2
12	17	ANZO	13214793600	13215312000	13226889600	2	27	3.15	1	2
13	21	ROPA	13214966400	13215830400	13217385600	1	35	3.15	2	2
14	8	ADJU	13212460800	13212806400	13219718400	2	23	3.20	1	2
15	18	MEVE	13214793600	13215571200	13236566400	2	32	3.32	2	2
16	1	GADI	13211856000	13212288000	13215484800	2	24	3.38	2	1
17	13	FUFE	13213756800	13214707200	13214361600	1	36	3.40	2	2
18	16	BOPE	13214448000	13215571200	13230777600	1	40	3.40	1	2
19	27	PERI	13215398400	13215830400	13230518400	2	25	3.44	1	2
20	11	LOKO	13212892800	13213411200	13217731200	1	26	3.45	1	2
21	12	FRFU	13212979200	13213584000	13234752000	2	29	3.45	2	2
22	28	MAPE	13215398400	13215830400	13225075200	1	24	3.53	2	2
23	5	PAVI	13212288000	13212806400	13212892800	1	27	3.60	2	2
24	15	LOLO	13214361600	13214448000	13214966400	1	17	3.60	1	2
25	22	LOMA	13215052800	13215571200	13234406400	2	27	3.70	1	2
26	24	CAGI	13215225600	13215398400	13215830400	2	18	3.75	2	2
27	23	CEMA	13215139200	13215571200	13216953600	1	24	3.79	1	2
28	20	PUPI	13214966400	13215225600	13222224000	2	21	4.46	2	2

hermanos

1 2

Por comodidad, se pasan los nombres de las variables a minúsculas

```
> names(datos) <- tolower(names(datos))
```

Para ver las etiquetas de todas las variables

```
> cbind(attr(datos, "variable.labels"))
```

```
      [,1]  
id      ""  
ini     "Iniciales del niño"  
dia_nac "Dia nacimiento"  
dia_entr "Dia entrada en el estudio"  
ulti_lac "Ultimo dia lactancia"  
tx       "Regimen visitas asignado"  
edad     "Edad de la madre"  
peso     "peso del niño"  
sexo     "sexo de la criatura"  
tip_par  "Tipo de parto"  
hermanos "Tiene hermanos "
```

Para ver las etiquetas de valor de una variable (p.e. sexo)

```
> attr(datos$sexo, "value.labels")
```

```
niña niño  
2     1
```


Para ver las etiquetas de valor de todas las variables, usamos la función `lapply`. Observe que las variables sin etiquetas de valor (por ejemplo `id`) devuelven `NULL`.

```
> lapply(datos, function(x) attr(x, "value.labels"))
```

```
$id
```

```
NULL
```

```
$ini
```

```
NULL
```

```
$dia_nac
```

```
NULL
```

```
$dia_entr
```

```
NULL
```

```
$multi_lac
```

```
NULL
```

```
$tx
```

Intensivo	Estándar
2	1

```
$edad
```

```
NULL
```

```
$peso
```

```
NULL
```

```
$sexo
```

niña	niño
2	1

Para arreglar las variables de formato fecha, y ponerlos en formato d/m/a:

```
> datos$dia_nac <- format(ISOdate(1582, 10, 14) + datos$dia_nac,
+ "%d/%m/%Y")
> datos$dia_entr <- format(ISOdate(1582, 10, 14) + datos$dia_entr,
+ "%d/%m/%Y")
> datos$ulti_lac <- format(ISOdate(1582, 10, 14) + datos$ulti_lac,
+ "%d/%m/%Y")
> datos
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo	tip_par
1	10	JUNA	23/06/2001	02/07/2001	29/09/2001	2	32	2.10	2	2
2	9	BEMI	22/06/2001	05/07/2001	31/08/2001	1	40	2.40	2	2
3	2	CAEL	15/06/2001	21/06/2001	15/02/2002	2	27	2.50	1	2
4	6	PASA	20/06/2001	01/07/2001	27/06/2001	1	36	2.65	2	1
5	19	TOPO	19/07/2001	26/07/2001	11/10/2001	1	29	2.65	2	2
6	4	VIMU	18/06/2001	23/06/2001	17/12/2001	2	25	2.74	1	1
7	26	GUMA	25/07/2001	31/07/2001	12/12/2001	2	27	2.90	2	1
8	25	GRSE	24/07/2001	03/08/2001	07/08/2001	1	34	2.95	2	2
9	7	VERI	20/06/2001	30/06/2001	12/09/2001	2	35	2.97	1	1
10	14	POCA	13/07/2001	24/07/2001	09/11/2001	2	36	3.05	1	2
11	3	COMO	16/06/2001	01/07/2001	23/06/2001	1	44	3.15	2	2
12	17	ANZO	18/07/2001	24/07/2001	05/12/2001	2	27	3.15	1	2
13	21	ROPA	20/07/2001	30/07/2001	17/08/2001	1	35	3.15	2	2
14	8	ADJU	21/06/2001	25/06/2001	13/09/2001	2	23	3.20	1	2
15	18	MEVE	18/07/2001	27/07/2001	27/03/2002	2	32	3.32	2	2
16	1	GADI	14/06/2001	19/06/2001	26/07/2001	2	24	3.38	2	1
17	13	FUFE	06/07/2001	17/07/2001	13/07/2001	1	36	3.40	2	2
18	16	BOPE	14/07/2001	27/07/2001	19/01/2002	1	40	3.40	1	2
19	27	PERI	25/07/2001	30/07/2001	16/01/2002	2	25	3.44	1	2
20	11	LOKO	26/06/2001	02/07/2001	21/08/2001	1	26	3.45	1	2
21	12	FRFU	27/06/2001	04/07/2001	06/03/2002	2	29	3.45	2	

Lectura de ficheros de STATA (.dta)

```
> datos <- read.dta("C:/cursoR/data/partoFin.dta", convert.dates = TRUE,
+   convert.factors = TRUE)
```

```
> datos
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo
1	1	GADI	2001-06-14	2001-06-19	2001-07-26	intensivo	24	3.38	niña
2	2	CAEL	2001-06-15	2001-06-21	2002-02-15	intensivo	27	2.50	niño
3	3	COMO	2001-06-16	2001-07-01	2001-06-23	estándar	44	3.15	niña
4	4	VIMU	2001-06-18	2001-06-23	2001-12-17	intensivo	25	2.74	niño
5	5	PAVI	2001-06-19	2001-06-25	2001-06-26	estándar	27	3.60	niña
6	6	PASA	2001-06-20	2001-07-01	2001-06-27	estándar	36	2.65	niña
7	7	VERI	2001-06-20	2001-06-30	2001-09-12	intensivo	35	2.97	niño
8	8	ADJU	2001-06-21	2001-06-25	2001-09-13	intensivo	23	3.20	niño
9	9	BEMI	2001-06-22	2001-07-05	2001-08-31	estándar	40	2.40	niña
10	10	JUNA	2001-06-23	2001-07-02	2001-09-29	intensivo	32	2.10	niña
11	11	LOKO	2001-06-26	2001-07-02	2001-08-21	estándar	26	3.45	niño
12	12	FRFU	2001-06-27	2001-07-04	2002-03-06	intensivo	29	3.45	niña
13	13	FUFE	2001-07-06	2001-07-17	2001-07-13	estándar	36	3.40	niña
14	14	POCA	2001-07-13	2001-07-24	2001-11-09	intensivo	36	3.05	niño
15	15	LOLO	2001-07-13	2001-07-14	2001-07-20	estándar	17	3.60	niño
16	16	BOPE	2001-07-14	2001-07-27	2002-01-19	estándar	40	3.40	niño
17	17	ANZO	2001-07-18	2001-07-24	2001-12-05	intensivo	27	3.15	niño
18	18	MEVE	2001-07-18	2001-07-27	2002-03-27	intensivo	32	3.32	niña
19	19	TOPO	2001-07-19	2001-07-26	2001-10-11	estándar	29	2.65	niña
20	20	PUPI	2001-07-20	2001-07-23	2001-10-12	intensivo	21	4.46	niña
21	21	ROPA	2001-07-20	2001-07-30	2001-08-17	estándar	35	3.15	niña
22	22	LOMA	2001-07-21	2001-07-27	2002-03-02	intensivo	27	3.70	niño
23	23	CEMA	2001-07-22	2001-07-27	2001-08-12	estándar	24	3.79	niño
24	24	CAGI	2001-07-23	2001-07-25	2001-07-30	intensivo	18	3.75	niña
25	25	GRSE	2001-07-24	2001-08-03	2001-08-07	estándar	34	2.95	niña
26	26	GUMA	2001-07-25	2001-07-31	2001-12-12	intensivo	27	2.90	niña
27	27	PERI	2001-07-25	2001-07-30	2002-01-16	intensivo	25	3.44	niño
28	28	MAPE	2001-07-25	2001-07-30	2001-11-14	estándar	24	3.53	niña

tip par hermanos fuma an fuma de horas an horas de naci ca masde12

Lectura de ficheros de CSV (comma-separated values, .csv)

```
> datos <- read.csv("parto2.csv", header = TRUE, sep = ";", quote = "\"",
+   dec = ".")
```

```
> datos
```

	ini	dia_nac	dia_entr	dia_ult	tx	edad	peso	sex
1	GADI	14-jun-01	19-jun-01	26-jul-01	Intensivo	24	3,38/3,48	Niña
2	CAEL	15-jun-01	21-jun-01	15-feb-02	Intensivo	27	2.5	Niño
3	COMO	16-jun-01	01-jul-01	23-jun-01	Estandar	44	3.15	Niña
4	VIMU	18-jun-01	23-jun-01	17-dic-01	Intensivo	25	2.74	Niño
5	PAVI	19-jun-01	25-jun-01	26-jun-01	Estandar	27	3.6	Niña
6	PASA	20-jun-01	01-jul-01	27-jun-01	Estandar	36	2.65	Niña
7	VERI	20-jun-01	30-jun-01	12-sep-01	Intensivo	35	2.97	Niño
8	ADJU	21-jun-01	25-jun-01	13-sep-01	Intensivo	23	3.2	Niño
9	BEMI	22-jun-01	05-jul-01	31-ago-01	Estandar	40	2.4	Niña
10	JUNA	23-jun-01	02-jul-01	29-sep-01	Intensivo	32	2.1	Niña
11	LOKO	26-jun-01	02-jul-01	21-ago-01	Estandar	26	3.45	Niño
12	FRFU	27-jun-01	04-jul-01	06-mar-02	Intensivo	29	3.45	Niña
13	FUFE	06-jul-01	17-jul-01	13-jul-01	Estandar	36	3.4	Niña
14	POCA	13-jul-01	24-jul-01	09-nov-01	Intensivo	36	3.05	Niño
15	LOLO	13-jul-01	14-jul-01	20-jul-01	Estandar	17	3.6	Niño
16	BOPE	14-jul-01	27-jul-01	19-ene-02	Estandar	40	3.4	Niño
17	ANZO	18-jul-01	24-jul-01	05-dic-01	Intensivo	27	3.15	Niño
18	MEVE	18-jul-01	27-jul-01	27-mar-02	Intensivo	32	3.32	Niña
19	TOPO	19-jul-01	26-jul-01	11-oct-01	Estandar	29	2.65	Niña
20	PUPI	20-jul-01	23-jul-01	12-oct-01	Intensivo	21	4.46	Niña
21	ROPA	20-jul-01	30-jul-01	17-ago-01	Estandar	35	3.15	Niña
22	LOMA	21-jul-01	27-jul-01	02-mar-02	Intensivo	27	3.7	Niño
23	CEMA	22-jul-01	27-jul-01	12-ago-01	Estandar	24	3.79	Niño
24	CAGI	23-jul-01	25-jul-01	30-jul-01	Intensivo	18	3.75	Niña
25	GRSE	24-jul-01	03-ago-01	07-ago-01	Estandar	34	2.95	Niña
26	GUMA	25-jul-01	31-jul-01	12-dic-01	Intensivo	27	2.9	Niña
27	PERI	25-jul-01	30-jul-01	16-ene-02	Intensivo	25	3.44	Niño
28	MAPE	25-jul-01	30-jul-01	14-nov-01	Estandar	24	3.53	Niña

tipparto hermano

Lectura de ficheros de SAS (.sas7bdat) hay dos formas:

```
> library(Hmisc)
> old.wd <- getwd()
> setwd("C:/Archivos de programa/SAS/SAS 9.1")
> datos <- sas.get("C:/cursoR/data", "partofin")
> setwd(old.wd)

> datos <- read.ssd("C:/cursoR/data/", "partofin", tmpXport = tempfile(),
+   tmpProgLoc = tempfile(), sascmd = "C:/Archivos de programa/SAS/SAS 9.1/sas.e
```

Obsérvese que en ambos casos hay que especificar dónde se encuentra `sas.exe`

Lectura de unos datos en (.RData)

```
> load("datos.RData")
```

Para ver los objetos e identificar el nombre de la base de datos que se acaba de leer.

```
> ls()
```

```
[1] "canal" "datos" "query" "x1"    "x2"    "xxx"
```

Para exportar una tabla de datos (`data.frame`) ó una matriz se usa la función `write.table`, análoga a `read.table`. La opción `quote=FALSE` hace que no se escriban comillas en las variables cadena.

```
> write.table(datos, "parto2ex.dat")
```

Para exportar una tabla de datos (`data.frame`) ó una matriz se usa la función `write.xls`, del paquete `xlsReadWrite`:

```
> write.xls(datos, "parto2ex.xls")
```


Para exportar una tabla de datos (`data.frame`) ó una matriz a una tabla de una base de datos ACCESS **ya existente**, por ejemplo a 'spss.mdb':

```
> canal <- odbcConnectAccess("spss.mdb")  
> sqlSave(canal, datos, "parto2ex")  
> odbcClose(canal)
```

NOTA: es importante mirar cómo quedan las variables fecha en la tabla de ACCESS
quizá hará falta hacer alguna recodificación o modificación dentro ACCESS.

Para exportar una tabla de datos (`data.frame`) ó una matriz a SPSS

```
> library(foreign)
> codefile <- tempfile()
> write.foreign(datos, "C:/cursoR/data/xxx.txt", codefile, package = "SPSS")
> file.show(codefile)
> unlink(codefile)
```

Posteriormente habrá que leer los datos almacenados (en el archivo "xxx.txt" en este ejemplo) desde SPSS como archivo de texto (ASCII).

Para exportar a STATA:

```
> library(foreign)  
> write.dta(datos, file = "C:/cursoR/data/xxx.dta", version = 7L)
```

Para exportar a SAS:

```
> help(package = "SASxport")
```

Para exportar a  un `data.frame`:

```
> save(datos, file = "C:/cursoR/data/xxx.Rdata")
```

Para exportar a  todo el *current workspace*:



```
> save.image(file = "C:/cursoR/data/xxx.Rdata")
```

La base de datos “ucias.sav” contiene información de la urgencias atendidas entre el 2-dic-2001 hasta el 31-mar-2002.

Se tienen datos de 8 diferentes hospitales. Esta información se quiere agrupar de manera que la base de datos resultante sea la suma de todas las urgencias que han atendido diariamente estos 8 hospitales.

```
> rm(list = ls())
> library(foreign)
> setwd("C:/cursoR/data")
> datos <- read.spss("ucias.sav", FALSE, TRUE)
> names(datos) <- tolower(names(datos))
> datos <- datos[order(datos$dia, datos$centre_c), ]
> datos[1:30, ]
```

	centre_c	dia	urg_ate	urg_ingr	c14hcon	c_crit
361	1	13226630400	304	35	11	14
601	2	13226630400	58	0	11	11
481	3	13226630400	51	5	1	2
1	4	13226630400	155	7	0	0
121	4	13226630400	291	27	13	26
241	4	13226630400	259	24	39	42
721	5	13226630400	192	19	34	34
841	8	13226630400	348	37	16	22
961	9	13226630400	8	1	0	0
1081	10	13226630400	107	15	0	0
362	1	13226716800	297	47	9	10
602	2	13226716800	90	1	0	0
482	3	13226716800	133	9	2	2

La variable **dia** está en formato SPSS (los segundos que han pasado desde 14-oct-1582). Éste es el **momento 0** para el SPSS. Para  el **momento 0** es 01-01-1070. Para pasar la variable día a formato fecha para que  pueda trabajar con ella, primero hay que cargar el paquete `chron` y después aplicar la siguiente transformación:

```
> library(chron)
> spss <- as.integer(chron("14-10-1582", format = "d-m-Y", out.format = "d-mon-Y"))
> datos$dia2 <- with(datos, dia/(24 * 60 * 60))
> datos$dia3 <- with(datos, dia2 + spss)
> datos$dia4 <- with(datos, chron(dia3, out.format = "d-mon-Y"))
> datos[1:30, ]
```

	centre_c	dia	urg_ate	urg_ingr	c14hcon	c_crit	dia2	dia3
361	1	13226630400	304	35	11	14	153086	11658
601	2	13226630400	58	0	11	11	153086	11658
481	3	13226630400	51	5	1	2	153086	11658
1	4	13226630400	155	7	0	0	153086	11658
121	4	13226630400	291	27	13	26	153086	11658
241	4	13226630400	259	24	39	42	153086	11658
721	5	13226630400	192	19	34	34	153086	11658
841	8	13226630400	348	37	16	22	153086	11658
961	9	13226630400	8	1	0	0	153086	11658
1081	10	13226630400	107	15	0	0	153086	11658
362	1	13226716800	297	47	9	10	153087	11659
602	2	13226716800	90	1	0	0	153087	11659
482	3	13226716800	133	9	2	2	153087	11659
2	4	13226716800	209	7	0	0	153087	11659

La base de datos resultante de agregar por días, debería devolver para el día 2-dic-2001:

$$304+58+51+155+291+259+192+348+8+107 = 1773$$

```
> pordia <- with(datos, aggregate(urg_ate, list(diaucias = dia4),  
+   sum))  
> pordia[1:30, ]
```

	diaucias	x
1	02-Dec-2001	1773
2	03-Dec-2001	2200
3	04-Dec-2001	1958
4	05-Dec-2001	1855
5	06-Dec-2001	1841
6	07-Dec-2001	1950
7	08-Dec-2001	1793
8	09-Dec-2001	1770
9	10-Dec-2001	2075
10	11-Dec-2001	1944
11	12-Dec-2001	1804
12	13-Dec-2001	1914
13	14-Dec-2001	1657
14	15-Dec-2001	1594
15	16-Dec-2001	1710
16	17-Dec-2001	1958
17	18-Dec-2001	1742
18	19-Dec-2001	1847
19	20-Dec-2001	1790
20	21-Dec-2001	1735
21	22-Dec-2001	1899
22	23-Dec-2001	1906
23	24-Dec-2001	1851
24	25-Dec-2001	1673
25	26-Dec-2001	1976

Obsérvese que el nombre de la variable suma de urgencias es “x” y que el día 30-dic-2001 devuelve “NA”. El primer problema se soluciona con:

```
> names(pordia)[2] <- "uciasot"
```

Para ver por que ocurre el segundo problema podemos ejecutar cualquiera de las dos siguientes instrucciones:

```
> datos[which(datos$dia4 == chron("30-12-2001", format = "d-m-Y",
+ out.format = "d-mon-Y")), ]
```

	centre_c	dia	urg_ate	urg_ingr	c14hcon	c_crit	dia2	dia3
389	1	13229049600	390	60	13	14	153114	11686
629	2	13229049600	61	2	0	0	153114	11686
509	3	13229049600	80	9	2	2	153114	11686
29	4	13229049600	168	3	6	6	153114	11686
149	4	13229049600	362	35	29	29	153114	11686
269	4	13229049600	275	22	91	91	153114	11686
749	5	13229049600	NA	122	45	45	153114	11686
869	8	13229049600	454	62	33	36	153114	11686
989	9	13229049600	18	5	0	0	153114	11686
1109	10	13229049600	120	26	0	0	153114	11686
	dia4							
389	30-Dec-2001							
629	30-Dec-2001							
509	30-Dec-2001							
29	30-Dec-2001							
149	30-Dec-2001							
269	30-Dec-2001							
749	30-Dec-2001							
869	30-Dec-2001							
989	30-Dec-2001							
1109	30-Dec-2001							

Se puede agregar por más de una variable:

```
> datos <- read.spss("partoFin.sav", FALSE, TRUE)
> names(datos) <- tolower(names(datos))
> with(datos, aggregate(sem_lac, list(tto = tx, sexo = sexo), mean))
```

	tto	sexo	x
1	1	1	9.750000
2	2	1	22.375000
3	1	2	5.333333
4	2	2	17.857143

Para convertir una variable continua en categorías:

```
> datos$edadcat <- car::recode(datos$edad, "lo:20=1;21:30=2;31:hi=3")  
> with(datos, cbind(edad, edadcat))
```

	edad	edadcat
[1,]	24	2
[2,]	27	2
[3,]	44	3
[4,]	25	2
[5,]	27	2
[6,]	36	3
[7,]	35	3
[8,]	23	2
[9,]	40	3
[10,]	32	3
[11,]	26	2
[12,]	29	2
[13,]	36	3
[14,]	36	3
[15,]	17	1
[16,]	40	3
[17,]	27	2
[18,]	32	3
[19,]	29	2
[20,]	21	2
[21,]	35	3
[22,]	27	2
[23,]	24	2
[24,]	18	1
[25,]	34	3
[26,]	27	2
[27,]	25	2
[28,]	24	2

La función **cut** hace lo mismo pero devuelve las categorías como un **factor**:

```
> datos$edadcat2 <- cut(datos$edad, c(-Inf, 20, 30, Inf), right = TRUE)
> datos[, c("edad", "edadcat", "edadcat2")]
```

	edad	edadcat	edadcat2
1	24	2	(20,30]
2	27	2	(20,30]
3	44	3	(30, Inf]
4	25	2	(20,30]
5	27	2	(20,30]
6	36	3	(30, Inf]
7	35	3	(30, Inf]
8	23	2	(20,30]
9	40	3	(30, Inf]
10	32	3	(30, Inf]
11	26	2	(20,30]
12	29	2	(20,30]
13	36	3	(30, Inf]
14	36	3	(30, Inf]
15	17	1	(-Inf,20]
16	40	3	(30, Inf]
17	27	2	(20,30]
18	32	3	(30, Inf]
19	29	2	(20,30]
20	21	2	(20,30]
21	35	3	(30, Inf]
22	27	2	(20,30]
23	24	2	(20,30]
24	18	1	(-Inf,20]
25	34	3	(30, Inf]
26	27	2	(20,30]
27	25	2	(20,30]
28	24	2	(20,30]

Para reconvertir los niveles ("levels") de un factor a números:

```
> levels(datos$edadcat2)
```

```
[1] "(-Inf,20]" "(20,30]" "(30, Inf]"
```

```
> datos$edadcat2 <- as.integer(datos$edadcat2)
```

```
> datos[, c("edad", "edadcat", "edadcat2")]
```

	edad	edadcat	edadcat2
1	24	2	2
2	27	2	2
3	44	3	3
4	25	2	2
5	27	2	2
6	36	3	3
7	35	3	3
8	23	2	2
9	40	3	3
10	32	3	3
11	26	2	2
12	29	2	2
13	36	3	3
14	36	3	3
15	17	1	1
16	40	3	3
17	27	2	2
18	32	3	3
19	29	2	2
20	21	2	2
21	35	3	3
22	27	2	2
23	24	2	2
24	18	1	1
25	34	3	3
26	27	2	2
27	25	2	2

Para crear grupos de igual tamaño (cuartiles por ejemplo):

```
> library(Hmisc)
> datos$edad4g <- cut2(datos$edad, g = 4)
> with(datos, table(edad4g))
```

```
edad4g
[17,25) [25,29) [29,36) [36,44]
      7       8       7       6
```

```
> datos$edad4g <- as.integer(datos$edad4g)
> with(datos, table(edad4g))
```

```
edad4g
1 2 3 4
7 8 7 6
```

El data frame “nausea.sav” contiene datos de episodios de nausea en 5 turnos de enfermería de 19 pacientes intervenidos quirúrgicamente

```
> datos <- read.spss("nausea.sav", FALSE, TRUE)
> names(datos) <- tolower(names(datos))
> datos
```

	ident	naus1	naus2	naus3	naus4	naus5
1	1	1	1	0	0	0
2	2	1	1	0	0	0
3	3	0	0	0	1	1
4	4	1	1	0	0	0
5	5	1	1	1	0	0
6	6	0	0	0	0	0
7	7	1	0	0	0	0
8	8	1	1	0	0	0
9	9	1	0	0	0	0
10	10	1	1	0	0	0
11	11	1	0	0	0	0
12	12	1	1	1	0	0
13	13	0	1	0	1	0
14	14	1	1	0	0	0
15	15	0	1	0	1	0
16	16	0	0	0	0	0
17	17	0	0	0	0	0
18	18	1	0	0	0	0
19	19	0	0	0	0	0

Si se quiere contar el numero total de turnos en los que el paciente ha presentado nauseas:

```
> datos$total <- apply(datos[, 2:6] == 1, 1, sum)
> datos
```

	ident	naus1	naus2	naus3	naus4	naus5	total
1	1	1	1	0	0	0	2
2	2	1	1	0	0	0	2
3	3	0	0	0	1	1	2
4	4	1	1	0	0	0	2
5	5	1	1	1	0	0	3
6	6	0	0	0	0	0	0
7	7	1	0	0	0	0	1
8	8	1	1	0	0	0	2
9	9	1	0	0	0	0	1
10	10	1	1	0	0	0	2
11	11	1	0	0	0	0	1
12	12	1	1	1	0	0	3
13	13	0	1	0	1	0	2
14	14	1	1	0	0	0	2
15	15	0	1	0	1	0	2
16	16	0	0	0	0	0	0
17	17	0	0	0	0	0	0
18	18	1	0	0	0	0	1
19	19	0	0	0	0	0	0

En este ejemplo se muestran las “edades” de los individuos de la base de datos “partoFin.sav” que tienen la misma edad (están repetidos por edad):

```
> datos <- read.spss("partoFin.sav", FALSE, TRUE)
> names(datos) <- tolower(names(datos))
> temp <- with(datos, table(edad))
> repes <- cbind(temp[temp > 1])
> repes
```

```
      [,1]
24      3
25      2
27      5
29      2
32      2
35      2
36      3
40      2
```


Para ver los “id” de las mujeres que tienen la misma edad, ordenado por edad y dentro de edad, por id:

```
> repedad <- datos[order(datos$edad, datos$id), c("id", "edad")]  
> repedad[repedad$edad %in% rownames(repes), c("id", "edad")]
```

	id	edad
1	1	24
23	23	24
28	28	24
4	4	25
27	27	25
2	2	27
5	5	27
17	17	27
22	22	27
26	26	27
12	12	29
19	19	29
10	10	32
18	18	32
7	7	35
21	21	35
6	6	36
13	13	36
14	14	36
9	9	40
16	16	40

La base de datos (data frame) “transfos.sav” servirá de ejemplo para estas transformaciones.

```
> datos <- read.spss("transfos.sav", FALSE, TRUE)
> names(datos) <- tolower(names(datos))
> datos
```

	ident	peso	talla		cip	imc
1	1	80.0	178	MAVU0561129008	25.24934	
2	2	60.6	170	SIC01871228001	20.96886	
3	3	53.0	168	EDIC1840423006	18.77834	
4	4	27.4	125	VIS00000501012	17.53600	
5	5	12.6	85	GAC00060302004	17.43945	

Para re-calcular el IMC y almacenarlo en "imc2":

```
> datos$imc2 <- round(with(datos, peso/(talla/100)^2), 1)
> datos
```

	ident	peso	talla		cip	imc	imc2
1	1	80.0	178	MAVU0561129008	25.24934	25.2	
2	2	60.6	170	SIC01871228001	20.96886	21.0	
3	3	53.0	168	EDIC1840423006	18.77834	18.8	
4	4	27.4	125	VIS00000501012	17.53600	17.5	
5	5	12.6	85	GAC00060302004	17.43945	17.4	

Para extraer el sexo en formato cadena:

```
> datos$sexo <- with(datos, substr(cip, 5, 5))  
> datos
```

	ident	peso	talla	cip	imc	imc2	sexo
1	1	80.0	178	MAVU0561129008	25.24934	25.2	0
2	2	60.6	170	SIC01871228001	20.96886	21.0	1
3	3	53.0	168	EDIC1840423006	18.77834	18.8	1
4	4	27.4	125	VIS00000501012	17.53600	17.5	0
5	5	12.6	85	GAC00060302004	17.43945	17.4	0

```
> datos$sexo
```

```
[1] "0" "1" "1" "0" "0"
```

Para convertir la cadena de sexo a numérico:

```
> datos$sexo <- as.integer(datos$sexo)
```

```
> datos
```

	ident	peso	talla	cip	imc	imc2	sexo
1	1	80.0	178	MAVU0561129008	25.24934	25.2	0
2	2	60.6	170	SIC01871228001	20.96886	21.0	1
3	3	53.0	168	EDIC1840423006	18.77834	18.8	1
4	4	27.4	125	VIS00000501012	17.53600	17.5	0
5	5	12.6	85	GAC00060302004	17.43945	17.4	0

```
> datos$sexo
```

```
[1] 0 1 1 0 0
```

Para extraer el día, el mes y el año:

```
> datos$dia <- with(datos, substr(cip, 10, 11))
> datos$dia <- as.integer(datos$dia)
> datos$mes <- with(datos, substr(cip, 8, 9))
> datos$mes <- as.integer(datos$mes)
> datos$ano <- with(datos, substr(cip, 6, 7))
> datos$ano <- as.integer(datos$ano)
> datos
```

	ident	peso	talla		cip	imc	imc2	sexo	dia	mes	ano
1	1	80.0	178	MAVU0561129008	25.24934	25.2	0	29	11	56	
2	2	60.6	170	SIC01871228001	20.96886	21.0	1	28	12	87	
3	3	53.0	168	EDIC1840423006	18.77834	18.8	1	23	4	84	
4	4	27.4	125	VISO00000501012	17.53600	17.5	0	1	5	0	
5	5	12.6	85	GAC00060302004	17.43945	17.4	0	2	3	6	

Para arreglar el año (añadiendo 2000 si es ≤ 6 y 1900 al resto):

```
> datos$ano <- with(datos, ifelse(ano <= 6, ano + 2000, ano + 1900))
> datos
```

	ident	peso	talla		cip	imc	imc2	sexo	dia	mes	ano
1	1	80.0	178	MAVU0561129008	25.24934	25.2	0	29	11	1956	
2	2	60.6	170	SIC01871228001	20.96886	21.0	1	28	12	1987	
3	3	53.0	168	EDIC1840423006	18.77834	18.8	1	23	4	1984	
4	4	27.4	125	VIS00000501012	17.53600	17.5	0	1	5	2000	
5	5	12.6	85	GAC00060302004	17.43945	17.4	0	2	3	2006	

Para crear la fecha de nacimiento (se convierte primero a cadena):

```
> datos$fnac <- with(datos, apply(cbind(dia, mes, ano), 1, paste,
+   collapse = "-"))
> datos
```

	ident	peso	talla	cip	imc	imc2	sexo	dia	mes	ano	fnac
1	1	80.0	178	MAVU0561129008	25.24934	25.2	0	29	11	1956	29-11-1956
2	2	60.6	170	SIC01871228001	20.96886	21.0	1	28	12	1987	28-12-1987
3	3	53.0	168	EDIC1840423006	18.77834	18.8	1	23	4	1984	23-4-1984
4	4	27.4	125	VIS00000501012	17.53600	17.5	0	1	5	2000	1-5-2000
5	5	12.6	85	GAC00060302004	17.43945	17.4	0	2	3	2006	2-3-2006

```
> datos$fnac
```

```
[1] "29-11-1956" "28-12-1987" "23-4-1984" "1-5-2000" "2-3-2006"
```


Para pasar la fecha de nacimiento de formato cadena a formato fecha:

```
> datos$fnac <- chron(datos$fnac, format = "d-m-y", out.format = "d-mon-Y")  
> datos$fnac
```

```
[1] 29-Nov-1956 28-Dec-1987 23-Apr-1984 01-May-2000 02-Mar-2006
```

Para calcular los días que han pasado desde la fecha de nacimiento hasta hoy.

Primero se calcula la fecha de hoy:

```
> hoy <- chron(format(Sys.time(), "%d-%m-%y"), format = "d-m-y",
+               out.format = "d-mon-Y")
> hoy
```

```
[1] 31-May-2011
```

```
> datos$dias <- with(datos, (hoy - fnac))
> datos
```

	ident	peso	talla		cip	imc	imc2	sexo	dia	mes	ano	fnac
1	1	80.0	178	MAVU0561129008	25.24934	25.2	0	29	11	1956	29-Nov-1956	
2	2	60.6	170	SIC01871228001	20.96886	21.0	1	28	12	1987	28-Dec-1987	
3	3	53.0	168	EDIC1840423006	18.77834	18.8	1	23	4	1984	23-Apr-1984	
4	4	27.4	125	VIS00000501012	17.53600	17.5	0	1	5	2000	01-May-2000	
5	5	12.6	85	GAC00060302004	17.43945	17.4	0	2	3	2006	02-Mar-2006	
	dias											
1	19906											
2	8555											
3	9899											
4	4047											
5	1916											

Si se quiere calcular los años que tiene el individuo (nótese la diferencia entre **trunc** y **ceiling**)

```
> datos$edad1 <- with(datos, dias/365.25)
> datos$edad2 <- with(datos, trunc(dias/365.25))
> datos$edad3 <- with(datos, ceiling(dias/365.25))
> datos
```

	ident	peso	talla		cip	imc	imc2	sexo	dia	mes	ano	fnac
1	1	80.0	178	MAVU0561129008	25.24934	25.2	0	29	11	1956	29-Nov-1956	
2	2	60.6	170	SIC01871228001	20.96886	21.0	1	28	12	1987	28-Dec-1987	
3	3	53.0	168	EDIC1840423006	18.77834	18.8	1	23	4	1984	23-Apr-1984	
4	4	27.4	125	VIS00000501012	17.53600	17.5	0	1	5	2000	01-May-2000	
5	5	12.6	85	GAC00060302004	17.43945	17.4	0	2	3	2006	02-Mar-2006	

	dias	edad1	edad2	edad3
1	19906	54.499658	54	55
2	8555	23.422313	23	24
3	9899	27.101985	27	28
4	4047	11.080082	11	12
5	1916	5.245722	5	6

Fecha del día de hoy.

```
> library(chron)
> hoy <- chron(format(Sys.time(), "%d-%m-%y"), format = "d-m-y",
+             out.format = "d-mon-Y")
> hoy
```

```
[1] 31-May-2011
```

días desde el día 01-Jan-1970.

```
> dias <- as.integer(hoy)
> dias
```

```
[1] 15125
```

De un entero se puede transformar en formato fecha.

```
> chron(dias, out.format = "d-mon-Y")
```

```
[1] 31-May-2011
```

“Por defecto” el programa utiliza las fechas calculando los días que han pasado desde el 1-ene-1970:

```
> chron(0, out.format = "d-mon-Y")
```

```
[1] 01-Jan-1970
```

Los días anteriores a 1-ene-1970 son valores negativos:

```
> fecha <- chron("25-12-1969", format = "d-m-Y", out.format = "d-mon-Y")
```

```
> as.integer(fecha)
```

```
[1] -7
```

Un número con decimales los transforma en h:m:s,

```
> chron(0.14, out.format = c(dates = "d-mon-Y", times = "h:m:s"))
```

```
[1] (01-Jan-1970 03:21:36)
```

Esta fecha la convierte a decimales,

```
> as.numeric(chron(0.14, out.format = c(dates = "d-mon-Y", times = "h:m:s")))
```

```
[1] 0.14
```

Para obtener el año de una fecha (el resultado es un factor!!):

```
> fecha <- chron("25-12-1969", format = "d-m-Y", out.format = "d-mon-Y")  
> ano <- years(fecha)  
> ano
```

```
[1] 1969  
Levels: 1969
```

Para convertir el factor en un entero:

```
> ano <- as.integer(as.character(ano))  
> ano
```

```
[1] 1969
```

Para obtener el mes:

```
> mes <- as.integer(months(fecha))  
> mes  
[1] 12
```

Para obtener el día:

```
> dia <- as.integer(days(fecha))  
> dia  
[1] 25
```

Para convertirlo otra vez en fecha:

```
> chron(paste(dia, mes, ano, sep = "-"), format = "d-m-Y", out.format = "d-mon-Y")  
[1] 25-Dec-1969
```


Para obtener el día de la semana:

```
> diasem <- weekdays(fecha, abbreviate = FALSE)
> diasem
```

```
[1] Thursday
```

```
7 Levels: Sunday < Monday < Tuesday < Wednesday < Thursday < ... < Saturday
```

Para convertirlo en número (1 = domingo, 2= lunes, etc.):

```
> diasem <- as.integer(diasem)
> diasem
```

```
[1] 5
```

Para saber si es fin de semana (Sábado o domingo):

```
> is.weekend(fecha)
```

```
[1] FALSE
```

Para saber si es: New Year, Memorial, Indepen., Labor, Thanksgiving o Christmas:

```
> is.holiday(fecha)
```

```
[1] FALSE
```

Supóngase que el nombre de tres individuos es (primero cargamos el paquete gdata):

```
> library(gdata)
> cadena <- as.data.frame(c("Salvador Vila", "Maria del Mar Muñoz",
+   "Joan Pol de la Casa"), stringsAsFactors = FALSE)
> names(cadena) <- "nombre"
> cadena
```

```
      nombre
1  Salvador Vila
2 Maria del Mar Muñoz
3 Joan Pol de la Casa
```

Para extraer información entre dos posiciones:

```
> cadena$posi2a5 <- with(cadena, substr(nombre, 2, 5))
> cadena
```

```
      nombre posi2a5
1  Salvador Vila   alva
2 Maria del Mar Muñoz   aria
3 Joan Pol de la Casa   oan
```

Para saber el número de caracteres que tiene una cadena:

```
> cadena$n_carac <- with(cadena, nchar(trim(nombre)))  
> cadena
```

	nombre	posi2a5	n_carac
1	Salvador Vila	alva	13
2	Maria del Mar Muñoz	aria	19
3	Joan Pol de la Casa	oan	19

Para separar en subcadenas las partes de una cadena. Obsérvese que se genera una **lista**:

```
> cadenaplit <- with(cadena, strsplit(nombre, " "))  
> cadenaplit
```

```
[[1]]  
[1] "Salvador" "Vila"  
  
[[2]]  
[1] "Maria" "del" "Mar" "Muñoz"  
  
[[3]]  
[1] "Joan" "Pol" "de" "la" "Casa"
```

De cada uno de estos objetos de la lista se puede extraer un elemento (p.e. el segundo):

```
> segundo <- lapply(cadenaplit, function(x) x[2])  
> segundo
```

```
[[1]]  
[1] "Vila"  
  
[[2]]  
[1] "del"  
  
[[3]]  
[1] "Pol"
```

Ahora el objeto “segundo” es una lista de tres objetos, con un elemento en cada uno.

Para convertir esta lista en un vector:

```
> elsegundo <- unlist(segundo)
> elsegundo
```

```
[1] "Vila" "del"  "Pol"
```

Este vector se añade al data frame cadena:

```
> cadena <- cbind(cadena, elsegundo)
> cadena
```

	nombre	posi2a5	n_carac	elsegundo
1	Salvador Vila	alva	13	Vila
2	Maria del Mar Muñoz	aria	19	del
3	Joan Pol de la Casa	oan	19	Pol

Para encontrar una cadena en un vector de caracteres (la función **grep/agrep**):

```
> library(foreign)
> setwd("C:/cursoR/data")
> datos <- read.spss("partoFin.sav", FALSE, TRUE)
> names(datos) <- tolower(names(datos))
> grep("dia", names(datos))
```

```
[1] 3 4
```

```
> grep("dia", names(datos), value = TRUE)
```

```
[1] "dia_nac" "dia_entr"
```

La función **grep** tiene que coincidir exactamente (al menos una parte):

```
> with(cadena, grep("María", nombre, value = TRUE))  
character(0)
```

La función **agrep** permite localizar una cadena de forma “aproximada”:

```
> with(cadena, agrep("María", nombre, value = TRUE))  
[1] "Maria del Mar Muñoz"
```


Para substituir un valor por su “coincidente” (i.e. Maria por María):

```
> cadena$nombre2 <- with(cadena, sub("Maria", "María", nombre))  
> cadena
```

	nombre	posi2a5	n_carac	elsegundo	nombre2
1	Salvador Vila	alva	13	Vila	Salvador Vila
2	Maria del Mar Muñoz	aria	19	del María del Mar Muñoz	
3	Joan Pol de la Casa	oan	19	Pol Joan Pol de la Casa	

Para juntar varias cadenas en un sólo vector. Supóngase que a tres individuos se les han registrado tres gérmenes:

Nota: la `I()` se utiliza para que no se convierta en factor.

```
> cadena$germen1 <- I(c("Calbicans", "EColi", "PAureoginosa"))
> cadena$germen2 <- I(c("Pcloacae", "Candida", "SAureus"))
> cadena$germen3 <- I(c("EColi", "spp", "Legionella"))
> cadena$germenes = with(cadena, apply(trim(cbind(germen1, germen2,
+ germen3)), 1, paste, collapse = ", "))
> cadena[, 5:9]
```

	nombre2	germen1	germen2	germen3
1	Salvador Vila	Calbicans	Pcloacae	EColi
2	María del Mar Muñoz	EColi	Candida	spp
3	Joan Pol de la Casa	PAureoginosa	SAureus	Legionella

	germenes
1	Calbicans, Pcloacae, EColi
2	EColi, Candida, spp
3	PAureoginosa, SAureus, Legionella

Se obtendría el mismo resultado con:


```
> cadena$xgermenes <- with(cadena, paste(germen1, germen2, germen3,
+   sep = ", "))
> cadena[, 6:10]
```

	germen1	germen2	germen3	germenes
1	Calbicans	Pcloacae	EColi	Calbicans, Pcloacae, EColi
2	EColi	Candida	spp	EColi, Candida, spp
3	PAureoginosa	SAureus	Legionella	PAureoginosa, SAureus, Legionella

	xgermenes
1	Calbicans, Pcloacae, EColi
2	EColi, Candida, spp
3	PAureoginosa, SAureus, Legionella

Para empezar, se eliminan todos los objetos y se fija la carpeta de trabajo donde están los datos. Se leen datos en SPSS, y por tanto también será necesario cargar el paquete `foreign`.

```
> rm(list = ls())  
> setwd("C:/cursoR/data")  
> library(foreign)
```

- Se añadirán los datos del fichero 'parto_extra.sav' a los que hay en 'parto4.sav'
- Primero se leen las bases de datos con la función `read.spss`, y se miran qué variables contiene cada una de ellas con la función `names`.
- Es importante pasar las variables de 'parto.extra' a minúscula; recuérdese que  distingue entre mayúsculas y minúsculas!:

```
> parto4 <- read.spss("parto4.sav", FALSE, TRUE)
> parto.extra <- read.spss("parto_extra.sav", FALSE, TRUE)
> names(parto.extra) <- tolower(names(parto.extra))
> names(parto4)
```

```
[1] "id"      "ini"      "dia_nac"  "dia_entr" "ulti_lac" "tx"
[7] "edad"    "peso"     "sexo"     "tip_par"  "hermanos" "nacion"
[13] "fuma_an" "fuma_de"  "horas_an" "horas_de"
```

```
> names(parto.extra)
```

```
[1] "id"      "ini"      "dia_nac"  "dia_entr" "ulti_lac" "tx"
[7] "edad"    "peso"     "sexo"     "tip_par"  "hermanos" "fuma_an"
[13] "fuma_de" "horas_an" "horas_de" "sem_lac"  "masde12"  "sem_12"
[19] "tx_2"
```

Para ver qué variables están en 'parto4' y no están en 'parto.extra', se usa el operador %in%

```
> names(parto4)[!names(parto4) %in% names(parto.extra)]
```

```
[1] "nacion"
```

Y qué variables están en 'parto.extra' y no están en 'parto4'

```
> names(parto.extra)[!names(parto.extra) %in% names(parto4)]
```

```
[1] "sem_lac" "masde12" "sem_12" "tx_2"
```

Se añadirán los casos y sólo las variables comunes en las dos bases de datos. Para ver las variables que están en las dos bases de datos, se usa la función `intersect`

```
> vars.comunes <- intersect(names(parto4), names(parto.extra))  
> cbind(vars.comunes)
```

```
      vars.comunes  
[1,] "id"  
[2,] "ini"  
[3,] "dia_nac"  
[4,] "dia_entr"  
[5,] "ulti_lac"  
[6,] "tx"  
[7,] "edad"  
[8,] "peso"  
[9,] "sexo"  
[10,] "tip_par"  
[11,] "hermanos"  
[12,] "fuma_an"  
[13,] "fuma_de"  
[14,] "horas_an"  
[15,] "horas_de"
```

Para añadir los casos, se usa la función `rbind` que combina por filas dos `data.frames` (ó también matrices) pero sólo se puede usar si las variables de los dos `data.frames` son las mismas y en el mismo orden; para ello se escribe `[,vars.comunes]`:

```
> parto5 <- rbind(parto4[, vars.comunes], parto.extra[, vars.comunes])
> parto5
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo	tip_par
1	1	GADI	13211856000	13212288000	13215484800	2	24	3.38	2	1
2	2	CAEL	13211942400	13212460800	13233110400	2	27	2.50	1	2
3	3	COMO	13212028800	13213324800	13212633600	1	44	3.15	2	2
4	4	VIMU	13212201600	13212633600	13227926400	2	25	2.74	1	1
5	5	PAVI	13212288000	13212806400	13212892800	1	27	3.60	2	2
6	6	PASA	13212374400	13213324800	13212979200	1	36	2.65	2	1
7	7	VERI	13212374400	13213238400	13219632000	2	35	2.97	1	1
8	8	ADJU	13212460800	13212806400	13219718400	2	23	3.20	1	2
9	9	BEMI	13212547200	13213670400	13218595200	1	40	2.40	2	2
10	10	JUNA	13212633600	13213411200	13221100800	2	32	2.10	2	2
11	11	LOKO	13212892800	13213411200	13217731200	1	26	3.45	1	2
12	12	FRFU	13212979200	13213584000	13234752000	2	29	3.45	2	2
13	13	FUFE	13213756800	13214707200	13214361600	1	36	3.40	2	2
14	14	POCA	13214361600	13215312000	13224643200	2	36	3.05	1	2
15	15	LOLO	13214361600	13214448000	13214966400	1	17	3.60	1	2
16	16	BOPE	13214448000	13215571200	13230777600	1	40	3.40	1	2
17	17	ANZO	13214793600	13215312000	13226889600	2	27	3.15	1	2
18	18	MEVE	13214793600	13215571200	13236566400	2	32	3.32	2	2
19	19	TOPO	13214880000	13215484800	13222137600	1	29	2.65	2	2
20	20	PUPI	13214966400	13215225600	13222224000	2	21	4.46	2	2
21	21	ROPA	13214966400	13215830400	13217385600	1	35	3.15	2	2
22	22	LOMA	13215052800	13215571200	13234406400	2	27	3.70	1	2
23	23	CEMA	13215139200	13215571200	13216953600	1	24	3.79	1	2
24	24	CAGI	13215225600	13215398400	13215830400	2	18	3.75	2	2

Se comprueba el número de filas de cada base de datos para ver que se ha añadido los individuos de 'parto.extra'

```
> dim(parto4)
```

```
[1] 28 16
```

```
> dim(parto.extra)
```

```
[1] 7 19
```

```
> dim(parto5)
```

```
[1] 35 15
```

Para añadir las variables del fichero 'parto3.sav' en la base de datos 'parto2', primero se lee el fichero 'parto3.sav':

```
> parto2 <- read.spss("parto2.sav", FALSE, TRUE)
> parto3 <- read.spss("parto3.sav", FALSE, TRUE)
> parto3
```

	id	nacion	fuma_an	fuma_de	horas_an	horas_de
1	1	Sudamérica	1	0	6	2
2	2	Española	0	0	2	2
3	3	Española	0	1	3	0
4	4	Otras	1	1	11	6
5	5	Española	1	0	10	22
6	6	Española	0	0	9	9
7	7	Española	0	0	8	8
8	8	Otras	0	0	5	2
9	9	Española	1	1	12	10
10	10	Sudamérica	1	1	7	0
11	11	Sudamérica	1	1	7	14
12	12	Sudamérica	1	0	12	11
13	13	Española	0	0	7	4
14	14	Española	1	1	7	3
15	15	Sudamérica	0	0	10	0
16	16	Española	0	0	5	9
17	17	Española	0	0	7	2
18	18	Otras	1	0	11	8
19	19	Española	0	1	3	1
20	20	Sudamérica	1	1	7	0
21	21	Española	1	0	5	4
22	22	Española	0	0	7	7
23	23	Sudamérica	1	1	4	10
24	24	Otras	1	0	11	7
25	25	Española	0	0	12	23
26	26	Otras	0	0	4	11
27	27	Otras	1	1	7	3

Se mira qué variables contiene 'parto3'

```
> cbind(names(parto3))
```

```
      [,1]  
[1,] "id"  
[2,] "nacion"  
[3,] "fuma_an"  
[4,] "fuma_de"  
[5,] "horas_an"  
[6,] "horas_de"
```

Se mira cuántas filas tiene 'parto3' y 'parto2'

```
> nrow(parto3)
```

```
[1] 28
```

```
> nrow(parto2)
```

```
[1] 28
```

- Finalmente se añaden las variables de 'parto3' a 'parto2', con la función merge.
- La fusión se hace a partir de la variable identificadora 'id'.
- A diferencia de SPSS, no hace falta ordenar la base de datos por 'id'.

```
> parto4 <- merge(parto2, parto3, by = "id")
> parto4
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo	tip_par
1	1	GADI	13211856000	13212288000	13215484800	2	24	3.38	2	1
2	2	CAEL	13211942400	13212460800	13233110400	2	27	2.50	1	2
3	3	COMO	13212028800	13213324800	13212633600	1	44	3.15	2	2
4	4	VIMU	13212201600	13212633600	13227926400	2	25	2.74	1	1
5	5	PAVI	13212288000	13212806400	13212892800	1	27	3.60	2	2
6	6	PASA	13212374400	13213324800	13212979200	1	36	2.65	2	1
7	7	VERI	13212374400	13213238400	13219632000	2	35	2.97	1	1
8	8	ADJU	13212460800	13212806400	13219718400	2	23	3.20	1	2
9	9	BEMI	13212547200	13213670400	13218595200	1	40	2.40	2	2
10	10	JUNA	13212633600	13213411200	13221100800	2	32	2.10	2	2
11	11	LOKO	13212892800	13213411200	13217731200	1	26	3.45	1	2
12	12	FRFU	13212979200	13213584000	13234752000	2	29	3.45	2	2
13	13	FUFE	13213756800	13214707200	13214361600	1	36	3.40	2	2
14	14	POCA	13214361600	13215312000	13224643200	2	36	3.05	1	2
15	15	LOLO	13214361600	13214448000	13214966400	1	17	3.60	1	2
16	16	BOPE	13214448000	13215571200	13230777600	1	40	3.40	1	2
17	17	ANZO	13214793600	13215312000	13226889600	2	27	3.15	1	2
18	18	MEVE	13214793600	13215571200	13236566400	2	32	3.32	2	2
19	19	TOPO	13214880000	13215484800	13222137600	1	29	2.65	2	2
20	20	PUPI	13214966400	13215225600	13222224000	2	21	4.46	2	2
21	21	LOPA	13214966400	13215830400	13217385600	1	35	3.15	2	2
22	22	LOMA	13215028800	13215571200	13234406400	2	27	3.70	1	2

Para comprobar que se han añadido las variables, se listan los nombres de las variables de 'parto4':

```
> cbind(names(parto4))
```

```
      [,1]  
[1,] "id"  
[2,] "ini"  
[3,] "dia_nac"  
[4,] "dia_entr"  
[5,] "ulti_lac"  
[6,] "tx"  
[7,] "edad"  
[8,] "peso"  
[9,] "sexo"  
[10,] "tip_par"  
[11,] "hermanos"  
[12,] "nacion"  
[13,] "fuma_an"  
[14,] "fuma_de"  
[15,] "horas_an"  
[16,] "horas_de"
```

- La función `merge`, fusiona por defecto todos los individuos que están tanto en 'parto2' como en 'parto3'. Para poner todos los individuos de 'parto2' y sólo aquellos de 'parto3' que estén en 'parto2', hay que añadir el argumento `all.x=TRUE`.
- Aunque en este caso es indiferente porque hay los mismos individuos en 'parto2' y en 'parto3'.

Para eliminar variables existe una función muy útil en el paquete `gdata` que se llama `remove.vars`. Por ejemplo, para eliminar las variables 'sexo' y 'edad' de la base de datos 'partoFin.sav':

```
> library(gdata)
> datos <- read.spss("partoFin.sav", FALSE, TRUE)
> names(datos) <- tolower(names(datos))
> datos2 <- remove.vars(datos, c("sexo", "edad"))
```

```
Removing variable 'sexo'
Removing variable 'edad'
```

```
> names(datos2)
```

```
[1] "id"      "ini"      "dia_nac"  "dia_entr" "ulti_lac" "tx"
[7] "peso"    "tip_par"  "hermanos" "fuma_an"  "fuma_de"   "horas_an"
[13] "horas_de" "naci_ca"  "masde12"  "sem_lac"
```


Para renombrar variables se puede usar la función `rename.vars` también del paquete `gdata`. Por ejemplo, para pasar los nombre de 'edad' y 'sexo' al inglés:

```
> datos2 <- rename.vars(datos, from = c("sexo", "edad"), to = c("gender",  
+ "age"))
```

Changing in datos

From: sexo edad

To: gender age

```
> names(datos2)
```

```
[1] "id"      "ini"      "dia_nac"  "dia_entr" "ulti_lac" "tx"  
[7] "age"     "peso"     "gender"   "tip_par"  "hermanos" "fuma_an"  
[13] "fuma_de" "horas_an" "horas_de" "naci_ca"  "masde12"  "sem_lac"
```

Al leer unos datos de SPSS, las etiquetas de valor son un atributo de **cada** variable. Si se quiere cambiar las etiquetas de valor de la variable 'sexo' al ingles:

```
> attr(datos$sexo, "value.labels") <- c(Male = 1, Female = 2)
> attr(datos$sexo, "value.labels")
```

```
Male Female
  1       2
```

En cambio, las etiquetas de las variables se almacenan como un atributo en el `data.frame`. *Esto es si se ha leído el archivo con la función `read.spss` del paquete `foreign`. Si se ha usado otra función para leer datos de SPSS como es `spss.get` del paquete `Hmisc`, eso ya no es así (véase `?spss.get` y `?label`). Así, si se quiere cambiar la etiqueta de la variable 'sexo':*

```
> attr(datos, "variable.labels")["sexo"] <- "Gender"
> cbind(attr(datos, "variable.labels"))
```

```
      [,1]
id      ""
ini     "Iniciales del niño"
dia_nac "Dia nacimiento"
dia_entr "Dia entrada en el estudio"
ulti_lac "Ultimo dia lactancia"
tx       "Regimen visitas asignado"
edad     "Edad de la madre"
peso     "peso del niño"
sexo     "Gender"
tip_par  "Tipo de parto"
hermanos "Tiene hermanos "
fuma_an  "Fuma antes embarazo"
fuma_de  "Fuma despues embarazo"
horas_an "Horas ejercicio semanal antes embarazo"
horas_de "Horas ejercicio semanal despues embarazo"
naci_ca  "nacionalidad"
masde12  "Lactancia mas de 12 semanas"
sem_lac  "Semanas de lactancia"
```

Separar datos (split file): separar los individuos de una base de datos según una variable categórica formando grupos. Por ejemplo, para separar la base de datos 'parto2', según el sexo del bebé y asignarlo al objeto 'datosseg'.

```
> datosseg <- split(datos, datos$sexo)
```

El resultado es una lista cuyas componentes son un `data.frame`, uno para niños y otro para niñas. Para acceder a la primera componente:

```
> datosseg[[1]]
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo	tip_par
2	2 CAEL	13211942400	13212460800	13233110400	2	27	2.50	1	2	
4	4 VIMU	13212201600	13212633600	13227926400	2	25	2.74	1	1	
7	7 VERI	13212374400	13213238400	13219632000	2	35	2.97	1	1	
8	8 ADJU	13212460800	13212806400	13219718400	2	23	3.20	1	2	
11	11 LOKO	13212892800	13213411200	13217731200	1	26	3.45	1	2	
14	14 POCA	13214361600	13215312000	13224643200	2	36	3.05	1	2	
15	15 LOLO	13214361600	13214448000	13214966400	1	17	3.60	1	2	
16	16 BOPE	13214448000	13215571200	13230777600	1	40	3.40	1	2	
17	17 ANZO	13214793600	13215312000	13226889600	2	27	3.15	1	2	
22	22 LOMA	13215052800	13215571200	13234406400	2	27	3.70	1	2	
23	23 CEMA	13215139200	13215571200	13216953600	1	24	3.79	1	2	
27	27 PERI	13215398400	13215830400	13230518400	2	25	3.44	1	2	

	hermanos	fuma_an	fuma_de	horas_an	horas_de	naci_ca	masde12	sem_lac
2	1	0	0	2	2	1	1	35
4	1	1	1	11	6	2	1	26
7	2	0	0	8	8	1	0	12
8	2	0	0	5	2	2	0	12
11	2	1	1	7	14	3	0	8
14	2	1	1	7	3	1	1	17
15	2	0	0	10	0	3	0	1
16	2	0	0	5	9	1	1	27
17	1	0	0	7	2	1	1	20
22	1	0	0	7	7	1	1	32
23	1	1	1	4	10	3	0	3
27	1	1	1	7	3	2	1	25

Y para acceder a la segunda componente:

```
> datosseg[[2]]
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo	tip_par
1	1	GADI	13211856000	13212288000	13215484800	2	24	3.38	2	1
3	3	COMO	13212028800	13213324800	13212633600	1	44	3.15	2	2
5	5	PAVI	13212288000	13212806400	13212892800	1	27	3.60	2	2
6	6	PASA	13212374400	13213324800	13212979200	1	36	2.65	2	1
9	9	BEMI	13212547200	13213670400	13218595200	1	40	2.40	2	2
10	10	JUNA	13212633600	13213411200	13221100800	2	32	2.10	2	2
12	12	FRFU	13212979200	13213584000	13234752000	2	29	3.45	2	2
13	13	FUFE	13213756800	13214707200	13214361600	1	36	3.40	2	2
18	18	MEVE	13214793600	13215571200	13236566400	2	32	3.32	2	2
19	19	TOPO	13214880000	13215484800	13222137600	1	29	2.65	2	2
20	20	PUPI	13214966400	13215225600	13222224000	2	21	4.46	2	2
21	21	ROPA	13214966400	13215830400	13217385600	1	35	3.15	2	2
24	24	CAGI	13215225600	13215398400	13215830400	2	18	3.75	2	2
25	25	GRSE	13215312000	13216176000	13216521600	1	34	2.95	2	2
26	26	GUMA	13215398400	13215916800	13227494400	2	27	2.90	2	1
28	28	MAPE	13215398400	13215830400	13225075200	1	24	3.53	2	2

	hermanos	fuma_an	fuma_de	horas_an	horas_de	naci_ca	masde12	sem_lac
1	2	1	0	6	2	3	0	6
3	1	0	1	3	0	1	0	1
5	1	1	0	10	22	1	0	1
6	2	0	0	9	9	1	0	1
9	2	1	1	12	10	1	0	10
10	2	1	1	7	0	3	1	14
12	1	1	0	12	11	3	1	36
13	2	0	0	7	4	1	0	1
18	2	1	0	11	8	2	1	36
19	2	0	1	3	1	1	0	12
20	2	1	1	7	0	3	0	12
21	2	1	0	5	4	1	0	4
24	2	1	0	11	7	2	0	1
25	1	0	0	12	23	1	0	2
26	1	0	0	4	11	2	1	20

Aplicando la función `lapply`, se pueden hacer cálculos separados para niños y para niñas. Por ejemplo un `summary`:

```
> lapply(datosseg, summary)
```

```
$`1`
```

id		ini	dia_nac		dia_entr
Min.	: 2.00	ADJU	:1	Min. :1.321e+10	Min. :1.321e+10
1st Qu.:	7.75	ANZO	:1	1st Qu.:1.321e+10	1st Qu.:1.321e+10
Median	:14.50	BOPE	:1	Median :1.321e+10	Median :1.321e+10
Mean	:13.83	CAEL	:1	Mean :1.321e+10	Mean :1.321e+10
3rd Qu.:	18.25	CEMA	:1	3rd Qu.:1.321e+10	3rd Qu.:1.322e+10
Max.	:27.00	LOKO	:1	Max. :1.322e+10	Max. :1.322e+10

```
(Other) :6
```

ulti_lac		tx	edad		peso
Min.	:1.321e+10	Min. :1.000	Min. :17.00	Min. :2.500	
1st Qu.:	1.322e+10	1st Qu.:1.000	1st Qu.:24.75	1st Qu.:3.030	
Median	:1.323e+10	Median :2.000	Median :26.50	Median :3.300	
Mean	:1.322e+10	Mean :1.667	Mean :27.67	Mean :3.249	
3rd Qu.:	1.323e+10	3rd Qu.:2.000	3rd Qu.:29.00	3rd Qu.:3.487	
Max.	:1.323e+10	Max. :2.000	Max. :40.00	Max. :3.790	

sexo		tip_par	hermanos	fuma_an	fuma_de
Min.	:1	Min. :1.000	Min. :1.0	Min. :0.0000	Min. :0.0000
1st Qu.:	1	1st Qu.:2.000	1st Qu.:1.0	1st Qu.:0.0000	1st Qu.:0.0000
Median	:1	Median :2.000	Median :1.5	Median :0.0000	Median :0.0000
Mean	:1	Mean :1.833	Mean :1.5	Mean :0.4167	Mean :0.4167
3rd Qu.:	1	3rd Qu.:2.000	3rd Qu.:2.0	3rd Qu.:1.0000	3rd Qu.:1.0000
Max.	:1	Max. :2.000	Max. :2.0	Max. :1.0000	Max. :1.0000

horas_an		horas_de	naci_ca	masde12
Min.	: 2.000	Min. : 0.00	Min. :1.00	Min. :0.0000
1st Qu.:	5.000	1st Qu.: 2.00	1st Qu.:1.00	1st Qu.:0.0000
Median	: 7.000	Median : 4.50	Median :1.50	Median :1.0000
Mean	: 6.667	Mean : 5.50	Mean :1.75	Mean :0.5833
3rd Qu.:	7.000	3rd Qu.: 3.00	3rd Qu.:2.00	3rd Qu.:1.0000

Para deshacer la segmentación ó separación, se usa la función `unsplit`

```
> datos <- unsplit(datosseg, datos$sexo)
```

```
> datos
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo	tip_par
1	1	GADI	13211856000	13212288000	13215484800	2	24	3.38	2	1
2	2	CAEL	13211942400	13212460800	13233110400	2	27	2.50	1	2
3	3	COMO	13212028800	13213324800	13212633600	1	44	3.15	2	2
4	4	VIMU	13212201600	13212633600	13227926400	2	25	2.74	1	1
5	5	PAVI	13212288000	13212806400	13212892800	1	27	3.60	2	2
6	6	PASA	13212374400	13213324800	13212979200	1	36	2.65	2	1
7	7	VERI	13212374400	13213238400	13219632000	2	35	2.97	1	1
8	8	ADJU	13212460800	13212806400	13219718400	2	23	3.20	1	2
9	9	BEMI	13212547200	13213670400	13218595200	1	40	2.40	2	2
10	10	JUNA	13212633600	13213411200	13221100800	2	32	2.10	2	2
11	11	LOKO	13212892800	13213411200	13217731200	1	26	3.45	1	2
12	12	FRFU	13212979200	13213584000	13234752000	2	29	3.45	2	2
13	13	FUFE	13213756800	13214707200	13214361600	1	36	3.40	2	2
14	14	POCA	13214361600	13215312000	13224643200	2	36	3.05	1	2
15	15	LOLO	13214361600	13214448000	13214966400	1	17	3.60	1	2
16	16	BOPE	13214448000	13215571200	13230777600	1	40	3.40	1	2
17	17	ANZO	13214793600	13215312000	13226889600	2	27	3.15	1	2
18	18	MEVE	13214793600	13215571200	13236566400	2	32	3.32	2	2
19	19	TOPO	13214880000	13215484800	13222137600	1	29	2.65	2	2
20	20	PUPI	13214966400	13215225600	13222224000	2	21	4.46	2	2
21	21	ROPA	13214966400	13215830400	13217385600	1	35	3.15	2	2
22	22	LOMA	13215052800	13215571200	13234406400	2	27	3.70	1	2
23	23	CEMA	13215139200	13215571200	13216953600	1	24	3.79	1	2
24	24	CAGI	13215225600	13215398400	13215830400	2	18	3.75	2	2
25	25	GRSE	13215312000	13216176000	13216521600	1	34	2.95	2	2
26	26	GUMA	13215398400	13215916800	13227494400	2	27	2.90	2	1
27	27	PERI	13215398400	13215830400	13230518400	2	25	3.44	1	2
28	28	MAPE	13215398400	13215830400	13225075200	1	24	3.53	2	2

Para ilustrar la selección de variables se trabajará con la base de datos 'parto2'. Primero, por comodidad, se pasan los nombres de las variables a minúsculas.

```
> names(parto2) <- tolower(names(parto2))
```

Hay varias maneras de seleccionar ó indexar variables:
A partir del nombre de la variable (con el \$ ó entre [])

```
> datos$sexo
```

```
[1] 2 1 2 1 2 2 1 1 2 2 1 2 2 1 1 1 1 2 2 2 2 1 1 2 2 2 1 2
```

```
> datos$sexo
```

```
[1] 2 1 2 1 2 2 1 1 2 2 1 2 2 1 1 1 1 2 2 2 2 1 1 2 2 2 1 2
```

```
> datos[, "sexo"]
```

```
[1] 2 1 2 1 2 2 1 1 2 2 1 2 2 1 1 1 1 2 2 2 2 1 1 2 2 2 1 2
```

ó a partir de su posición actual en la base de datos.

```
> datos[, 9]
```

```
[1] 2 1 2 1 2 2 1 1 2 2 1 2 2 1 1 1 1 2 2 2 2 1 1 2 2 2 1 2
```

Si se quiere que al seleccionar sólo una variable (ó sólo una columna en tratarse de una matriz) el resultado continúe siendo un `data.frame` (ó una matriz) en lugar de un vector, hay que especificar el argumento `drop=FALSE`

```
> datos[, 9, drop = FALSE]
```

```
      sexo  
1         2  
2         1  
3         2  
4         1  
5         2  
6         2  
7         1  
8         1  
9         2  
10        2  
11        1  
12        2  
13        2  
14        1  
15        1  
16        1  
17        1  
18        2  
19        2  
20        2  
21        2  
22        1  
23        1  
24        2  
25        2
```

Para seleccionar varias variables, análogamente a antes, se puede hacer a partir del nombre o de la posición pero sólo con []:

```
> datos[, c("sexo", "edad")]
```

	sexo	edad
1	2	24
2	1	27
3	2	44
4	1	25
5	2	27
6	2	36
7	1	35
8	1	23
9	2	40
10	2	32
11	1	26
12	2	29
13	2	36
14	1	36
15	1	17
16	1	40
17	1	27
18	2	32
19	2	29
20	2	21
21	2	35
22	1	27
23	1	24
24	2	18
25	2	34
26	2	27
27	1	25
28	2	24

```
> datos[, c(7, 9)]
```

Para seleccionar todas las variables excepto una, hay que hacerlo a partir de la posición de la variable y el signo menos:

```
> datos[, -1]
```

	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo	tip_par
1	GADI	13211856000	13212288000	13215484800	2	24	3.38	2	1
2	CAEL	13211942400	13212460800	13233110400	2	27	2.50	1	2
3	COMO	13212028800	13213324800	13212633600	1	44	3.15	2	2
4	VIMU	13212201600	13212633600	13227926400	2	25	2.74	1	1
5	PAVI	13212288000	13212806400	13212892800	1	27	3.60	2	2
6	PASA	13212374400	13213324800	13212979200	1	36	2.65	2	1
7	VERI	13212374400	13213238400	13219632000	2	35	2.97	1	1
8	ADJU	13212460800	13212806400	13219718400	2	23	3.20	1	2
9	BEMI	13212547200	13213670400	13218595200	1	40	2.40	2	2
10	JUNA	13212633600	13213411200	13221100800	2	32	2.10	2	2
11	LOKO	13212892800	13213411200	13217731200	1	26	3.45	1	2
12	FRFU	13212979200	13213584000	13234752000	2	29	3.45	2	2
13	FUFE	13213756800	13214707200	13214361600	1	36	3.40	2	2
14	POCA	13214361600	13215312000	13224643200	2	36	3.05	1	2
15	LOLO	13214361600	13214448000	13214966400	1	17	3.60	1	2
16	BOPE	13214448000	13215571200	13230777600	1	40	3.40	1	2
17	ANZO	13214793600	13215312000	13226889600	2	27	3.15	1	2
18	MEVE	13214793600	13215571200	13236566400	2	32	3.32	2	2
19	TOPO	13214880000	13215484800	13222137600	1	29	2.65	2	2
20	PUPI	13214966400	13215225600	13222224000	2	21	4.46	2	2
21	ROPA	13214966400	13215830400	13217385600	1	35	3.15	2	2
22	LOMA	13215052800	13215571200	13234406400	2	27	3.70	1	2
23	CEMA	13215139200	13215571200	13216953600	1	24	3.79	1	2
24	CAGI	13215225600	13215398400	13215830400	2	18	3.75	2	2
25	GRSE	13215312000	13216176000	13216521600	1	34	2.95	2	2
26	GUMA	13215398400	13215916800	13227494400	2	27	2.90	2	1
27	PERI	13215398400	13215830400	13230518400	2	25	3.44	1	2
28	MAPE	13215398400	13215830400	13225075200	1	24	3.53	2	2

```
hermanos fuma_an fuma_de horas_an horas_de naci_ca masde12 sem_lac
```

Para quitar una variable cuya posición no se sabe, se puede usar la función `which` que devuelve la posición de esta variable:

```
> datos[, -which(names(datos) == "sexo")]
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	tip_par
1	1	GADI	13211856000	13212288000	13215484800	2	24	3.38	1
2	2	CAEL	13211942400	13212460800	13233110400	2	27	2.50	2
3	3	COMO	13212028800	13213324800	13212633600	1	44	3.15	2
4	4	VIMU	13212201600	13212633600	13227926400	2	25	2.74	1
5	5	PAVI	13212288000	13212806400	13212892800	1	27	3.60	2
6	6	PASA	13212374400	13213324800	13212979200	1	36	2.65	1
7	7	VERI	13212374400	13213238400	13219632000	2	35	2.97	1
8	8	ADJU	13212460800	13212806400	13219718400	2	23	3.20	2
9	9	BEMI	13212547200	13213670400	13218595200	1	40	2.40	2
10	10	JUNA	13212633600	13213411200	13221100800	2	32	2.10	2
11	11	LOKO	13212892800	13213411200	13217731200	1	26	3.45	2
12	12	FRFU	13212979200	13213584000	13234752000	2	29	3.45	2
13	13	FUFE	13213756800	13214707200	13214361600	1	36	3.40	2
14	14	POCA	13214361600	13215312000	13224643200	2	36	3.05	2
15	15	LOLO	13214361600	13214448000	13214966400	1	17	3.60	2
16	16	BOPE	13214448000	13215571200	13230777600	1	40	3.40	2
17	17	ANZO	13214793600	13215312000	13226889600	2	27	3.15	2
18	18	MEVE	13214793600	13215571200	13236566400	2	32	3.32	2
19	19	TOPO	13214880000	13215484800	13222137600	1	29	2.65	2
20	20	PUPI	13214966400	13215225600	13222224000	2	21	4.46	2
21	21	ROPA	13214966400	13215830400	13217385600	1	35	3.15	2
22	22	LOMA	13215052800	13215571200	13234406400	2	27	3.70	2
23	23	CEMA	13215139200	13215571200	13216953600	1	24	3.79	2
24	24	CAGI	13215225600	13215398400	13215830400	2	18	3.75	2
25	25	GRSE	13215312000	13216176000	13216521600	1	34	2.95	2
26	26	GUMA	13215398400	13215916800	13227494400	2	27	2.90	1
27	27	PERI	13215398400	13215830400	13230518400	2	25	3.44	2
28	28	MAPE	13215398400	13215830400	13225075200	1	24	3.53	2

```
hermanos fuma_an fuma_de horas_an horas_de naci_ca masde12 sem_lac
```

Para seleccionar todas la variables excepto algunas:

```
> datos[, -c(7, 9)]
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	peso	tip_par	hermanos
1	1	GADI	13211856000	13212288000	13215484800	2	3.38	1	2
2	2	CAEL	13211942400	13212460800	13233110400	2	2.50	2	1
3	3	COMO	13212028800	13213324800	13212633600	1	3.15	2	1
4	4	VIMU	13212201600	13212633600	13227926400	2	2.74	1	1
5	5	PAVI	13212288000	13212806400	13212892800	1	3.60	2	1
6	6	PASA	13212374400	13213324800	13212979200	1	2.65	1	2
7	7	VERI	13212374400	13213238400	13219632000	2	2.97	1	2
8	8	ADJU	13212460800	13212806400	13219718400	2	3.20	2	2
9	9	BEMI	13212547200	13213670400	13218595200	1	2.40	2	2
10	10	JUNA	13212633600	13213411200	13221100800	2	2.10	2	2
11	11	LOKO	13212892800	13213411200	13217731200	1	3.45	2	2
12	12	FRFU	13212979200	13213584000	13234752000	2	3.45	2	1
13	13	FUFE	13213756800	13214707200	13214361600	1	3.40	2	2
14	14	POCA	13214361600	13215312000	13224643200	2	3.05	2	2
15	15	LOLO	13214361600	13214448000	13214966400	1	3.60	2	2
16	16	BOPE	13214448000	13215571200	13230777600	1	3.40	2	2
17	17	ANZO	13214793600	13215312000	13226889600	2	3.15	2	1
18	18	MEVE	13214793600	13215571200	13236566400	2	3.32	2	2
19	19	TOPO	13214880000	13215484800	13222137600	1	2.65	2	2
20	20	PUPI	13214966400	13215225600	13222224000	2	4.46	2	2
21	21	ROPA	13214966400	13215830400	13217385600	1	3.15	2	2
22	22	LOMA	13215052800	13215571200	13234406400	2	3.70	2	1
23	23	CEMA	13215139200	13215571200	13216953600	1	3.79	2	1
24	24	CAGI	13215225600	13215398400	13215830400	2	3.75	2	2
25	25	GRSE	13215312000	13216176000	13216521600	1	2.95	2	1
26	26	GUMA	13215398400	13215916800	13227494400	2	2.90	1	1
27	27	PERI	13215398400	13215830400	13230518400	2	3.44	2	1
28	28	MAPE	13215398400	13215830400	13225075200	1	3.53	2	1

	fuma_an	fuma_de	horas_an	horas_de	naci_ca	masde12	sem_lac
1	1	0	6	2	3	0	6
2	0	0	2	2	1	1	35
3	0	1	2	0	1	0	1

Y, como antes, si no se sabe la posición de estas variables:

```
> datos[, -which(names(datos) %in% c("edad", "sexo"))]
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	peso	tip_par	hermanos
1	1	GADI	13211856000	13212288000	13215484800	2	3.38	1	2
2	2	CAEL	13211942400	13212460800	13233110400	2	2.50	2	1
3	3	COMO	13212028800	13213324800	13212633600	1	3.15	2	1
4	4	VIMU	13212201600	13212633600	13227926400	2	2.74	1	1
5	5	PAVI	13212288000	13212806400	13212892800	1	3.60	2	1
6	6	PASA	13212374400	13213324800	13212979200	1	2.65	1	2
7	7	VERI	13212374400	13213238400	13219632000	2	2.97	1	2
8	8	ADJU	13212460800	13212806400	13219718400	2	3.20	2	2
9	9	BEMI	13212547200	13213670400	13218595200	1	2.40	2	2
10	10	JUNA	13212633600	13213411200	13221100800	2	2.10	2	2
11	11	LOKO	13212892800	13213411200	13217731200	1	3.45	2	2
12	12	FRFU	13212979200	13213584000	13234752000	2	3.45	2	1
13	13	FUFE	13213756800	13214707200	13214361600	1	3.40	2	2
14	14	POCA	13214361600	13215312000	13224643200	2	3.05	2	2
15	15	LOLO	13214361600	13214448000	13214966400	1	3.60	2	2
16	16	BOPE	13214448000	13215571200	13230777600	1	3.40	2	2
17	17	ANZO	13214793600	13215312000	13226889600	2	3.15	2	1
18	18	MEVE	13214793600	13215571200	13236566400	2	3.32	2	2
19	19	TOPO	13214880000	13215484800	13222137600	1	2.65	2	2
20	20	PUPI	13214966400	13215225600	13222224000	2	4.46	2	2
21	21	ROPA	13214966400	13215830400	13217385600	1	3.15	2	2
22	22	LOMA	13215052800	13215571200	13234406400	2	3.70	2	1
23	23	CEMA	13215139200	13215571200	13216953600	1	3.79	2	1
24	24	CAGI	13215225600	13215398400	13215830400	2	3.75	2	2
25	25	GRSE	13215312000	13216176000	13216521600	1	2.95	2	1
26	26	GUMA	13215398400	13215916800	13227494400	2	2.90	1	1
27	27	PERI	13215398400	13215830400	13230518400	2	3.44	2	1
28	28	MAPE	13215398400	13215830400	13225075200	1	3.53	2	1

	fuma_an	fuma_de	horas_an	horas_de	naci_ca	masde12	sem_lac
1	1	0	6	2	3	0	6
2	0	0	2	2	1	1	35
3	0	1	2	0	1	0	1

Para seleccionar individuos, se usa la función `subset`, dónde el primer argumento es la base de datos, y el segundo una condición lógica que deben cumplir los individuos que se desea seleccionar. El resultado es un `data.frame` con sólo los individuos seleccionados.

Por ejemplo, si se quiere seleccionar los niños de 'parto2'

```
> subset(datos, sexo == 1)
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo	tip_par
2	2 CAEL	13211942400	13212460800	13233110400	2	27	2.50	1	2	
4	4 VIMU	13212201600	13212633600	13227926400	2	25	2.74	1	1	
7	7 VERI	13212374400	13213238400	13219632000	2	35	2.97	1	1	
8	8 ADJU	13212460800	13212806400	13219718400	2	23	3.20	1	2	
11	11 LOKO	13212892800	13213411200	13217731200	1	26	3.45	1	2	
14	14 POCA	13214361600	13215312000	13224643200	2	36	3.05	1	2	
15	15 LOLO	13214361600	13214448000	13214966400	1	17	3.60	1	2	
16	16 BOPE	13214448000	13215571200	13230777600	1	40	3.40	1	2	
17	17 ANZO	13214793600	13215312000	13226889600	2	27	3.15	1	2	
22	22 LOMA	13215052800	13215571200	13234406400	2	27	3.70	1	2	
23	23 CEMA	13215139200	13215571200	13216953600	1	24	3.79	1	2	
27	27 PERI	13215398400	13215830400	13230518400	2	25	3.44	1	2	

	hermanos	fuma_an	fuma_de	horas_an	horas_de	naci_ca	masde12	sem_lac
2	1	0	0	2	2	1	1	35
4	1	1	1	11	6	2	1	26
7	2	0	0	8	8	1	0	12
8	2	0	0	5	2	2	0	12
11	2	1	1	7	14	3	0	8
14	2	1	1	7	3	1	1	17
15	2	0	0	10	0	3	0	1
16	2	0	0	5	9	1	1	27
17	1	0	0	7	2	1	1	20
22	1	0	0	7	7	1	1	32
23	1	1	1	4	10	3	0	3
27	1	1	1	7	3	2	1	25

O si se quiere seleccionar las niñas cuya madre tenga más de 20 años.

```
> subset(datos, sexo == 2 & edad > 20)
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo	tip_par
1	1	GADI	13211856000	13212288000	13215484800	2	24	3.38	2	1
3	3	COMO	13212028800	13213324800	13212633600	1	44	3.15	2	2
5	5	PAVI	13212288000	13212806400	13212892800	1	27	3.60	2	2
6	6	PASA	13212374400	13213324800	13212979200	1	36	2.65	2	1
9	9	BEMI	13212547200	13213670400	13218595200	1	40	2.40	2	2
10	10	JUNA	13212633600	13213411200	13221100800	2	32	2.10	2	2
12	12	FRFU	13212979200	13213584000	13234752000	2	29	3.45	2	2
13	13	FUFE	13213756800	13214707200	13214361600	1	36	3.40	2	2
18	18	MEVE	13214793600	13215571200	13236566400	2	32	3.32	2	2
19	19	TOPO	13214880000	13215484800	13222137600	1	29	2.65	2	2
20	20	PUPI	13214966400	13215225600	13222224000	2	21	4.46	2	2
21	21	ROPA	13214966400	13215830400	13217385600	1	35	3.15	2	2
25	25	GRSE	13215312000	13216176000	13216521600	1	34	2.95	2	2
26	26	GUMA	13215398400	13215916800	13227494400	2	27	2.90	2	1
28	28	MAPE	13215398400	13215830400	13225075200	1	24	3.53	2	2

	hermanos	fuma_an	fuma_de	horas_an	horas_de	naci_ca	masde12	sem_lac
1	2	1	0	6	2	3	0	6
3	1	0	1	3	0	1	0	1
5	1	1	0	10	22	1	0	1
6	2	0	0	9	9	1	0	1
9	2	1	1	12	10	1	0	10
10	2	1	1	7	0	3	1	14
12	1	1	0	12	11	3	1	36
13	2	0	0	7	4	1	0	1
18	2	1	0	11	8	2	1	36
19	2	0	1	3	1	1	0	12
20	2	1	1	7	0	3	0	12
21	2	1	0	5	4	1	0	4
25	1	0	0	12	23	1	0	2
26	1	0	0	4	14	0	1	20

Para seleccionar aleatoriamente un subconjunto de individuos (filas) de un `data.frame` se usa la función `sample`. De forma equivalente se pueden seleccionar filas ó columnas (de un `data.frame` o de una `matrix`), o elementos de un vector.

Ejemplo: selección de 20 individuos al azar

```
> datos[sample(1:nrow(datos), 20), ]
```

	id	ini	día_nac	día_entr	ulti_lac	tx	edad	peso	sexo	tip_par
21	1	ROPA	13214966400	13215830400	13217385600	1	35	3.15	2	2
6	6	PASA	13212374400	13213324800	13212979200	1	36	2.65	2	1
17	17	ANZO	13214793600	13215312000	13226889600	2	27	3.15	1	2
11	11	LOKO	13212892800	13213411200	13217731200	1	26	3.45	1	2
26	26	GUMA	13215398400	13215916800	13227494400	2	27	2.90	2	1
22	22	LOMA	13215052800	13215571200	13234406400	2	27	3.70	1	2
5	5	PAVI	13212288000	13212806400	13212892800	1	27	3.60	2	2
3	3	COMO	13212028800	13213324800	13212633600	1	44	3.15	2	2
12	12	FRFU	13212979200	13213584000	13234752000	2	29	3.45	2	2
8	8	ADJU	13212460800	13212806400	13219718400	2	23	3.20	1	2
9	9	BEMI	13212547200	13213670400	13218595200	1	40	2.40	2	2
25	25	GRSE	13215312000	13216176000	13216521600	1	34	2.95	2	2
18	18	MEVE	13214793600	13215571200	13236566400	2	32	3.32	2	2
2	2	CAEL	13211942400	13212460800	13233110400	2	27	2.50	1	2
19	19	TOPO	13214880000	13215484800	13222137600	1	29	2.65	2	2
7	7	VERI	13212374400	13213238400	13219632000	2	35	2.97	1	1
27	27	PERI	13215398400	13215830400	13230518400	2	25	3.44	1	2
10	10	JUNA	13212633600	13213411200	13221100800	2	32	2.10	2	2
1	1	GADI	13211856000	13212288000	13215484800	2	24	3.38	2	1
23	23	CEMA	13215139200	13215571200	13216953600	1	24	3.79	1	2

	hermanos	fuma_an	fuma_de	horas_an	horas_de	naci_ca	masde12	sem_lac
21	2	1	0	5	4	1	0	4
6	2	0	0	9	9	1	0	1
17	1	0	0	7	2	1	1	20

O también es útil la función `sample` para seleccionar los individuos con reemplazamiento. Esto es el caso de la técnica *bootstrap*. Ahora aparecerán el mismo número de individuos pero algunos de ellos repetidos al azar.

Nótese la opción `replace=TRUE` en la función `sample`

```
> datos[sample(1:nrow(datos), nrow(datos), replace = TRUE), ]
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo	tip_par
1	1 GADI	13211856000	13212288000	13215484800	2	24	3.38	2	1	
17	17 ANZO	13214793600	13215312000	13226889600	2	27	3.15	1	2	
15	15 LOLO	13214361600	13214448000	13214966400	1	17	3.60	1	2	
25	25 GRSE	13215312000	13216176000	13216521600	1	34	2.95	2	2	
18	18 MEVE	13214793600	13215571200	13236566400	2	32	3.32	2	2	
12	12 FRFU	13212979200	13213584000	13234752000	2	29	3.45	2	2	
5	5 PAVI	13212288000	13212806400	13212892800	1	27	3.60	2	2	
22	22 LOMA	13215052800	13215571200	13234406400	2	27	3.70	1	2	
21	21 ROPA	13214966400	13215830400	13217385600	1	35	3.15	2	2	
24	24 CAGI	13215225600	13215398400	13215830400	2	18	3.75	2	2	
27	27 PERI	13215398400	13215830400	13230518400	2	25	3.44	1	2	
27.1	27 PERI	13215398400	13215830400	13230518400	2	25	3.44	1	2	
21.1	21 ROPA	13214966400	13215830400	13217385600	1	35	3.15	2	2	
20	20 PUPI	13214966400	13215225600	13222224000	2	21	4.46	2	2	
8	8 ADJU	13212460800	13212806400	13219718400	2	23	3.20	1	2	
10	10 JUNA	13212633600	13213411200	13221100800	2	32	2.10	2	2	
1.1	1 GADI	13211856000	13212288000	13215484800	2	24	3.38	2	1	
9	9 BEMI	13212547200	13213670400	13218595200	1	40	2.40	2	2	
3	3 COMO	13212028800	13213324800	13212633600	1	44	3.15	2	2	
20.1	20 PUPI	13214966400	13215225600	13222224000	2	21	4.46	2	2	
26	26 GUMA	13215398400	13215916800	13227494400	2	27	2.90	2	1	
13	13 FUFU	13213756800	13214707200	13214361600	1	36	3.40	2	2	
17.1	17 ANZO	13214793600	13215312000	13226889600	2	27	3.15	1	2	

Se cargan los paquetes necesarios, y se fija la carpeta de trabajo.

```
> library(chron)
> library(foreign)
> setwd("C:/cursoR/data")
```

- Con la función `c`, (*concatenate*), se forman vectores cuyos elementos son de la misma clase.
- Los componentes de los vectores son de tipo 'atomic', eso es, enteros, caracteres, etc. pero no pueden ser complejos, o sea, ni del tipo matriz, ni `data.frame` ni otros vectores.
- A continuación se muestran unos ejemplos.

```
> # concatenación de enteros
```

```
> c(4,5,2,1,0,10)
```

```
[1] 4 5 2 1 0 10
```

```
> c("Joan", "Isaac", "Marta", "Pere")
```

```
[1] "Joan" "Isaac" "Marta" "Pere"
```

Existe la función `seq`, para concatenar una secuencia de números. Su uso es: primer número, último número y longitud del salto. Ejemplo: secuencia entre 10 y 20 de dos en dos:

```
> seq(10, 20, 2)
```

```
[1] 10 12 14 16 18 20
```

Por defecto la longitud del salto es 1. Así, para crear una secuencia de 10 a 20 de 1 en 1:

```
> seq(10, 20)
```

```
[1] 10 11 12 13 14 15 16 17 18 19 20
```

O también se puede usar la instrucción `a:b`, lo cual es equivalente a `seq(a,b)`:

```
> 10:20
```

```
[1] 10 11 12 13 14 15 16 17 18 19 20
```


Si se mezclan enteros y caracteres, como el resultado tiene que ser un conjunto de elementos de la misma clase, todo se convierte a la clase más general, que en este caso será la clase carácter:

```
> c("a", 3, "b", 4)
```

```
[1] "a" "3" "b" "4"
```

Finalmente, se pueden nombrar los elementos de un vector, guardando primero el vector y después asignándole los nombres con la función `names`

```
> v <- c(10, 12, -15)
> names(v) <- c("e1", "e2", "e3")
> v
```

```
e1 e2 e3
10 12 -15
```

o directamente, en una sola instrucción:

```
> v <- c(e1 = 10, e2 = 12, e3 = -15)
```

Para ordenar un vector se usa la función `sort`

```
> sort(v)
```

```
  e3  e1  e2  
-15  10  12
```

Y para hacerlo de forma descendente, hay que especificar el argumento `decreasing=TRUE`:

```
> sort(v, decreasing = TRUE)
```

```
  e2  e1  e3  
 12  10 -15
```

No hay que confundir el `sort` con lo que hace la función `order`. Esta última asigna los rangos a los elementos. El rango uno es el elemento cuyo valor es el menor, y así sucesivamente.

```
> order(v)
```

```
[1] 3 1 2
```

Para extraer los nombres de los elementos

```
> names(v)
```

```
[1] "e1" "e2" "e3"
```

Para poner los elementos del vector en orden inverso al que están

```
> rev(v)
```

```
  e3  e2  e1  
-15  12  10
```

Para seleccionar los elementos de un vector se puede hacer:

- con la posición que ocupan

```
> v[1]
```

```
e1  
10
```

```
> v[c(1, 2)]
```

```
e1 e2  
10 12
```

- con el nombre del elemento

```
> v["e1"]
```

```
e1  
10
```

```
> v[c("e1", "e2")]
```

```
e1 e2  
10 12
```

- ó usando la función `which`. Por ejemplo, para ver los elementos mayores que 1:

```
> v[which(v > 1)]
```

```
e1 e2  
10 12
```

Para saber cuántos elementos tiene un vector

```
> length(v)
```

```
[1] 3
```

Las operaciones con vectores, se hacen elemento a elemento

```
> v1 <- c(1, 2, 3)
> v2 <- c(2, 2, -1)
> v1 + v2
```

```
[1] 3 4 2
```

```
> v1 * v2
```




```
[1] 2 4 -3
```

```
> v1/v2
```

```
[1] 0.5 1.0 -3.0
```

```
> v1 > v2
```

```
[1] FALSE FALSE TRUE
```


- En  existen diferentes tipos básicos de elementos "atómicos" enteros (`integer`), reales (`double`), caracteres (`character`), factores (`factor`),...
- Los factores son un tipo de objetos especiales en  y hay que ir con cuidado al trabajar con ellos. Internamente están codificados como enteros, aunque sus valores sean las etiquetas (niveles del factor). Para ver cuáles son sus niveles se usa la función `levels`.
- Por defecto  lee las variables carácter y las transforma en factor. Esto ocurre al leer datos desde un fichero externo con la función `read.table`, o cuando se crea un `data.frame` con una variable carácter (si es que no se usa la función `I()` como se ha visto en secciones anteriores).

- Para crear un factor, por ejemplo convertir la variable tratamiento (que está codificado con números) a factor con los niveles especificados en el atributo "value labels", se usa la función factor:

```
> datos <- read.spss("partoFin.sav", FALSE, TRUE)
> vl <- attr(datos$tx, "value.labels")
> datos$txfac <- factor(datos$tx, labels = names(sort(vl)))
> summary(datos$txfac)
```

```
Estándar Intensivo
    13         15
```

- También se puede usar la función as.factor para "coercionar" una variable a factor. Pero tiene la desventaja que no se pueden especificar sus niveles, y estos son los distintos valores que toma la variable:

```
> levels(as.factor(datos$tx))

[1] "1" "2"
```

- Tratar las variables cualitativas/carácter como factores puede ser muy útil: por ejemplo, al hacer una descriptiva con la función `summary` ya no se calculan los cuantiles o la media, sino que se muestran las frecuencias. Ahora, esta variable ya no será tratada como numérica sino como una variable cualitativa (al hacer regresión, o al hacer un gráfico, por ejemplo). Pero algunas manipulaciones pueden ser un poco laboriosas de escribir, como por ejemplo:

```
> subset(datos, txfac == "Estándar")
```

Pero lo siguiente ya no funciona...

```
> subset(datos, txfac == 1)
```

- En este curso se ha evitado trabajar con factores, aunque es conveniente tenerlos en cuenta.

- Para combinar filas y columnas se usa la función `rbind` (*row bind*) y `cbind` (*column bind*), respectivamente.
- Para combinar por columnas los vectores 'v1' y 'v2':

```
> cbind(v1, v2)
```

```
      v1 v2  
[1,]  1  2  
[2,]  2  2  
[3,]  3 -1
```

Y para combinarlos en filas

```
> rbind(v1, v2)
```

```
  [,1] [,2] [,3]  
v1    1    2    3  
v2    2    2   -1
```

El resultado de combinar vectores en filas y/o columnas es una matriz:

```
> m <- rbind(v1, v2)
```

```
> m
```

```
      [,1] [,2] [,3]  
v1      1   2   3  
v2      2   2  -1
```

```
> class(m)
```

```
[1] "matrix"
```

Y para combinarlos en columnas

```
> m <- cbind(v1, v2)
```

```
> m
```

```
      v1 v2  
[1,]  1  2  
[2,]  2  2  
[3,]  3 -1
```

```
> class(m)
```

```
[1] "matrix"
```

- Una matriz es un conjunto de elementos 'atómicos' dispuestos en filas y columnas.
- Todos los elementos de una matriz han de ser del mismo tipo, como pasa con los vectores.
- Como ejemplo, se crean dos matrices (con la función `matrix`) y un vector.

```
> A <- matrix(c(1, 2, 3, 4, 5, 6, 7, 10, -3), nrow = 3, ncol = 3,  
+           byrow = FALSE)  
> B <- matrix(c(0, 1, 3, 3, 5, 8, -1, -3, 0), nrow = 3, ncol = 3,  
+           byrow = FALSE)  
> b <- c(1, 3, 5)
```

Véase qué pasa si se crea una matriz con números y caracteres.
Todo se convertirá al tipo más general, en este caso, caracteres.

```
> C <- matrix(c(1, "a", 3, 4, "b", 6, 7, "c", -3), nrow = 3, ncol = 3,  
+           byrow = FALSE)  
> C
```

```
      [,1] [,2] [,3]  
[1,] "1"  "4"  "7"  
[2,] "a"  "b"  "c"  
[3,] "3"  "6"  "-3"
```

Para saber el número de columnas:

```
> ncol(A)
```

```
[1] 3
```

Para saber el número de filas

```
> nrow(A)
```

```
[1] 3
```

El número de filas y de columnas:

```
> dim(A)
```

```
[1] 3 3
```


Para seleccionar filas y columnas.

```
> A[1, 2]
```

```
[1] 4
```

```
> A[, 3]
```

```
[1] 7 10 -3
```

```
> A[-2, ]
```

	[,1]	[,2]	[,3]
[1,]	1	4	7
[2,]	3	6	-3

- Si se selecciona sólo una fila ó sólo una columna, el resultado es un vector.
- Para que se quede como una matriz fila ó columna hay que especificar el argumento `drop=FALSE`. Exactamente lo mismo ocurre con los `data.frames`.

```
> A[1, , drop = FALSE]
```

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

```
> A[, 3, drop = FALSE]
```

```
      [,1]  
[1,]     7  
[2,]    10  
[3,]    -3
```

Para seleccionar filas o columnas en un orden determinado

```
> A[c(3, 2, 1), ]
```

	[,1]	[,2]	[,3]
[1,]	3	6	-3
[2,]	2	5	10
[3,]	1	4	7

```
> A[, c(1, 3, 2)]
```

	[,1]	[,2]	[,3]
[1,]	1	7	4
[2,]	2	10	5
[3,]	3	-3	6

Para sumar los elementos de una matriz:

```
> # Sumar los elementos de la primera fila:  
> sum(A[1,])
```

```
[1] 12
```

```
> # Sumar todos los elementos excepto la segunda columna  
> sum(A[,-2])
```

```
[1] 20
```

```
> # Sumar todos los elementos  
> sum(A)
```

```
[1] 35
```

- Por defecto, las filas y las columnas de las matrices no tienen nombre.
- Así, que si nos preguntamos qué nombre tienen las columnas de la matriz 'A' el resultado será NULL.
- Es importante remarcar que los nombres de las columnas ó de las filas pueden estar repetidos.
- En cambio, un `data.frame` no puede tener nombres de filas repetidos, ni tampoco de las variables.

```
> # para las filas  
> rownames(A)
```

NULL

```
> # para las columnas  
> colnames(A)
```

NULL

Para poner nombre a sus filas:

```
> rownames(A) <- c("fila1", "fila2", "fila3")
```

```
> rownames(A)
```

```
[1] "fila1" "fila2" "fila3"
```

y a sus columnas

```
> colnames(A) <- c("columna1", "columna2", "columna3")
```

```
> colnames(A)
```

```
[1] "columna1" "columna2" "columna3"
```

```
> A
```

	columna1	columna2	columna3
fila1	1	4	7
fila2	2	5	10
fila3	3	6	-3

- Para seleccionar filas ó columnas aleatoriamente: función `sample`.
- Por ejemplo, para seleccionar dos filas de las 3 al azar de la matriz 'A':

```
> selec <- sample(1:nrow(A), 2)
> selec <- sort(selec)
> A[selec, ]
```

	columna1	columna2	columna3
fila1	1	4	7
fila3	3	6	-3

Multiplicación de dos matrices (producto matricial)

```
> A %*% B
```

```
      [,1] [,2] [,3]  
fila1    25    79   -13  
fila2    35   111   -17  
fila3    -3    15   -21
```

Multiplicación de dos matrices (elemento a elemento)

```
> A * B
```

```
      columna1 columna2 columna3  
fila1         0        12       -7  
fila2         2        25      -30  
fila3         9        48        0
```


Transponer una matriz

```
> t(A)
```

```
      fila1 fila2 fila3  
columna1    1    2    3  
columna2    4    5    6  
columna3    7   10   -3
```

Determinante de una matriz

```
> det(A)
```

```
[1] 48
```

Inversa de una matriz

```
> solve(A)
```

```
      fila1  fila2      fila3  
columna1 -1.5625  1.125  0.10416667  
columna2  0.7500 -0.500  0.08333333  
columna3 -0.0625  0.125 -0.06250000
```

Valores y vectores propios de una matriz

```
> eigen(A)
```

```
$values
```

```
[1] 11.9058461 -8.4274533 -0.4783928
```

```
$vectors
```

```
      [,1]      [,2]      [,3]  
[1,] -0.5329223 -0.3527924 -0.90203789  
[2,] -0.7423581 -0.5245250  0.42824065  
[3,] -0.4060766  0.7748620 -0.05419947
```

Multiplicación de una matriz por un vector

```
> A %% b
```

```
      [,1]  
fila1  48  
fila2  67  
fila3   6
```

Resolución de un sistema de ecuaciones lineales (compatible determinado)

$$Ax = b$$

```
> solve(A, b)
```

```
[1]  2.333333e+00 -3.333333e-01  2.313078e-17
```

Para combinar diferentes matrices en filas y en columnas (si es posible) con las funciones `cbind` y `rbind`

```
> cbind(A, B)
```

	columna1	columna2	columna3
fila1	1	4	7 0 3 -1
fila2	2	5	10 1 5 -3
fila3	3	6	-3 3 8 0

```
> rbind(A, B)
```

	columna1	columna2	columna3
fila1	1	4	7
fila2	2	5	10
fila3	3	6	-3
	0	3	-1
	1	5	-3
	3	8	0


ó una matriz y un vector

```
> cbind(A, b)
```

	columna1	columna2	columna3	b
fila1	1	4	7	1
fila2	2	5	10	3
fila3	3	6	-3	5

```
> rbind(A, b)
```

	columna1	columna2	columna3
fila1	1	4	7
fila2	2	5	10
fila3	3	6	-3
b	1	3	5

- Un *array* es como una matriz, sólo que puede tener más de 2 dimensiones.
- En realidad una matriz es un caso particular de un *array* con 2 dimensiones, y un vector también es un caso particular de una *array* con una sola dimensión.
- Véase un ejemplo de cómo se construye un array en .

```
> arr <- array(c(1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3,  
+ 3, 3), dim = c(3, 2, 3), dimnames = list(c("fila1", "fila2",  
+ "fila3"), c("col1", "col2"), c("capa1", "capa2", "capa3")))
```

```
> arr
```

```
, , capa1
```

	col1	col2
fila1	1	1
fila2	1	1
fila3	1	1

```
, , capa2
```

	col1	col2
fila1	2	2
fila2	2	2
fila3	2	2

```
, , capa3
```

	col1	col2
fila1	3	3

Para seleccionar las diferentes dimensiones, es similar a las matrices. Por ejemplo la primera fila:

```
> arr[1, , ]
```

```
      capa1 capa2 capa3
col1      1      2      3
col2      1      2      3
```

ó la segunda capa

```
> arr[, , 2]
```

```
      col1 col2
fila1      2      2
fila2      2      2
fila3      2      2
```

ó la tercera fila de la segunda capa

```
> arr[3, , 2]
```

```
col1 col2
  2      2
```

También se puede construir un array a partir de dos matrices de las mismas dimensiones (una por capa).

Primero hay que inicializarlo

```
> arr <- array(NA, dim = c(3, 3, 2))
```

En la primera capa se pone la matriz 'A'

```
> arr[, , 1] <- A
```

y en la segunda la matriz 'B'

```
> arr[, , 2] <- B
```

```
> arr
```

```
, , 1
```

	[,1]	[,2]	[,3]
[1,]	1	4	7
[2,]	2	5	10
[3,]	3	6	-3

```
, , 2
```

	[,1]	[,2]	[,3]
[1,]	0	3	-1
[2,]	1	5	-3
[3,]	3	8	0

- Un `data.frame` es similar a una matriz, en el sentido que los datos están dispuestos en filas (registros) y columnas (variables). Pero a diferencia de una matriz, cada columna puede ser de una clase distinta.
- Por ejemplo, la primera variable puede ser el nombre (cadena), la segunda la edad (entero) y la tercera la fecha de nacimiento (fecha).
- Para almacenar esta información no es posible hacerlo en una matriz.
- Nótese el uso de la función `I()`. Esto hace que las variables cadena, al formarse el `data.frame`, no se transformen en factores.

```
> id <- 1:5
> nombre <- c("Joan", "Isaac", "Marta", "Cristina", "Julia")
> edad <- c(51, 29, 31, 28, 56)
> naci <- chron(c("17-10-1979", "20-5-1958", "12-2-1983", "1-12-1978",
+               "28-3-1965"), format = c("d-m-Y"), out.format = c("d-mon-Y"))
> datos <- data.frame(id = id, nombre = I(nombre), edad = edad,
+                     naci = naci)
> datos
```

	id	nombre	edad	naci
1	1	Joan	51	17-Oct-1979
2	2	Isaac	29	20-May-1958
3	3	Marta	31	12-Feb-1983
4	4	Cristina	28	01-Dec-1978
5	5	Julia	56	28-Mar-1965

- Se pueden seleccionar columnas de un `data.frame`.
- Por ejemplo para crear dos `data.frames`, `'datos1'` y `'datos2'`, el primero que contenga las dos primeras variables de `'datos'` y el segundo las otras dos:

```
> datos1 <- datos[, 1:2]
```

```
> datos1
```

```
  id  nombre
1  1    Joan
2  2   Isaac
3  3   Marta
4  4 Cristina
5  5   Julia
```

```
> datos2 <- datos[, 3:ncol(datos)]
```

```
> datos2
```

```
  edad      naci
1   51 17-Oct-1979
2   29 20-May-1958
3   31 12-Feb-1983
4   28 01-Dec-1978
5   56 28-Mar-1965
```

Y se pueden seleccionar filas. Por ejemplo, para poner en 'datos3' las dos primeras filas de 'datos' y en 'datos4' las restantes:

```
> datos3 <- datos[1:2, ]  
> datos3
```

	id	nombre	edad	naci
1	1	Joan	51	17-Oct-1979
2	2	Isaac	29	20-May-1958

```
> datos4 <- datos[3:nrow(datos), ]  
> datos4
```

	id	nombre	edad	naci
3	3	Marta	31	12-Feb-1983
4	4	Cristina	28	01-Dec-1978
5	5	Julia	56	28-Mar-1965

- Se pueden combinar por columnas dos `data.frames` (siempre y cuando tengan, por supuesto, el mismo número de filas), con la función `cbind`, tal y como se hace para las matrices, y el resultado es un `data.frame`.
- Por ejemplo, combinar `'datos1'` y `'datos2'` para volver a obtener el `data.frame` `'datos'`.

```
> cbind(datos1, datos2)
```

	id	nombre	edad	naci
1	1	Joan	51	17-Oct-1979
2	2	Isaac	29	20-May-1958
3	3	Marta	31	12-Feb-1983
4	4	Cristina	28	01-Dec-1978
5	5	Julia	56	28-Mar-1965

- También se puede combinar por filas usando la función `rbind`, al igual que con las matrices, .
- Al combinar por filas dos `data.frames` hay que ir con cuidado que las columnas sean de la misma clase y estén en el mismo orden, a parte, por supuesto, que tengan el mismo número de columnas.
- El resultado de combinar por filas dos `data.frames` es también otro `data.frame`.

```
> rbind(datos3, datos4)
```

	id	nombre	edad	naci
1	1	Joan	51	17-Oct-1979
2	2	Isaac	29	20-May-1958
3	3	Marta	31	12-Feb-1983
4	4	Cristina	28	01-Dec-1978
5	5	Julia	56	28-Mar-1965

Para inicializar una nueva variable vacía:

```
> datos$nueva <- NA  
> datos
```

	id	nombre	edad	naci	nueva
1	1	Joan	51	17-Oct-1979	NA
2	2	Isaac	29	20-May-1958	NA
3	3	Marta	31	12-Feb-1983	NA
4	4	Cristina	28	01-Dec-1978	NA
5	5	Julia	56	28-Mar-1965	NA

Para inicializar una nueva variable a una constante:

```
> datos$nueva2 <- 1  
> datos
```

	id	nombre	edad	naci	nueva	nueva2
1	1	Joan	51	17-Oct-1979	NA	1
2	2	Isaac	29	20-May-1958	NA	1
3	3	Marta	31	12-Feb-1983	NA	1
4	4	Cristina	28	01-Dec-1978	NA	1
5	5	Julia	56	28-Mar-1965	NA	1

También se puede usar `[, "nueva3"]` para inicializar una nueva variable, ó `datos$"nueva4"`

- Se puede convertir un `data.frame` en una matriz.
- Nótese que ahora todas las columnas tienen que ser del mismo tipo: será de tipo cadena.

```
> as.matrix(datos)
```

	id	nombre	edad	naci	nueva	nueva2
[1,]	"1"	"Joan"	"51"	"17-Oct-1979"	NA	"1"
[2,]	"2"	"Isaac"	"29"	"20-May-1958"	NA	"1"
[3,]	"3"	"Marta"	"31"	"12-Feb-1983"	NA	"1"
[4,]	"4"	"Cristina"	"28"	"01-Dec-1978"	NA	"1"
[5,]	"5"	"Julia"	"56"	"28-Mar-1965"	NA	"1"

- Los `data.frames` tienen algunas propiedades que también tienen las matrices.
- Por ejemplo, se puede pedir el número de filas, columnas, la dimensión ó el nombre de las filas, de la misma manera que se haría con una matriz:

```
> nrow(datos)
```

```
[1] 5
```

```
> ncol(datos)
```

```
[1] 6
```

```
> dim(datos)
```

```
[1] 5 6
```

```
> rownames(datos)
```

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

Incluso también, como en las matrices, se puede usar la función `apply`, como se verá más adelante. Aunque hay que tener en cuenta que `apply` transforma internamente el `data.frame` en matriz. Así, si se desea saber de qué clase es cada variable no será apropiado usar la función `apply` ya que resultará que son de la misma clase, porque que el `data.frame` se ha convertido internamente en matriz. Para ello, habrá que usar la función `lapply`, que se verá más adelante.

- Para contar el número de missings: OK !!

```
> # se pone algún missing de forma "manual"
> datosna<-datos
> datosna[1,1]<-NA
> datosna[3,2]<-NA
> # cuenta el número de missings por columnas (variables).
> apply(is.na(datosna),2,sum)

  id nombre  edad  naci  nueva nueva2
1     1     1     0     0     5       0
```

- Para saber de qué clase es cada variable: No OK !!

```
> apply(datos, 2, class)

  id  nombre  edad  naci  nueva  nueva2
"character" "character" "character" "character" "character" "character"
```

- Un `data.frame` comparte algunas propiedades con los objetos de clase lista (que se presentarán a continuación).
- Por ejemplo, al igual que en las listas, para obtener el nombre de cada componente (para un `data.frame`, variable):

```
> names(datos)
```

```
[1] "id"      "nombre" "edad"   "naci"    "nueva"  "nueva2"
```

- Ó también se puede aplicar la función `lapply`, que en el caso de una lista sirve para hacer una misma operación para cada componente de ella, cosa que en un `data.frame` se traduce a cada variable.
- `lapply` convierte el input del primer argumento a una lista (recuérdase que `apply` convierte el primer argumento a una matriz).
- Por ejemplo, para saber la clase de cada variable:

```
> lapply(datos, class)
```

```
$id  
[1] "integer"
```

```
$nombre  
[1] "AsIs"
```

```
$edad  
[1] "numeric"
```

```
$naci  
[1] "dates" "times"
```

```
$nueva  
[1] "logical"
```

```
$nueva2  
[1] "numeric"
```

- Una lista es una colección de elementos.
- Cada elemento de una lista puede ser un número, un vector, una matriz ó incluso otra lista.
- Véase un ejemplo.

```
> # se inicializa la lista
> lista<-list()
> # para crear cada elemento de la lista
> lista$nombre<-c("Isaac","Subirana","Cachinero")
> lista$amigos<-data.frame(nom=c("Joan","Jordi","Mireia"), edad=c(34,30,45))
> lista$telefono<-93345656
> lista$naci<-chron("17-10-1979",format="d-m-Y", out.format="d-mon-Y")
> lista$notas<-list()
> lista$notas$mates<-c(4,6.7,8.5)
> lista$notas$historia<-c(5.6,5,8,10,9,9.5)
> lista$mifuncion<-function(x) log(x)+1.5
```

Una lista también se puede crear de la siguiente manera:

```
> lista<-list(nombre=c("Isaac","Subirana","Cachinero"),
+             amigos=data.frame(nom=c("Joan","Jordi","Mireia"),
+                                 edad=c(34,30,45)), telefono=93345656,
+             naci=chron("17-10-1979",format="d-m-Y",out.format="d-mon-Y"),
+             notas=list(mates=c(4,6.7,8.5),historia=c(5.6,5,8,10,9,9.5)),
+             mifuncion=function(x) log(x)+1.5)
> lista
```

```
$nombre
```

```
[1] "Isaac"      "Subirana"   "Cachinero"
```

```
$amigos
```

```
      nom edad
1   Joan   34
2  Jordi   30
3 Mireia   45
```

```
$telefono
```

```
[1] 93345656
```

```
$naci
```

```
[1] 17-Oct-1979
```

```
$notas
```

```
$notas$mates
```

```
[1] 4.0 6.7 8.5
```

```
$notas$historia
```

```
[1] 5.6 5.0 8.0 10.0 9.0 9.5
```

```
$mifuncion
```

- Cada elemento de una lista puede ser del tipo que sea
- Incluso un elemento de una lista puede ser otra lista (como son las notas de las distintas asignaturas).
- Como las distintas asignaturas pueden tener distinto número de parciales, no se pueden poner en una matriz ni un `data.frame`, sino que hay que ponerlo en una lista.

```
> lista
```

```
$nombre
```

```
[1] "Isaac"      "Subirana"   "Cachinero"
```

```
$amigos
```

```
      nom edad  
1  Joan   34  
2  Jordi  30  
3 Mireia  45
```

```
$telefono
```

```
[1] 93345656
```

```
$naci
```

```
[1] 17-Oct-1979
```

```
$notas
```

```
$notas$mates
```

```
[1] 4.0 6.7 8.5
```

```
$notas$historia
```

```
[1] 5.6 5.0 8.0 10.0 9.0 9.5
```


Para ver los nombres de los elementos de una lista, al igual que en un `data.frame`, se usa la función `names`.

```
> names(lista)
```

```
[1] "nombre"    "amigos"    "telefono"  "naci"      "notas"     "mifuncion"
```

Para saber cuántos elementos tiene una lista

```
> length(lista)
```

```
[1] 6
```

Para referenciar un elemento de la lista

```
> lista[[4]]
```

```
[1] 17-Oct-1979
```

```
> lista$naci
```

```
[1] 17-Oct-1979
```

```
> lista$"naci"
```

```
[1] 17-Oct-1979
```

Y si no hay ambigüedades, se puede teclear de forma parcial el nombre del elemento después el signo \$ y sin comillas. Esto es igual para los `data.frames`.

```
> lista$nac
```

```
[1] 17-Oct-1979
```

```
> datos$nom
```

```
[1] "Joan"      "Isaac"     "Marta"     "Cristina"  "Julia"
```

Para referenciar varios elementos de una lista hay que usar `[]`, **y no** `[[]]`, similar a como se hace con los vectores:

- Por la posición

```
> lista[3:4]
```

```
$telefono  
[1] 93345656
```

```
$naci  
[1] 17-Oct-1979
```

- O por el nombre de los elementos

```
> lista[c("telefono", "naci")]
```

```
$telefono  
[1] 93345656
```

```
$naci  
[1] 17-Oct-1979
```

Un `data.frame` se puede convertir en una lista, sencillamente quitándole la clase

```
> datoslista <- unclass(datos)
```

```
> datoslista
```

```
$id
```

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

```
$nombre
```

```
[1] "Joan"      "Isaac"      "Marta"      "Cristina" "Julia"
```

```
$edad
```

```
[1] 51 29 31 28 56
```

```
$naci
```

```
[1] 17-Oct-1979 20-May-1958 12-Feb-1983 01-Dec-1978 28-Mar-1965
```

```
$nueva
```

```
[1] NA NA NA NA NA
```

```
$nueva2
```

```
[1] 1 1 1 1 1
```


```
attr(,"row.names")
```

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


```
> class(datoslista)
```

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

- Un `data.frame` se puede entender como una lista "especial", en que todas sus componentes son del tipo vector con la misma longitud.
- Si sus componentes fueran vectores de distinta longitud (como las notas en el anterior ejemplo), no queda más remedio que ponerlo en una lista dónde cada componente es un vector de las notas de cada asignatura.

- Finalmente, véase otra lista más compleja, como puede ser el resultado de una regresión lineal en  usando la función `lm`.
- Por ejemplo, el peso del bebé según la edad de la madre.
- Nótese que el resultado es una lista que consta de muchos elementos de distinto tipo: vectores, listas, etc.
- Primero se leerá la base de datos 'partoFin', y se pasan los nombres de las variables a minúscula:

```
> datos <- read.spss("partoFin.sav", FALSE, TRUE)
> names(datos) <- tolower(names(datos))
```

y después se realiza una regresión lineal. Para saber la estructura del resultado de la regresión se usa la función `str` (La función `str` es una función genérica que devuelve la estructura de cualquier objeto en ):

```
> result <- lm(peso ~ edad, data = datos)
> str(result)
```

List of 12

```
$ coefficients : Named num [1:2] 4.2288 -0.0348
..- attr(*, "names")= chr [1:2] "(Intercept)" "edad"
$ residuals : Named num [1:28] -0.0124 -0.7879 0.4546 -0.6176 0.3121 ...
..- attr(*, "names")= chr [1:28] "1" "2" "3" "4" ...
$ effects : Named num [1:28] -16.976 -1.221 0.121 -0.541 0.345 ...
..- attr(*, "names")= chr [1:28] "(Intercept)" "edad" "" "" ...
$ rank : int 2
$ fitted.values: Named num [1:28] 3.39 3.29 2.7 3.36 3.29 ...
..- attr(*, "names")= chr [1:28] "1" "2" "3" "4" ...
$ assign : int [1:2] 0 1
$ qr :List of 5
..$ qr : num [1:28, 1:2] -5.292 0.189 0.189 0.189 0.189 ...
.. ..- attr(*, "dimnames")=List of 2
.. .. ..$ : chr [1:28] "1" "2" "3" "4" ...
.. .. ..$ : chr [1:2] "(Intercept)" "edad"
.. ..- attr(*, "assign")= int [1:2] 0 1
..$ qraux: num [1:2] 1.19 1.04
..$ pivot: int [1:2] 1 2
..$ tol : num 1e-07
..$ rank : int 2
..- attr(*, "class")= chr "qr"
$ df.residual : int 26
$ xlevels : Named list()
$ call : language lm(formula = peso ~ edad, data = datos)
```


si se desea extraer, por ejemplo, el elemento pivot del elemento qr:

```
> result$qr$pivot
```

```
[1] 1 2
```

ó extraer los atributos del elemento qr del elemento qr

```
> attributes(result$qr$qr)
```

```
$dim
```

```
[1] 28 2
```

```
$dimnames
```

```
$dimnames[[1]]
```


```
[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10" "11" "12" "13" "14" "15"  
[16] "16" "17" "18" "19" "20" "21" "22" "23" "24" "25" "26" "27" "28"
```

```
$dimnames[[2]]
```

```
[1] "(Intercept)" "edad"
```

```
$assign
```

```
[1] 0 1
```

Las funciones `apply`, `lapply`, `sapply` son muy útiles para ejecutar instrucciones ó cálculos repetitivos en forma de bucle. En  es más eficiente usar estas funciones que la instrucción `for`, la cual se verá más adelante.

Primero, se fija la carpeta de trabajo y el paquete necesario:

```
> rm(list = ls())  
> setwd("C:/cursoR/data")  
> library(foreign)
```

- La función `apply` es útil cuando hay que hacer una misma operación, procedimiento ó cálculo sobre cada una de las columnas ó filas de una matriz ó `data.frame`.
- El primer argumento es la matriz ó el `data.frame`, el segundo es un 1 (si es por filas), un 2 (si es por columnas) o el vector `c(1,2)` (si es for filas y columnas), y el tercer argumento es una función.
- A continuación, algunos de los ejemplos más usuales:

A modo de ejemplo, se creará una matriz, como ya se vio en secciones anteriores:

```
> A <- matrix(c(1, 2, 3, 4, 5, 6, 7, 10, -3), nrow = 3, ncol = 3,  
+           byrow = FALSE)  
> A
```

```
      [,1] [,2] [,3]  
[1,]    1    4    7  
[2,]    2    5   10  
[3,]    3    6   -3
```

```
> # suma por filas de la matriz A
> apply(A,1,sum)

[1] 12 17 6

> # suma por filas (equivalente al anterior)
> rowSums(A)

[1] 12 17 6

> # suma por columnas
> apply(A,2,sum)

[1] 6 15 14

> # suma por columnas (equivalente al anterior)
> colSums(A)

[1] 6 15 14
```

```
> # media por filas  
> apply(A,1,mean)  
  
[1] 4.000000 5.666667 2.000000  
  
> # media por filas (equivalente al anterior)  
> rowMeans(A)  
  
[1] 4.000000 5.666667 2.000000
```

```
> # media por columnas  
> apply(A,2,mean)  
  
[1] 2.000000 5.000000 4.666667  
  
> # media por columnas (equivalente al anterior)  
> colMeans(A)  
  
[1] 2.000000 5.000000 4.666667  
  
> # variancia por columnas  
> apply(A,2,var)  
  
[1] 1.00000 1.00000 46.33333
```

También se puede especificar alguna función más compleja, definida por el usuario

```
> apply(A, 1, function(x) x[1]^2 + x[2]^3 - x[3])
```

```
[1] 58 119 228
```


- O para mirar cuántos elementos mayores de 0 hay por filas. Óbserve, que en lugar de dar la matriz directamente se pone ' $A > 0$ '. Esto resulta una matriz de VERDADERO/FALSO.
- Lo que se hace es sumar por filas esta matriz resultante de VERDADERO/FALSO, en lugar de trabajar con la matriz ' A '. Así se obtiene cuántos elementos, por filas, son mayores a 0.

```
> apply(A > 0, 1, sum)
```

```
[1] 3 3 2
```

- La función `sapply` se utiliza para aplicar a cada elemento del vector 'v' una función 'f'.
- Ejemplo 1: ejemplo muy simple para empezar. Dado un vector 'v', se desea saber si cada uno de sus elementos es mayor que dos ó no.


```
> v <- c(3, 4, 6, 0, 10)
> f <- function(x) x > 2
> sapply(v, f)
```

```
[1] TRUE TRUE TRUE FALSE TRUE
```

esto también se podría hacer simplemente

```
> v > 2
```

```
[1] TRUE TRUE TRUE FALSE TRUE
```

ya que  opera por defecto con los vectores y matrices elemento a elemento.

Ejemplo 2. un poco más complicado. Dada una función que calcula la media y la DS ó una tabla de frecuencias según si la variable es numérica y con más de 4 valores diferentes ó no, respectivamente.

```
> summaryfun <- function(x, misdatos) {
+   variable <- misdatos[, x]
+   cat("---Variable:", x, "----\n")
+   cat("n. missings:", sum(is.na(variable)), "\n")
+   frec <- table(variable)
+   if (length(frec) > 4 & is.numeric(variable)) {
+     cat("media=", round(mean(variable), 3), "\n")
+     cat("desv tipo=", round(sd(variable), 3), "\n")
+   }
+   else {
+     cat("Tabla de frecuencias:\n")
+     print(cbind(frec))
+   }
+   cat("\n\n")
+ }
```

Más tarde se presentarán las funciones y las instrucciones condicionales (if).

- El input de la función será el nombre de una variable y el segundo argumento será el `data.frame`
- El output de la función será el sumario de esta variable.
- Se llama a la función para hacer un sumario de la edad (para una sola variable) de la base de datos 'partoFin.sav'.

```
> datos <- read.spss("partoFin.sav", FALSE, TRUE)
> names(datos) <- tolower(names(datos))
> summaryfun("edad", datos)
```

```
---Variable: edad ----
n. missings: 0
media= 29.286
desv tipo= 6.743
```

- Para hacer un sumario de **todas** las variables de 'datos' se usará la función `sapply`.
- Es importante fijarse en el tercer argumento '`misdatos=datos`'. La función `summaryfun` necesita dos argumentos: el primero ya se especifica en el primer argumento de `sapply` con `names(datos)`, pero el segundo hay que especificarlo y se pone al final de todo de la función `sapply` especificando su nombre y su valor `misdatos=datos`.

```
> sapply(names(datos), summaryfun, misdatos = datos)
```

```
---Variable: id ----
n. missings: 0
media= 14.5
desv tipo= 8.226
```

```
---Variable: ini ----
n. missings: 0
Tabla de frecuencias:
      frec
ADJU      1
ANZO      1
BEMI      1
BOPE      1
CAEL      1
CAGI      1
CEMA      1
COMO      1
```

- La función `lapply` es parecida a `sapply`, pero sirve para aplicar una función a cada elemento de una lista.
- Ejemplo. Notas de los exámenes de cada asignatura.
- Nótese que se han hecho un número distinto de exámenes según la asignatura: por lo tanto no se puede poner ni en un `data.frame` ni en una matriz.

```
> notas <- list()  
> notas$mates = c(5, 6.5, 7, 8, 6.5)  
> notas$calatan = c(4, 4.5, 4.75, 7, 3.25, 5.5, 7.7)  
> notas$castellano = c(5.5, 4.5, 6, 6.5, 7, 7, 4.5)  
> notas$ingles = c(8, 9, 9.5, 7.5)  
> notas$historia = c(8, 9, 9.5, 10, 10, 4.5)
```

Si se desea saber cuántos exámenes se han hecho por asignatura

```
> lapply(notas, length)
```

```
$mates  
[1] 5
```

```
$calatan  
[1] 7
```

```
$castellano  
[1] 7
```

```
$ingles  
[1] 4
```

```
$historia  
[1] 6
```

o si se quiere saber cuál es la media por asignatura

```
> lapply(notas, mean)
```

```
$mates  
[1] 6.6
```

```
$calatan  
[1] 5.242857
```

```
$castellano  
[1] 5.857143
```

```
$ingles  
[1] 8.5
```

```
$historia  
[1] 8.5
```


- Para realizar cálculos (medias, sumarios, etc.) de una variable agrupando ("by") por los grupos definidos por otra variable categórica ó una combinación de variables categóricas.
- Por ejemplo, calcular la media de la edad de la madre según el sexo del niño:

```
> with(datos, by(edad, sexo, mean, na.rm = TRUE))
```

```
sexo: 1  
[1] 27.66667
```

```
-----  
sexo: 2  
[1] 30.5
```

- ó la media de la edad de la madre según el sexo del niño y el tipo de parto.
- Nótese que hay que poner la combinación de sexo y tipo de parto en una lista (*list*)

```
> with(datos, by(edad, list(sexo, tip_par), mean))
```


```
: 1
: 1
[1] 30
-----
: 2
: 1
[1] 29
-----
: 1
: 2
[1] 27.2
-----
: 2
: 2
[1] 30.84615
```

- El resultado de hacer un `by` es una lista.
- Si hace falta, se puede transformar en un vector, simplemente con la función `unlist`.

```
> unlist(with(datos, by(edad, sexo, mean)))
```

```
sexo: 1  
[1] 27.66667
```

```
-----  
sexo: 2  
[1] 30.5
```

Las funciones en  se escriben con su nombre y después entre paréntesis se escriben sus argumentos. Los argumentos van separados por comas, en orden. Si se cambia el orden hay que especificar el nombre del argumento.

```
> datos <- read.spss("partoFin.sav", FALSE, TRUE)
> datos <- read.spss(to.data.frame = FALSE, use.value.labels = TRUE,
+   file = "partoFin.sav")
```

Para ver los argumentos de una función y saber en qué orden están, hay que usar la función `args`

```
> args(read.spss)
```

```
function (file, use.value.labels = TRUE, to.data.frame = FALSE,  
  max.value.labels = Inf, trim.factor.names = FALSE, trim_values = TRUE,  
  reencode = NA, use.missings = to.data.frame)  
NULL
```

Es posible ver el código de algunas funciones simplemente tecleando el nombre de la función sin paréntesis ni comillas, (o usando la función body),

```
> read.spss
```

```
function (file, use.value.labels = TRUE, to.data.frame = FALSE,
  max.value.labels = Inf, trim.factor.names = FALSE, trim_values = TRUE,
  reencode = NA, use.missings = to.data.frame)
{
  trim <- function(strings, trim = TRUE) if (trim)
    sub(" +$", "", strings)
  else strings
  knownCP <- c(`UCS-2LE` = 1200, `UCS-2BE` = 1201, macroman = 10000,
    `UCS-4LE` = 12000, `UCS-4BE` = 12001, `koi8-r` = 20866,
    `koi8-u` = 21866, latin1 = 28591, latin2 = 28592, latin3 = 28593,
    latin4 = 28594, `latin-9` = 28605, `ISO-2022-JP` = 50221,
    `euc-jp` = 51932, `UTF-8` = 65001, ASCII = 20127, CP1250 = 1250,
    CP1251 = 1251, CP1252 = 1252, CP1253 = 1253, CP1254 = 1254,
    CP1255 = 1255, CP1256 = 1256, CP1257 = 1257, CP1258 = 1258,
    CP874 = 874, CP936 = 936)
  if (length(grep("(http|ftp|https)://", file))) {
    tmp <- tempfile()
    download.file(file, tmp, quiet = TRUE, mode = "wb")
    file <- tmp
    on.exit(unlink(file))
  }
  rval <- .Call(do_read_SPSS, file)
  codepage <- attr(rval, "codepage")
  if (is.null(codepage))
    codepage <- 2
  if (!capabilities("iconv"))
    reencode <- FALSE
  if (!identical(reencode, FALSE)) {
```

Se lee la base de datos 'partoFin.sav' y se ponen los nombres de las variables en minúsculas, por comodidad.

```
> datos <- read.spss("partoFin.sav", FALSE, TRUE)  
> names(datos) <- tolower(names(datos))
```

- Se usa la instrucción `function` para crear nuevas funciones
- Ejemplo 1. Cálculo del coeficiente de variación
- El coeficiente de variación CV se define como el ratio entre la desviación estándar y la media en valor absoluto, todo ello multiplicado por 100:

$$CV = 100 \frac{\tilde{S}}{|\bar{x}|} \quad (1)$$

```
> coef.var <- function(x) {  
+   media <- mean(x)  
+   desv <- sd(x)  
+   return(100 * (desv/abs(media)))  
+ }
```


Luego se llama a la función con su nombre, y entre paréntesis se especifica el argumento: la edad de la madre.

```
> coef.var(datos$edad)
```

```
[1] 23.0256
```

Ejemplo 2. Creación de la fecha a partir del día, el mes y el año.

```
> convert.dates <- function(day, month, year) {  
+   ans <- apply(cbind(day, month, year), 1, paste, collapse = "-")  
+   ans <- chron(ans, format = "d-m-Y", out.format = "d-mon-Y")  
+   return(ans)  
+ }
```

Supóngase que se tienen las siguientes fechas, con los días, meses y años:

```
> dia <- c(10, 23, 17, 5, 6)
> mes <- c(1, 7, 12, 11, 12)
> año <- c(1978, 1932, 1967, 1982, 1967)
```

en columnas:

```
> cbind(dia, mes, año)
```

```
      dia mes  año  
[1,]  10   1 1978  
[2,]  23   7 1932  
[3,]  17  12 1967  
[4,]   5  11 1982  
[5,]   6  12 1967
```

Se llama a la función `convert.dates` poniendo los argumentos en orden y el resultado se guarda en la variable 'fecha'

```
> fecha <- convert.dates(dia, mes, año)
```

Cuando se llama a una función con más de un argumento, como en este caso donde hay 3 (day, month, year), no hace falta respetar el orden en que se ponen. Aunque si se pone un argumento en una posición que no es la suya hay que especificar su nombre. Por ejemplo,

```
> fecha <- convert.dates(month = mes, year = año, day = dia)
> fecha
```

```
[1] 10-Jan-1978 23-Jul-1932 17-Dec-1967 05-Nov-1982 06-Dec-1967
```

- Finalmente se coloca el resultado en columnas juntamente a las variables día, mes y año, para visualizarlo.
- Aunque es más adecuado ponerlo en un `data.frame`, porque las variables son de diferente tipo (día, mes y año son de tipo entero, mientras que la fecha es de clase `chron`).

```
> data.frame(fecha, dia, mes, año)
```

```
      fecha dia mes año
1 10-Jan-1978 10  1 1978
2 23-Jul-1932 23  7 1932
3 17-Dec-1967 17 12 1967
4 05-Nov-1982  5 11 1982
5 06-Dec-1967  6 12 1967
```

- Ejemplo 3. Cálculo del intervalo de confianza de la media.
- Esta es una función con un argumento obligatorio (la variable) y uno opcional (el error α , que tiene por defecto el valor 0.05).

$$IC = \bar{x} \pm t_{\alpha; n-1} \frac{\tilde{S}}{\sqrt{n}} \quad (2)$$

```
> ic.media <- function(x, alfa = 0.05) {  
+   media <- mean(x)  
+   desv <- sd(x)  
+   n <- length(x)  
+   tv <- qt(1 - alfa/2, n - 1)  
+   inf <- media - tv * desv/sqrt(n)  
+   sup <- media + tv * desv/sqrt(n)  
+   return(c(`Inf` = inf, Sup = sup))  
+ }
```


intervalo de confianza al 95 % (por defecto).

```
> ic.media(datos$edad)
```

Inf	Sup
26.67097	31.90046

intervalo de confianza al 99 %: hay que especificar el error de tipo-I (α) a 0.01:

```
> ic.media(datos$edad, alfa = 0.01)
```

Inf	Sup
25.75490	32.81653

Los objetos definidos dentro de una función sólo existen dentro de ellas. De la función 'ic.media', si se desea saber cuanto vale el objeto 'tv'

```
> tv
```

aparece un mensaje de error.

- A veces puede ser interesante saber el valor de algunos objetos que sólo están definidos dentro de la función, para chequear que se ha definido bien.
- Por ejemplo, para ver cuánto vale 'tv' hay que redefinir la función, y asignar a 'tv' con una flecha de doble punta y volver a llamar a la función.

```
> ic.media <- function(x, alfa = 0.05) {  
+   media <- mean(x)  
+   desv <- sd(x)  
+   n <- length(x)  
+   tv <- qt(1 - alfa/2, n - 1)  
+   inf <- media - tv * desv/sqrt(n)  
+   sup <- media + tv * desv/sqrt(n)  
+   return(c(`Inf` = inf, Sup = sup))  
+ }  
> ic.media(datos$edad)
```

```
      Inf      Sup  
26.67097 31.90046
```

Ahora el objeto 'tv' ya existe fuera de la función

```
> tv
```

```
[1] 2.051831
```

- Supóngase que se quiere hacer una prueba Fisher de asociación entre el sexo y cada una de las siguientes variables: tip_par, fuma_an, fuma_de y masde12.
- Primero se almacena en el objeto vars el nombre de estas 4 variables

```
> vars <- c("tip_par", "fuma_an", "fuma_de", "masde12")
```

- Se va a usar un bucle (for) para calcular un test de Fisher para cada una de las 4 variables y el sexo
- Dentro del bucle se usará la función print y cat para ver por pantalla el nombre de la variable y el resultado del test.

```
> for (i in 1:length(vars)) {
+   nom.var <- vars[i]
+   var <- datos[, nom.var]
+   tt <- table(datos$sexo, var)
+   cat("---Prueba Fisher entre sexo y", nom.var, "-----\n")
+   print(fisher.test(tt))
+ }
```

```
---Prueba Fisher entre sexo y tip_par -----
```

```
Fisher's Exact Test for Count Data
```

```
data: tt
p-value = 1
alternative hypothesis: true odds ratio is not equal to 1
95 percent confidence interval:
 0.06160374 9.21621060
sample estimates:
odds ratio
 0.8710761
```

```
---Prueba Fisher entre sexo y fuma_an -----
```

```
Fisher's Exact Test for Count Data
```

```
data: tt
```

Ejemplo. Creación de una función que calcula la media ó la desviación estándar según se indique en el segundo argumento ('what'):

```
> calcula<-function(x,what=c("media","desv")){  
+   if (what=="media") return(mean(x))  
+   if (what=="desv") return(sd(x))  
+ }
```

```
> # calula la media de la edad de la madre  
> calcula(datos$edad,"media")
```

```
[1] 29.28571
```

```
> # calula la desviación estándar de la edad de la madre  
> calcula(datos$edad,"desv")
```


```
[1] 6.743211
```


- Ejemplo. Supóngase que la probabilidad de éxito en un experimento es del 5 %.
- ¿Cuántos experimentos hay que hacer para conseguir el primer éxito?
- Para ello se usará la instrucción `while`; mientras no se consiga un éxito, se va repitiendo el experimento.

```
> x <- 0
> intentos <- 0
> max.intentos <- 1000
> while (x == 0 & intentos < max.intentos) {
+   x <- rbinom(1, 1, 0.05)
+   intentos <- intentos + 1
+ }
> intentos
```

```
[1] 20
```

NOTA: repitiendo estas instrucciones y mirando el valor de `intentos`, hay que darse cuenta que éste es variante (aleatorio) ya que va tomando distintos valores al azar (aunque en promedio será 20).


En  existen gran variedad de funciones que permiten generar y trabajar con variables aleatorias (funciones de densidad, etc.). En el ejemplo anterior se ha visto la función para generar binomiales/bernoullis `rbinom`, pero existen muchas más. He aquí una tabla resumen, con algunas de las distribuciones más importantes:

Distribución	densidad/probabilidad	probabilidad acumulada	quantiles	números aleatorios
Normal	<code>dnorm</code>	<code>pnorm</code>	<code>qnorm</code>	<code>rnorm</code>
t-Student	<code>dt</code>	<code>pt</code>	<code>qt</code>	<code>rt</code>
χ^2	<code>dchisq</code>	<code>pchisq</code>	<code>qchisq</code>	<code>rchisq</code>
F de Fisher	<code>df</code>	<code>pf</code>	<code>qf</code>	<code>rf</code>
Binomial ¹	<code>dbinom</code>	<code>pbinom</code>	<code>qbinom</code>	<code>rbinom</code>
Poisson	<code>dpois</code>	<code>ppois</code>	<code>qpois</code>	<code>rpois</code>
Exponencial	<code>dexp</code>	<code>pexp</code>	<code>qexp</code>	<code>rexp</code>
Gamma	<code>dgamma</code>	<code>pgamma</code>	<code>qgamma</code>	<code>rgamma</code>
Weibull	<code>dweibull</code>	<code>pweibull</code>	<code>qweibull</code>	<code>rweibull</code>
Catagórica	<code>dmultinom</code>	<code>—</code> ²	<code>—</code> ³	<code>rmultinom</code>


¹ especificando el argumento `size=1` se obtiene la Bernoulli

² No tiene sentido la distribución de probabilidad acumulada al tratarse de una variable multidimensional

³ No tiene sentido los quantiles al tratarse de una variable multidimensional

-  es especialmente potente en crear gráficos de todo tipo.
- Hay múltiples opciones, funciones y packages, para hacer todo tipo de gráficos.
- He aquí unos ejemplos:

```
> demo(graphics)
```

- Para ilustrar como se crean los gráficos en  se mostrarán algunos ejemplos.
- Para ello se trabajará con algunos datos de "partoFin.sav".

```
> rm(list = ls())  
> setwd("C:/cursoR/data")  
> library(foreign)  
> datos <- read.spss("partoFin.sav", FALSE, TRUE)
```

La función más básica y principal para crear gráficos es `plot`. `plot` es una función genérica (métodos y clases): según el input genera diferentes tipos de gráfico:

- Si el input es un vector numérico genera un scatterplot:

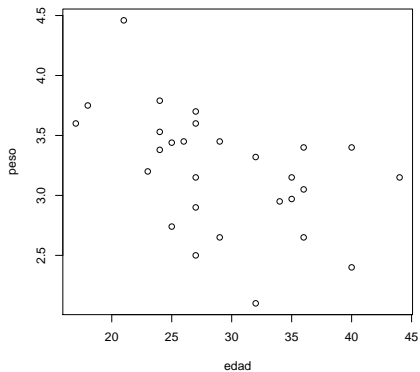
```
> plot(datos$edad)
```
- Si el input es un factor o un carácter se genera un gráfico de barras:

```
> tx.fac <- factor(datos$tx, labels = names(sort(attr(datos$tx,  
+ "value.labels"))))  
> plot(tx.fac)
```
- o si el input es el resultado de una regresión lineal, el resultado son diferentes gráficos más complejos:

```
> rl <- lm(peso ~ edad, data = datos)  
> par(mfrow = c(2, 2))  
> plot(rl)
```
- etc.

- En el primer ejemplo, se creará un gráfico *Scatterplot* de correlación entre dos variables cuantitativas: la edad de la madre y el peso del bebé.
- En este primer ejemplo se verá como modificar las etiquetas de los ejes, el título, colores, etc.
- Se usará la función `plot` (`?plot.default`).
- Se pondrá la edad en el eje 'x' y el peso en el eje 'y'.

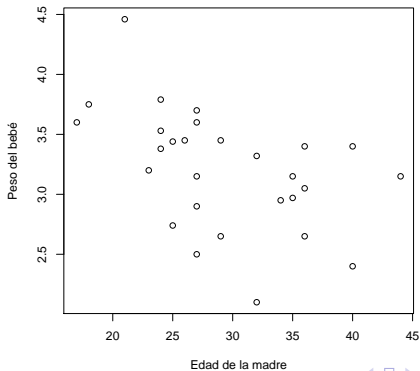
```
> #graphics.off() # cierra todos los gráficos  
> with(datos,plot(edad,peso))
```



El gráfico anterior es como queda por defecto. En seguida se presentan algunas instrucciones para efectuar cambios sobre el anterior gráfico.

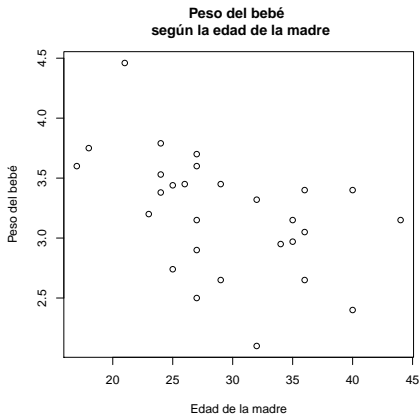
Para cambiar las etiquetas de los ejes, en los argumentos xlab y ylab

```
> with(datos, plot(edad, peso, xlab = "Edad de la madre", ylab = "Peso del bebé"))
```



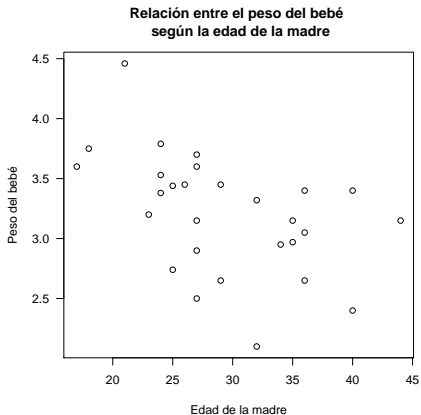
Para añadir un título, se especifica en el argumento `main`. Nótese en este ejemplo el carácter `"\n"` que significa cambio de línea.

```
> with(datos, plot(edad, peso, xlab = "Edad de la madre", ylab = "Peso del bebé",  
+   main = "Peso del bebé \n según la edad de la madre"))
```



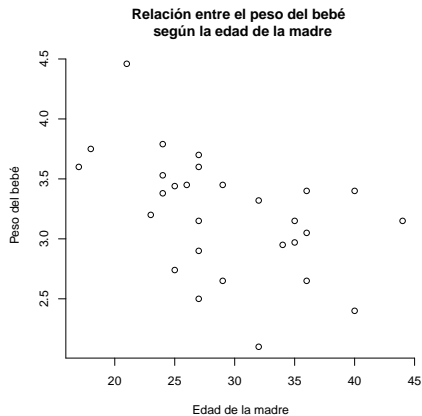
- También se puede añadir un título con la función `title` una vez hecho el gráfico.
- Por defecto, los números en los ejes están paralelos a los ejes.
- Puede ser interesante que los números siempre estén en horizontal.
- Para ello se cambia el argumento `las=1` de la función `par`.

```
> par(las = 1)
> with(datos, plot(edad, peso, xlab = "Edad de la madre", ylab = "Peso del bebé"))
> title("Relación entre el peso del bebé \n según la edad de la madre")
```



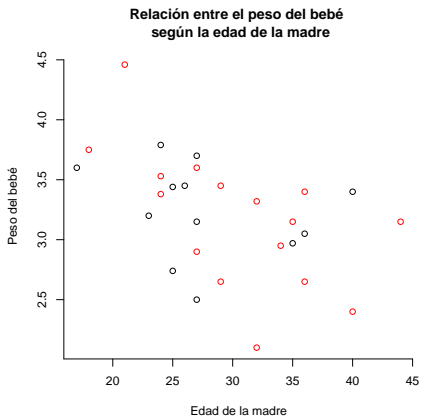
- Si se quiere que aparezca solamente el eje de las x's y de las y's, primero hay que quitar todos los ejes especificando el argumento `axes=FALSE`, y posteriormente añadiendo cada uno de los ejes con la función `axis`.
- Nótese la función `range` que devuelve el mínimo y el máximo, y la función `pretty` que devuelve una secuencia de números (5 ó 6 por defecto) entre el mínimo y el máximo sin ó con pocos decimales de forma que quede "bonito".
- También es posible especificar los números que se desea que aparezca en los ejes (con `c` ó con `seq`).


```
> with(datos, plot(edad, peso, xlab = "Edad de la madre", ylab = "Peso del bebé",  
+ axes = FALSE, las = 1))  
> title("Relación entre el peso del bebé \n según la edad de la madre")  
> axis(1, pretty(range(datos$edad, na.rm = TRUE)))  
> axis(2, pretty(range(datos$peso, na.rm = TRUE)))
```



Diferentes colores de los puntos según una tercera covariable (p.e. sexo del bebé) - argumento `col`:

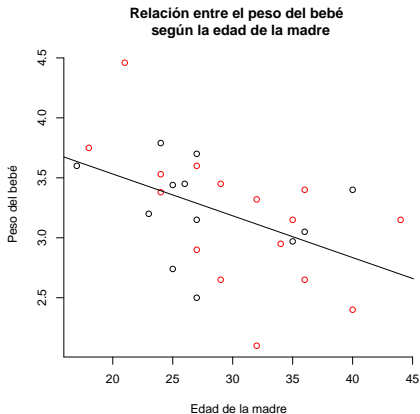
```
> with(datos, plot(edad, peso, xlab = "Edad de la madre", ylab = "Peso del bebé",  
+ axes = FALSE, col = datos$sexo, las = 1))  
> title("Relación entre el peso del bebé \n según la edad de la madre")  
> axis(1, pretty(range(datos$edad, na.rm = TRUE)))  
> axis(2, pretty(range(datos$peso, na.rm = TRUE)))
```



Como la variable sexo está codificado como 1-niño y 2-niña, y como el 1 en  es negro y el 2 es rojo, salen los puntos correspondientes a los niños en negro y a las niñas en rojo. Para especificar cualquier otro color (por ejemplo azul para niños y rosa para niñas):

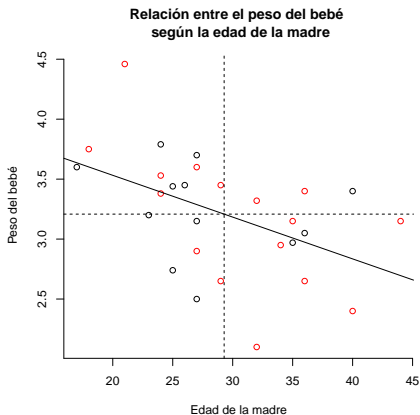
```
> col = ifelse(datos$sexo == 1, "blue", "pink")
```

Para añadir una recta de regresión se usará la función `abline`, y los coeficientes de la regresión lineal (`coef,lm`).



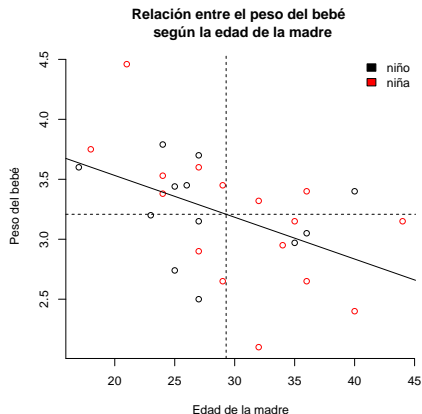
```
> abline(coef(lm(peso ~ edad, data = datos)))
```


Añadir rectas horizontal y vertical en las medias de edad y peso → función `abline` con el argumento `h=` y `v=`. Tipo de línea → argumento `lty`:



```
> abline(v = mean(datos$edad, na.rm = TRUE), h = mean(datos$peso,  
+       na.rm = TRUE), lty = 2)
```

Para poner una leyenda con los códigos de colores de los puntos se usa la función `legend`



```
> legend("topright", c("niño", "niña"), fill = c(1, 2), bty = "n")
```

Otras cosas que se pueden hacer en el gráfico son

- añadir un texto con la función `text`, especificando las coordenadas interactivamente con la función `locator`:

```
> text(locator(1), "punto")
```

- añadir puntos con la función `points`:

```
> # añade dos puntos en forma de cuadrado (pch) y más grandes (cex)
```

```
> points(c(25,30),c(3,4),pch=22,cex=2)
```

- añadir líneas con la función `lines`:

```
> # conecta los dos puntos anteriores con una línea más gruesa (lwd)
```

```
> lines(c(25,30),c(3,4),lwd=2)
```

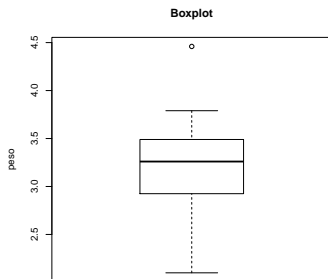
```
> # identificar un punto por la variable 'ini'
```

```
> with(datos,identify(edad,peso,ini))
```

- ...

Para hacer una descripción de una variable cuantitativa es muy útil realizar un *boxplot*. Para ello, se usa la función `boxplot`. Por ejemplo, para la variable peso,

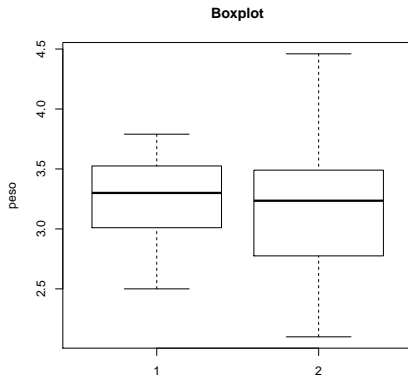
```
> boxplot(datos$peso, ylab = "peso", main = "Boxplot")
```



Nótese que se pueden añadir los argumentos `xlab`, `ylab` y `main` como en la función `plot`. En general hay muchos de estos argumentos que son comunes para muchas funciones de gráficos.

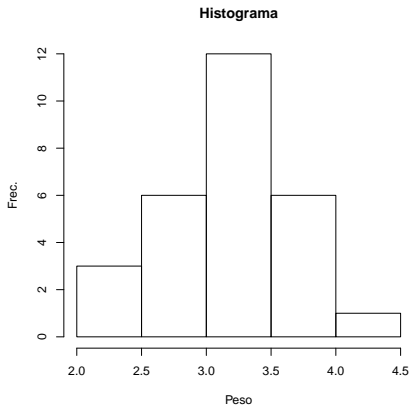
Si se quiere hacer un boxplot separado según el sexo del bebé (un boxplot al lado del otro)

```
> boxplot(peso ~ sexo, data = datos, ylab = "peso", main = "Boxplot")
```



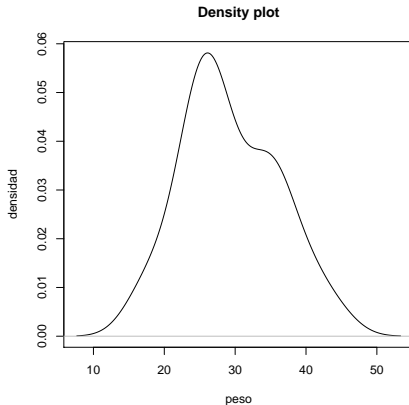
También, para describir visualmente variables cuantitativas hay la opción de realizar histogramas. Para ello, existe la función `hist`:


```
> hist(datos$peso, xlab = "Peso", ylab = "Frec.", main = "Histograma")
```



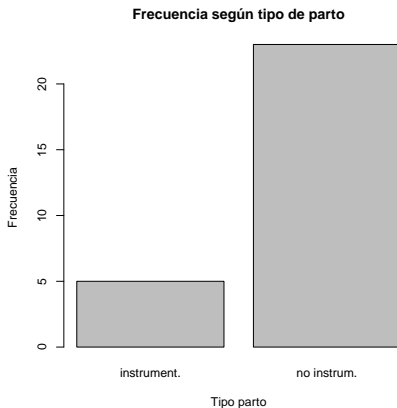
Un gráfico similar al histograma, pero más "fino" es el *density plot*. Realiza técnicas no paramétricas de suavizado para estimar la curva de función de densidad de una variable cuantitativa, mediante la función `density`:

```
> plot(density(datos$edad, na.rm = TRUE), xlab = "peso", ylab = "densidad",  
+      main = "Density plot")
```



La función en  para hacer gráficos de barras es `barplot`. Por ejemplo si se desea representar la frecuencia del tipo de parto:

```
> frec <- table(datos$tip_par)
> vl <- names(sort(attr(datos$tip_par, "value.labels")))
> barplot(frec, names.arg = vl, xlab = "Tipo parto", ylab = "Frecuencia")
> title("Frecuencia según tipo de parto")
```



Los gráficos de pastel son usados muy frecuentemente, aunque no son muy recomendables desde el punto de vista estadístico. Para hacer estos gráficos se puede usar la función `pie`. Por ejemplo, si se desea representar la frecuencia del tipo de parto:

```
> frec <- table(datos$tip_par)
> vl <- names(sort(attr(datos$tip_par, "value.labels")))
> pie(frec, labels = vl, xlab = "Tipo parto", ylab = "Frecuencia")
> title("Frecuencia según tipo de parto")
```



- A veces puede ser interesante representar más de un gráfico en una misma ventana (*Device*).
- Para ello, se especifica en el argumento `mfrow` de la función par un vector de dos componentes que indica cuántos gráficos se desean hacer por filas y por columnas.
- Todos los gráficos resultantes tendrán el mismo tamaño. Si se quiere dividir la ventana en diferentes gráficos de distinto tamaño hay que usar la función `layout`, aunque esto se escapa de este curso.

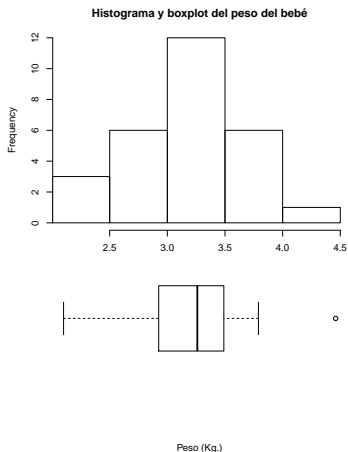
- Ejemplo: representación de un histograma arriba y un boxplot de forma horizontal debajo, de la variable peso del bebé. Se pondrán los mismos límites para el eje que represente el peso del bebé.
- Con la función `windows` se abre una ventana de los tamaños especificados (amplitud y altura respectivamente); en este ejemplo se abre una ventana más alta que ancha (*sólo para Windows*).



```
> windows(height = 8, width = 6)
```

```

> limites <- range(datos$peso)
> par(mfrow = c(2, 1))
> par(mar = c(0, 4, 3, 1))
> hist(datos$peso, xlim = limites, main = "Histograma y boxplot del peso del bebé")
> par(mar = c(5, 4, 0, 1))
> boxplot(datos$peso, horizontal = TRUE, ylim = limites, axes = FALSE,
+         xlab = "Peso (Kg.)")

```



- Para almacenar los gráficos en archivos en  hay diferentes maneras.
- La primera y más "naive", una vez se hace el gráfico y aparece por pantalla, a través del menú de la consola de , ir a archivo → guardar como... y especificar el formato.
- Hay muchos formatos: pdf, jpg, Windows metafile, etc.
- También hay una función que permite guardar un gráfico una vez está en pantalla: `savePlot`, la sintaxis es:

```
> savePlot("C:/cursoR/figura.pdf", type = "pdf")
```



pero hay que tener en cuenta que esta instrucción guardará el gráfico que tengamos hecho (, y sólo es válido para Windows). Si se cierra y no hay ninguno activo dará un error, y si hay varios abiertos, guardará el último que se haya realizado.


- Pero hay otras funciones que permiten guardar gráficos antes de realizarlos; por ejemplo, para guardar un gráfico en pdf:

```
> pdf("C:/cursoR/figura.pdf")  
> hist(datos$peso, xlab = "peso", ylab = "frec.", main = "Histograma")  
> dev.off()
```

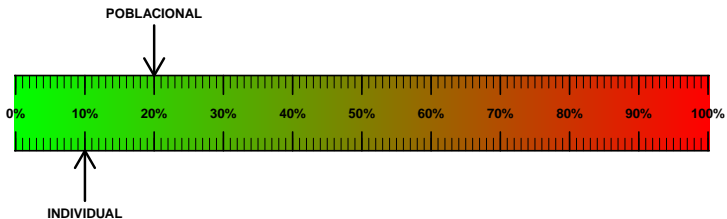
- Similarmente, existe la función `png` para guardar gráficos en formato `png`, `jpeg`, `tiff`, etc.
- Todas estas funciones (`pdf`, `png`, ...) tienen argumentos que permiten mejorar notablemente la resolución del gráfico.

- En general, para modificar aspectos del gráfico, desde el tamaño de las letras hasta el grosor de las líneas pasando por cuantos gráficos se quieren en un mismo *Device* ó los márgenes, etc., se usa la función `par`.
- Es muy útil mirar la ayuda de esta función `?par`, y ver todos sus argumentos con un poco de paciencia ya que son muchos.
- Existen múltiples `packages` con funciones para crear gráficos más sofisticados o específicos, como los típicos de un metaanálisis, ó para crear gráficos en 3 dimensiones, o para hacer mapas, etc. Esto ya daría para muchas más horas y está fuera del alcance de este curso.

- Para especificar los colores, se pueden indicar con un número ( tiene codificados los colores principales con números; 1-negro, 2-rojo, etc.), con un texto en inglés ("black","red","green",etc.), o también con el código rgb que es multiplataforma con la función `rgb`.
- Hacer gráficos en  puede ser muy laborioso o muy simple, todo depende del tipo de gráfico que se quiera hacer y de si hay alguna función ya implementada que sea de ayuda. Pero hay que tener en cuenta (y esto es una gran ventaja) que se puede guardar todas las instrucciones para poder reproducir exactamente el mismo gráfico en un futuro. O fácilmente modificar algún texto (título ó las etiquetas de los ejes).

He aquí un ejemplo para mostrar la potencia de  para tratar con colores. En este caso se grafica una barra con colores graduales, y se usa la función `image`:

```
> a<-0.3
> par(xaxs="i",yaxs="i")
> zz<-matrix(rep(seq(0,100,len=256),100),byrow=TRUE,ncol=256,
+   nrow=100)
> colors<-rgb(0:(256-1),(256-1):0,0,maxColorValue=255)
> image(seq(0,100,len=256),seq(-a,a,len=101),t(zz), col=colors,
+   ylim=c(-1,1),axes=FALSE,xlab="",ylab="")
> abline(h=c(a,-a),lw=2)
> arrows(seq(0,100,10),a,seq(0,100,10),a-a/2,len=0,lw=2)
> arrows(seq(0,100,1),a,seq(0,100,1),a-a/3,len=0)
> arrows(seq(0,100,10),-a,seq(0,100,10),-a+a/2,len=0,lw=2)
> arrows(seq(0,100,1),-a,seq(0,100,1),-a+a/3,len=0)
> par(xpd=NA)
> text(seq(0,100,10),rep(0,11),paste(seq(0,100,10),"%",sep=""),
+   font=2)
> arrows(20,0.7,20,a,code=2,lw=3)
> arrows(10,-0.7,10,-a,code=2,lw=3)
> text(20,0.8,"POBLACIONAL",font=2)
> text(10,-0.8,"INDIVIDUAL",font=2)
```




Hay más ejemplos en la carpeta 'misc' (Miscelania) del material del curso (Ejemplo de gráficos). Es muy recomendable que intentéis ejecutarlos a ver qué os dá. En los ejemplos hay las instrucciones comentadas.

... Y muchos más gráficos en la web

<http://addictedtor.free.fr/graphiques/>

¿Propuestas?

- En esta sección se presentan algunas funciones que pueden ser útiles para el manejo de datos y el análisis estadístico.
- Estas funciones se han creado en la unidad de la URLEC como uso interno, y son un ejemplo de como se puede aprovechar el potencial de  para resolver problemas cotidianos del día a día.
- Las siguientes y algunas más se encuentran en la carpeta 'rutinas' del material del curso. Aunque son funciones propias de la unidad, y por lo tanto no están documentadas...

- Para ilustrar algunas algunas funciones, como siempre se empieza cargando los paquetes necesarios y fijando la carpeta de trabajo.
- Además, en este caso, se indica en qué carpeta hay los *scripts* de estas funciones propias.

```
> rm(list = ls())  
> library(foreign)  
> library(chron)  
> setwd("C:/cursoR/data")  
> RutinasLocales <- "C:/cursoR/rutinas"
```

Con la función `read.spss4` se lee un archivo de SPSS. A diferencia de `read.spss`, `read.spss4` lee los nombres de las variables aunque tenga más de 8 caracteres, pone las variables en su formato correcto (inclusive las fechas) y convierte por defecto a NA los valores que están definidos como missing por el usuario en SPSS.

```
> source(file.path(RutinasLocales, "read.spss4.r"))
```

```
> datos <- read.spss4("partoFin.sav")
```


```
> datos
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo	tip_par
1	1	GADI	14-Jun-2001	19-Jun-2001	26-Jul-2001	2	24	3.38	2	1
2	2	CAEL	15-Jun-2001	21-Jun-2001	15-Feb-2002	2	27	2.50	1	2
3	3	COMO	16-Jun-2001	01-Jul-2001	23-Jun-2001	1	44	3.15	2	2
4	4	VIMU	18-Jun-2001	23-Jun-2001	17-Dec-2001	2	25	2.74	1	1
5	5	PAVI	19-Jun-2001	25-Jun-2001	26-Jun-2001	1	27	3.60	2	2
6	6	PASA	20-Jun-2001	01-Jul-2001	27-Jun-2001	1	36	2.65	2	1
7	7	VERI	20-Jun-2001	30-Jun-2001	12-Sep-2001	2	35	2.97	1	1
8	8	ADJU	21-Jun-2001	25-Jun-2001	13-Sep-2001	2	23	3.20	1	2
9	9	BEMI	22-Jun-2001	05-Jul-2001	31-Aug-2001	1	40	2.40	2	2
10	10	JUNA	23-Jun-2001	02-Jul-2001	29-Sep-2001	2	32	2.10	2	2
11	11	LOKO	26-Jun-2001	02-Jul-2001	21-Aug-2001	1	26	3.45	1	2
12	12	FRFU	27-Jun-2001	04-Jul-2001	06-Mar-2002	2	29	3.45	2	2
13	13	FUFE	06-Jul-2001	17-Jul-2001	13-Jul-2001	1	36	3.40	2	2
14	14	POCA	13-Jul-2001	24-Jul-2001	09-Nov-2001	2	36	3.05	1	2
15	15	LOLO	13-Jul-2001	14-Jul-2001	20-Jul-2001	1	17	3.60	1	2
16	16	BOPE	14-Jul-2001	27-Jul-2001	19-Jan-2002	1	40	3.40	1	2
17	17	ANZO	18-Jul-2001	24-Jul-2001	05-Dec-2001	2	27	3.15	1	2
18	18	MEVE	18-Jul-2001	27-Jul-2001	27-Mar-2002	2	32	3.32	2	2
19	19	TOPO	19-Jul-2001	26-Jul-2001	11-Oct-2001	1	29	2.65	2	2
20	20	PUPI	20-Jul-2001	23-Jul-2001	12-Oct-2001	2	21	4.46	2	2
21	21	BOBA	20-Jul-2001	20-Jul-2001	17-Aug-2001	1	35	2.15	2	2

Recuérdase que con la función `read.spss`, los nombres de las variables quedan truncados a 8 caracteres, y las variables fecha no están en este formato, sino que son enteros.

```
> read.spss("partoFin.sav", FALSE, TRUE)
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo	tip_par
1	1	GADI	13211856000	13212288000	13215484800	2	24	3.38	2	1
2	2	CAEL	13211942400	13212460800	13233110400	2	27	2.50	1	2
3	3	COMO	13212028800	13213324800	13212633600	1	44	3.15	2	2
4	4	VIMU	13212201600	13212633600	13227926400	2	25	2.74	1	1
5	5	PAVI	13212288000	13212806400	13212892800	1	27	3.60	2	2
6	6	PASA	13212374400	13213324800	13212979200	1	36	2.65	2	1
7	7	VERI	13212374400	13213238400	13219632000	2	35	2.97	1	1
8	8	ADJU	13212460800	13212806400	13219718400	2	23	3.20	1	2
9	9	BEMI	13212547200	13213670400	13218595200	1	40	2.40	2	2
10	10	JUNA	13212633600	13213411200	13221100800	2	32	2.10	2	2
11	11	LOKO	13212892800	13213411200	13217731200	1	26	3.45	1	2
12	12	FRFU	13212979200	13213584000	13234752000	2	29	3.45	2	2
13	13	FUFE	13213756800	13214707200	13214361600	1	36	3.40	2	2
14	14	POCA	13214361600	13215312000	13224643200	2	36	3.05	1	2
15	15	LOLO	13214361600	13214448000	13214966400	1	17	3.60	1	2
16	16	BOPE	13214448000	13215571200	13230777600	1	40	3.40	1	2
17	17	ANZO	13214793600	13215312000	13226889600	2	27	3.15	1	2
18	18	MEVE	13214793600	13215571200	13236566400	2	32	3.32	2	2
19	19	TOPO	13214880000	13215484800	13222137600	1	29	2.65	2	2
20	20	PUPI	13214966400	13215225600	13222224000	2	21	4.46	2	2
21	21	ROPA	13214966400	13215830400	13217385600	1	35	3.15	2	2
22	22	LOMA	13215052800	13215571200	13234406400	2	27	3.70	1	2
23	23	CEMA	13215139200	13215571200	13216953600	1	24	3.79	1	2
24	24	CAGI	13215225600	13215398400	13215830400	2	18	3.75	2	2
25	25	GRSE	13215312000	13216176000	13216521600	1	34	2.95	2	2
26	26	GUMA	13215398400	13215916800	13227494400	2	27	2.90	2	1
27	27	PERI	13215398400	13215830400	13230518400	2	25	3.44	1	2
28	28	MADE	13215398400	13215830400	13230518400	1	24	3.52	2	2

Con la función `fix2` se visualiza un `data.frame` o una matriz en una parrilla o en la misma consola de , según se especifique el argumento `print=TRUE` o no. Además, para las variables etiquetadas con el atributo `value.labels`, es posible ver sus etiquetas de valor, sus valores o ambos a la vez especificando el argumento `view`.

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo
1	1	GADI	14-Jun-2001	19-Jun-2001	26-Jul-2001	Intensivo	24	3.38	niña
2	2	CAEL	15-Jun-2001	21-Jun-2001	15-Feb-2002	Intensivo	27	2.5	niño
3	3	COMO	16-Jun-2001	01-Jul-2001	23-Jun-2001	Estándar	44	3.15	niña
4	4	VIMU	18-Jun-2001	23-Jun-2001	17-Dec-2001	Intensivo	25	2.74	niño
5	5	PAVI	19-Jun-2001	25-Jun-2001	26-Jun-2001	Estándar	27	3.6	niña
6	6	PASA	20-Jun-2001	01-Jul-2001	27-Jun-2001	Estándar	36	2.65	niña
7	7	VERI	20-Jun-2001	30-Jun-2001	12-Sep-2001	Intensivo	35	2.97	niño
8	8	ADJU	21-Jun-2001	25-Jun-2001	13-Sep-2001	Intensivo	23	3.2	niño
9	9	BEMI	22-Jun-2001	05-Jul-2001	31-Aug-2001	Estándar	40	2.4	niña
10	10	JUNA	23-Jun-2001	02-Jul-2001	29-Sep-2001	Intensivo	32	2.1	niña
11	11	LOKO	26-Jun-2001	02-Jul-2001	21-Aug-2001	Estándar	26	3.45	niño
12	12	FRFU	27-Jun-2001	04-Jul-2001	06-Mar-2002	Intensivo	29	3.45	niña
13	13	FUFE	06-Jul-2001	17-Jul-2001	13-Jul-2001	Estándar	36	3.4	niña
14	14	POCA	13-Jul-2001	24-Jul-2001	09-Nov-2001	Intensivo	36	3.05	niño
15	15	LOLO	13-Jul-2001	14-Jul-2001	20-Jul-2001	Estándar	17	3.6	niño
16	16	BOPE	14-Jul-2001	27-Jul-2001	19-Jan-2002	Estándar	40	3.4	niño
17	17	ANZO	18-Jul-2001	24-Jul-2001	05-Dec-2001	Intensivo	27	3.15	niño
18	18	MEVE	18-Jul-2001	27-Jul-2001	27-Mar-2002	Intensivo	32	3.32	niña
19	19	TOPO	19-Jul-2001	26-Jul-2001	11-Oct-2001	Estándar	29	2.65	niña
20	20	PUPI	20-Jul-2001	23-Jul-2001	12-Oct-2001	Intensivo	21	4.46	niña
21	21	ROPA	20-Jul-2001	30-Jul-2001	17-Aug-2001	Estándar	35	3.15	niña
22	22	LOMA	21-Jul-2001	27-Jul-2001	02-Mar-2002	Intensivo	27	3.7	niño
23	23	CEMA	22-Jul-2001	27-Jul-2001	12-Aug-2001	Estándar	24	3.79	niño

Con la función `table2` se crea una tabla de frecuencias al igual que con la función `table`, pero con la diferencia que aparecen las etiquetas de valor de la/s variable/s si existen, y imprime por pantalla las etiquetas y el número de missings. Además también es posible dar el porcentaje con el argumento `margin`.

```
> source(file.path(RutinasLocales, "table2.r"))
> table(datos$tx)
```

```
 1  2
13 15
```

```
> table2(datos$tx)
```

```
variable name: datos$tx
variable label: Regimen visitas asignado
      num. of missings: 0 ( 0.00 %)
```

```
      freq
1: Estándar  13 (46.4%)
2: Intensivo 15 (53.6%)
```

```
> with(datos, table(tx, sexo))
```

```
      sexo
tx  1  2
  1  4  9
  2  8  7
```

```
> with(datos, table2(tx, sexo))
```

```
variable name: tx
variable label: Regimen visitas asignado
      num. of missings: 0 ( 0.00 %)
```

```
variable name: sexo
variable label: sexo de la criatura
      num. of missings: 0 ( 0.00 %)
```

```
Global: num. of missings: 0 ( 0.00 %)
```

```
      sexo
tx      1: niño  2: niña
  1: Estándar  4 (14.3%) 9 (32.1%)
  2: Intensivo 8 (28.6%) 7 (25.0%)
```

```
> with(datos, table2(tx, sexo, margin = 0))
```

```
variable name: tx
variable label: Regimen visitas asignado
      num. of missings: 0 ( 0.00 %)
```

```
variable name: sexo
variable label: sexo de la criatura
      num. of missings: 0 ( 0.00 %)
```

```
Global: num. of missings: 0 ( 0.00 %)
```

	sexo	
tx	1: niño	2: niña
1: Estándar	4	9
2: Intensivo	8	7

Para fusionar dos data.frames, en el sentido de añadir registros, se había usado la función `rbind`, poniendo las variables comunes en los dos data.frames en el mismo orden:

```
> source(file.path(RutinasLocales, "add.cases.r"))
> parto4 <- read.spss4("parto4.sav")
> parto.extra <- read.spss4("parto_extra.sav")
> vars.comunes <- intersect(names(parto4), names(parto.extra))
> fusion <- rbind(parto4[, vars.comunes], parto.extra[, vars.comunes])
> fix2(fusion, print = TRUE)
```

	id	ini	dia_nac	dia_entr	ulti_lac	tx	edad	peso	sexo
1	1	GADI	14-Jun-2001	19-Jun-2001	26-Jul-2001	Intensivo	24	3.38	niña
2	2	CAEL	15-Jun-2001	21-Jun-2001	15-Feb-2002	Intensivo	27	2.5	niño
3	3	COMO	16-Jun-2001	01-Jul-2001	23-Jun-2001	Estándar	44	3.15	niña
4	4	VIMU	18-Jun-2001	23-Jun-2001	17-Dec-2001	Intensivo	25	2.74	niño
5	5	PAVI	19-Jun-2001	25-Jun-2001	26-Jun-2001	Estándar	27	3.6	niña
6	6	PASA	20-Jun-2001	01-Jul-2001	27-Jun-2001	Estándar	36	2.65	niña
7	7	VERI	20-Jun-2001	30-Jun-2001	12-Sep-2001	Intensivo	35	2.97	niño
8	8	ADJU	21-Jun-2001	25-Jun-2001	13-Sep-2001	Intensivo	23	3.2	niño
9	9	BEMI	22-Jun-2001	05-Jul-2001	31-Aug-2001	Estándar	40	2.4	niña
10	10	JUNA	23-Jun-2001	02-Jul-2001	29-Sep-2001	Intensivo	32	2.1	niña
11	11	LOKO	26-Jun-2001	02-Jul-2001	21-Aug-2001	Estándar	26	3.45	niño
12	12	FRFU	27-Jun-2001	04-Jul-2001	06-Mar-2002	Intensivo	29	3.45	niña
13	13	FUFE	06-Jul-2001	17-Jul-2001	13-Jul-2001	Estándar	36	3.4	niña
14	14	POCA	13-Jul-2001	24-Jul-2001	09-Nov-2001	Intensivo	36	3.05	niño
15	15	LOLO	13-Jul-2001	14-Jul-2001	20-Jul-2001	Estándar	17	3.6	niño
16	16	BOPE	14-Jul-2001	27-Jul-2001	19-Jan-2002	Estándar	40	3.4	niño
17	17	ANZO	18-Jul-2001	24-Jul-2001	05-Dec-2001	Intensivo	27	3.15	niño
18	18	MEVE	18-Jul-2001	27-Jul-2001	27-Mar-2002	Intensivo	32	3.32	niña
19	19	TOPO	19-Jul-2001	26-Jul-2001	11-Oct-2001	Estándar	29	2.65	niña
20	20	PUPI	20-Jul-2001	23-Jul-2001	12-Oct-2001	Intensivo	21	4.46	niña
21	21	ROPA	20-Jul-2001	30-Jul-2001	17-Aug-2001	Estándar	35	3.15	niña

Con la función `add.cases` es posible añadir todas la variables de un `data.frame` o de otro, o sólo las variables comunes, con una sola instrucción, sin necesidad de poner las variables en el mismo orden. Por defecto se añaden todas las variables:

```
> fusion <- add.cases(parto4, parto.extra, font = "origen")
```

```
>Advertencia: las siguientes variables quedaran vacías para y  
nacion
```

```
>Advertencia: las siguientes variables quedaran vacías para x  
sem_lac, masde12, sem_12, tx_2
```

```
-----Arreglando la variable ' id '-----
```

```
Se le ha asignado el formato 'SPSS': F3.0
```

```
-----Arreglando la variable ' ini '-----
```

```
Se le ha asignado el formato 'SPSS': A4
```

```
-----Arreglando la variable ' dia_nac '-----
```

```
Se le ha asignado el formato 'SPSS': DATE11
```

```
Antes tenía el formato 'SPSS' DATE11
```

Con el argumento `all=2`, el resultado contiene todas la variables de 'parto4'.

```
> fusion <- add.cases(parto4, parto.extra, all = 2)
```

```
> Advertencia: las siguientes variables quedaran vacías  
nacion
```

```
> names(fusion)
```

```
[1] "id"      "ini"      "dia_nac"  "dia_entr" "ulti_lac" "tx"  
[7] "edad"    "peso"     "sexo"     "tip_par"  "hermanos" "nacion"  
[13] "fuma_an" "fuma_de"  "horas_an" "horas_de"
```

Con el argumento `all=1`, el resultado contiene todas la variables comunes.

```
> fusion <- add.cases(parto4, parto.extra, all = 1)  
> names(fusion)
```

```
[1] "id"      "ini"      "dia_nac"  "dia_entr" "ulti_lac" "tx"  
[7] "edad"    "peso"     "sexo"     "tip_par"  "hermanos" "fuma_an"  
[13] "fuma_de" "horas_an" "horas_de"
```


Con la función `intervals` se visualiza por pantalla las estimaciones de los parámetros, su intervalo de confianza y el p-valor de forma más "bonita", tanto de un modelo de regresión lineal, logística o de Cox. En los dos últimos casos, se muestran los odds ratio y los hazard ratios respectivamente.

Por ejemplo, para mostrar la constante y la pendiente, con su intervalo y su p-valor de una regresión lineal del peso del niño según la edad de la madre:

```
> source(file.path(RutinasLocales, "intervals.r"))
> fit <- with(datos, lm(peso ~ edad))
> intervals(fit)
```

	coef	95%	C.I.	p-value
(Intercept)	4.23 (3.48 -	4.97)	0.000
edad	-0.03 (-0.06 -	-0.01)	0.011

- En el grupo de Epidemiología genética y cardiovascular-IMIM hemos escrito funciones para construir tablas bivariadas que se han juntado en un paquete disponible en CRAN

<http://www.r-project.org/>: `compareGroups` . Aunque la versión que hay en la carpeta 'packages' del material del curso pueda ser más nueva que la que haya disponible en CRAN

- Por ejemplo para construir una tabla con la variable sexo y las siguientes variables de partoFin:

tx,edad,peso,tip_par,hermanos,fuma_an,fuma_de,horas_an,horas_de,naci_ca,masde12,sem_lac

Las instrucciones son muy simples

```
> library(compareGroups)
> datos <- spss.get("C:/cursoR/data/partoFin.sav", TRUE, TRUE)
> ans <- compareGroups(sexo ~ tx + edad + peso + tip.par + hermanos +
+   fuma.an + fuma.de + horas.an + horas.de + naci.ca + masde12 +
+   sem.lac, data = datos, method = NA)
```

- Hay un manual con muchos más detalles en la 'vignette' del package. Además, todas la funciones están documentadas.

```
> createTable(ans, show.all = FALSE)
```

```
-----Summary descriptives table by 'sexo de la criatura'-----
```


	niño N=12	niña N=16	p.overall
Regimen visitas asignado:			0.412
Estándar	4 (33.3%)	9 (56.2%)	
Intensivo	8 (66.7%)	7 (43.8%)	
Edad de la madre	27.7 (6.34)	30.5 (6.98)	0.273
peso del niño	3.25 (0.39)	3.18 (0.57)	0.697
Tipo de parto:			1.000
instrument.	2 (16.7%)	3 (18.8%)	
no instrum.	10 (83.3%)	13 (81.2%)	
Tiene hermanos :			0.783
si	6 (50.0%)	6 (37.5%)	
no	6 (50.0%)	10 (62.5%)	
Fuma antes embarazo:			0.703
No	7 (58.3%)	7 (43.8%)	
Si	5 (41.7%)	9 (56.2%)	
Fuma despues embarazo:			0.698
No	7 (58.3%)	11 (68.8%)	
Si	5 (41.7%)	5 (31.2%)	
Horas ejercicio semanal antes embarazo	6.67 (2.46)	8.00 (3.20)	0.224
Horas ejercicio semanal despues embarazo	4.50 [2.00; 8.25]	6.00 [1.75; 10.2]	0.675
nacionalidad:			1.000
Española	6 (50.0%)	8 (50.0%)	
Otras	3 (25.0%)	4 (25.0%)	
Sudamérica	3 (25.0%)	4 (25.0%)	
Lactancia mas de 12 semanas:			0.295
No	5 (41.7%)	11 (68.8%)	
Si	7 (58.3%)	5 (31.2%)	
Semanas de lactancia	18.5 [11.0; 26.2]	8.00 [1.00; 14.5]	0.084

- En el grupo de Epidemiología genética y cardiovascular-IMIM, hemos creado una función para estimar el riesgo coronario a 10 años según unas características del individuo.
- Se basa en una función estadística de Cox ⁴.
- Por ejemplo, si se quiere calcular la probabilidad de que un hombre de 60 años con un colesterol total de 210, un HDL de 42, con una tensión arterial sistólica de 150 y una diastólica de 90, no diabético y fumador, tenga un evento coronario:

```
> source(file.path(RutinasLocales, "calculadora.riesgo.r"))  
> calculadora.riesgo(sex = 1, age = 60, coltot = 210, hdl = 42,  
+   tas = 150, tad = 90, diab = 1, smoke = 1)
```

```
[1] 0.1683645
```

⁴ Marrugat et al., Estimación del riesgo coronario en España mediante la ecuación de Framingham calibrada, Rev. Esp. Cardio. 2003.

En esta sección se presentan algunas funciones útiles de  que no se han explicado en las secciones anteriores para el manejo de datos, y otras funciones más de manejo de software.

La función rep sirve para crear vectores que contienen elementos repetidos. He aquí algunos ejemplos:

```
> rep(1,10) # repite el 1 10 veces
```

```
[1] 1 1 1 1 1 1 1 1 1 1
```

```
> # repite la letra 'a' 3 veces y la 'b' 5 veces
```

```
> rep(c("a","b"),c(3,5))
```

```
[1] "a" "a" "a" "b" "b" "b" "b" "b"
```

```
> # repite la combinación 'a','b' 2 veces
```

```
> rep(c("a","b"),2)
```

```
[1] "a" "b" "a" "b"
```

```
> # repite cada letra 2 veces.
```

```
> rep(c("a","b"),each=2)
```

```
[1] "a" "a" "b" "b"
```

- Las funciones `grep` y `agrep` son muy útiles para buscar fragmentos de texto en una variable de tipo carácter.
- La función `grep` busca qué elemento de un vector contiene el fragmento de texto especificado, y la función `agrep` es más laxa ya que busca un fragmento parecido.
- Si no se especifica el argumento `value`, retorna la posición del elemento o elementos que ha encontrado.

```
> nombres <- c("Joan Vila", "Juan Vila", "Isaac Subirana", "Jordi Sala")  
> grep("Joan", nombres)
```

```
[1] 1
```

```
> grep("Joan", nombres, value = TRUE)
```

```
[1] "Joan Vila"
```

```
> agrep("Joan", nombres, value = TRUE)
```

```
[1] "Joan Vila" "Juan Vila"
```

Con la función `dir.create` se pueden crear carpetas. Por ejemplo, para crear una nueva carpeta dentro de la carpeta de trabajo:

```
> dir.create("./carpeta_nueva")
```


Con la función `file.copy` se copian ficheros de una carpeta a otra, o con otro nombre. Por ejemplo, para copiar el fichero de SPSS "partoFin.sav" a la nueva carpeta creada.

```
> file.copy("partoFin.sav", "./carpeta_nueva/copia de partoFin.sav")
```

```
[1] TRUE
```



Con la función `file.remove` se eliminan archivos. Por ejemplo, para eliminar el fichero acabado de copiar en la nueva carpeta:

```
> file.remove("./carpeta_nueva/copia de partoFin.sav")
```

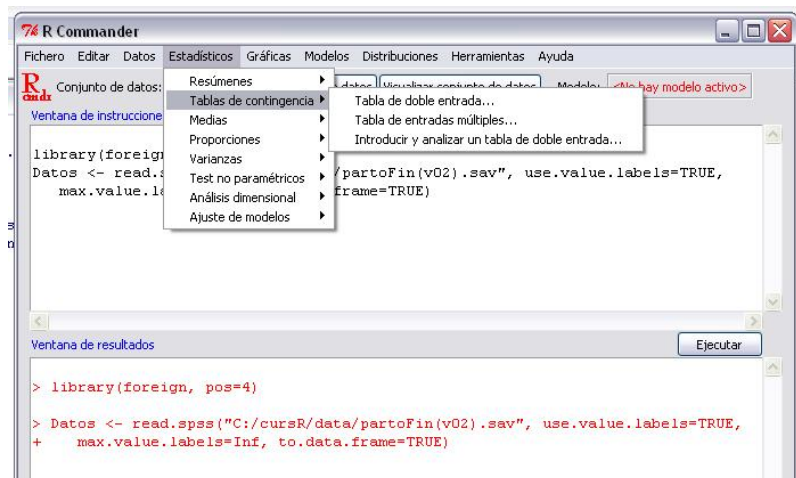
```
[1] TRUE
```



Con la función `shell.exec` se pueden abrir archivos. El programa que abrirá el archivo estará de acuerdo con su extensión. Así, por ejemplo si se abre el archivo "partoFin.sav" se usará el programa SPSS.

```
> shell.exec(file.path(getwd(), "partoFin.sav"))
```

El paquete Rcmdr permite usar el  de forma más interactiva. Es muy útil para empezar a manejar el  de forma didáctica aunque es un poco limitado para hacer análisis (por ejemplo, no permite realizar análisis de supervivencia).

> library(Rcmdr)



El paquete Sweave sirve para crear documentos a partir de análisis y cálculos de : inserta código de  con texto del editor de documentos \LaTeX . Seguidamente se verá un ejemplo ilustrativo, cuyo material se encuentra en la carpeta 'misc' del material del curso. Aunque para ejecutarlo, se debería tener instalado algún compilador de \LaTeX , como por ejemplo Miktex. No obstante, no es objetivo de este curso, sino más bien de un curso introductorio de \LaTeX , que el alumno realice este ejemplo. El material está sólo para la ilustración que se hará en clase. Por supuesto, si en un futuro queréis realizar el ejemplo en vuestro computador, no dudéis en preguntárnoslo.