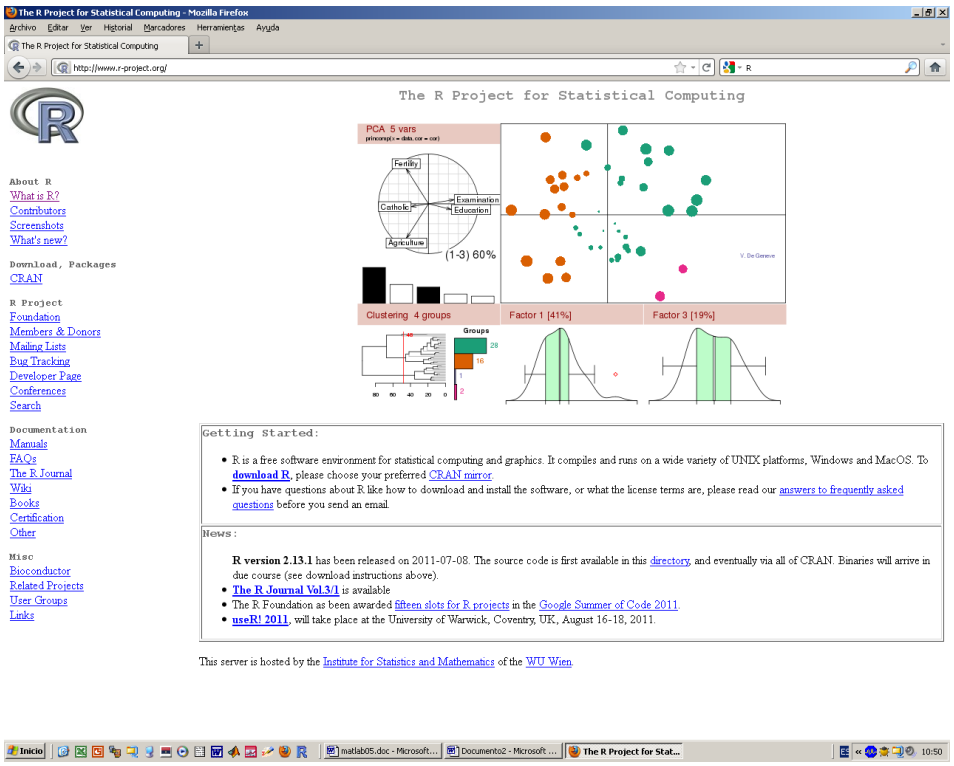
INICIACIÓN AL LEGUAJE DE PROGRAMACION R

Carlos Carleos Artime - Norberto Corral Blanco - Teresa López García

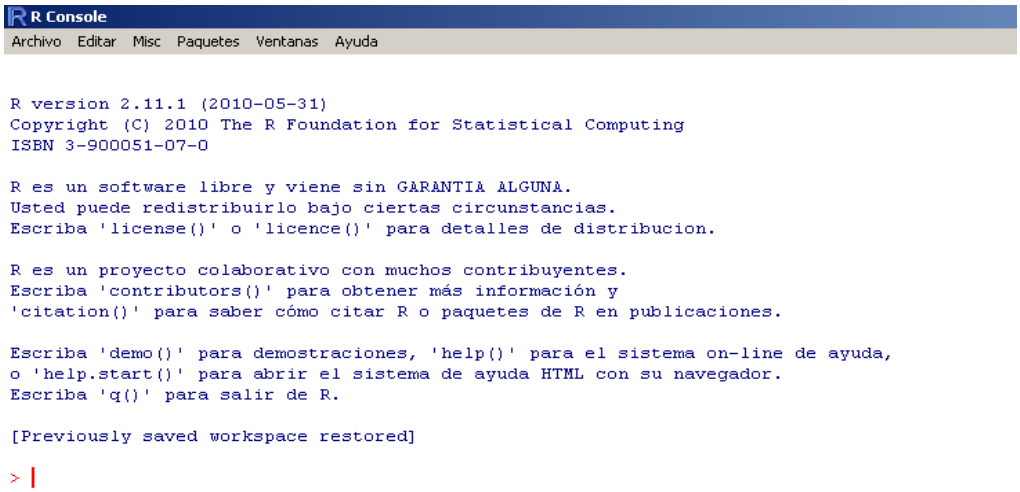
¿Qué es R?
R es un entorno informático con una colección de programas libres diseñados para el análisis estadístico de datos y presentaciones gráficas.

El lenguaje utilizado en R es una versión libre del lenguaje estadístico S. Sus códigos fuentes están disponibles bajo GPL (General Public License) en el proyecto GNU (http://www.gnu.org/) y funciona en distintos sistemas operativos como Linux, Windows, Mac. La página oficial de R es http://www.r-project.org/



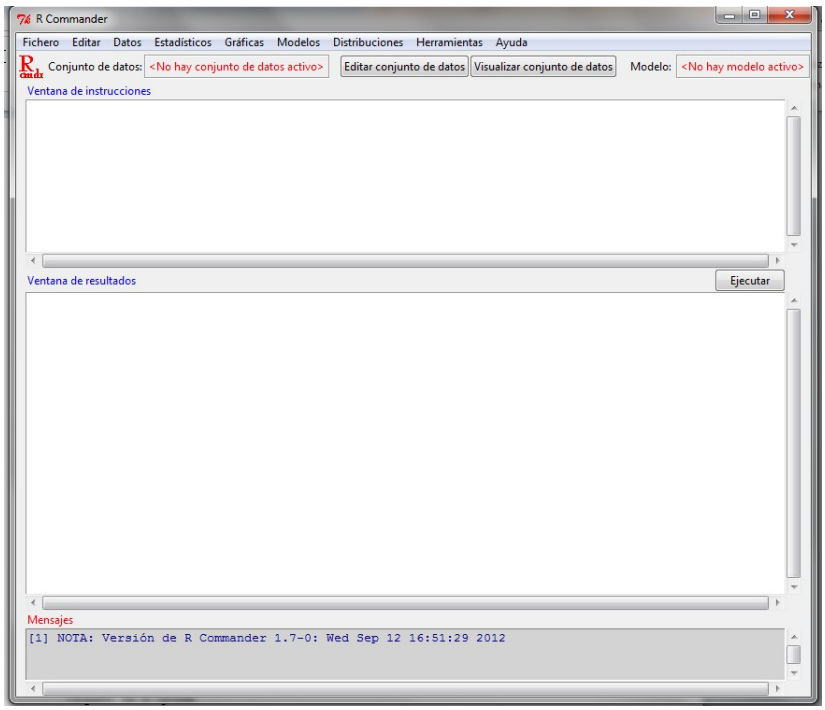
Entre las características de R pueden mencionarse las siguientes:
· El lenguaje R es similar al octave o matlab pero con la sintaxis orientada al manejo estadístico de datos.
· Existe un amplio grupo de programadores en R, lo que permite la existencia de paquetes para muchas aplicaciones estadísticas
· Existen diferentes interfaces gráficas que permiten realizar los análisis más usuales mediante menús. Entre ellas se encuentra la que vamos a usar Rcmdr, que funciona tanto en entorno Windows como Linux.

Inicio y salida de R:
R se puede utilizar de forma interactiva a través de la línea de comandos, introduciendo las ordenes después de



y se abandona con la opción salir del menú fichero o
> q()

Se puede instalar el interface gráfico R-comander utilizando los menús de la consola:
Paquetes > seleccionar espejo CRAN… > Spain(Madrid)
Paquetes > instalar paquetes…> Rcmdr (puede tardar un poco tiempo)
install.packages("Rcmdr")
Posteriormente para arrancar el interface se escribirá en la línea de comandos de R
>library(Rcmdr)
Con eso aparecen las ventanas del Rcommander y se puede trabajar en modo menú:



Una vez que ya se inició el Rcmdr si se sale de él, sin salir de R, se puede utilizar
>commander() (Para que vuelvan a aparecer las ventanas).

TIPOS DE OBJETOS EN R

Los objetos elementales de R son de tres tipos básicos: numérico, incluidos el Inf (ej: 3/0) y NaN (Not a Number) (ej: log(-1)), y NA(Not Avaliable), lógico (T o TRUE, F o FALSE) y cadenas de caracteres que están delimitadas por " o ' ("pepe", 'casa') (si se quiere utilizar el carácter " dentro del texto delimitado por " se deberá escribirse \").

Cuando a estos elementos se les desee asignar un nombre para guardarlos en memoria se utilizará <- , por ejemplo:

```
> a<-3       # crea el elemento numérico a con valor 3

> A<-1e4     # crea el objeto numérico A con valor 1000.

> b<-T       # crea el elemento lógico b y le asigna el valor verdadero
> B<-FALSE   # crea el elemento lógico  B   con valor falso
```

obsérvese que se distingue entre mayúsculas y minúsculas,

```
> b = =B
[1] FALSE

> p<-   'la "p" es letra'          # crea la variable p de tipo carácter cuyo contenido es
> p
[1] "la \"p\" es letra"
> name <- "Carmen"; n1 <- 10; n2 <- 100; m <- 0.5  # crea  cuatro variables
```

Note el uso del punto y coma para separar comandos diferentes en la misma línea.

Todo objeto tiene dos atributos intrínsecos: longitud y tipo

```
> x <- 1
> length(x)          [1] 1
> mode(x)            [1] "numeric"
```

IMPORTANTE: Hay que acostumbrarse a no usar "=" como operador de asignación en R. Normalmente su uso no devuelve ningún mensaje de error, pero tampoco realiza lo deseado y puede dar lugar a errores lógicos difíciles de encontrar.

Para ver los objetos que están almacenados en memoria se usa **ls()**

```
> ls()
[1] "a" "b" "B" "p"
```

ls(pat='b') listará los objetos que contiene en su nombre el carácter b
ls(pat='^b') listará los objetos que su nombre empieza por b
ls.str() nuestra la estructura de todos los objetos en memoria

```
> ls.str()
a :  num 3
b :  logi TRUE
B :  logi FALSE
p :  chr "la \"p\" es letra"
```

rm() sirve para borrar objetos
> rm(b)
> ls()
[1] "a" "B" "p"
> rm(list=ls(all=TRUE)) # borra todos los objetos en memoria

A los objetos de tipo numérico se pueden aplicar las operaciones usuales de +, -, * ,/, ^
> a+4 #sumar 4 a la variable a (que vale 3)
[1] 7
> b<-a^2 # se almacena en b el resultado de elevar a al cuadrado
> a*b # se realiza la operación a *b, o sea a^3
[1] 27

Operadores

Aritméticos		Comparativos		Lógicos	
+	adición	<	menor que	! x	NO lógico
-	substracción	>	mayor que	x & y	Y lógico
*	multiplicación	<=	menor o igual que	x && y	id.
/	división	>=	mayor o igual que	x | y	O lógico
^	potencia	==	igual	x || y	id.
%%	módulo	!=	diferente de	xor(x, y)	O exclusivo
%/%	división de enteros				

y también se les pueden aplicar las funciones usuales:

Otras funciones matemáticas: `help("abs")`	
`abs(x)`	valor absoluto
`sqrt(x)`	raíz cuadrada
Funciones especiales: `help("Special")`	
`factorial(x)`	x!
`choose(n,x)`	binomio n sobre x

Operaciones de redondeo: `help("round")`

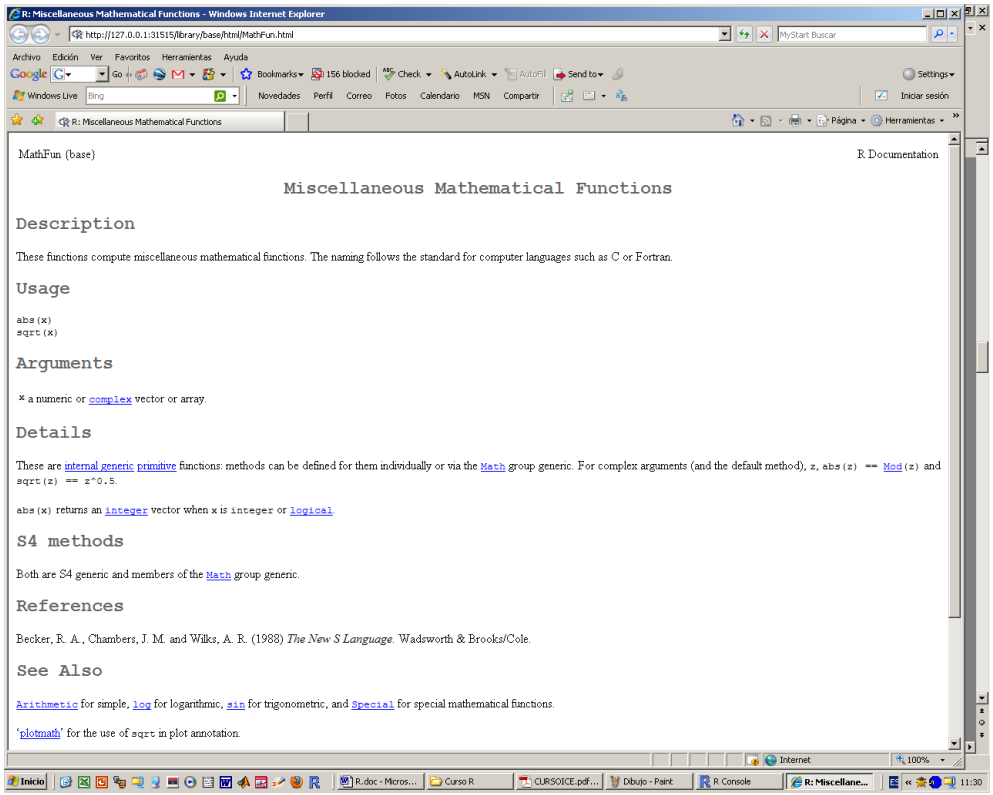
Función	Significado
`ceiling(x)`	redondeo hacia arriba
`floor(x)`	redondeo hacia abajo
`round(x)`	redondeo al más cercano
`signif(x)`	redondeo al número de dígitos significativos especificado
`trunc(x)`	eliminar decimales

Funciones matemáticas

Función	Significado
Funciones logarítmicas: `help("log")`	
`log(x)`	logaritmo natural
`log10(x)`	logaritmo en base 10
`log2(x)`	logaritmo en base 2
`logb(x,base)`	logaritmo en cualquier base
`log1p(x)`	calcula $\log(1+x)$ con precisión para $\|x\| \ll 1$
`exp(x)`	función exponencial
`expm1(x)`	calcula $\exp(x)-1$ con precisión para $\|x\| \ll 1$
Funciones trigonométricas: `help("Trig")`	
`cos(x)`	coseno
`sin(x)`	seno
`tan(x)`	tangente
`acos(x)`	arcocoseno
`asin(x)`	arcoseno
`atan(x)`	arcotangente
`atan2(y,x)`	ángulo entre el eje X y el vector del origen al punto (x, y)

Los objetos se pueden cambiar de tipo a través de las funciones **as.numeric** o **as.logical** . as.numeric transforma T o TRUE =1, F o FALSE=0, "25"= 25, 'a'=NAN. as.logical transforma 0=FALSE, >0=TRUE, 'F'=FALSE, 'T'=TRUE, 'B'=NA.

Las ordenes **help()** o **? o ??** sirven para pedir ayuda sobre una función o procedimiento de R (necesitan la conexión a internet)
> ?sqrt ## accede a la ayuda de R sobre la función sqrt
> help(sqrt) ## equivale a la anterior
>??norm (búsqueda más exhaustiva todos los paquetes que contienen esta sentencia)



La orden **apropos()** busca en memoria las funciones disponibles con el nombre que se indique

```
> apropos("help")
[1] "help"         "help.request" "help.search"  "help.start"
```

Hay cuatro funciones en memoria con help.

Tipos de elemntos compuestos en R.

Los elementos compuestos que puede utilizar R son: vectores, matrices, arrays, listas, data.frames.

Vectores

Es una colección de varios elementos del mismo tipo. Se pueden crear por asignación a través de la función **c(),** que combina distintos elementos separados por comas. Por ejemplo si se quieren almacenar los resultados de 5 lanzamientos de una moneda o de un dado se pueden construir dos vectores de longitud 5, uno numérico y otro de tipo carácter.

```
> moneda<-c('c','+','c','c','+')
> dado<-c(4, 5, 6, 7, 8 )
> ls.str()
dado :  num [1:5] 4 5 6 7 8
moneda :  chr [1:5] "c" "+" "c" "c" "+"
> dado
[1] 4 5 6 7 8
```

Los vectores también se pueden crear a través de secuencias **seq()** (indicado inicio , final y paso si es distinto de 1) o repeticiones **rep()**

```
> a<-(1:10)  #números del 1 al 10
> a
 [1]  1  2  3  4  5  6  7  8  9 10

> par<-seq(2,11,2)   # secuencia empezando en 2 con paso 2 y acabo en 11
> par
[1]  2  4  6  8 10
```

otros ejemplos de secuencias: seq(0, 1, length.out=11) seq(1, 6, by = 3)
seq(1.25, 0.25, -0.05)) seq(17) # igual que 1:17

```
> dos<-rep(2,10)  #repetir el valor 2 10 veces
> dos
 [1] 2 2 2 2 2 2 2 2 2 2

> A<-rep('a',5)   # se crea el vector de tipo carácter A  de dimensión 5  formado por
> A
[1] "a" "a" "a" "a" "a"
```

otros usos de rep

```
> rep(1:4, 2)         [1] 1 2 3 4 1 2 3 4
> rep(1:4, each = 2)         [1] 1 1 2 2 3 3 4 4
```

> rep(1:4, c(2,1,2,1)) [1] 1 1 2 3 3 4
> rep(1:4, each = 2, len = 10) [1] 1 1 2 2 3 3 4 4 1 1
> rep(1:4, each = 2, times = 3) # vector de longitud 24
 [1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
 > rep(seq(2,20,2),rep(2,10))
 [1] 2 2 4 4 6 6 8 8 10 10 12 12 14 14 16 16 18 18 20 20

otra forma de construir vectores es por combinación de otros anteriores
> b<-c(a,dos)
> b
 [1] 1 2 3 4 5 6 7 8 9 10 2 2 2 2 2 2 2 2 2 2
d<-c(A,dos) # vector formado por la unión de dos vectores de distinto tipo
> d
 [1] "a" "a" "a" "a" "a" "2" "2" "2" "2" "2" "2" "2" "2" "2" "2"

> l<-c(T,F)
> c(a,l) # transforma la variable lógica l en numérica (T=1, F=0) para añadírsela al vector numérico a
 [1] 1 2 3 4 5 6 7 8 9 10 1 0

La referencia a los elementos de un l vector se realiza indicando su posición entre **[]**
> par[4] # elemento 4 del vector par
[1] 8
> par[2:4] # elementos 2, 3, 4 del vector par
[1] 4 6 8
> par[c(1,3)] # elementos 1, 3 del vector par
[1] 2 6

Para eliminar elementos de un vector se indicará su posición con signo menos
> par[-1] # elimina el primer elemento del vector par
  [1] 4 6 8 10
> par[-c(1,3)] [1] 4 8 10
se pueden asignar valores a elementos de un vector haciendo referencia a sus indices de múltiples formas:
x<-1:10
x[x>5]<-20 # asigna 20 a todos los elementos del vector cuyo contenido actual es superior a 5.
x[x==1] <-5 # asigna 5 a los elementos de x que valgan 1.

A veces interesa quitar los valores NA y NaN de un vector, esto se haría con el comando **!is.na(x)**

x<-x[!is.na(x)] # elimina del vector x todos aquellos elementos que sean NA o NaN, l x[is.nan(x)]<-0 reemplazaría todos los NaN de x por 0.

Las operaciones +,-,*,/,^ entre vectores numéricos actúan elemento a elemento
> impar<-seq(1,10,2)
> impar [1] 1 3 5 7 9
> par<-seq(2,10,2)
> par+impar [1] 3 7 11 15 19

> par^impar [1] 2 64 7776 2097152 1000000000

Una función sobre un vector actúa sobre cada elemento del vector
> sqrt(par)
[1] 1.414214 2.000000 2.449490 2.828427 3.162278

Es importante tener en cuenta que en operaciones entre dos vectores de distinta dimensión reutiliza el más pequeño las veces que sea necesario:
> a<-c(1,2,3)
> par+a
[1] 3 6 9 9 12 # hizo la operación siguiente: 2+1 4+2 6+3 8+1 10+2

Mensajes de aviso perdidos
In par + a :
longitud de objeto mayor no es múltiplo de la longitud de uno menor

Aunque un vector se escriba como fila su posterior tratamiento en cálculo matricial será como vector columna, en la mayoría e los casos.

>t(par) #transpuesto de par es un matriz 1x5
[,1] [,2] [,3] [,4] [,5]
[1,] 2 4 6 8 10

> par%*%t(impar) # producto de los vectores par y transpuesto de impar es una matriz 5 x5
[,1] [,2] [,3] [,4] [,5]
[1,] 2 6 10 14 18
[2,] 4 12 20 28 36
[3,] 6 18 30 42 54
[4,] 8 24 40 56 72
[5,] 10 30 50 70 90

> t(par)%*%impar
[,1]
[1,] 190
y también considerando los dos como vectores:
>par%*%impar
[,1]
[1,] 190

Las funciones más usuales sobre vectores son:

Ordenación de vectores: `help("rank"), help("order"), help("sort")`

Función	Significado
`sort(x)`	vector x ordenado de menor a mayor
`rev(sort(x))`	vector x ordenado de mayor a menor
`sort(x,dec=T)`	vector x ordenado de mayor a menor
`rank(x)`	vector de rangos de x
`order(x)`	vector de permutaciones

Ordenes sort, order y rank :

```
sort(x, decreasing = FALSE, ...)
order(..., na.last = TRUE, decreasing = FALSE)
rank(x, na.last = TRUE, ties.method = c("average", "first", "random",
"max", "min"))
```

```
> x<- c(4, 3, 5, 8, 7 ,6,6,5)
> sort(x)          [1] 3 4 5 5 6 6 7 8
> order(x)         [1] 2 1 3 8 6 7 5 4
> rank(x)          [1] 2.0 1.0 3.5 8.0 7.0 5.5 5.5 3.5
```

Operaciones de extremos: `help("max")`

Función	Significado
`max(x)`	valor máximo en x
`min(x)`	valor mínimo en x
`pmax(x,y,z)`	máximos en paralelo
`pmin(x,y,z)`	mínimos en paralelo

Es importante que los elementos sobre los que se calculan máximos (mínimos) tengan todos sus elementos bien definidos, ya que en caso de tener elementos Nan pueden aparecer errores salvo que se utilice la orden **max(x, na.rm=TRUE)** que indica que se borren los elementos na del vector x

which(x>3)	índices satisfacen una determinada condición
which.max(x)	Posición en que se alcanza un máximo(mínimo)
length(x)	Longitud del vector

Operaciones acumulativas: `help("sum")`, `help("prod")`, `help("cumsum")`

Función	Significado
`sum(x)`	suma de los valores de x
`prod(x)`	producto de los valores de x
`cumsum(x)`	vector que contiene las sumas acumuladas
`cumprod(x)`	vector que contiene los productos acumulados
`cummax(x)`	vector que contiene los máximos acumulados
`cummin(x)`	vector que contiene los mínimos acumulados

Algunos descriptivos

Función	Significado
`mean(x)`	Media aritmética
`mean(x,trim=0.2)`	Media recortada al 20% (p.ej.)
`median(x)`	Mediana
`quantile(x,0.95)`	Cuantiles (p.ej. percentil 95)
`sd(x)`	Cuasi-desviación típica
`var(x)`	Cuasi-varianza
`IQR(x)`	Rango intercuartílico
`range(x)`	Rango (mínimo y máximo)
`summary(x)`	Resumen estadístico

Cuando se quieran varios cuantiles (por ejemplo el 0.25, el 0.8…..)

```
> quantile(x,c(.25,.8,.9,.95))
    25%      80%      90%      95%
17.08961 23.05216 25.86564 27.79499
```

En todas estas órdenes descriptivas si algún dato falta es conveniente eliminarlos para evitar errores, como por ejemplo

```
> x<-c(1,2,3,0/0)
> mean(x)                 [1] NaN
> mean(x,na.rm=T)         [1] 2
```

Factores

A veces nos encontramos con que los datos de un vector indican una "cualidad" y pueden ser agrupados de acuerdo a un criterio. Imaginemos que tenemos un vector prov que nos indica la provincia a la que pertenece cada individuo de un grupo:

```
prov<-c("lu","co","lu","po","co","or","co","lu","lu","po","co","or","co","co", "lu","po")
```

El comando **factor**() convierte este vector en un factor. Es decir un nuevo vector que además contiene la información relativa a los distintos valores de los elementos del vector que va a llamar niveles. Serán útiles para gráficos y procedimientos estadísticos.

```
provf<-factor(prov)
provf
[1] lu co lu po co or co lu lu po co or co co lu po
Levels: co lu or po
```

La variables factor son importantes para los análisis estadísticos ya que hay muchos procedimientos que sólo trabajan con variables de este tipo, por ejemplo para comparar la media en dos grupos se necesita que el grupo esté definido por una variable factor. Muchas veces se pueden utilizar códigos numéricos para los distintos grupos, pero si la variable está definida como factor no se podrán aplicar procedimientos de vectores numéricos, por ejemplo la suma o la media.

Matrices

Son objetos cuyos elementos están organizados en filas y columnas.

Para crear una matriz se emplea el comando: **matrix(datos, nfilas, ncol, byrow=F)**

datos= datos de la matriz, nfilas=número de filas, ncol=número de columnas,

byrow=F (opción por defecto) indica que los datos se ordenan por columnas.

```
a<-c(1:10);  A<-matrix(a,2); A
#A es una matriz con 2 filas cuyos elementos son los números del 1 al 10
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10

> A1<-matrix(a,2,byrow=T); A1
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
```

Cuando el tamaño de la matriz no coincide con el número de datos disponibles R "ajusta" los datos.

```
> matrix(3,2,2)  # matriz 2x2 cuyos elementos toman el valor 3
```

```
     [,1] [,2]
[1,]    3    3
[2,]    3    3
> D<-matrix(1:5, 2); D
```
Mensajes de aviso perdidos In matrix(1:5, 2) la longitud de los datos [5] no es un submúltiplo o múltiplo del número de filas [2] en la matriz
```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    1

D<-matrix(1:15, 2,5); D
```
Mensajes de aviso perdidos In matrix(1:15, 2, 5) :la longitud de los datos [15] no es un submúltiplo o múltiplo del número de filas [2] en la matriz
```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

Para obtener los elementos de la matriz se debe hacer referencia a los índices de las filas y las columnas (cuando un índice se deja en blanco hace referencia a toda la dimensión)
```
> A[1,3]; A[, 2]; A[1,c(1,3,5)] A[1,]
[1] 5
[1] 3 4
[1] 1 5 9
[1] 1 3 5 7 9
```

Para eliminar filas o columnas se pone el subíndice correspondiente en negativo
```
> A<-matrix(1:6,2); A[,-1]
     [,1] [,2]
[1,]    3    5
[2,]    4    6
```
Al igual que en el caso de los vectores, todos los elementos de una matriz tienen que ser del mismo tipo; en otro caso R los transforma de manera automática.

Para unir vectores o matrices por filas o columnas se utilizan las funciones **rbind()** y **cbind()**:
```
> rbind(1:3,c(4,5,6))
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
> cbind(1:3,c(4,5,6))
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
```

Si los vectores que se quieren unir no cumplen las condiciones adecuadas, R reutiliza los elementos del vector de menor dimensión (y da un aviso)
```
> cbind(1:3,4:7)
     [,1] [,2]
[1,]    1    4
```

[2,] 2 5
[3,] 3 6
[4,] 1 7
Mensajes de aviso perdidos
In cbind(1:3, 4:7) : number of rows of result is not a multiple of vector length (arg 1)

Operaciones con matrices
Los operadores +, -, * , / entre matrices actúan elemento a elemento.
> A<-matrix(1:6,2,3); B<-matrix(2,2,3); B*A
  [,1] [,2] [,3]
[1,] 2 6 10
[2,] 4 8 12

El producto de usual de matrices se representa con %*%
> B%*%t(A)
  [,1] [,2]
[1,] 18 24
[2,] 18 24

> B%*%A
Error en B %*% A : argumentos no compatibles

> vector%*%matriz multiplica como si el vector fuese fila, resultado vector
> matriz%*%vector multiplica como si el vector fuese columna, resultado vector

Matrices especiales
I<- diag(n) # matriz identidad de tamaño n
J<- matrix(1, n,n) # matriz de unos de tamaño n

Operaciones con matrices

Función	Significado
`t`	Traspuesta
`+`	Suma elemento a elemento
`*`	Producto elemento a elemento
`%*%`	Producto matricial
`crossprod`	Producto cruzado: `crossprod(x,y)` ≡ `t(x) %*% y`
`outer`	Producto exterior de dos vectores
`scale`	Escala las columnas de una matriz
`diag`	Crea una matriz diagonal o extrae la diagonal de una matriz
`det`	Determinante de una matriz
`solve`	Inversa; resuelve sistemas de ecuaciones lineales
`eigen`	Valores propios
`svd`	Descomposición en valores singulares
`qr`	Descomposición QR
`chol`	Descomposición de Choleski

eigen(B) # es una lista que contiene los valores y vectores propios de una matriz B.
B<-matrix (1:9,3); T<-eigen(B) ; T
> T
$values

```
[1]  1.611684e+01 -1.116844e+00 -4.054215e-16
$vectors
          [,1]       [,2]       [,3]
[1,] -0.4645473 -0.8829060  0.4082483
[2,] -0.5707955 -0.2395204 -0.8164966
[3,] -0.6770438  0.4038651  0.4082483
```

A los valores propios se puede acceder como una lista o como un vector:
Valpropio<-T[1] # se accede en forma de lista (no se puede operar como vector)
Valpropio<-T[[1]] # accede como vector (T[[1]] es equivalente T$values)

A los vectores propios se accede de la misma forma
Vecpropio<-T[2] # se accede en forma de lista (no se puede operar como matriz)
Vecpropio<-T[[2]] # accede como matriz (T[[2]] es equivalente T$vectors)

Algunas funciones sobre matrices

Función	Significado
`dim`	Dimensiones de una matriz
`dimnames`	Nombres de las dimensiones de una matriz
`colnames`	Nombres de las columnas de una matriz
`rownames`	Nombres de las filas de una matriz
`mode`	Tipo de datos de los elementos de una matriz
`length`	Número total de elementos de una matriz
`is.matrix`	Devuelve `T` si el objeto es una matriz, `F` si no lo es
`[ , ]`	Accede a elementos dentro de la matriz
`apply`	Aplica una función sobre las filas o columnas de una matriz
`cbind`	Pega matrices (o vectores) por columnas
`rbind`	Pega matrices (o vectores) por filas

Función **apply**() ya que nos permite aplicar distintas funciones a las filas, o columnas de una matriz

```
> A<-matrix(1:6,2,3); apply(A,1,sum)          [1] 9   12
> apply(A,2,sum)                              [1] 3   7   11
> apply(A,2,mean)  (media por columnas)       [1] 1.5   3.5   6.5
```

Hay que tener cuidado ya que max(A) nos da el máximo global de toda la matriz.

Array

Generaliza una matriz al caso de más de dos dimensiones. La sintaxis que se usa es: **array(datos,v)** # v es un vector con las dimensiones del array

```
AA<-array(1:24, c(2,6,2)); AA
, , 1                                   , , 2
     [,1] [,2] [,3] [,4] [,5] [,6]           [,1] [,2] [,3] [,4] [,5] [,6]
[1,]    1    3    5    7    9   11     [1,]   13   15   17   19   21   23
[2,]    2    4    6    8   10   12     [2,]   14   16   18   20   22   24
> AA[1,3,2]                            [1] 17
```

Lista

Una lista es un objeto cuyas componentes pueden ser arrays, listas, Data Frames,
Se usan para guardar en un mismo objeto elementos de distinto tipo.
Su sintaxis de esta función es: **list(objeto1, objeto2, …)**

```
> Lista<-list(c(1:3), matrix(11:20,5,2), c("pepe", "juan")); Lista
[[1]]
[1] 1 2 3

[[2]]
     [,1] [,2]
[1,]   11   16
[2,]   12   17
[3,]   13   18
[4,]   14   19
[5,]   15   20

[[3]]
[1] "pepe" "juan"
```

Es importante distinguir entre cada objeto y su contenido;

- con Lista[1], hacemos referencia al primer objeto de Lista, pero no podemos manipular su contenido como vector, es una sublista.
- con Lista[[1]] hacemos referencia al contenido del primer objeto de Lista, el vector (1,2,3)

```
> vv<-Lista[[1]]; vvv<-Lista[1]; ls.str(pat="^vv")
vv :  int [1:3] 1 2 3
vvv : List of 1
 $ : int [1:3] 1 2 3
```

Cuando se construye una lista se pierde la “identidad” de los objetos que la forma y su referencia se hace por su posición, por ejemplo Lista[2] o Lista[[2]]. . Para asignarle un nombre hay que ponérselo en la construcción de la lista, antes del objeto ,de la siguiente forma:

```
>v<-c(1:3); M<-matrix(11:20,5,2);
>  Lista<-list(v=v,M=M,nombre=c("pepe", "juan")); Lista
$v
[1] 1 2 3

$M
     [,1] [,2]
[1,]   11   16
[2,]   12   17
[3,]   13   18
[4,]   14   19
[5,]   15   20
$nombre
[1] "pepe" "juan"

> Lista$M # accede a la matriz contenida en el segundo objeto de la lista
     [,1] [,2]
[1,]   11   16
```

```
[2,]   12   17
[3,]   13   18
[4,]   14   19
[5,]   15   20
```

Observese la diferencia de los siguientes ejemplos:

```
> apply(Lista$M,1,sum)  # suma las filas la matriz M de la Lista
[1] 27 29 31 33 35

> apply(Lista[2],1,sum) # el objeto  Lista[2] es una sublista
Error en apply(Lista[2], 1, sum) : dim(X) must have a positive length ,

> apply(Lista[[2]],1,sum)
[1] 27 29 31 33 35
```

Data.frame

Es objeto cuyo contenido son vectores que pueden ser de diferentes tipos. Se suelen emplear para almacenar datos estadísticos, por lo que en general cada columna representa una variable y cada fila un individuo (hay que tener cuidado cuando los vectores no son de la misma longitud).
Salvo que se tengan pocos datos es preferible construir el fichero de datos con otra aplicación (una base de datos tipo excel, o un editor) e importar los datos desde R.

Para crear una base de datos con dos variables, edad y sexo, y 5 individuos se puede hacer lo siguiente. Como la segunda variable es cadena será considerada como una variable factor con dos niveles que son los diferentes valores que puede tomar la variable sexo.

```
> datos<-data.frame(edad=c(22,33,44,55),sexo=c("v","m","v","m")) ;datos
  edad sexo
1   22    v
2   33    m
3   44    v
4   55    m
> datos[1,2]  #contenido del elemento [1,2], valor del sexo para el primer individuo
[1] v
Levels: m v
> datos[,1]  #datos de la primera variable
[1] 22 33 44 55
> datos[1,]  # datos del individuo 1
  edad sexo
1   22    v
> datos[,2]  #datos de la segunda variable, cadena  con dos valores v y m,  será considerada como una variable factor con dos niveles.
[1] v m v m
Levels: m v
```

En los data.frame utilizaremos bastantes veces la función **tapply** para aplicar una función a los datos de una variable dentro de los grupos definidos por otra

```
tapply(X, INDEX, FUN = NULL, ..., simplify = TRUE, na.rm = TRUE)
```

na.rm = TRUE solo aplica la función cuando hay dato, para evitar los problemas con datos ausentes

```
> tapply(datos$edad, datos$sexo,mean)
> tapply(datos[[1]],datos$sexo,mean)
          # calcula la media de la primera variable en los grupos definidos por sexo
 m  v
44 33
```

```
n <- 17; fac <- factor(rep(1:3, length = n), levels = 1:5); fac
```
#se crea el objeto fac que es de tipo factor con 17 elementos que son rep del 1:3 y se le dice que hay 5 posibles valores 1:5 (aunque algunos de momento no aparecen)
```
> fac
 [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3 1 2
Levels: 1 2 3 4 5
```

```
>table(fac)  #tabla de frecuencias de fac
fac
1 2 3 4 5
6 6 5 0 0
```

>tapply(1:n, fac, sum) #suma los elementos del 1:17 dentro de cada índice definido por fac (cuando fac=1 1+4+7+10+13+15; ……)
```
 1  2  3  4  5
51 57 45 NA NA
```

```
> tapply(1:n, fac, range) #calcula el rango dentro de cada grupo definido por fac
$`1`              $`2`              $`3`              $`4'              $`5`
 [1]  1 16         [1]  2 17         [1]  3 15         NULL              NULL
```

```
> tapply(1:n, fac, quantile)  # calcula el mínimo, los cuartiles y el máximo para los
          grupos indicados por el nivel de fac
$`1`   # formado por los números 1,4,7,10,13,16
  0%      25%     50%     75%     100%
1.00     4.75     8.50    12.25   16.00
$`2`
  0%      25%     50%     75%     100%
2.00     5.75     9.50    13.25   17.00
$`3`
  0%      25%     50%     75%     100%
  3        6       9       12      15
$`4`
NULL
$`5`
NULL
```

Expresiones

Son una colección de caracteres que tienen sentido en R, posteriormente se pueden valorar o utilizar para otras funciones como la derivada.

nombre**<-expresion(**definición**)**

```
> exp1<-expression(x^2/y^3)
> D(exp1,'x')  #calcula la derivada de la expresión respecto de x
2 * x/y^3
> x=2;y=3;eval(exp1) # evalua la expresión cuando x=2 e y=3
[1] 0.1481481
```

Funciones

Una característica muy útil del paquete R es la posibilidad de ampliar las funciones existentes y escribir fácilmente las funciones habituales. De hecho, la mayoría del software R puede ser visto como una serie de funciones R. La forma genérica de crear una función es

nombre <- function(argumento1 , argumento2, ...) { comandos de definición}

```
> flineal<-function(x)  2*x^2+5    #se crea la función 2x^2+5 que se llama flineal
> flineal(2)       # se valora la función anterior en 2
[1] 13
> flineal(1:5)  #se valora la función flineal en los enteros  del 1 al 5
[1]  7 13 23 37 55
> g<-function(x,y) x^2+3+2*y   #se crea la función g de dos variables  x^2+3+2y
> g(2,3)
[1] 13
```

Las funciones así definidas pueden utilizarse como argumentos de otras funciones

```
>  tapply(datos$edad,datos$sexo,flineal)  aplica la flineal a los datos de edad en cada grupo de sexo
$m
[1] 2183 6055
$v
[1]  973 3877
```

Por ejemplo, podemos definir una función que calcule la desviación típica:

```
x<-c(1,2,3)
dt <-function(x){ # Definimos la función
media<-sum(x)/length(x)
var<-sum(x^2)/length(x)-media^2
cuasivar<-length(x)/(length(x)-1)*var
sqrt(cuasivar)
}
```

(si no se pone return la función devuelve el último valor que se ha calculado)

Las variables definidas dentro del cuerpo de una función son locales, y desaparecen al terminar la ejecución de la función. Por ejemplo:

```
> y <- 10 # Definimos la variable y
> cuadrado = function(x){ y <- x^2 ; return(y)} # Definimos otra y local
> x <- 2 # Asignamos valor a x
> cuadrado(x) # Calculamos el cuadrado de x : Se hace y=4 (localmente)
[1] 4
```

> y # Sin embargo, y no ha cambiado. La y local desaparece
[1] 10

>solve resuelve polinomios
>uniroot ceros de funciones unidimensionales

Funciones para la aplicación repetida de otras funciones varias veces

sapply aplica a cada elemento del vector x , la function f devolviendo un vector o una matriz o un array
sapply(x, f, simplify=FALSE, USE.NAMES=FALSE)

replicate es similar a sapply pero se utiliza para la evaluación repetida de una función, es importante para la simulación de estadisticos

ks<-function(n) {x<-sort(runif(n,0,1));empix<-(1:n)/n; max(empix- x,x-(empix-1/n)) }
estks<-replicate (nr,ks(n)) # nr valores del estadistico de k-s

tapply aplica una función a los elementos de un vector en función de los grupos definidos por uno o varios factores
tapply(X, INDEX, FUN = NULL, ..., simplify = TRUE)
tapply(X, sexo, mean)# calcula para cada valor e sexo la media de x

Funciones asociadas al manejo de distribuciones estadísticas

Además de las funciones descriptivas, ya mencionadas anteriormente,

Algunos descriptivos

Función	Significado
`mean(x)`	Media aritmética
`mean(x,trim=0.2)`	Media recortada al 20 % (p.ej.)
`median(x)`	Mediana
`quantile(x,0.95)`	Cuantiles (p.ej. percentil 95)
`sd(x)`	Cuasi-desviación típica
`var(x)`	Cuasi-varianza
`IQR(x)`	Rango intercuartílico
`range(x)`	Rango (mínimo y máximo)
`summary(x)`	Resumen estadístico
`max(x)`	valor máximo en x
`min(x)`	valor mínimo en x

Por ejemplo para calcular los cuantiles de unos datos

quantile(x,0.75) calcula el cuantil 0.75 del vector x
quantile(x,c(.01,.99) calcula las percentiles 1 y 99 del vector x

El paquete R dispone de numerosas funciones para el manejo de las distribuciones más usuales, tanto su función de densidad o probabilidad, función de distribución y generación de valores aleatorios con determinada distribución.

Distribuciones

binom	**Binomial**
cauchy	**Cauchy**
chisq	**Chi Cuadrado, centrada o descentrada**
beta	**Beta**
exp	**Exponencial**
f	**F centrada o descentrada**
gamma	**Gamma**
geom	**Geométrica**
hyper	**Hipergeométrica**
lnorm	**Log-normal**
logis	**Logística**
nbinom	**Binomial negativa**
nchisq	**Chi cuadrado no central**
norm	**Normal**
pois	**Poisson**
signrank	**Distribucion del test de Wilcoxon de rangos con signo**
t	**Student centrada o descentrada**
tukey	**Rango studentizado**
unif	**Uniforme**
weibull	**Weibull**
wilcox	**Distribución de la suma de rangos de Wilcoxon**

Estas distribuciones se pueden utilizar para el cálculo de las densidades (o probabilidad), función de distribución, cuantiles y generación de números aleatorios escribiendo delante de los anteriores nombres una de las siguientes letras

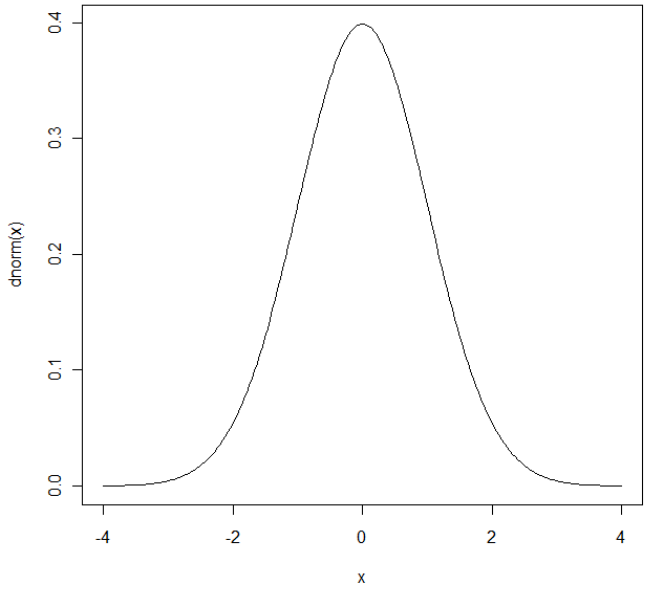
d:Densidad de probabilidad	Funcion de densidad o de probabilidad
p:Probabilidad	Función de distribución
q:Quantil	Percentiles
r: Aleatorio(random)	Generación de muestras aleatorias

Ejemplo con la distribución normal

```
> rnorm(5) # Generación de 5 datos de la normal estandar
[1] -0.3220499 -0.5556478 -0.1899898 -0.3450181 -2.5807986
> rnorm(5,mean=1,sd=3) # Generación de 5  datos  de una normal no estándar, N(1,3)
[1] -0.4035896 -0.8089832 3.2513373 4.9641722 -1.8603231
> dnorm(c(0,1,3)) # Evaluación de la función de densidad normal en los puntos 0,1,3
[1] 0.3989423      0.2419707     0.004431848
>dbinom(5,7,.5)  #P( B(7,.5)=5)
> pnorm(3) # Probabilidad acumulada bajo la normal en el punto 3 o sea F(3)
[1] 0.9986501
> qnorm(0.5) # El cuantil 50% de la normal que es el 0
[1] 0
> qnorm(0.9986501)
[1] 3.00000
```

Estas órdenes se pueden utilizar para dibujar las funciones de densidad o de distribución de un variable. Por ejemplo para dibujar la gráfica de la función de densidad de la normal estandar

```
> x<-seq(-4,4,length=200) # puntos en los que se va a valorar
> plot(x,dnorm(x),type="l")  #dibujo tipo línea en los puntos anteriores
```



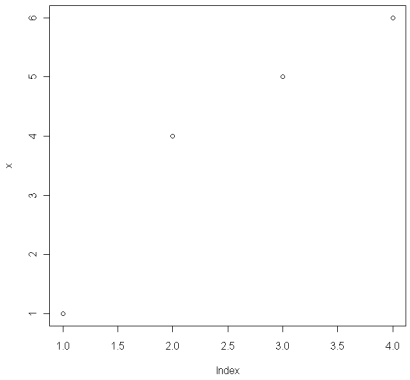
Otras funciones estadísticas de interés son las relacionadas con la función de distribución empírica:

ecdf(x) función de distribución empírica asociada al vector x , luego se puede dibujar directamente con plot(ecdf(x))
ecdf(x,y) calcula la función de distribución empírica de x y la valora en los puntos dados por y

GRÁFICAS

El programa R también realiza gráficas que pueden exportarse a distintos formatos PDF, Latex, PNG, JPEG. Los gráficos disponibles en R son de gran calidad y versatilidad. Para tener una idea se puede ejecutar la demo del programa con: **demo("graphics")**
La función básica para los gráficos es la función **plot()** que puede ser modificada por **lines(), points(), legend(), text(),** etc. ya que R distingue dos grandes grupos:
Gráficos de alto nivel: Crean un nuevo gráfico en la ventana de gráficas.
Gráficos de bajo nivel: Permiten añadir líneas, puntos, etiquetas... a otro gráfico

La función básica para los gráficos de alto nivel es **plot()**
La función plot sobre un vector realiza una nube de puntos con ordenadas el indice de posición en el vector y ordenada el elemento correspondiente. pares (i,x[i])
x<-c(1,4,5,6); plot(x)



entre dos vectores x,y plot dibuja la nube de puntos (x[i],y[i]).
Si f es un vector de tipo factor entonces plot(f) dibuja un diagrama de barras. Cuando f=factor, y=vector numérico plot(f,y) da un diagrama de cajas de y por cada nivel de f.
Si d es un data.frame plot(d) realiza nubes de puntos de todos los pares de variables.

La función *plot* puede ser modificada por distintos parámetros
plot(x, y = NULL, type = "p", xlim = NULL, ylim = NULL,log = "", main = NULL, sub = NULL, xlab = NULL, ylab = NULL, axes = TRUE, col= ...)

type	un carácter que da el tipo de dibujo. "p" para puntos, "l" para línea, "o" ambos superpuestos, "s" gráfico de escaleras y "h" barras como un diagrama de barras.
lwd	Ancho de linea
lty	tipo de línea, (0=blank, 1=solid (default), 2=dashed, 3=dotted, 4=dotdash, 5=longdash, 6=twodash)
pch	Carácter par el dibujo: ".","*"……
log	"x" para que este eje sea logarítmico, "y" para el eje y o "xy" para ambos ejes
main	'Título para el dibujo'.
sub	'Subtítulo'.
xlab	'Etiqueta para el eje x'
ylab	'Etiqueta para el eje y'
axes	Un valor lógico para indicar si se dibujan los ejes.

col=v especifica colores del gráfico. Si v es un vector los colores se aplican sucesivamente (0-negro,2-rojo,3-verde, 4-azul, 5-cian, 6-magenta, 7-amarillo, 8-gris ...)
para más ayuda utilizar help(par).

R, dispone de otras funciones para añadir elementos a un gráfico, las más representativas se muestran en la siguiente tabla:

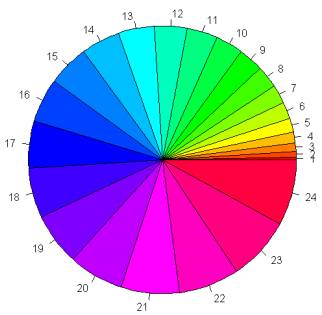
R: Herramientas Gráficas.	
points()	Añade puntos al gráfico activo.
lines()	Añade líneas al gráfico activo.
text()	Añade un texto en el gráfico actual.
abline	Añade líneas en el gráfico actual. abline(a,b) Añade una línea de pendiente a y que corta al origen en b abline(h=y) Añade línea horizontal que corta al eje y en h=y abline(v=x) Lo análogo para línea vertical
polygon()	Dibuja un polígono.
title()	Añade título y/o subtitulo al gráfico actual.
axis()	Añade ejes al gráfico actual (De 1 a 4 maneras diferentes)
persp	gráficos 3D
contour	Gráficos de contorno

Gráficos estadísticos de alto nivel

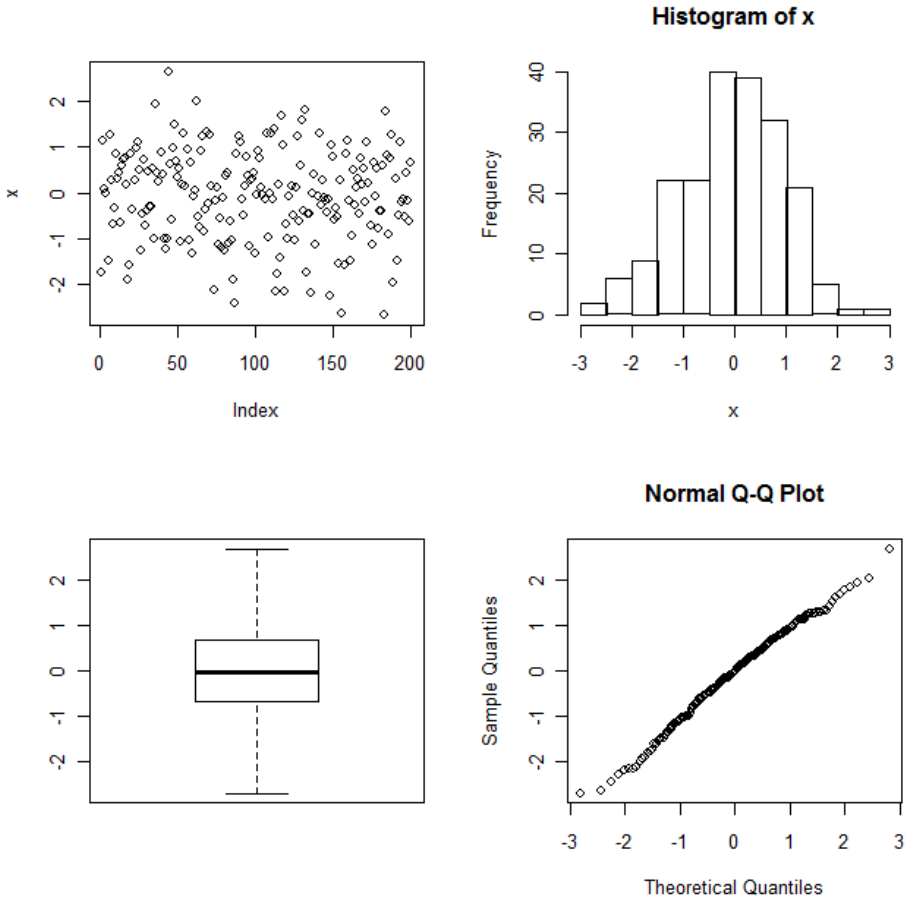
barplot	Diagrama de barras
pie	Diagrama de sectores
boxplot	Diagrama de caja
hist	Histograma
pairs	Pares de gráficos de dispersión por variables
qqnormal	Gráfico cuantil-cuantil

Tambien hay dos funciones para localizar las coordenadas de un punto en el gráfico: La función **locator()** que sitúa el cursor en la ventana de gráficos y al pulsar el botón izquierdo del ratón devuelve las coordenadas del punto marcado. La función **identify()** devuelve, al marcar un punto de la gráfica, el índice del vector que dio lugar a ese punto

Ejemplos
pie((1:24), col= rainbow(24), radius= 0.9) # diagrama de sectores con frecuencias

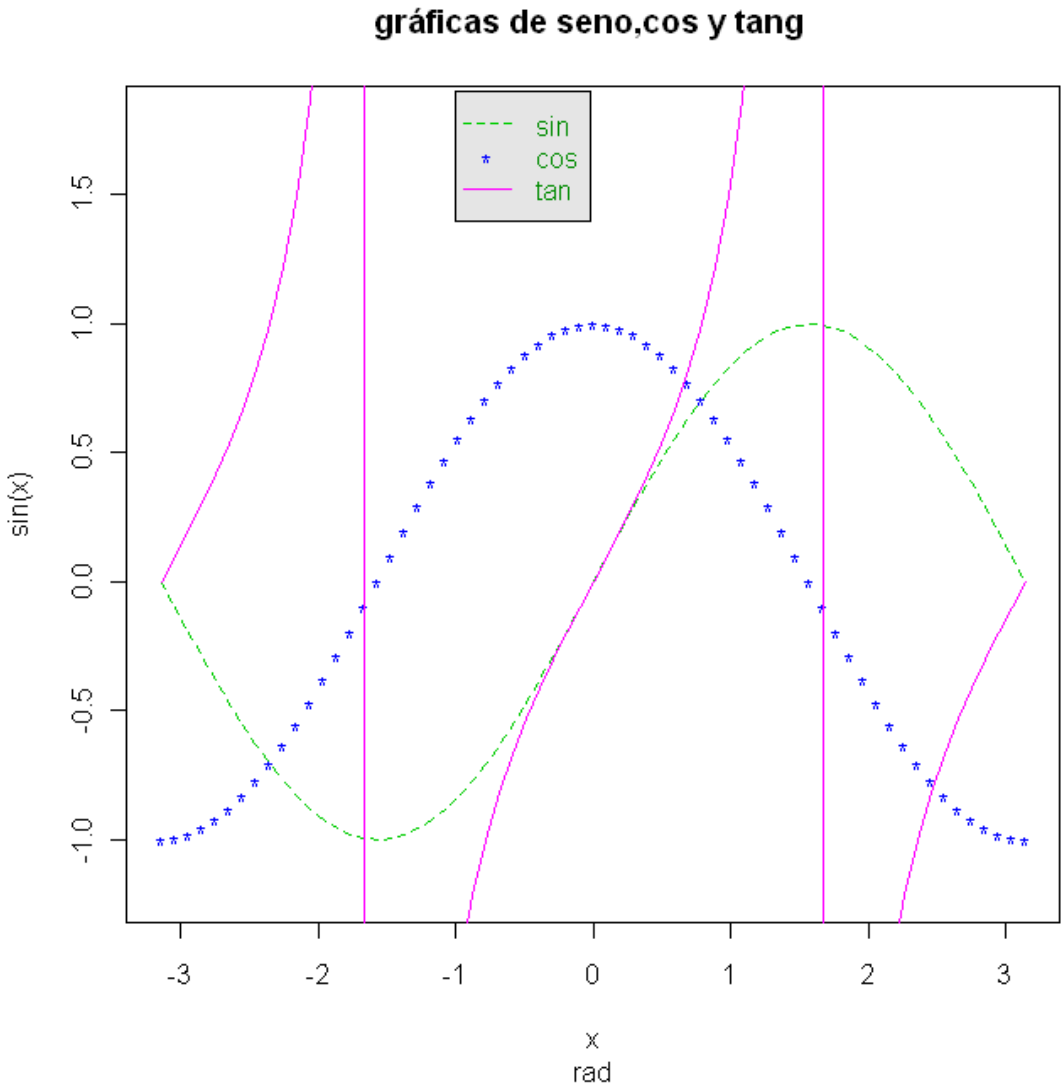


> x = rnorm(200) ;
> par(mfrow=c(2,2)) # divide la ventana gráfica en una matriz de gráficas 2x2
> plot(x) ; hist(x) ; boxplot(x) ; qqnorm(x)



Para combinar funciones en una misma gráfica se utiliza plot(), y se modifica con las ordenes de dibujo de bajo nivel, por ejemplo para dibujar simultáneamente seno, coseno y tangente

```
x <- seq(-pi, pi, len = 65)
plot(x, sin(x), type = "l", ylim = c(-1.2, 1.8), col = 3, lty = 2, main="gráficas de seno,cos y tang ",sub="rad")
points(x, cos(x), pch = "*", col = 4)
lines(x, tan(x), type = "l", col = 6)
legend(-1, 1.9, c("sin", "cos", "tan"), col = c(3,4,6),
       text.col = "green4", lty = c(2, -1, 1), pch = c("","*",""),
       merge = TRUE, bg = 'gray90')
```



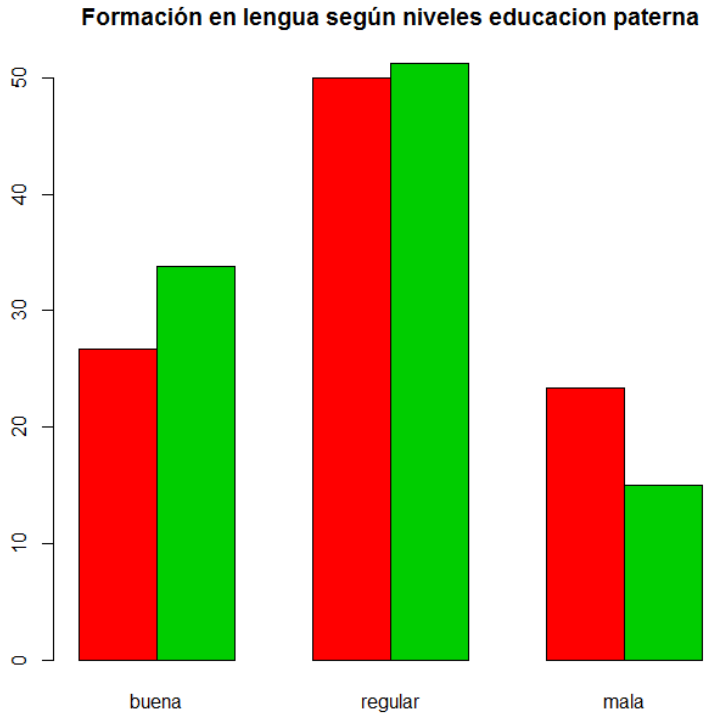
Gráficas estadísticas:

barplot(height, width = 1, space = NULL,
names.arg = NULL, legend.text = NULL, beside = FALSE,
horiz = FALSE, density = NULL, angle = 45,
col = NULL, border = par("fg"),
main = NULL, sub = NULL, xlab = NULL, ylab = NULL,
xlim = NULL, ylim = NULL, xpd = TRUE, log = "",
axes = TRUE, axisnames = TRUE,
cex.axis = par("cex.axis"), cex.names = par("cex.axis"),
inside = TRUE, plot = TRUE, axis.lty = 0, offset = 0,
add = FALSE, args.legend = NULL, ...)

pie(x, labels = names(x), edges = 200, radius = 0.8,
clockwise = FALSE, init.angle = if(clockwise) 90 else 0,
density = NULL, angle = 45, col = NULL, border = NULL,
lty = NULL, main = NULL, ...)

para realizar diagramas de barras apareados primero se realiza una tabla cruzada y luego los porcentajes por fila para luego dibujarlos:

```
Table <- xtabs(~nivele+FormaLen, data=Datos)
a<-rowPercents(Table) # Row Percentages
dime<-dim(a)
barplot(a[1:2,1:(dime[2]-2)], beside=T,col=c(2,3), main='Formación en lengua según niveles educación paterna') # se elimina del objeto a la columna de total y conteo que se obtiene al pedir la matriz de porcentajes
```



PROGRAMACIÓN

R permite realizar programas para poder solucionar nuestros problemas utilizando el sofware que ya está disponible para adaptarlo a cada situación para lo cual agruparan una serie de comandos entre {}. Las instrucciones que más se utilizan en la programación en R son for, if, while que permiten realizar bucles.

For

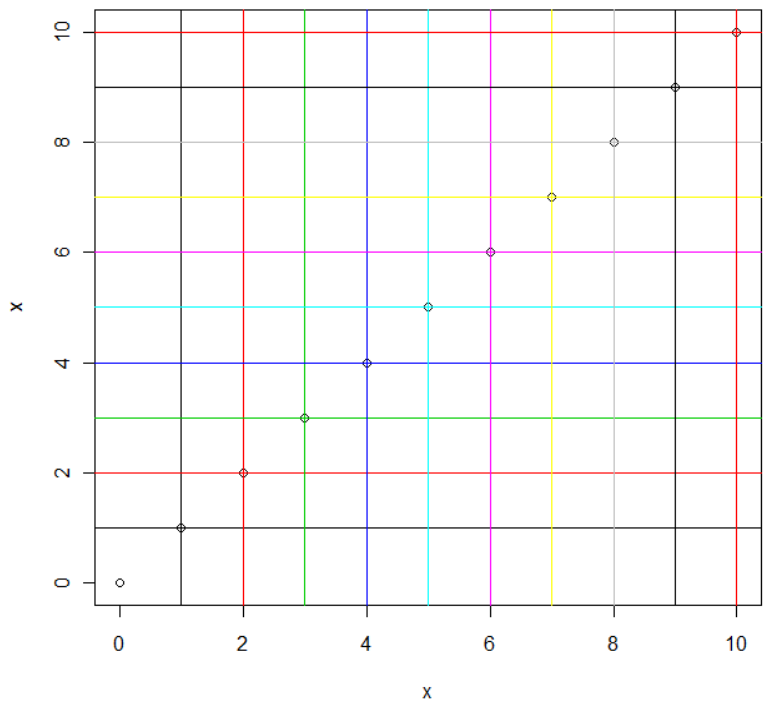
Esta instrucción se utiliza para crear bucles de la siguiente manera:

for (i in lista de valores) { secuencia de comandos }

```
> n<-3; x=numeric(n)
> for(i in 1:n) {
+    x[i]<-i^2;
+    print(x[i])
+ }
[1] 1
[1] 4
[1] 9
```

Otro ejemplo:

```
>x<-1:10; plot(x,x,xlim=c(0,10),ylim=c(0,10))
>for(i in 1:10){
+ abline(h=i,col=i)
+  abline(v=i,col=i)
+ }# dibuja una línea horizontal y otra vertical en cada punto de distintos colores
```



Los bucles **for** son lentos y deben ser evitados en la medida de lo posible por ello muchas veces se puede utilizar la función **replicate**

replicate(n,{}) n el número de repeticiones

While

while realiza un conjunto de instrucciones mientras no se cumpla cierta condición.

while (condición lógica) { expresiones a ejecutar }

Ejemplo.- Calcular el mayor número natural cuyo cuadrado no excede de 1000,

```
cuadrado = 0;  n<-0
 while(cuadrado<=1000)  {
 n<-n+1
 cuadrado<-n^2  }
 m<-n-1; m; m^2; cuadrado
[1] 31
[1] 961
[1] 1024
```

If

Sirve para ejecutar instrucciones cuando se verifique una condición.

La sintaxis general es:

if (condición) {comandos1} else {comandos2}

Por ejemplo, vamos a crear dos vectores; uno para guardar los números pares de 1 a 10, y otro para los impares:

```
 n <- 10 # Se inicializa n
 pares <- c() # Se crea un vector vacío
 impares <- c() # Idem
 for(i in 1:n){ # Se van a procesar los números de 1 a n
 if(i%%2==0) pares<-c(pares,i) # Si al dividir por 2 sale 0
   else
impares<-c(impares,i)} # el numero es impar en otro caso
 pares; impares
[1]  2  4  6  8 10
[1] 1 3 5 7 9
```

Directorios para grabar y leer datos

R utiliza el directorio de trabajo para leer y escribir archivos. Para saber cual es este directorio puede utilizar el comando **getwd()** (get working directory) .

Para cambiar el directorio de trabajo se utiliza la función **setwd()**; por ejemplo, setwd("C:/data") o setwd("/home//R") .

Lectura de datos

Además de lo visto anteriormente creación de vectores, matrices, ….. para introducir datos por teclado existen otras funciones scan(); data.entry(), read.table() …

La función mas usual para leer datos en forma tabular es **read.table()** que crea un objeto de tipo data frame. Entre sus argumentos se puede decir cuál es el separador de campo, si tiene encabezado o no, etc. Por ejemplo:

```
misdatos <- read.table("data.dat")
```

creará un data.frame (o matriz de datos) denominado misdatos, cada variable recibirá por defecto el nombre V1, V2, . . . y puede ser accedida individualmente escribiendo misdatos$V1, misdatos$V2, . . . , escribiendo misdatos["V1"], misdatos["V2"], . . . , o, escribiendo misdatos[,1] , misdatos[,2] .

x<- read.table("c:/octave/qtoctve……../data.dat") para leer un fichero en una determinada dirección

La función **scan**() es más flexible que read.table ya que permite especificar el modo de las objetos con la opción what y crear diferentes objetos como vectores, matrices, marcos de datos, listas, ... mientras que con la función read.table sólo se crean vectores.
> misdatos <- scan("data.dat", what = list("", 0, 0))
En este ejemplo scan lee tres variables del archivo data.dat; el primero es un carácter y los siguientes dos son numéricos.
Por defecto, es decir si se omite el argumento what, scan() crea un vector numérico. Si los datos leídos no corresponden al modo (o modos) esperado se genera un mensaje de error.

La función **read.fwf**() permite la lectura de datos en formato fijo.
read.fwf(file, widths, header = FALSE,sep = "\t", skip = 0, row.names, col.names, n = -1, buffersize = 2000, …)

Ejemplo:
ff <- tempfile()
cat(file=ff, "123456", "987654", sep="\n")
read.fwf(ff, widths=c(1,2,3))
#> 1 23 456 \ 9 87 654 lee 3 variables la primera de un carácter, la segunda de 2 y la tercera de 3.
read.fwf(ff, widths=c(1,-2,3)) #> 1 456 \ 9 654 lee 2 variables la primera de un carácter, salta 2 espacios y lee la segunda variable con 3 caracteres.

La función **load()** permite cargar datos en memoria grabados anteriormente con save

Escritura de datos
Las funciones **print()** y **cat()** nos permiten imprimir datos en pantalla durante la ejecución de un programa. Son útiles tanto para presentar la salida de resultados como para depurar el código.

La función cat es para visualizar objetos en panatalla
cat("Distribución: ",i," \n") # escribe por pantalla Distribución: y el valor de i.

Para escribir datos en archivos se utiliza las funciones **save(), write() o write.table()**:
Una manera sencilla de escribir los contenidos de un objeto en un archivo es utilizando la función write(x, file="data.txt", append=F), donde append es un elemento lógico que agrega los datos al archivo sin borrar datos ya existentes (TRUE) o borra cualquier dato que existe en el archivo (FALSE, por defecto).

La función save permite guardar objetos de diferentes tipos como un archivo de datos R
save(x, y, z, file= "xyz.RData").
Para intercambiar datos entre distintas máquinas se puede usar la opción ascii = TRUE .

El programa R trae múltiples conjuntos de datos. Para ver una lista de los mismos basta teclear **data()**, para cargar uno concreto se ha de usar el comando data(nombre).

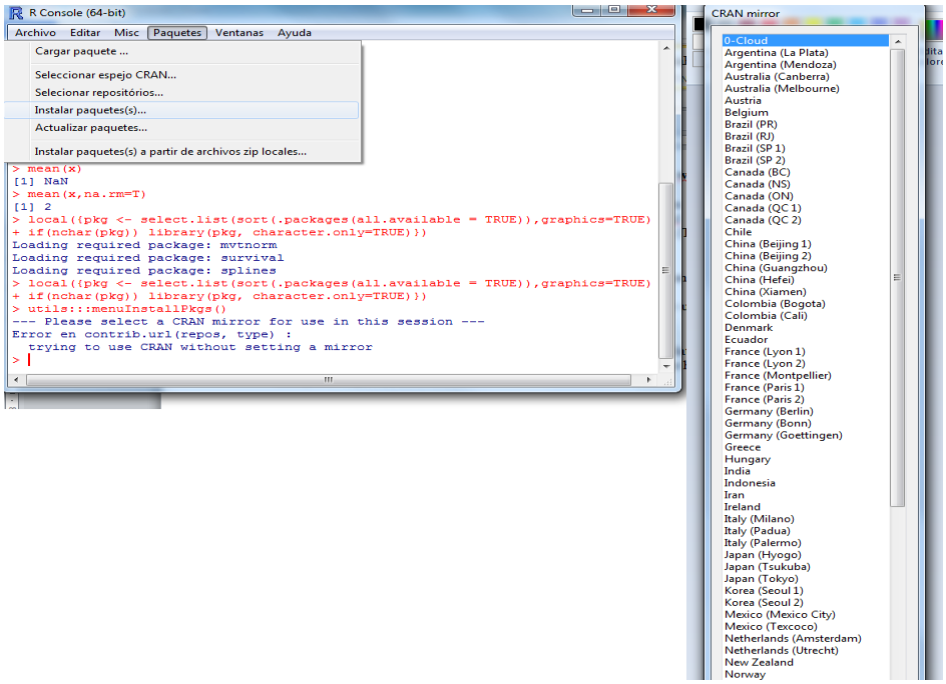
LIBRERÍAS (PAQUETES)

Una de las ventajas de trabajar con R es que hay gran número de programas libres y disponibles como sofware gratuito en forma de paquete o librería.
Al iniciar una sesión en R se cargan una serie de librerías básicas, pero en muchas ocasiones es necesario cargar librerías específicas que tengan implementadas algunas funciones concretas.

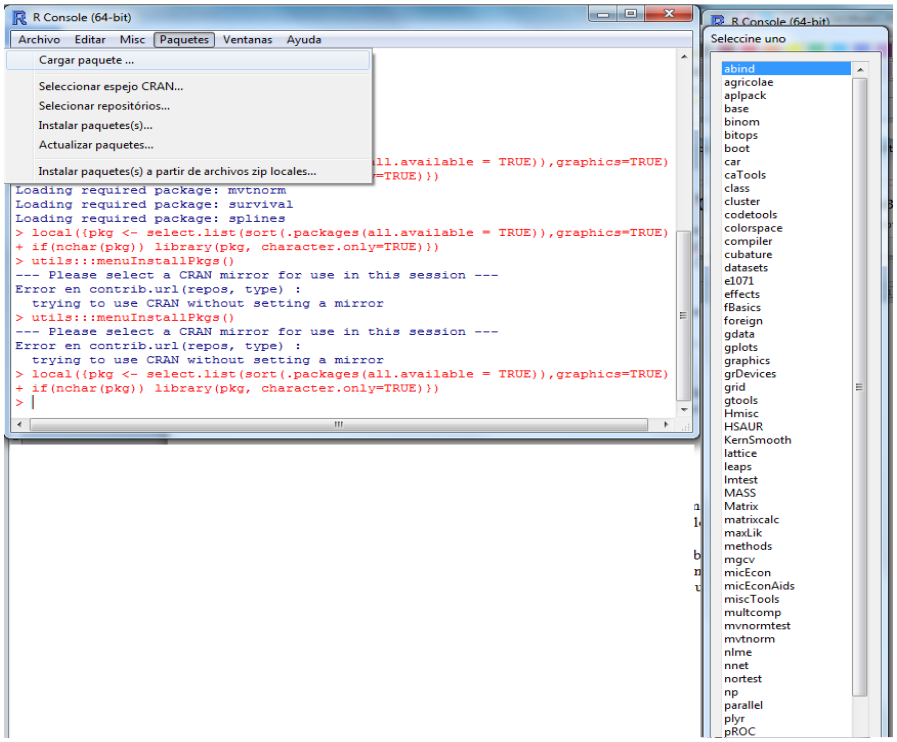
Librerías básicas (se cargan por defecto)
base: Contiene las bases de R.
normtest: contiene test de normalidad
nls: Librería para llevar a cabo regresiones no lineales.
mva: Análisis multivariante.

Para cargar una librería de R se utiliza el menú P*aquetes* de la barra de comandos o hacerlo directamente con el comando
library(nombre de la librería).
Si no estuviera instalada, se puede hacer desde el menú *Paquetes* o usando el comando:
install.packages("nombre")
y luego cargarla en memoria con la orden

Ejemplo:
install.packages(“mvtnorm”) ## Para trabajar con normales multivariantes
library(mvtnorm) ## carga la librería para poder trabajar con ella



Y luego cargarla en memoria también desde el menú paquetes



Entre los distintos paquetes que se cargarán tenemos el Rcmdr que nos permitirá el acceso al interfaz de ejecución de R con menús para los procedimientos estadísticos más usuales.
Y una lista completa de los paquetes disponible puede verse en
http://cran.r-project.org/web/packages/

help.start() buca ayuda en internet y nos da acceso a los paquetes disponibles.
library (help= '……') da información sobre las rutinas del paquete

paquete **xtable** se utiliza para escribir en latex. Es muy útil para escribir informes porque permite coger las salidas del R y crear el código necesario para que esa tabla pueda ser usada en latex.

a<- objeto tipo tabla
print(xtable(a)) # crea el código latex para imprimir a

Para controlar el tiempo de ejecución de ciertas órdenes
system.time({ ……})

Bibliografía de referencia para R.

https://cran.r-project.org/doc/contrib/R-intro-1.1.0-espanol.1.pdf
http://knuth.uca.es/repos/ebrcmdr/pdf/13marzo/ebrcmdr.pdf
https://cran.r-project.org/doc/contrib/Saez-Castillo-RRCmdrv21.pdf
Probability and Statistics with R. Ugarte, D., Militin, A., Arnholt, A. Chapman and Hall