Apuntes de R

Créditos: Estos apuntes se basan en una traducción de parte de un documento facilitado por Pablo E. Verde, del AMS y otros documentos sobre SPLUS o R de dominio público.

Inicio de R

R es un programa de 32 bits y precisa Windows 95 o superior para funcionar. Existen versiones para Linux, pero no para Windows 3 o DOS.

Existen dos versiones de R para windows. **Rgui.exe** emplea un entorno gráfico con ventanas y **Rterm.exe**, que es más rápido pero no dispone de gráficos y se ejecuta en una ventana de DOS bajo windows. En teste curso usaremos Rgui.exe .

Al arrancar Rgui aparece un entorno con una barra horizontal superior que contiene menús y botones y una ventana de comandos con el símbolo “>” que nos invita a escribir.

Expresiones y Objetos

R se usa escribiendo expresiones en la ventana de comandos tras el símbolo “>”. Una expresión es algo que R es capaz de evaluar.
Ejemplos:

```
> 3 + 7*4
[1] 31
```

esto es un vector de longitud 1, es decir de 1 elemento.

R es un lenguaje que emplea funciones que operan sobre objetos para generar otros objetos. Los argumentos de una función siempre deben incluirse entre paréntesis, y si se desea ejecutar una función sin argumentos, los paréntesis siguen siendo obligatorios, por eso siempre que nos refiramos a una función la nombraremos con los paréntesis.

Por ejemplo, para finalizar la sesión de R se puede emplear la función **q()**:

```
> q()
```

Se pueden combinar varios elementos en un vector mediante la función **c()**

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

Otra función útil para crear vectores es **seq()**, que genera una secuencia de números:

```
seq(1,10)
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(4,10,2)
[1]  4  6  8 10
```

Para generar secuencias simples también se puede usar el operador **:**

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

A pesar de que cada operación que se realiza en R resulta en un objeto, éstos no se guardan a no ser que se les asigne un nombre. Una vez nombrados, un objeto se guarda permanentemente hasta que no se borre explícitamente. El operador de asignación es <-. Por ejemplo, la siguiente expresión crea un vector de 4 elementos y lo asigna al objeto x

```
> x<-c(4,3,2,1)
> x
```

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

Una vez se dispone de un objeto nominado, se puede emplear en expresiones:

```
> x*10
[1] 40 30 20 10
> mean(x)
[1] 2.5
> var(x)
[1] 1.666667
```

Si aparece un mensaje de error, casi siempre se debe a sintaxis incorrecta:

```
> 3 x
Error: syntax error
```

Para ver que objetos se han creado se emplea la función **ls()**. Para borrar objetos se usa **rm()**. Por ejemplo:

```
> ls()
[1] "x"
> rm(x)
```

Sintaxis básica y convenciones

Espacios

R ignora la mayoría de los espacios. Por lo tanto, se pueden incluir tantos espacios como se desee entre los operadores y los números u objetos:

```
3+  9
[1] 12
```

Los espacios son importantes en el medio de un nombre o un número. Por ejemplo:

```
> 21 1+2
Error: syntax error
```

También, los espacios son importantes en el operador de asignación <-:

```
> x < -3
[1] F F F F
> x <- 3
> x
[1] 3
```

Mayúsculas y minúsculas

R diferencia minúsculas y mayúsculas en todos sus objetos, argumentos, nombres, funciones etc. Hay que tener mucho cuidado con los nombres que se emplea.

Continuación en otra línea

Las expresiones en R pueden ser tan largas como se quiera, pero se puede continuar en múltiples líneas si se desea. El símbolo de continuación que aparece para indicar que la expresión está incompleta es “+”:

```
> x*
+ 23
[1] 69
```

Nombres de directorios y el símbolo “\”

El símbolo contrabarra “\” actúa como secuencia de composición de caracters, es decir, indica que el siguiente carácter es especial.Por ejemplo “\n” significa salto de línea. Como en DOS los directorios se separan con contrabarras, éstas deben duplicarse en R. Por ejemplo “c:\\datos\\estudio.dat” resulta en el nombre del fichero de DOS “c:\datos\estudio.dat”.

```
>read.table(“c:\\datos\\estudio.dat”)
```

Ayuda Online

Existen dos sistemas de ayuda en R. Una basada en archivos de R y otra basada en archivos hipertexto html, que se leen con un navegador de internet. Para pedir ayuda sobre un tema específico se puede emplear el menú de ayuda o bien las siguientes instrucciones:

```
> help(rnorm)
> ?rnorm
```

Funciones

Una función es una expresión de R que devuelve un valor tras realizar operaciones sobre uno a más argumentos. Ejemplo:

```
> rnorm(5)
[1] -0.02040495 -1.01193289  0.91630813 -1.38298488 -0.46955270
> rnorm(5,mean=10, sd=1000)
[1]  -793.5893   912.6408 -1145.8699   114.9803   240.2155
```

Para asignar el resultado de una función a un objeto permanente debe emplearse el operador de asignación:

```
x<-rnorm(5,mean=10,sd=1000)
> x
[1]  2405.68112    92.49998   -14.88167   762.56178 -1097.84235
```

Operadores

Un operador es una función que como máximo acepta dos argumentos. Por ejemplo, las operaciones aritméticas típicas se representan con los operadores `+`, `-`, `*`, y `/`. Algunos ejemplos:

```
> 4+4
[1] 8
> 3*50
[1] 150
> (5.7-9)/5
[1] -0.66
> 2^3
[1] 8
```

Los operadores lógicos se emplean para comparaciones y expresiones lógicas. El igual de comparación emplea dos signos igual “==” para diferenciarlo del igual de asignación de argumentos a funciones.

```
== (igual), != (diferente)
>, <, >=, <=
| (o), & (y)
```

Ejemplos de uso:

```
> 24==(6*4)
[1] TRUE
> 7>7
[1] FALSE
> 7>=7
```

```
[1] TRUE
> (7>5)|(6<=10)
[1] TRUE
```

Expresiones

Una expresión es cualquier combinación de funciones, operadores y objetos de datos:

```
>3 * runif(10)
 [1] 2.7368648 2.7806944 0.3488282 1.0469554 1.5058319 2.5762746 1.1441339
0.0203806
 [9] 0.4741212 2.5809784
> 3* runif(5)
[1] 1.4249556 2.7961005 2.3872305 0.7703814 1.9126908
> 5* c(3,4)-1
[1] 14 19
> c(2*runif(5),4*rnorm(5,mean=10,sd=1))
 [1]  1.3105429  1.4122571  0.7469029  0.1724914  0.5172468 42.2367480 50.7732304
 [8] 44.3738271 40.3949496 36.3289406
```

Jerarquía de precedencia

Al igual que la mayoría de lenguajes de programación, las operaciones se evalúan primero en los paréntesis más internos y después los más externos. Por ejemplo:

```
x<-5
> 1:(x-1)
[1] 1 2 3 4
> 1:x-1
[1] 0 1 2 3 4
```

Si no se emplean paréntesis, la precedencia es:

funciones especiales > productos | división > suma | resta

Objetos de datos en R

Existen siete tipos básicos de objetos: *vector, matrix, array, list, factor, time series,* y *data frame*. Todos se crean mediante generalización del más sencillo: el v*ector*.

Nombres de objetos de datos

Los nombres de objetos de datos deben comenzar por una letra y pueden contener combinaciones de letras mayúsculas y minúsculas, números y puntos (.). Ejemplos de nombres válidos:

```
Misdatos, datos.ma.vcv, DatosHoy
```

Tipos de valores de datos

`logical`	Valores lógicos, **TRUE** y **FALSE**, representan datos binarios. “Verdadero” o “falso”, “si” o “no”, “presencia” o “ausencia” se pueden representar por valores lógicos. Se `pueden usar de manera indistinta` **T** o TRUE y **F** o FALSE.
`numeric`	Representan números reales. En R los decimales sólo se pueden representar con punto (no coma). Se pueden expresar de las siguientes formas: • Números *decimales* ordinales como `11`, `-2.3`, ó `14.948` • Expresiones de R que generam valores reales como `pi`, `exp(1)`, o `39/4`.

	• En notación científica (forma exponencial), que representa números como potencias de 10. Por ejemplo, `100` se representa como `1e2` en notación científica y `0.002` es `2e-3`.
	• El valor **`Inf`**, que representa el infinito, se puede asignar a objetos o puede ser el resultado de operaciones como dividir por cero:
	
	`>c(5/0, -2.1/0)`
	`[1] Inf -Inf`

`complex`	Números complejos, similar a los numéricos, excepto por la componente imaginaria. Se especifican como `a+bi`, donde a es la parte real y b es la imaginaria. Ejemplos: `2-3i` y `4.2+5.4i.`
`character`	Cualquier texto incluido entre comillas (" ") es un valor de tipo carácter. Por ejemplo: "Estudio 1", "Hospital Comarcal"

`NA`	`NA` es el código para valores perdidos en R para datos logical, numeric y complex. Son las inciales del inglés "Not Available". Puede emplearse como un valor más es operaciones:
	
	`> c(5,NA,3.9)`
	`[1] 5.0 NA 3.9`
	
	`NA` también representa "No numérico" y es el resultado de operaciones no determinadas como `0/0`.
	No hay código de valor perdido para datos de tipo carácter, pero suele emplearse una cadena vacía `""`.

`NULL`	`NULL` representa el valor nulo, y es útil para que una función no devuelva ningún valor o para establecer que un argumento no se pasa a una función. Por ejemplo, si se piden los nombre de los valores de un vector, y éstos no existen, se a devuelve `NULL`:
	
	`> names(1:4)`
	`NULL`

Conversión de tipos

Algunos tipos de objetos sólo permiten datos del mismo tipo, y si se intentan combinar se convierten al tipo más general. Por ejemplo:

```
> c(T,F,F)
[1]  TRUE FALSE FALSE
> c(T,F,F,1,6)
[1] 1 0 0 1 6
> c(T,F,F,1,6,6+2i)
[1] 1+0i 0+0i 0+0i 1+0i 6+0i 6+2i
> c(T,F,F,1,6,6+2i,"Azul")
[1] "TRUE"  "FALSE" "FALSE" "1"     "6"     "6+2i"  "Azul"
```

Las mismas conversiones ocurren en cálculos:

```
T+1
[1] 2
> 3*T+1
[1] 4
```

Vectores

Funciones útiles para crear vectores

Función	Descripción	Ejemplos
`c`	Combina valores	c(1,2,3), c(“si”, “no”)
`rep`	Repite valores	rep(NA,5), rep(c(1,2,3),3)
`:`	Secuencias numéricas	1:5 , –2:8
`seq`	Secuencias numéricas	seq(-pi,pi,.5)
`vector`	Inicia vectores a cero	vector(“complex”,5) vector("double",9)
`logical`	Inicia vectores lógicos	logical(3)
`numeric`	Inicia vectores numéricos	numeric(4)
`complex`	Inicia vectores complejos	complex(3)
`character`	Inicia vectores carácter	character(5)

Atributos de los vectores

Atributo	Descripción
“length”	Número de valores
“mode”	Tipo de valores
“names”	Etiquetas (nombres)

```
> numb.leters<-letters
> length(numb.leters)
[1] 26
> mode(numb.leters)
[1] "character"
> numb.leters[1:5]
[1] "a" "b" "c" "d" "e"
> names(numb.leters[1:5])
NULL
> numb.leters
 [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s"
"t"
[21] "u" "v" "w" "x" "y" "z"
> names(numb.leters)<-1:26
> numb.leters
   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
21
 "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l" "m" "n" "o" "p" "q" "r" "s" "t"
"u"
  22 23 24 25 26
 "v" "w" "x" "y" "z"
>
```

Matrices

Para crear una matriz a partir de un vector existente se puede emplear la función **dim()**, que asigna el número de filas y de columnas. Por ejemplo:

```
> mat<-rep(1:4,rep(3,4))
> mat
 [1] 1 1 1 2 2 2 3 3 3 4 4 4
> dim(mat)<- c(3,4)
> mat
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    1    2    3    4
[3,]    1    2    3    4
```

Para combinar vectores (y matrices) para formar matrices se pueden emplear las funciones **cbind()** y **rbind()**. La función cbind() une columnas y la rbind() une filas:

```
rbind(c(234,234,123,34),c(23,344,112,12))
     [,1] [,2] [,3] [,4]
[1,]  234  234  123   34
[2,]   23  344  112   12

cbind(c(234,234,123,34),c(23,344,112,12))
     [,1] [,2]
[1,]  234   23
[2,]  234  344
[3,]  123  112
[4,]   34   12
```

También se puede usar la función `matrix() para crear matrices:`

```
> matrix(1:12,ncol=3,byrow=T)
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
[3,]    7    8    9
[4,]   10   11   12

> matrix(1:12,ncol=3)
     [,1] [,2] [,3]
[1,]    1    5    9
[2,]    2    6   10
[3,]    3    7   11
[4,]    4    8   12
```

Atributos de las matrices

Atributo	Descripción
"length"	Número de valores
"mode"	Tipo de valores
"dim"	Número de filas y columnas
"dimnames"	Nombres de filas y columnas

Para comprobar la dimensión de una matriz también se emplea la función **dim()**:

```
mat
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    1    2    3    4
[3,]    1    2    3    4
> dim(mat)
[1] 3 4
```

A las matrices se les puede etiquetar las filas y las columnas con la función **dimnames()**. Los nombres deben suministrarse como una lista:

```
dimnames(mat)<-list(paste("row",letters[1:3]),paste("col",LETTERS[1:4]))
> mat
      col A col B col C col D
row a     1     2     3     4
row b     1     2     3     4
row c     1     2     3     4
```

Para suprimir las etiquetas, se asigna el valor NULL al elemento corespondiente de la lista:

```
dimnames(mat)<-list(NULL,paste("col",LETTERS[1:4]))
> mat
     col A col B col C col D
[1,]     1     2     3     4
[2,]     1     2     3     4
[3,]     1     2     3     4
```

Para extraer elementos de una matriz se emplean índices entre corchetes con las posiciones de filas y columnas:

```
mat
     col A col B col C col D
[1,]     1     2     3     4
[2,]     1     2     3     4
[3,]     1     2     3     4
> mat[1,2]
 col B
     2
> mat[c(1,3),c(3,4)]
     col C col D
[1,]     3     4
[2,]     3     4

> mat[-c(1,3),c(3,4)]
 col C col D
     3     4

> mat[-c(1,3),mat[3,]>2]
 col C col D
     3     4

> mat[-c(1,3),mat[3,]= =2]
 col B
     2
> mat[-c(1,3),mat[3,]!=2]
 col A col C col D
     1     3     4

> mat[,c("col A", "col D")]
     col A col D
[1,]     1     4
[2,]     1     4
[3,]     1     4

> mat[1,]
 col A col B col C col D
     1     2     3     4
```

La misma técnica sirve para extraer valores de otros tipos de objeto.

Arrays

Los Arrays generalizan las matrices extendiendo el atributo dim a más de 2 dimensiones. Se crean con la función array(), similar a la función matrix(). Si no se suministran datos, se crea el array lleno de valores NA. Cuando se pasan valores para crear un array, la primera dimensión cambia la más rápida, la segunda después y así sucesivamente.

```
myarray<-array(c(1:8,11:18,111:118),dim=c(2,4,3))
> myarray

, , 1
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
```

```
[2,]    2    4    6    8

, , 2
     [,1] [,2] [,3] [,4]
[1,]   11   13   15   17
[2,]   12   14   16   18

, , 3
     [,1] [,2] [,3] [,4]
[1,]  111  113  115  117
[2,]  112  114  116  118
```

Atributos de los arrays

Atributo	Descripción
"length"	Número de valores
"mode"	Tipo de valores
"dim"	Tamaño de cada dimensión
"dimnames"	Nombres de cada dimensión

Listas

Hasta ahora, todos los objetos descritos sólo permitían elementos del mismo tipo (mode). El objeto list permite combinar elementos de diferente tipo, y que cada elemento de la lista conserve su tipo original, sin conversiones. Las listas son un tipo de objetos muy general. Se componen de elementos que pueden ser de diverso tipo. Muchas funciones de R devuelven listas.

Para crear una lista se emplea la función **list().** Cada argumento es un elemento de la lista. Para nombrar los elementos se emplea la forma *nombre=componente*:

```
mylist<-list(NUMbers=1:10, COLORS=c("black","brown","green"),
sublist=list(logicalval=logical(4),Description="list data example"))
> mylist
$NUMbers:
 [1]  1  2  3  4  5  6  7  8  9 10

$COLORS:
[1] "black" "brown" "green"

$sublist:
$sublist$logicalval:
[1] F F F F

$sublist$Description:
[1] "list data example"
```

Para acceder a un elemento de una lista se puede emplear su nombre precedido de un $:

```
> mylist$COLORS
[1] "black" "brown" "green"
> mylist$sublist$Description
[1] "list data example"
```

De manera más general (y la única posible si los componentes no tienen nombre) , se pueden emplear índices entre doble corchetes [[]]. Para referirse a los elementos dentro de un componente se emplean corchetes simples:

```
> mylist$COLOR
[1] "black" "brown" "green"
> mylist[[2]]
[1] "black" "brown" "green"
> mylist[[2]][c(2,3)]
```

```
[1] "brown" "green"
> mylist[[3]]
$logicalval:
[1] F F F F

$Description:
[1] "list data example"


> mylist[[3]][1]
$logicalval:
[1] F F F F
>
```

Atributos de una lista

Atributo	Descripción
"length"	Número de componentes
"mode"	List
"names"	Nombres de cada componente

```
length(mylist)
[1] 3
> mode(mylist)
[1] "list"
> names(mylist)
[1] "NUMbers" "COLORS"  "sublist"
> names(mylist)<-c("NUM","COL","SList")
> names(mylist)
[1] "NUM"   "COL"   "SList"
>
```

Los resultados de ajustar modelos a unos datos devuelven una lista con muchos componentes que contienen información de interés.

```
> y<-rnorm(5)
> x<-runif(5)
> modelo<-lm(y~x)
> mode(modelo)
[1] "list"
> names(modelo)
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "xlevels"       "call"          "terms"         "model"
```

Factores y Factores ordenados

R permite tratar de manera especial variables de naturaleza categórica en el análisis de datos. Los datos categóricos se deben definir como un tipo especial llamado *factor*.

Para crear un factor, se usa la función **`factor()`**. Esta función toma los diferentes valores y los asigna un código interno. Se puede especificar el número de niveles y el orden que interesa que tengan los niveles. Los códigos internos dependen se asignarán por orden a cada nivel especificado. Si no se especifican se convierten los valores a tipo carácter y se ordenan alfabéticamente. La función factor() crea un objeto que pertenece a la clase "factor", lo cuál es útil para verificar el tipo de objeto.

```
> vola.intens <-factor( c("Hi","Med","Lo","Hi","Hi"),levels=c("Lo","Me","Hi"))
> vola.intens
[1] Hi NA Lo Hi Hi
Levels: Lo Me Hi
```

Para crear un factor donde el orden es fundamental, se puede emplear la función **`ordered()`**, que crea un factor ordenado.

```
> vola.intens <-ordered( c("Hi","Med","Lo","Hi","Hi"),levels=c("Lo","Me","Hi"))
> vola.intens
 [1] Hi NA Lo Hi Hi
Levels:  Lo < Me < Hi
```

Para crear una variable categórica a partir de una numérica se puede emplear la función **cut()**. El argumento breaks acepta una lista con los líimtes de los grupos, incluyendo el mínimo y el máximo.

```
> x<-seq(1,12)
> x
 [1]  1  2  3  4  5  6  7  8  9 10 11 12

> cut(x,breaks=c(0,3,6,9,12),labels=c("q1","q2","q3","q4"))
 [1] q1 q1 q1 q2 q2 q2 q3 q3 q3 q4 q4 q4
Levels:  q1 q2 q3 q4
```

Por defecto los grupos no incluyen el mínimo y sí el máximo, por eso se empleado como mínimo el 0:

```
> cut(x,breaks=c(1,3,6,9,12),labels=c("q1","q2","q3","q4"))
 [1] NA q1 q1 q2 q2 q2 q3 q3 q3 q4 q4 q4
Levels:  q1 q2 q3 q4
>
```

Si se desea un número concreto de grupos de aproximadamente igual tamaño se puede pasar el número como argumento break:

```
> cut(x,breaks=4,labels=c("q1","q2","q3","q4"))
 [1] q1 q1 q1 q2 q2 q2 q3 q3 q3 q4 q4 q4
Levels:  q1 q2 q3 q4
>
```

Atributos de un factor

Atributo	Descripción
"length"	Número de valores
"mode"	"numeric"
"names"	Etiquetas de valores
"levels"	Niveles del factor: valores diferentes posibles.
"class"	"factor"

Tablas de datos: Data Frames

Las data frames son unos objetos diseñados fundamentalmente para análisis de datos mediante modelos. Se trata de una generalización de las matrices, de manera que pueden contener datos de diferentes tipos en columnas. Corresponde a la matriz de datos donde cada columna es una variable y cada fila una observación. En realidad son una lista, en la que cada variable es un elemento, pero todos deben tener la misma longitud y se representan como columnas.

Para crear una data frame se pueden emplear varias funciones:

- `read.table()` Lee datos de un archivo externo.
- `as.data.frame()` Convierte una matriz en una data frame.
- data.frame() Une varios vectores del mismo tamaño en una data frame

Se pueden combinar data frames existentes de diferentes maneras mediante las funciones `cbind()`, `rbind()` y `merge()`

```
> mi.logical<-sample(c(T,F),size=10,replace=T)
> mi.factor<-factor( sample(c("A","B"),size=10,replace=T) )
> mi.numeric<-rpois(10,7)
> x<-seq(1,10,1)
```

```
> w<-1+x/2
> y<-x+w*rnorm(x)
> mi.df<-data.frame(mi.logical,mi.factor,mi.numeric,x,y,w)
> mi.df
   mi.logical mi.factor mi.numeric  x         y   w
1       FALSE         B          5  1 2.4119674 1.5
2       FALSE         A         11  2 0.6519663 2.0
3        TRUE         B         11  3 1.2367506 2.5
4       FALSE         A          6  4 4.8522928 3.0
5        TRUE         B          7  5 4.1885153 3.5
6       FALSE         A          3  6 7.6435818 4.0
7       FALSE         B          7  7 7.5932973 4.5
8       FALSE         B          8  8 5.5866054 5.0
9        TRUE         A          7  9 2.8659266 5.5
10      FALSE         A          9 10 5.2355681 6.0
>
```

Lectura de datos externos

El archivo de datos debe tener cada variable en columnas separadas entre sí por al menos un espacio. No puede haber valores vacíos en ninguna columna. Si hay missings debe existir un código alfanumérico especial. El código por defecto es el valor NA.

Para leer los datos se emplea la función **read.table()**:

```
> datos<- read.table("a:\\datos.dat")
```

Si el archivo de datos tiene los nombres de las variables en la primera fila, éstos pueden ser leídos para nombrar las columnas de la data frame. Deben ser nombres válidos de R separados entre sí por al menos un espacio. Para que R los lea se especificará el argumento "**header=T**":

```
> datos<- read.table("a:\\datos.dat", header=T)
```

Los nombres de las variables se pueden suministrar también como argumento mediante **col.names=**

```
> datos<- read.table("a:\\datos.dat",
+                    col.names=c("caso","edad","sexo","peso","talla"))

> datos
   caso edad sexo peso talla
1     1   83 Dona  112   179
2     2   46 Home   67   157
3     3   49 Dona   85   138
4     4   74 Home  122   156
5     5   78 Home   75   173
6     6   69 Home   79   193
7     7   59 Dona   84   160
8     8   79 Dona   40   157
9     9   30 Home   47   158
10   10   73 Home   65   161
```

Una vez leídos los datos, para ver los nombres de las variables se puede usar la función **names()**. Esta misma función permite asignar (o cambiar) los nombres:

```
> names(datos)
[1] "caso","edad","sexo","peso","talla"

> names(datos)<- c("Caso","Edad","Sexo","Peso","Talla")
> names(datos)
[1] "Caso","Edad","Sexo","Peso","Talla"
```

Para seleccionar una variable concreta podemos usar su nombre con la sintasix “data.frame$variable”. Por ejemplo para seleccionar la variable Sexo de datos se usa:

```
> datos$Sexo
 [1] Dona Home Dona Home Home Home Dona Dona Home Home
Levels:  Dona Home
```

Para extraer elementos de una data frame se emplea la misma estrategia que con una matriz. Hay que indicar las filas y columnas de interés. Para obtener la submatriz de las mujeres:

```
> datos[datos$Sexo=="Dona",]
  Caso Edad Sexo Peso Talla
1    1   83 Dona  112   179
3    3   49 Dona   85   138
7    7   59 Dona   84   160
8    8   79 Dona   40   157
```

Para obtener las variables peso y talla del caso 7:

```
> datos[datos$Caso==7,c("Peso","Talla")]
  Peso Talla
7   84   160
```

Algunas funciones útiles para manipular datos

1. Funciones matemáticas comunes: `abs, sign, log, log10, sqrt, exp, sin, cos, tan, acos, asin, atan, cosh, sinh, tanh`

   Estas funciones son vectoriales, es decir, operan elemento a elemento sobre un vector. Si se combian vectores con diferentes tamaños, los más pequeños se “reciclan”.

```
> x<-1:5
> x
[1] 1 2 3 4 5
> y<-10:19
> y
[1] 10 11 12 13 14 15
> y
 [1] 10 11 12 13 14 15 16 17 18 19
> log(x)
[1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379
> log(x)/y
 [1] 0.00000000 0.06301338 0.09155102 0.10663803 0.11495985 0.00000000
 [7] 0.04332170 0.06462425 0.07701635 0.08470726
>
> z<-1:3
> log(x)/z
[1] 0.0000000 0.3465736 0.3662041 1.3862944 0.8047190
Warning message:
longer object length
        is not a multiple of shorter object length in: log(x)/z
```

2. Las funciones `sum` y `prod` calculan la suma y el producto de los elementos de un vector. También existen las versiones acumuladas `cumsum` y `cumprod`.

```
> x<-1:5
> sum(x)
[1] 15
> cumsum(x)
[1]  1  3  6 10 15
> prod(x)
```

```
[1] 120
> cumprod(x)
[1]   1   2   6  24 120
```

3. Las funciones `max(x)` y `min(x)` seleccionan el mayor y menor valor de los elementos del vector x, respectivamente.

```
> min(datos$Talla)
 [1] 138
```

4. La función `range(x)` devuelve `c(min(x), max(x))`.

```
> range(datos$Talla)
[1] 138 193
```

5. Hay múltiples funciones estadísticas. Las más sencillas son `mean(x)` que calcula la media de la muestra: `sum(x)/length(x)`, y `var(x)` que calcula la varianza muestral: `sum((x-mean(x))^2)/(length(x)-1)`. Si x es una matriz nxp, var devuelve la matriz de covarianza pxp.

```
> mean(datos$Peso)
[1] 77.6
> sqrt(var(datos$Peso))
[1] 25.56126

> var(datos[,c("Peso","Talla")])
           Peso     Talla
Peso  653.37778  57.64444
Talla  57.64444 226.62222
>
```

6. Para redondear el número de decimales hay una serie de funciones útiles:

```
ceiling(x)              #menor entero no menor que x
floor(x)                #mayor entero no mayor que x
round(x, digits = 0)    #redondea a digits
signif(x, digits = 6)   #selecciona un digits cifras significativas
trunc(x)                #trunca a la parte entera
```

7. Buscar duplicados y quedarse con casos sin duplicar. La función **duplicated()** devuelve T o F según el valor del vector aparezca más de una vez. La función **unique()** extrae los valores no repetidos de un vector.

8. Buscar valores perdidos (missings). La función **is.na()** devuelve T si un valor tiene asignado el código NA. Para cambiar los valores NA por otro código se puede emplear:

```
> x[is.na(x)]<-999
```

9. Conversión de tipo de objeto. Las siguientes funciones fuerzan, siempre que sea posible la conversión de un objeto al tipo deseado:

```
> z <- as.numeric(x)
> z <- as.factor(x)
> z <- as.ordered(x)
> z <- as.character(x)
> z <- as.data.frame(x)
```

Ordenar vectores y data frames

La función `sort(x)` devuelve un vector del mismo tamaño que x con los valores ordenados en orden creciente. Para ordenar también es útil la función order(x), que devuelve los valores de las posiciones de los elementos de x en orden creciente. sort(x) equivale a x[order(x)].

```
> x<-sample(1:10)
> x
 [1]  2  6  7  3 10  5  1  9  4  8
> sort(x)
 [1]  1  2  3  4  5  6  7  8  9 10
> order(x)
 [1]  7  1  4  9  6  2  3 10  8  5
> x[order(x)]
 [1]  1  2  3  4  5  6  7  8  9 10
>
```

Este último método es muy útil si se quiere ordenar una data.frame. El primer elemento de los indices son las filas, y el segundo las columnas. Para indicar que queremos ver la data frame ordenada según valores crecientes de la variable var podemos emplear: data[order(data$var),]. Por ejemplo, para ordenar datos por Peso:

```
> datos[order(datos$Peso),]
   Caso Edad Sexo Peso Talla
8     8   79 Dona   40   157
9     9   30 Home   47   158
10   10   73 Home   65   161
2     2   46 Home   67   157
5     5   78 Home   75   173
6     6   69 Home   79   193
7     7   59 Dona   84   160
3     3   49 Dona   85   138
1     1   83 Dona  112   179
4     4   74 Home  122   156
>
```

Para ordenar en sentido descendente se puede usar la función rev(), que cambia el orden:

```
> datos[rev(order(datos$Peso)),]
```

Usar individualmente las variables de una data frame

Se puede evitar la necesidad de tener que escribir el nombre de la data frame repetidamente delante del nombre de la variable. Para ello se emplea la función **attach()** que fija la data frame:

```
> attach(datos)
> Sexo
 [1] Dona Home Dona Home Home Home Dona Dona Home Home
Levels:  Dona Home
> Edad
 [1] 83 46 49 74 78 69 59 79 30 73
```

Para liberar la data frame se usa la función **detach()**:

```
> detach()
> Edad
Error: Object "Edad" not found
> datos$Edad
 [1] 83 46 49 74 78 69 59 79 30 73
>
```

Para ver que data frame tenemos attachada podemos usar la función **search()**:

```
> search()
[1] ".GlobalEnv"   "datos"        "Autoloads"    "package:base"
>
```

Cambiar datos en una data frame

Se pueden realizar cambios en los datos de una data frame o crear una copia de la columna que queramos cambiar. Por ejemplo, queremos cambiar la escala de la variable Talla de cm a m:

```
> Talla
 [1] 179 157 138 156 173 193 160 157 158 161
> Talla<-Talla/100
> Talla
 [1] 1.79 1.57 1.38 1.56 1.73 1.93 1.60 1.57 1.58 1.61
```

El nuevo objeto Talla es una copia modificada del de datos, pero en datos tenemos el original:

```
> datos$Talla
 [1] 179 157 138 156 173 193 160 157 158 161
> ls()
[1] "Talla" "datos"
```

Para hacer un cambio permanente en la data frame hay que asignarlo específicamente:

```
> rm(Talla) # para eliminar el objeto Talla externo
> ls()
[1] "datos"
> datos$Talla<-Talla/100
> datos
   Caso Edad Sexo Peso Talla
1     1   83 Dona  112  1.79
2     2   46 Home   67  1.57
3     3   49 Dona   85  1.38
4     4   74 Home  122  1.56
5     5   78 Home   75  1.73
6     6   69 Home   79  1.93
7     7   59 Dona   84  1.60
8     8   79 Dona   40  1.57
9     9   30 Home   47  1.58
10   10   73 Home   65  1.61
>
```

Después de hacer cambios en una data frame fijada es mejor liberarla y volverla a fijar de nuevo.

```
> detach()
> attach(datos)
```

Si deseamos añadir una columna a una data frame sólo hace falta asignar un nombre nuevo:

```
> datos$imc<-Peso/(Talla^2)
> datos
   Caso Edad Sexo Peso Talla      imc
1     1   83 Dona  112  1.79 34.95521
2     2   46 Home   67  1.57 27.18163
3     3   49 Dona   85  1.38 44.63348
4     4   74 Home  122  1.56 50.13149
5     5   78 Home   75  1.73 25.05931
6     6   69 Home   79  1.93 21.20862
7     7   59 Dona   84  1.60 32.81250
8     8   79 Dona   40  1.57 16.22784
9     9   30 Home   47  1.58 18.82711
10   10   73 Home   65  1.61 25.07619
> detach()
> attach(datos)
```

Aunque no siempre es necesario:

```
> joves<-factor(Edad<65,levels=c(F,T),labels=c("vell","jove"))
```

```
> joves
 [1] vell jove jove vell vell vell jove vell jove vell
Levels:  vell jove
> cbind(datos,joves)
   Caso Edad Sexo Peso Talla      imc joves
1     1   83 Dona  112  1.79 34.95521  vell
2     2   46 Home   67  1.57 27.18163  jove
3     3   49 Dona   85  1.38 44.63348  jove
4     4   74 Home  122  1.56 50.13149  vell
5     5   78 Home   75  1.73 25.05931  vell
6     6   69 Home   79  1.93 21.20862  vell
7     7   59 Dona   84  1.60 32.81250  jove
8     8   79 Dona   40  1.57 16.22784  vell
9     9   30 Home   47  1.58 18.82711  jove
10   10   73 Home   65  1.61 25.07619  vell
```

Operaciones con caracteres

Ejemplos:

```
> paste(c("X","Y"),1:5)
[1] "X 1" "Y 2" "X 3" "Y 4" "X 5"
> paste(c("X","Y"),1:5,sep="")
[1] "X1" "Y2" "X3" "Y4" "X5"
> paste(c("X","Y"),1:5,sep="",collapse=" + ")
[1] "X1 + Y2 + X3 + Y4 + X5"

substring("Pablo E Verde",6,7)
[1] " E"
```

Operaciones elementales con vectores y matrices

```
> x<-seq(1:20)
> x<-matrix(x,4)
> x
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
>
```

Matriz transpuesta

```
t(x)
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    9   10   11   12
[4,]   13   14   15   16
[5,]   17   18   19   20
```

escalar x Matriz

```
> 4*x
     [,1] [,2] [,3] [,4] [,5]
[1,]    4   20   36   52   68
[2,]    8   24   40   56   72
[3,]   12   28   44   60   76
[4,]   16   32   48   64   80
```

Multiplicación matricial

```
> x %*% y
     [,1]
[1,]   45
[2,]   50
[3,]   55
[4,]   60
> x %*% t(x)
     [,1] [,2] [,3] [,4]
[1,]  565  610  655  700
[2,]  610  660  710  760
[3,]  655  710  765  820
[4,]  700  760  820  880
```

Valores propios

```
> eigen(x %*% t(x))$values
[1]  2.864414e+003  5.585784e+000  8.714195e-014 -3.315622e-013
> max(eigen(x %*% t(x))$values)
[1] 2864.414
> min(eigen(x %*% t(x))$values)
[1] -3.315622e-013
```

Matriz diagonal

```
> diag(c(1,2))
     [,1] [,2]
[1,]    1    0
[2,]    0    2
```

Matriz inversa

```
> d_diag(c(1,2))
> solve(d)
     [,1] [,2]
[1,]    1  0.0
[2,]    0  0.5
```

Exploración de datos

Tabulación de frecuencias

Función: **table()** calcula frecuencias observadas para valor diferente del vector x o la combinación de vectores x,y si se emplean varios argumentos:

```
> x<-sample(c("Home","Dona"),size=50,replace=T)
> y<-sample(c("Jove","Vell"),size=50,replace=T)
> table(x)
Dona Home
  27   23
> table(y)
Jove Vell
  20   30
> table(x,y)
      Jove Vell
Dona   12   15
Home    8   15
```

Resumen estadístico

Para variables cuantitativas ya se han descrito las funciones**: mean(), median(), var(), min(), max(), range(), quantile(), cor()**. La función **summary()** presenta un resumen:

```
> edad<-rnorm(50,mean=55,sd=10)
> summary(edad)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  30.41   46.32   53.16   53.86   61.99   81.07
```

Para variables categóricas, si se han definido como factores, **summary()** calcula las frecuencias.

```
> summary(as.factor(x))
Dona Home
  27   23
```

Para obtener resúmenes de una variable cuantitativa en función de las categorías de un factor se emplea la función **tapply()**. Los argumentos son tapply(variable, factor, función resumen), por ejemplo:

```
> tapply(edad,x,mean)
    Dona     Home
54.61110 52.97955
>
> tapply(edad,x,summary)
$Dona
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  35.23   46.42   55.68   54.61   61.83   81.07

$Home
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  30.41   46.66   51.29   52.98   62.67   72.68
```

La función resumen se aplica a cada valor diferente del factor

```
> tapply(edad,paste(x,y),mean)
Dona Jove Dona Vell Home Jove Home Vell
 46.17576  63.69531  46.42526  65.26884
```

Podemos crear nuestras funciones resumen personalizadas. Más adelante se verá cómo programar en R:

```
> tapply(edad,x,function(o){cbind(media=mean(o),d.e.=sqrt(var(o)),n=length(o))})

$Dona
       media     d.e.  n
[1,] 54.6111 10.67709 27

$Home
        media     d.e.  n
[1,] 52.97955 11.23648 23
```

Gráficos

Histogramas de frecuencias: **hist()** Argumentos de interés: **breaks=** para definir el número de puntos de corte (define breaks+1 grupos) y **probability=T** para estandarizar el área a 1.

```
> hist(edad,breaks=3)
```

La función general de gráficos es **plot(),** que espera como primeros argumentos los vectores con coordenadas x e y. Hay muchas opciones para elaborar las gráficas a gusto del usuario. Aquí mencionaremos algunas. Por defecto plot() genera un diagrama de dispersión de puntos:

```
> plot(x,y)
```

Para una gráfica de líneas: **type='l'** y para que aparezcan puntos y líneas **type='o'**. También podemos hacer que aparezcan los ejes sin puntos para posteriormente añadirlos: **type='n'**.

Para añadir series se usan las funciones **lines()** o **points()**.

```
> lines(x2,y2)
> points(x3,y3)
```

Otros parámetros para adaptar el plot:

Límites de los ejes: **xlim=**c(min,max), **ylim=**c(min,max).

Títulos y etiquetas: **main=**"Titulo superior", **sub=**"titulo inferior". Etiquetas de ejes: **xlab=**"eje x", **ylab=**"eje y".

Como argumentos de interés se pueden pasar parámetros de definición del tipo de línea (**lty=**), grosor (**lwd=**) o color (**col=**), con números que codifican las diferentes opciones. Ver **help(par)** para obtener una lista de los parámetros posibles.

Para añadir texto: **text()**. El argumento **pch=** controla el tamaño de la letra.

```
> etiquetas<-c("A","B","C")
> text(x,y,etiquetas, pch=2)
```

Para dibujar una línea recta: **abline()**

```
> abline(a,b)     # a es la ordenada en el origen y b la pendiente
> abline(h=c)     #una línea horizontal de altura c
> abline(v=c)     #una línea vertical en c
```

Para añadir una leyenda en una caja en posición x,y: **legend()**. Como argumentos de interés se pueden pasar parámetros de definición del tipo de línea (**lty=**), grosor (**lwd=**) o color (**col=**)

```
> legend(x,y,textos,lty=tipo.linea,lwd=grosor.linea,col=color.linea)
```

Modelos

El ajuste de modelos emplea funciones cuyo primer argumento es un objeto fórmula que representa el modelo deseado según los nombres de vectores. Las fórmulas en R siguen la sintaxis: respuesta ~ expresión de covariables

Por ejemplo para expresar un modelo de regresión lineal con "y" como respuesta (variable dependiente) y "x" como covariable (variable independiente) se usa la función **lm()**:

```
> modelo.lineal <- lm(y ~ x)
```

Otros ejemplos de fórmulas válidas:

```
peso ~ talla + sexo + region       # modelo de regresión múltiple
peso ~ talla + sexo + talla:sexo  # modelo con interacción equivale a peso ~ talla*sexo
peso ~ sexo - 1                   # modelo sin constante. Genera un coeficiente para cada categoría de sexo
peso ~ poly(talla,3)               # modelo que ajusta un polinomio de grado 3 a talla.
```

Otros argumentos de interés de la función **lm()** son:

data= para indicar una data frame donde buscar los vectores de la fórmula

subset= para indicar una expresión lógica que defina una condición que selecciona las observaciones que deben emplearse para el modelo. Por ejemplo, para seleccionar sólo los hombres puede usarse: subset=(sexo=="hombre"). Los paréntesis son opcionales

weights= para indicar un vector con pesos si se desea un modelo de regresión ponderada

na.action= función que indica que debe hacerse con los valores NA (perdidos). Por defecto es **na.omit**, por lo que se realiza un análisis con casos completos. Si se desea que el análisis acabe con un mensaje de error si hay NAs debe ponerse **na.action=na.fail**

Modelos lineales generalizados

La función para estos modelos es **glm().** Además de los anteriores argumentos, precisa uno más que define la familia del modelo.

family= normal | binomial | poisson | gamma

cada familia basada en una distribución de probabilidad del error tiene una transformación asociada. Otras transformaciones válidas pueden indicarse entre paréntesis:

normal	identity
binomial	logistic
poisson	log
gamma	inverse

Exploración de los resultados

Normalmente el resultado de ajustar un modelo con lm() o glm() se asigna a un objeto que es de clase lm o glm respectivamente. Las siguientes funciones permiten explorar diferentes aspectos del modelo:

print()
summary()
coef()
resid()
fitted()
deviance()
anova()
predict()
plot()

Análisis de la supervivencia

Estimación de la función de supervivencia. Se precisan dos variables, una con el tiempo de seguimiento y otra con el estado del individuo al final de ese tiempo, codificada 0:censura / 1:evento. Para estimar la supervivencia global se emplea la función survfit() que genera un objeto con metodos print(), summary() y plot() especificos:

```
> s0<- survfit(Surv(tiempo,estado)~1)
> s0
> summary(s0)
> plot(s0)
```

Para eliminar los intervalos de confianza del plot se puede usar el argumento **conf.int=F**. La gráfica de la transformación log(-logS) vs logT se obtiene con el argumento **fun='cloglog'**.
Para analizar la supervivencia según categorías de variables:

```
> s.sexo<- survfit(Surv(tiempo,estado)~sexo)
```

La función survdiff() calcula el estadístico logrank:

```
> d.sexo<- survdiff(Surv(tiempo,estado)~sexo)
```

El argumento **rho=1** cambia la ponderación de las observaciones y calcula el test de Peto.

Modelos de Cox

```
> c.sexo<- coxph(Surv(tiempo,estado)~sexo)
> c.sexo
> summary(c.sexo)
```

Para evaluar la proporcionalidad en los riesgos:

```
> z.sexo<-cox.zph(c.sexo)
> z.sexo
> plot(z.sexo, resid=F)
```

Modelos paramétricos

```
> w.sexo<- survregv(tiempo,estado)~sexo,dist='weibull')
> w.sexo
> summary(w.sexo)
```

Los nombres de las distribuciones disponibles son: weibull, exponential, lognormal, gaussian, loglogistic
Hay métodos summary() y anova()