# Tema 1

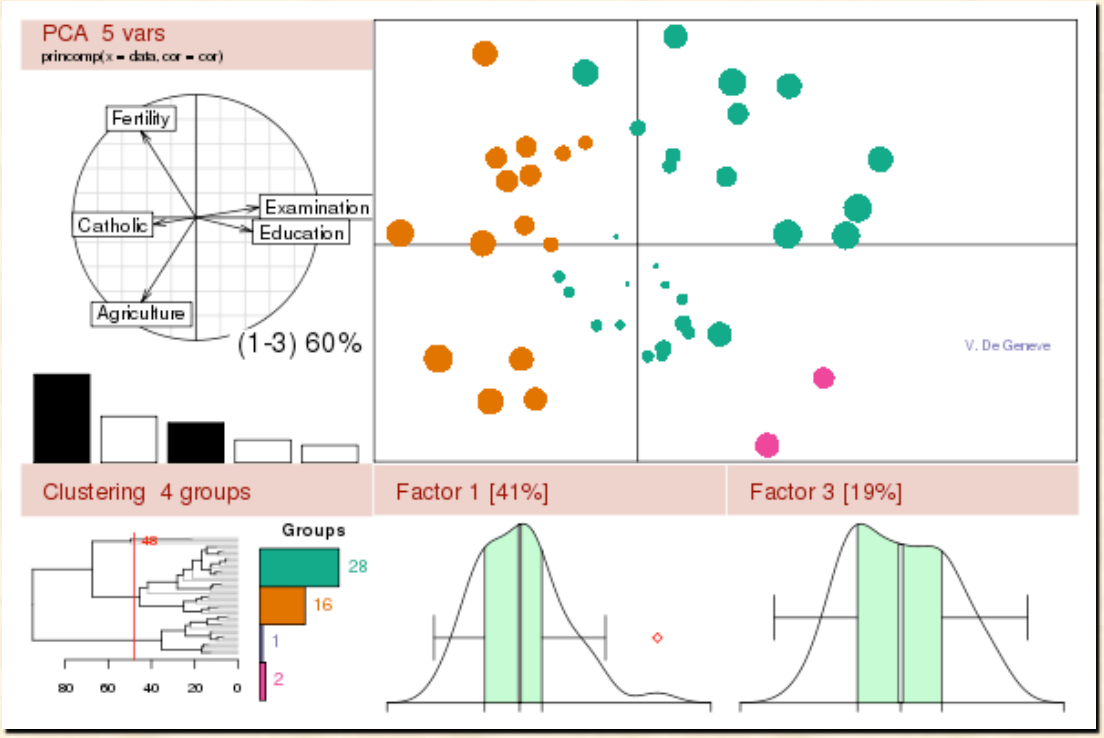
## Introducción al paquete de software *R*

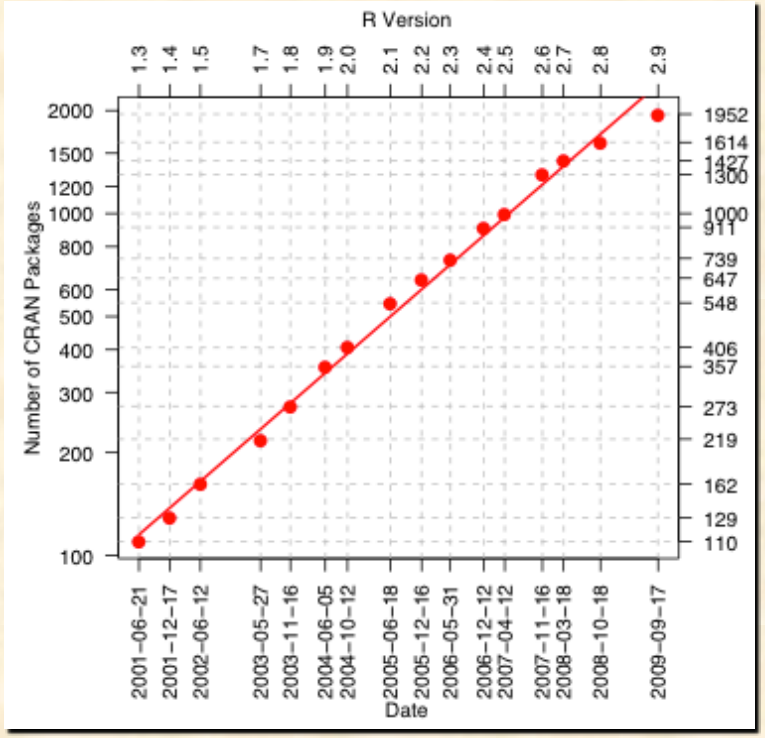
- Introducción
  - Instalación. *R* Studio.
- Características principales
- Estructuras de datos
  - Vectores
  - Matrices y arrays
  - Factores
  - Listas
  - Data frames
  - Tablas
  - Funciones
  - Valores especiales
  - Subgrupos
- Algunas operaciones básicas
- Estructuras de control
- Lectura y escritura de datos
- Paquetes y ejemplos en R
- Gráficas
- Referencias

# Introducción

## ¿Por qué *R*?

- Es un entorno integrado, no una colección de herramientas, especialmente desarrollado para el análisis de datos, los cálculos estadísticos y las representaciones gráficas.
- Es un lenguaje de programación muy sencillo
- Es software LIBRE
- Disponible para diferentes plataformas (Unix, MacOS, Windows)
- Muy usado en la investigación científica

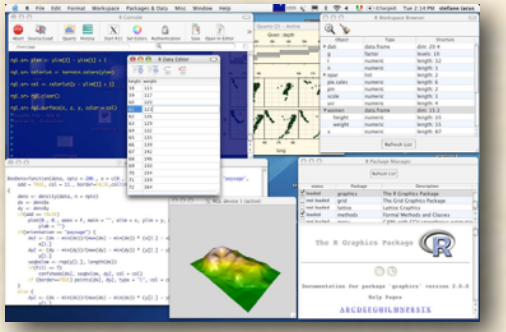


# Instalación

Directamente desde: *The R Project for Statistical Computing* (http://www.r-project.org)

Ver también CRAN (*Comprehensive R Archive Network*) (http://cran.r-project.org)



**Download and Install R**

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

Última versión: R 3.3.1 (21/06/16)

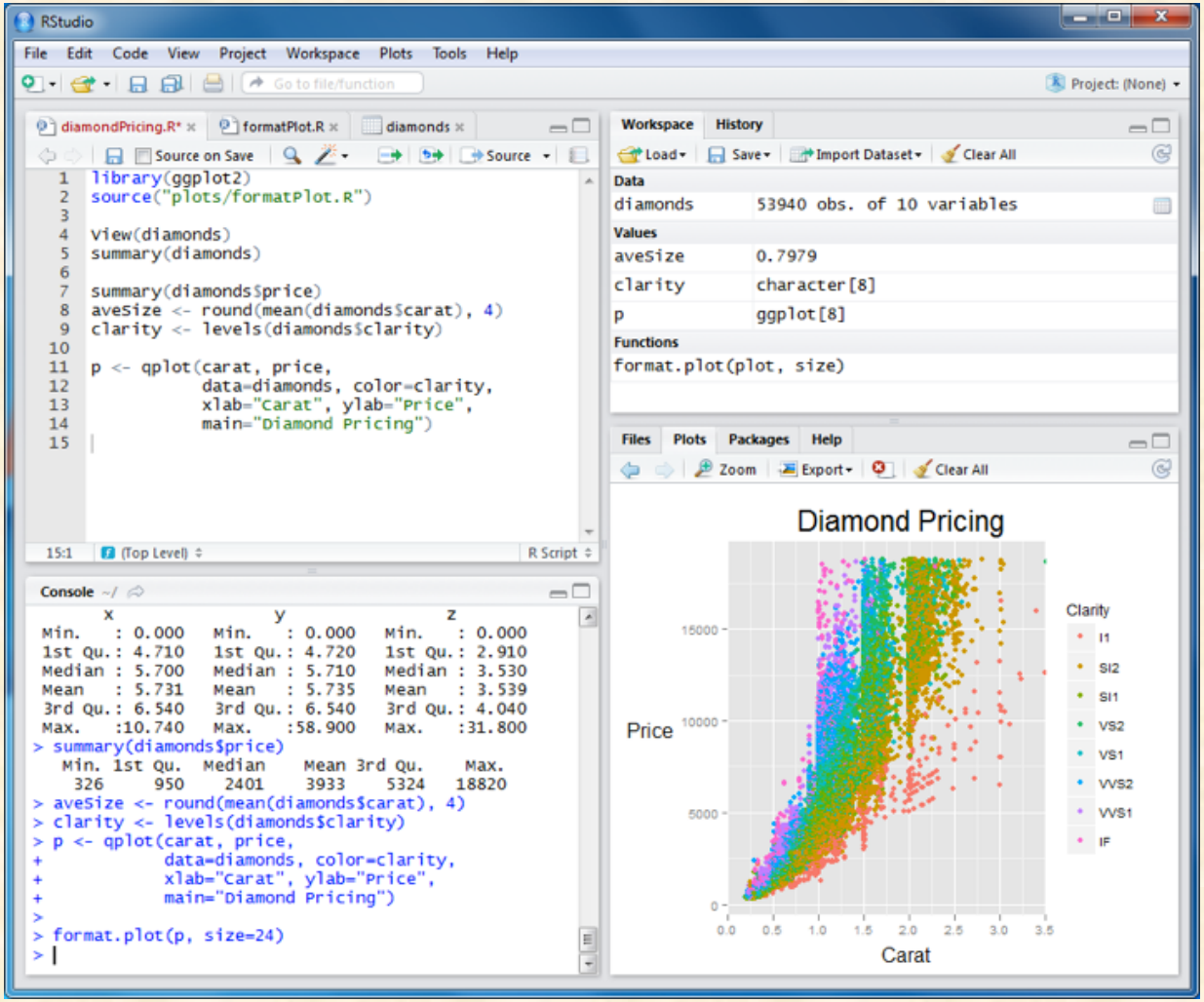
http://www.rstudio.com

# Interface de usuario para *R*, potente, sencillo

Software LIBRE. Versiones para linux, Mac, Windows

## Diferentes ventanas para:

- Scripts
- Consola de comandos
- Lista de variables
- Variables individuales
- Historial
- Gráficas
- Manejo de paquetes
- Directorio
- Help



TUTORIAL BASADO EN ***First Steps in R*** por M. Ceballos y N. Cardiel
(http://venus.ifca.unican.es/Rintro/)

# Características principales

Antes de empezar a trabajar con R conviene crear un subdirectorio exclusivo para el trabajo en R, o, mucho mejor, un directorio para cada proyecto diferente

Arrancando R:

en linux
```
[user@pc]$ mkdir work
[user@pc]$ cd work
[user@pc work]$ R
```

o arrancando *RStudio* desde el icono correspondiente

Saliendo de R:

```
> quit()
Save workspace image? [y/n/c]: # posibilidad de salvar información
> quit(save="no")              # para la próxima sesión
> Ctrl-D
```

**Cuidado**: R distingue entre mayúsculas y minúsculas:

```
> a = 1
> A = 2
> a == A
[1] FALSE
```

Se puede usar la tecla TAB para completar comandos

```
> his<Tab>
hist          hist.default
history
```

En *RStudio* situando el ratón sobre las diferentes opciones se obtiene información

## Buscando AYUDA:

```
> help.start()                          # abre ventana con web para ayuda
> help("pp")                            # ayuda con la función "pp"
> ?pp                                   # ayuda con la función "pp"
> help.search("pp")                     # busca apariciones de la cadena "pp"
> ??pp                                  # busca apariciones de la cadena "pp"
> apropos("pp", mode="function")        # lista funciones con "pp" en el nombre
> example(topic)                        # corre el código en R con ejemplos en un tópico
                        # determinado (por ejemplo “example(plot)”)
```

o usar directamente el buscador del sistema de ayuda de *RStudio*

## Directorios y entrada/salida

```
> R.home()                              # vuelve al directorio raíz de R
[1] "/usr/lib64/R"
> getwd()                               # vuelve al directorio de trabajo
[1] "/home/user/R"
> setwd("/home/user/newRdir")           # establece un nuevo directorio de trabajo
> dir()                                 # muestra el contenido del directorio actual
> ls()...                               # muestra el contenido del espacio de trabajo
> history(n)                            # muestra los últimos n comandos
> source("filename.R")                  # ejecuta comandos en el script filename.R
> sink("register.txt")                  # envía la salida de R a un fichero externo
> sink()                                # termina el envío anterior (vuelve a terminal)
> save.image("filename”)                # guarda espacio de trabajo
> load("filename”)                      # carga espacio de trabajo
> save(objectlist,file="filename”)      # guarda objetos específicos
> options()                             # muestra las opciones
> options(opt=...)                      # fija una opción
```

# Estructuras de datos

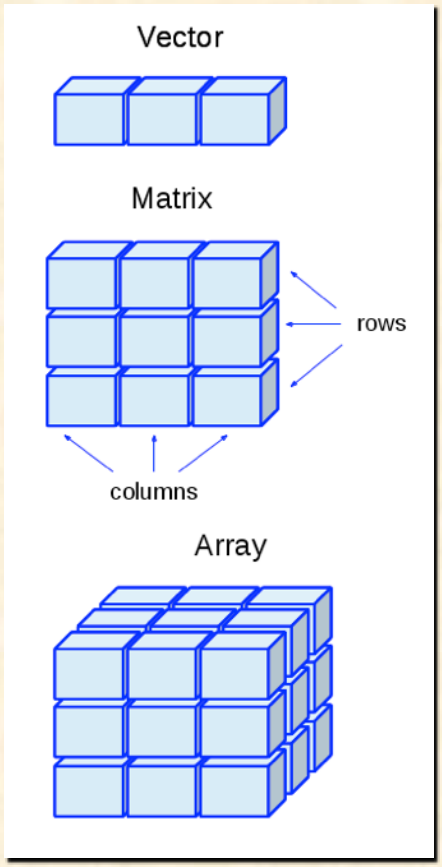
*R* es un lenguaje orientado a objetos (un objeto es algo que puede ser asignado a una variable)

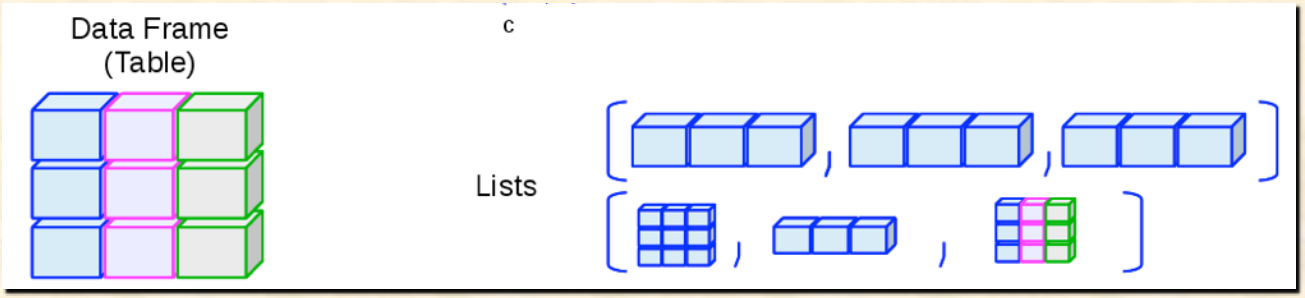
## TIPOS DE ESTRUCTURAS

- **Vectores** (unidimensionales)
- **Matrices** (bidimensionales)
- ***Arrays*** (multidimensionales)

de tipo (*mode*): *numeric*, *complex*, *logical*, *character*)

- **Factores** (vectores de variables categóricas, para agrupar los componentes de otro vector)
- **Listas** (colección de objetos, cada uno puede ser de un tipo)
- ***Data Frames*** (generalización de matrices; cada fila es un elemento, cada columna una variable de diferente tipo)
- **Funciones** (objetos creados para hacer operaciones)




Función: "`str(objeto)`": da el tipo e información sobre el objeto

# Vectores numéricos

Asignación de valores

```
> a <- 1.7
> 1.7 -> a
> a = 1.7
> assign("a", 1.7)
> b <- c(10, 11, 15, 19)
```

(concatenar)

Mostrar valores

```
> a
[1] 1.7
> print(a)
[1] 1.7
> b
[1] 10 11 15 19
```

Las operaciones se ejecutan sobre todos los elementos de la variable:

```
> b*b
[1] 100 121 225 361
> 1/b
[1] 0.10000000 0.09090909 0.06666667 0.05263158
> c <- b-1
> c
[1]  9 10 14 18
```

Refiriéndose a elementos:

```
> x[n]    # elemento en la posición n
> x[c(n1,n2)]   # elemento en las
                # posiciones n1 y n2
> x[n1:n2] # elemento en las
           # posiciones de n1 a n2
```

Generando una secuencia:

```
> 2:10
[1]  2  3  4  5  6  7  8  9 10
> 5:1
[1] 5 4 3 2 1
> seq(from=1, to=10, by=3)
[1]  1  4  7 10
> seq(1, 10, 3)
[1]  1  4  7 10
> seq(length=10, from=1, by=3)
[1]  1  4  7 10 13 16 19 22 25 28
```

Generando repeticiones:

```
> a <- 1:3; b <- rep(a, times=3); c <- rep(a, each=3)
> a
[1] 1 2 3
> b
[1] 1 2 3 1 2 3 1 2 3
> c
[1] 1 1 1 2 2 2 3 3 3
```

Pueden combinarse vectores de diferente tamaño siempre que el tamaño de uno sea múltiplo del otro

# Vectores lógicos

Toma valores TRUE o FALSE

```
> a <- seq(1:10)
> a
[1]  1  2  3  4  5  6  7  8  9 10
> b <- (a>5)                          # asigna valores TRUE o FALSE a partir
                  # de una desigualdad
> b
[1] FALSE FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE
> a[b]                                # muestra los valores que cumplen una condición
[1]  6  7  8  9 10
> a[a>5]                              # igual, pero evitando variables intermedias
[1]  6  7  8  9 10
```

# Vectores con caracteres

```
> a <- "This is an example"
> a
[1] "This is an example"
> x <- 1.5
> y <- -2.7
> paste("Point is (",x,",",y,")", sep="")     # concatenando x, y más un texto usando 'paste'
[1] "Point is (1.5,-2.7)"

> a <- "This is an example"                   # extrayendo una parte de la cadena
> substr(a,6,11)
[1] "is an "
```

## Matrices

Variable bidimensional

```
> a <- matrix(1:12, nrow=3, ncol=4)
> a
   [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> dim(a)
[1] 3 4
> a <- matrix(1:8, nrow=4, ncol=4, byrow=TRUE)
> a
     [,1] [,2] [,3] [,4]
[1,]    1    2    3    4
[2,]    5    6    7    8
[3,]    1    2    3    4
[4,]    5    6    7    8
```

Los elementos se reciclan si hay menos elementos que espacios a a rellenar.

Si no se dice lo contrario, se rellenan por columnas, para hacerlo por filas incluir "byrow=TRUE"

Refiriéndose a elementos:

```
> x[i,]     # fila i
> x[,j]     # columna j
> x[i,j]    # elemento i,j
```

## *Arrays*

Variable n-dimensional

```
> z <- array(1:24, dim=c(2,3,4))
> z
, , 1
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

, , 2
     [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12

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

, , 4
     [,1] [,2] [,3]
[1,]   19   21   23
[2,]   20   22   24
> dim(z)
[1] 2 3 4
```

# Factores

Vectores que contienen información categórica útil para agrupar los elementos de otros vectores del mismo tamaño

```
> bv <- c(0.92,0.97,0.87,0.91,0.92,1.04,0.91,0.94,0.96,
+         0.90,0.96,0.86,0.85)                          # colores (B-V) para 13 galaxias
> morfo <- c("Sab","E","Sab","S0","E","E","S0","S0","E",
+            "Sab","E","Sab","S0")                      # información morfológica
> length(morfo)                                        # comprobamos que es el mismo tamaño
[1] 13
> fmorfo <- factor(morfo)                              # se crea el factor con 'factor()'
> fmorfo
[1] Sab E   Sab S0  E   E   S0  S0  E   Sab E   Sab S0
Levels: E S0 Sab                                       # muestra los diferentes factores
> levels(fmorfo)                                       # muestra los niveles
[1] "E"   "S0"  "Sab"
> bv[fmorfo=="E"]                                      # separa un determinado nivel
[1] 0.97 0.92 1.04 0.96 0.96
> mean(bv[fmorfo=="E"])                                # hace una operación sobre un nivel
[1] 0.97
```

Útil para identificar submuestras y realizar operaciones sólo sobre sus elementos

# Listas

Colección ordenadas de objetos, donde se pueden agrupar objetos de diferentes tipos (por ejemplo una combinación de vectores, matrices, factores, otras listas, etc.)

```
> gal <- list(name="NGC3379", morf="E", T.RC3=-5, colours=c(0.53,0.96))
> gal
$name
[1] "NGC3379"
$morf
[1] "E"
$T.RC3
[1] -5
$colours
[1] 0.53 0.96
> gal$<Tab>                    # con la tecla Tab después de $ se muestran los elementos
gal$name     gal$morf     gal$T.RC3    gal$colours
> length(gal)                  # muestra cuántos elementos tiene
[1] 4
> names(gal)                   # muestra los nombres de los elementos
[1] "name"    "morf"    "T.RC3"   "colours"
> gal$radio <- TRUE                          # pueden añadirse nuevos elementos
> gal$redshift <- 0.002922                   # add a numeric element
> names(gal)
[1] "name"     "morf"     "T.RC3"    "colours"  "radio"    "redshift"
> list123 <- c(list1, list2, list3) # se pueden concatenar varias listas para producir otra
```

Muy versátiles porque pueden almacenar cualquier tipo de información, pero pueden convertirse en estructuras muy complejas.

# Data frames

Tipo especial de lista de gran utilidad estadística. Sus componente son vectores, o factores, de igual longitud.

La información se organiza en una tabla. Típicamente, cada fila corresponde a un elemento de la muestra. Cada columna a una variable medida en toda (o parte de) la muestra.

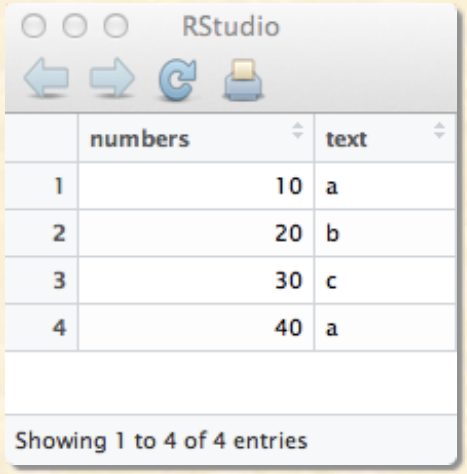
Los elementos dentro de cada columna son del mismo tipo. Cada columna puede ser de un tipo diferente.

Se crean con la función: "`data.frame(...)`"

Cada columna (variable) tiene un título o nombre

**Cuidado**: Al crearse, los caracteres se convierten automáticamente en factores. Para evitarlo usar: "options(stringsAsFactors = `FALSE`)"

```
> options(stringsAsFactors = FALSE)
> df <- data.frame(numbers=c(10,20,30,40),text=c("a","b","c","a"))
> df
  numbers text
1      10    a
2      20    b
3      30    c
4      40    a
> df$text
[1] "a" "b" "c" "a"
> options(stringsAsFactors = TRUE)
> df <-data.frame(numbers=c(10,20,30,40),text=c("a","b","c","a"))
> df$text
[1] a b c a
Levels: a b c
```




En RStudio, pinchando sobre la variable en la ventana "*Environment*" se muestra la tabla

# Data frames (II)

```
> x <- data.frame(vector1,vector2,vector3) # vectores para cada columna
>                 # los nombres de cada variable (columna) serán "vector1", etc
```

Para referirse a las columnas: `x[n]`, `x[n1,n2]`, `x[c(n1,n2)]`

O por el nombre: `x[c("vector1","vector2")]`, `x$vector1`

Función: "`attach(x)`": en los comandos siguientes se usará el *dataframe* x por defecto, sin dar su nombre. Ej. `vector1` se referirá a `x$vector1`

Función: "`detach(x)`": termina la asignación automática

Se pueden identificar los elementos de la muestra (filas) indicando en qué columna están los nombres:

```
> x <- data.frame(vector1,vector2,row.names=vector1)
```

Anotando dataframes:

- `names(dataframe)`: da un vector con los nombres de las columnas
- `names(dataframe)[n] <- ...`: cambia el nombre para la columna n
- `row.names(dataframe)`: da un vector con los nombres de las filas

Añadiendo una nueva variable:

```
> dfname$y <- dfname$x1 + dfname$x2    # En los dos casos crea una nueva variable
> dfname <- transform(dfname, y=x1+x2) # "y" sumando los contenidos de x1 y x2
```

# Tablas

Se pueden generar factores a partir de una variable continua usando el comando `cut`. Si el parámetro `break` es un número, indica el número de intervalos. **Con `table` se construye una tabla de frecuencias**:

```
> bv <- c(0.92,0.97,0.87,0.91,0.92,1.04,0.91,0.94,0.96,
+         0.90,0.96,0.86,0.85)                  # colores (B-V) para 13 galaxias
> fbv <- cut(bv,breaks=3)                       # divide 'bv' en 3 intervalos de igual longitud
> fbv                                           # muestra en que intervalo está cada galaxia
[1] (0.913,0.977] (0.913,0.977] (0.85,0.913]  (0.85,0.913]  (0.913,0.977]
[6] (0.977,1.04]  (0.85,0.913]  (0.913,0.977] (0.913,0.977] (0.85,0.913]
[11] (0.913,0.977] (0.85,0.913]  (0.85,0.913]
Levels: (0.85,0.913] (0.913,0.977] (0.977,1.04]
> table(fbv)                                    # genera una tabla contando el número de galaxias
fbv                                             # (frecuencias) en cada intervalo
  (0.85,0.913] (0.913,0.977]  (0.977,1.04]
             6             6             1
```

Si el parámetro `break` es un vector, indica los extremos de los intervalos:

```
> ffbv <- cut(bv,breaks=c(0.80,0.90,1.00,1.10))
> table(ffbv)
ffbv
  (0.8,0.9]   (0.9,1]   (1,1.1]
          4         8         1
```

Para dividir por cuantiles, usar la opción `quantile`:

```
> ffffbv <- cut(bv,quantile(bv,(0:4)/4))
> table(ffffbv)
ffffbv
  (0.85,0.9]  (0.9,0.92] (0.92,0.96]
(0.96,1.04]
           3           4           3           2
```

Se puede usar la opción `pretty` para dividir en n intervalos más o menos redondeados:

```
> fffbv <- cut(bv,pretty(bv,3))
```

# Tablas multidimensionales

Construcción de tablas multidimensionales

```
> heights <- c(1.64,1.76,1.79,1.65,1.68,1.65,1.86,1.82,1.73,1.75,1.59,1.87,
+              1.73,1.57,1.63,1.71,1.68,1.73,1.53,1.82)
> weights <- c(64,77,82,62,71,72,85,68,72,75,81,88,72,71,74,69,81,67,65,73)
> ages <- c(12,34,23,53,23,12,53,38,83,28,28,58,38,63,72,44,33,27,32,38)
> fheights <- cut(heights,c(1.50,1.60,1.70,1.80,1.90))
> fweights <- cut(weights,c(60,70,80,90))
> fages    <- cut(ages,seq(10,90,10))
```

Tabla entre dos variables:

```
> ta <- table(fheights, fweights)
> ta
           fweights
fheights    (60,70] (70,80] (80,90]
  (1.5,1.6]       1       1       1
  (1.6,1.7]       2       3       1
  (1.7,1.8]       2       4       1
  (1.8,1.9]       1       1       2
```

Añadiendo frecuencias marginales:

```
> addmargins(ta)
           fweights
fheights    (60,70] (70,80] (80,90] Sum
  (1.5,1.6]       1       1       1   3
  (1.6,1.7]       2       3       1   6
  (1.7,1.8]       2       4       1   7
  (1.8,1.9]       1       1       2   4
  Sum             6       9       5  20
```

Frecuencias relativas (con el comando `prop.table`):

```
> tta <- prop.table(ta)
> addmargins(tta)
           fweights
fheights    (60,70] (70,80] (80,90]  Sum
  (1.5,1.6]    0.05    0.05    0.05 0.15
  (1.6,1.7]    0.10    0.15    0.05 0.30
  (1.7,1.8]    0.10    0.20    0.05 0.35
  (1.8,1.9]    0.05    0.05    0.10 0.20
  Sum          0.30    0.45    0.25 1.00
```

De la misma forma se pueden crear tablas tridimensionales, etc

```
> table(fheights, fweights, fages)
```

# Matrices y tablas

Se pueden crear tablas bidimensionales a partir de matrices:

```
> mtab <- matrix(c(30,12,47,58,25,32), ncol=2, byrow=TRUE)
> colnames(mtab) <- c("ellipticals","spirals")
> rownames(mtab) <- c("sample1","sample2","new sample")
> mtab
           ellipticals spirals
sample1             30      12
sample2             47      58
new sample          25      32
```

Pero no son “tablas reales”. Para transformarlas hacer:

```
> rtab <- as.table(mtab)
```

```
> class(mtab);class(rtab)
[1] "matrix"
[1] "table"
```

El comando `summary` devuelve información diferente para una matriz y para una tabla:

```
> summary(mtab)
       V1              V2
 Min.   :25.0   Min.   :12
 1st Qu.:27.5   1st Qu.:22
 Median :30.0   Median :32
 Mean   :34.0   Mean   :34
 3rd Qu.:38.5   3rd Qu.:45
 Max.   :47.0   Max.   :58
```

```
> summary(rtab)
Number of cases in table: 204
Number of factors: 2
Test for independence of all factors:
        Chisq = 9.726, df = 2, p-value = 0.007726
```

# Funciones

Objetos que pueden ser creados por el usuario para hacer, y repetir, operaciones específicas:

Ejemplo: función para calcular la desviación típica de un vector:

```
> stddev <- function(x) {
+   res = sqrt(sum((x-mean(x))^2) / (length(x)-1))
+   return(res)
+ }
```

Se pueden usar y definir funciones dentro de funciones.

El valor devuelto por una función es el resultado de la última expresión evaluada o el especificado con el comando `return`

Los argumentos de las funciones pueden especificarse por su posición o por su nombre.
Puede haber argumentos con valores por defecto.

```
> mynumbers <- c(1, 2, 3, 4, 5)
> stddev(mynumbers)
[1] 1.581139
> stddev(x = mynumbers)
[1] 1.581139
```

Ej. Función `sd` de *R* (calcula la desviación típica):

```
> sd(x=mynumbers)  # No se indica el valor por defecto
[1] 1.581139       # na.rm=FALSE (no elimina valores NA)
> sd(x=mynumbers, na.rm=TRUE) # Argumentos por nombre
[1] 1.581139
> sd(mynumbers, na.rm=TRUE)   # Argumentos por posición
[1] 1.581139
> sd(na.rm=TRUE, x=mynumbers) # OK pero no recomendado
[1] 1.581139
```

# Funciones iterativas

- **lapply** (calcula una función para todos los elementos de una lista)
- **sapply** (igual pero simplificando el resultado)
- **apply** (calcula una función para parte, columnas o filas, de una matriz)
- **tapply** (calcula una función para un subconjunto de un vector; ej. usando un factor)

```
> a <- matrix(1:12, nrow=3, ncol=4)
> a
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> apply(a,1,mean)  # medias por filas ("1")
[1] 5.5 6.5 7.5
> rowMeans(a)
[1] 5.5 6.5 7.5
> apply(a,1,sum)
[1] 22 26 30
> rowSums(a)
[1] 22 26 30
> apply(a,2,mean)# medias por columnas ("2")
[1]  2  5  8 11
> apply(a,2,sum)
[1]  6 15 24 33
```

```
> bv.list <-  list(
+   colsSab=c(0.92,0.87,0.90,0.86),
+   colsE=c(0.97,0.92,1.04,0.96,0.96),
+   colsSO=c(0.91,0.91,0.94,0.85))
> lapply(bv.list, mean)
$colsSab           # devuelve una lista
[1] 0.8875
$colsE
[1] 0.97
$colsSO
[1] 0.9025
> sapply(bv.list, mean) # devuelve un vector
colsSab   colsE  colsSO
0.8875  0.9700  0.9025
```

```
> bv <- c(0.92,0.97,0.87,0.91,0.92,1.04,0.91,
+         0.94,0.96,0.90,0.96,0.86,0.85)
> morfo <- c("Sab","E","Sab","S0","E","E","S0",
+            "S0","E","Sab","E","Sab","S0")
> fmorfo <- factor(morfo)       # crea factor
> tapply(bv,fmorfo,mean)
     E     S0    Sab
0.9700 0.9025 0.8875
# aplica la función media a los colores
segregando por tipo morfológico
```

# Valores especiales

- ***NA*** (*Not available*): valores no disponibles
- ***NaN*** (*Not a Number*): valores numéricos imposibles o sin sentido (ej. imaginarios, etc.)
- ***Inf***: valores infinitos

```
> a <- c(0:2, NA, NA, 5:7)
> a
[1]  0  1  2 NA NA  5  6  7
> a*a
[1]  0  1  4 NA NA 25 36 49
> unavail <- is.na(a)
> unavail
[1] FALSE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE
```

En muchas funciones el argumento `na.rm` se puede usar para eliminar o no los NAs de los cálculos

```
> a <- c(0:2, NA, NA, 5:7)
> a
[1]  0  1  2 NA NA  5  6  7
> mean(a)
[1] NA
> mean(a, na.rm=TRUE)
[1] 3.5
```

Se puede seguir operando con estructuras con elementos que tomen estos valores.

```
> a <- log(-1)
> a
[1] NaN
> a <- 1/0; b <- 0/0; c <- log(0)
> d <- c(a,b,c)
> d
[1]  Inf  NaN -Inf
> 1/Inf
[1] 0
> is.infinite(d)
[1]  TRUE FALSE  TRUE
> is.nan(d)
[1] FALSE  TRUE FALSE
```

`is.na()` sirve para comprobar si tenemos *NAs*

`is.nan()` sirve para comprobar si tenemos *NaNs*

`is.infinite()` sirve para comprobar si tenemos *Infs*

# Subgrupos

Se pueden extraer subgrupos o elementos particulares de otras estructuras de datos. Gran utilidad.

- `[ ]` : extrae uno o varios elementos, misma clase
- `[[ ]]` :extrae un elemento de un *dataframe* o lista, puede ser de diferente clase
- `$`: extrae elementos de una variable con un cierto nombre de un *dataframe* o una lista

```
> a <- 1:15
> a <- a*a
> a
[1]   1   4   9  16  25  36  49  64  81 100 121 144
169 196 225
> a[3]          # tercer valor del vector
[1] 9
> a[3:5]        # valores del tercero al noveno
[1]  9 16 25
> a[c(1,3,10)] # valores con posiciones determinadas
[1]   1   9 100
> a[-1]         # se elimina el primer valor
[1]   4   9  16  25  36  49  64  81 100 121 144 169
196 225
> a[c(-1,-3,-5,-7)]  # se eliminan varios valores
[1]   4  16  36  64  81 100 121 144 169 196 225
> a[a>100]      # valores que cumplen una condición
[1] 121 144 169 196 225
> b <- c("A", "B", "C", "C", "D", "E")
> b[b>"C"]      # también con caracteres
[1] "D" "E"
```

Para una matriz se usan dos subíndices (fila, columna):

```
> a <- matrix(1:12, nrow=3, ncol=4)
> a
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> a[2,3]   # 2ª fila, 2ª columna
[1] 8
> a[[2,3]]
[1] 8
> a[2,]    # 2ª fila
[1]  2  5  8 11
> a[,3]    # 3ª columna
[1] 7 8 9
```

(los resultados son vectores, no matrices)

# Subgrupos (II)

Pueden accederse a elementos de una matriz con índices almacenados en matrices auxiliares:

```
> ind <- matrix(c(1:3,3:1), nrow=3, ncol=2)
> ind    # matrix auxiliar para los índices i,j
     [,1] [,2]
[1,]    1    3
[2,]    2    2
[3,]    3    1
> a[ind] <- 0 # se ponen a 0 los índices  especificados
> a           #  en 'ind' (1,3), (2,2), (3,1)
     [,1] [,2] [,3] [,4]
[1,]    1    4    0   10
[2,]    2    0    8   11
[3,]    0    6    9   12
```

Para extraer elementos de una lista:

```
> gal <- list(name="NGC3379", morf="E", colours=c(0.53,0.96))
> gal[3]          # devuelve el tercer elemento como una lista
$colours          # con un elemento llamado "colours"
[1] 0.53 0.96
> gal["colours"]  # hace lo mismo
$colours
[1] 0.53 0.96

> gal[[3]]        # accede al tercer elemento de la lista
[1] 0.53 0.96     # y devuelve solo la secuencia
> gal[["colours"]]# hace lo mismo
[1] 0.53 0.96

> gal$colours     # elemento con el nombre 'colours'
[1] 0.53 0.96     # (igual que con [[ ]])
> gal$colours[1]
[1] 0.53
```

Para extraer recursivamente un elemento:

```
> gal[[c(3,1)]]
[1] 0.53
> gal[[3]][[1]]
[1] 0.53

> gal[c(3,1)]
$colours
[1] 0.53 0.96
$name
[1] "NGC3379"
```

Nótese la diferencia entre [ ] y [[ ]]

# Subgrupos (III)

Los nombres de las variables pueden aproximarse a sus primeros caracteres:

```
> gal <- list(name="NGC3379", morf="E", colours=c(0.53,0.96))
> gal$na
[1] "NGC3379"
> gal[["na"]]
NULL
> gal[["na", exact=FALSE]]
[1] "NGC3379"
```

```
> airquality   # dataframe en la librería de R
> airquality[1:5, ]
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
> class(airquality[1:5, ])
[1] "data.frame"
> airquality[1,1]
[1] 41
> airquality[[1,1]]
[1] 41
> airquality[1,]
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
> class(airquality[1,])
[1] "data.frame"
> as.numeric(airquality[1,])
[1]  41.0 190.0   7.4  67.0   5.0   1.0
```

Ejemplo de extracción de elementos de un *dataframe:*

```
> airquality$Ozone # extrae una col. en un vector
  [1]  41  36  12  18  NA  28  23  19   8  NA   7
 [12]  16  11  14  18  14  34   6  30  11   1  11
 [23]   4  32  NA  NA  NA  23  45 115  37  NA  NA
 [34]  NA  NA  NA  NA  29  NA  71  39  NA  NA  23
 [45]  NA  NA  21  37  20  12  13  NA  NA  NA  NA
 [56]  NA  NA  NA  NA  NA  NA 135  49  32  NA  64
 [67]  40  77  97  97  85  NA  10  27  NA   7  48
 [78]  35  61  79  63  16  NA  NA  80 108  20  52
 [89]  82  50  64  59  39   9  16  78  35  66 122
[100]  89 110  NA  NA  44  28  65  NA  22  59  23
[111]  31  44  21   9  NA  45 168  73  NA  76 118
[122]  84  85  96  78  73  91  47  32  20  23  21
[133]  24  44  21  28   9  13  46  18  13  24  16
[144]  13  23  36   7  14  30  NA  14  18  20
> class(airquality$Ozone)
[1] "integer"
```

# Subgrupos de un dataset

Fusionando datasets: Es necesario tener una variable común que sirva de ancla:

```
> newdataframe <- merge(dataframe1,dataframe2,by="variable_ancla")
> newdataframe <- cbind(dataframe1,dataframe2)
```

Añadiendo filas: (deben tener el mismo número de variables):

```
> newdataframe <- rbind(dataframe1,dataframe2)
```

Seleccionando o excluyendo variables:

```
> new <- dataframe[c(n1,n2)]                    # seleccionando variables
> new <- dataframe[c("name1","name2",...)]
> new <- dataframe[c(-n1,-n2)]                  # excluyendo variables
```

Seleccionando elementos de acuerdo con un criterio según las variables:

```
> new <- dataframe[which(dataframe$x1 > 0 & dataframe$x2 < 10),]
```

Función `subset(dataframe,subset,select)` (subset=condiciones para mantener los elementos, select=variables que se extraen)

```
> subset(dataframe, age>35 | age<24, select=c("name1","name2")
> subset(dataframe, gender == "M", select=name1:name2)
```

Extrayendo muestras aleatorias:

```
> new <- dataframe[sample(1:nrow(dataframe),n,replace=FALSE),]
```

(`sample` extrae n números aleatorios)

< <= > >= == !=

!x (not x) x|y (x ó y) x&y (x e y)

isTRUE(x)

# Subgrupos: eliminando *NAs*

Eliminando *NAs* con la función `is.na()`:

```
> a <- c(0:2, NA, NA, 5:7)
> aa <- a[!is.na(a)]      # "!" indica negación
> aa                      # de la función
[1] 0 1 2 5 6 7           # Nuevo vector sin NAs
```

Para eliminarlos de vectores multidimensionales: `complete.cases`

```
> a <- c( 1,  2, 3, NA, 5, NA, 7)
> b <- c("A","B",NA,"D",NA,"E","F")

> valsok <- complete.cases(a,b)
> valsok     # vector lógico con las posiciones sin NAs
[1]  TRUE  TRUE FALSE FALSE FALSE FALSE  TRUE

> a[valsok]
[1] 1 2 7
> b[valsok]
[1] "A" "B" "F"
```

Para *dataframes*:

```
> airquality
> airquality[1:7, ]
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
5    NA      NA 14.3   56     5   5
6    28      NA 14.9   66     5   6
7    23     299  8.6   65     5   7
```

```
> valsok <- complete.cases(airquality)
> airquality[valsok, ][1:7,]
  Ozone Solar.R Wind Temp Month Day
1    41     190  7.4   67     5   1
2    36     118  8.0   72     5   2
3    12     149 12.6   74     5   3
4    18     313 11.5   62     5   4
7    23     299  8.6   65     5   7
8    19      99 13.8   59     5   8
9     8      19 20.1   61     5   9
```

También con:

```
> newdata <- na.omit(airquality)
```

# Algunas operaciones básicas

```
> a <- c(7+4,7-4,7*4,7/4)
> a
[1] 11.00  3.00 28.00  1.75
> length(a)              # longitud de un vector
[1] 4
> dim(b)                 # Dimensión de un objeto
> c(min(a),max(a))       # valores mínimo y máximo
[1]  1.75 28.00
> which.min(a)           # índice del mínimo
[1] 4
> which.max(a)           # índice del máximo
[1] 3
> sort(a)                # ordena de menor a mayor
[1]  1.75  3.00 11.00 28.00
> sum(a)                 # suma de los valores
[1] 43.75
> prod(a)                # producto de los valores
[1] 1617
> cumsum(1:10)           # suma acumulativa
[1]  1  3  6 10 15 21 28 36 45 55
```

Gran cantidad de funciones disponibles:
`exp()`, `sin()`, `cos()`, `tan()`, `log()`, `log10()`, `beta(a,b)`, `gamma(x)`, etc

Las operaciones se pueden vectorizar (gran flexibilidad):

```
> a <- seq(10,30,10)
> b <- seq(1:3)
> a + b
[1] 11 22 33
> a * b
[1] 10 40 90
> a / b
[1] 10 10 10
> a > 5
[1] TRUE TRUE TRUE
> b == 2
[1] FALSE  TRUE FALSE
```

```
> m1 <- matrix(1:9, 3, 3)
> m2 <- matrix(11:19, 3, 3)
> m1 * m2    # producto elemento a elemento
     [,1] [,2] [,3]
[1,]   11   56  119
[2,]   24   75  144
[3,]   39   96  171
> m1 %*% m2          # producto de matrices
     [,1] [,2] [,3]
[1,]  150  186  222
[2,]  186  231  276
[3,]  222  276  330
```

# Más operaciones con objetos

Combinando objetos:

- `c(object1,object2)` : combina objetos secuencialmente en un vector
- `cbind(object1,object2)`: combina objetos en columnas sucesivas
- `rbind(object1,object2)`: combina objetos en filas sucesivas

- `head(object)` : muestra las primeras líneas
- `tail(object1)` : muestra las últimas líneas
- `ls()` : lista objetos
- `rm(object1,object2)`: elimina objetos

Ordenado los datos:

```
> a <-c(5,8,2,1,3)
> order(a)      # devuelve las posiciones
[1] 4 3 5 1 2   # de los elementos en orden
> sort(a)       # ordena los elementos
[1] 1 2 3 5 8
```

Comprobar el tipo de un objeto
(devuelve TRUE o FALSE)

- `is.numeric(...)`
- `is.character(...)`
- `is.vector(...)`
- `is.matrix(...)`
- `is.data.frame(...)`
- `is.factor(...)`
- `is.logical(...)`

Cambiar el tipo de un objeto

- `as.numeric(...)`
- `as.character(...)`
- `as.vector(...)`
- `as.matrix(...)`
- `as.data.frame(...)`
- `as.factor(...)`
- `as.logical(...)`

Editando un objeto: desde *RStudio* o con `fix(object)`

# Programando en *R*: estructuras de control

Normalmente las instrucciones se ejecutan secuencialmente. Con las estructuras de control se pueden repetir instrucciones o alterar su secuencia dependiendo de condiciones:

## Ejecución condicional:

```
> if(cond) expr
> if(cond) cons.expr  else  alt.expr
```

## Repeticiones y bucles:

```
> for (name in expr_1) expr
> while (condition) expr
> repeat expr
```

Se pueden usar varias estructuras aninadas con `{...}`

ejemplo:

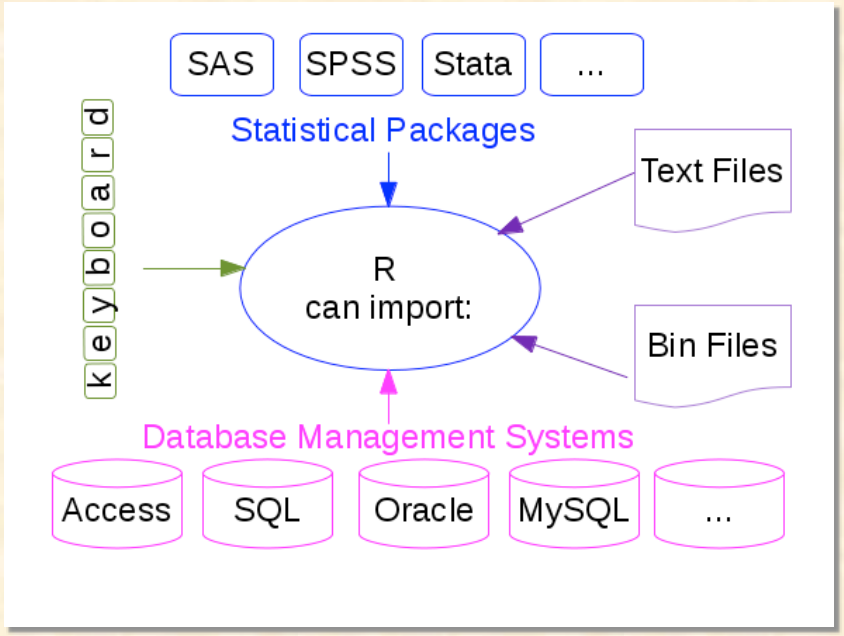
```
for (x in seq(-3,3)) {
  if (x < 0) {
    print("Caso A:")
    y <- sqrt(-x)
    cat("y=",y,"\n")
  } else {
    print("Caso B:")
    y <- sqrt(x)
    cat("y=",y,"\n")
  }
}
```

Flujo de control: comandos para interrumpir el flujo y alterar las estructuras anteriores:

- `break` : sale de un bucle `for`, `while` o `repeat`
- `next` : interrumpe la iteración en curso y avance al siguiente índice
- `return`: devuelve un valor y sale de una función

# Lectura y escritura de datos

Ver: http://cran.r-project.org/doc/manuals/R-data.pdf




Ej. fichero `galaxies.dat`

```
GALAXY morf T.RC3 U-B B-V
NGC1357 Sab 2    0.25 0.87
NGC1832 Sb  4   -0.01 0.63
NGC2276 Sc  5   -0.09 0.52
NGC3245 S0 -2    0.47 0.91
NGC3379 E  -5    0.53 0.96
NGC1234 Sab 3   -0.56 0.84
NGC5678 E  -4    0.45 0.92
```

***R*** puede importar datos de numerosas fuentes: paquetes estadísticos, bases de datos, ficheros de texto, binarios, hojas de cálculo, teclado, etc.

## Lectura de ficheros en ASCII

```
> gal <- read.table("galaxies.dat",header=TRUE)
```

(con cabecera). Los datos se almacenan en un *dataframe*:

```
> class(gal)
[1] "data.frame"
> names(gal)
[1] "GALAXY" "morf"   "T.RC3"  "U.B"    "B.V"
> gal$morf   # las cadenas de texto se convierten
[1] Sab Sb  Sc  S0  E               # en factores
Levels: E S0 Sab Sb Sc
> options(stringsAsFactors = FALSE)   # evitándolo
> gal <- read.table("galaxies.dat",header=TRUE)
> gal$morf
[1] "Sab" "Sb"  "Sc"  "S0"  "E"
> tapply(gal$U-B,gal$morf,mean) # color medio para
    E     S0    Sab     Sb     Sc      # cada tipo
0.490  0.470 -0.155 -0.010 -0.090
```

# Lectura y escritura de datos (II)

Puede especificarse la clase de datos para cada columna:

```
> gal <- read.table("galaxies.dat",header=TRUE,colClasses=
+   c("character","character","integer","numeric","numeric"))
```

Se puede especificar el separador:

```
> gal <- read.table("file.dat",sep="...",header=TRUE)
```

Si el separador es una coma puede usarse directamente:

```
> gal <- read.csv("file.csv",header=TRUE)
```

## Lectura de ficheros desde excel:

Pueden convertirse de xls a ficheros delimitados por "," y usar `read.csv`, o:

```
> gal <- read.xlsx("file.xlsx",n)
```

**n** es el número de página con los datos

Para ello es necesario tener instalado el paquete `xlsx` (ver más adelante)

---

Si el fichero sólo contiene números, puede leerse directamente a una matriz

```
> a <- matrix(data=scan("numbers.dat",0),ncol=3,byrow=TRUE)
Read 36 items
```

Si no se especifica `ncol`, se lee a un vector unidimensional

```
> a1 <- matrix(data=scan("numbers.dat",0))
Read 36 items
```

# Lectura y escritura de datos (III)

## Escritura de datos: Comandos `write.table` y `write.csv`

```
> # generamos unos vectores numéricos de interés
> x <- seq(-5,5)
> y <- x^3
> lista.output <- data.frame(x=x, y=y)

> write.table(lista.output, file="ejemplo1.txt")
> # el fichero generado tiene etiquetas para cada fila

> write.table(lista.output, file="ejemplo2.txt", row.names=FALSE)
> # sin etiquetas para cada fila
```

Para que en la lectura identifique bien la cabecera:

```
> # Lectura
> lista.input <- read.table("ejemplo1.txt")
> # interpreta bien la primera fila como cabecera ya que tiene esa
fila tiene una columna menos

> lista.input <- read.table("ejemplo2.txt", header=TRUE)
> # en este caso hay que decir que la primera fila es cabecera
```

`write.csv` : igual pero separa las columnas con comas (u otro separador)

```
> save(objectlist,file="filename")            # guarda objetos en binario
> save(objectlist,file="filename",ascii=TRUE) # en ascii
> save.image("filename")  o  save.image()     # guarda espacio de trabajo
> load("filename")                            # carga espacio de trabajo
```

ejemplo1.txt

```
"x" "y"
"1" -5 -125
"2" -4 -64
"3" -3 -27
"4" -2 -8
"5" -1 -1
"6" 0 0
"7" 1 1
"8" 2 8
"9" 3 27
"10" 4 64
"11" 5 125
```

ejemplo2.txt

```
"x" "y"
-5 -125
-4 -64
-3 -27
-2 -8
-1 -1
0 0
1 1
2 8
3 27
4 64
5 125
```

# Paquetes y ejemplos en *R*

R tienen muchos ejemplos y cientos de paquetes especializados.
Ver: http://cran.r-project.org/web/packages/

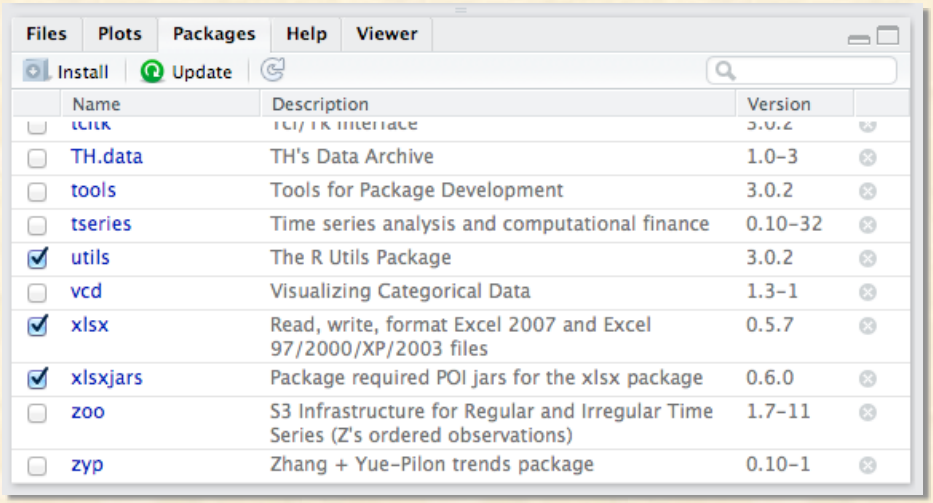
Lista de paquetes instalados:
```
> library()
```

Lista de paquetes cargados:
```
> search()
```

Ayuda sobre un determinado paquete:
```
> library(help=xlsx)
```

Para usar un paquete hay que 1) instalarlo, y 2) cargarlo:
```
> install.packages("xlsx")     #1
> library("xlsx")              #2
```

Todo este proceso se puede hacer de forma muy sencilla en la ventana "*Packages*" de ***RStudio***




Los datos disponibles en cada momento pueden verse con:
```
> data()
```

Y los correspondientes a un determinado paquete con:
```
> data(package="cluster")
```

# Gráficas

paquete básico `graphics`

- **plot**: nueva gráfica, ej. diagrama de dispersión
- **lines**: añade líneas a un gráfico abierto
- **abline**: añade una línea recta
- **segments**: añade segmentos
- **points**: añade puntos a un gráfico abierto
- **polygon**: añade polígonos
- **arrows**: añade flechas
- **text**: añade texto
- **title**: añade títulos a ejes, gráfico, etc.
- **axis**: añade y modifica propiedades de los ejes
- **hist**: histograma

El proceso para realizar un gráfico será:

```
> # 1. Abrir una salida gráfica
> # No es necesario desde RStudio
> pdf(myfile.pdf,width=10.,height=7.1)
> postscript(myfile.ps)
> png(myfile.png)
> jpeg(myfile.jpeg)

> # 2. Realizar el gráfico
> plot(x,y)

> # 3. Cerrar la salida gráfica
> # No es necesario desde RStudio
> dev.off()
```

Añadiendo texto:

```
> plot(x,y,main="titulo principal", sub="subtitulo",
+    xlab="título en x",ylab="titulo en y")
```

Cambiando los límites:

```
> plot(x,y,xlim=c(xmin,xmax),ylim=c(ylim,ymax))
```

Tipo de gráfico:

```
> plot(x,y,type="p") # sólo puntos (default)
> plot(x,y,type="l") # sólo líneas
> plot(x,y,type="b") # líneas y puntos
> plot(x,y,type="s") # tipo escalera
> plot(x,y,type="h") # histograma
> plot(x,y,type="n") # no dibuja los datos
```

Anotaciones matemáticas:

```
> plot(x,y,xlab=expression(sum(hat(omega)[j]/N, j=1,10)))
```

$$\sum_{1}^{10} \hat{\omega}_j / N$$

# Gráficas: parámetros

```
> par() # Muestra los valores de los parámetros
> par("pch") # Muestra el valor de un determinado parámetro
> par(pch=2,optionname=valor) # Cambia el valor de los parámetros para la sesión
> ?par # ayuda
```

- **pch**: tipo de símbolo
- **cex**: tamaño de símbolo (1=def, 1.5=50% mayor, etc)
- **lty**: tipo de línea
- **lwd**: anchura de línea (1=def, 1.5=50% mayor, etc)
- **font**: fuente (1=plain, 2=bold, 3=italic, etc.)

Tipo de líneas (`lty=`)

1. continua
2. discontinua (rayas)
3. puntos
4. punto-raya
5. rayas largas
6. punto-raya larga

Símbolos (`pch=`):


Pueden modificarse los colores de los símbolos 21:25 con `col()` (para el contorno) y `bg()` (para el fondo)

Varias gráficas en la misma página:

```
> par(mfrow=c(nºfilas,nºcolumnas)) # llena los gráficos por filas
> par(mfcol=c(nºfilas,nºcolumnas)) # llena los gráficos por columnas
> par(mfrow=c(1,1))                # vuelve al defecto
```

Cambiando parámetros de parte del gráfico:

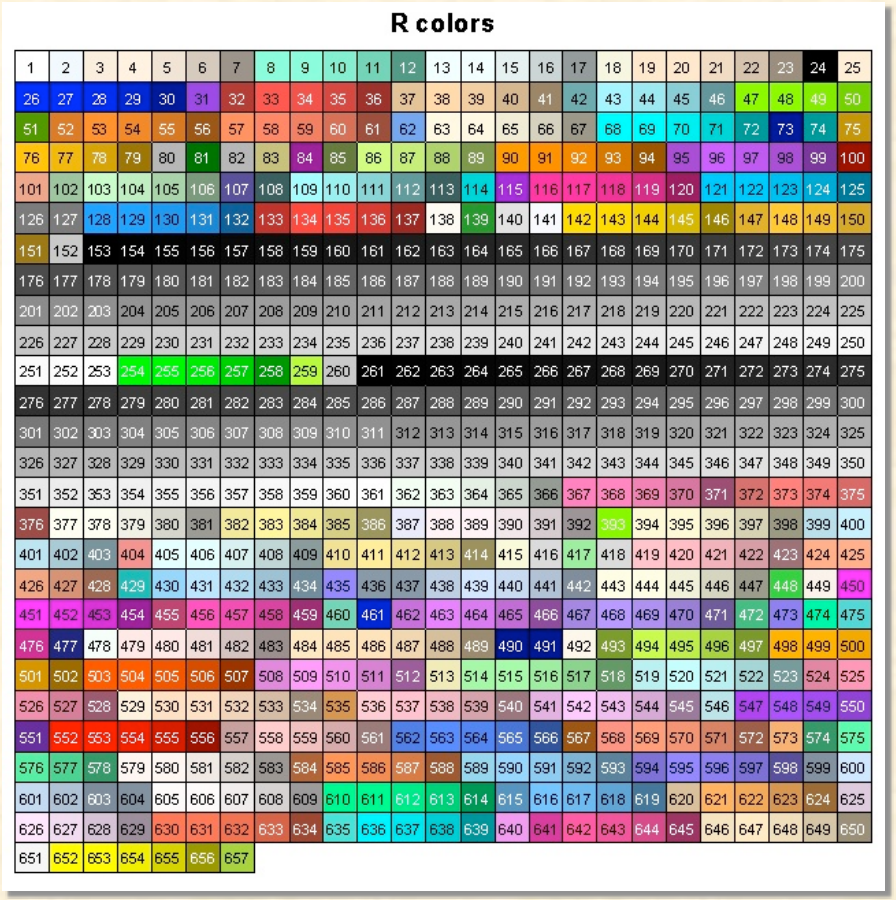
```
cex=
cex.axis=
cex.lab=
cex.main=
cex.sub=
```

# Gráficas: colores

```
> colors() # lista de los 657 colores activos
> demo(colors) # demostración
```

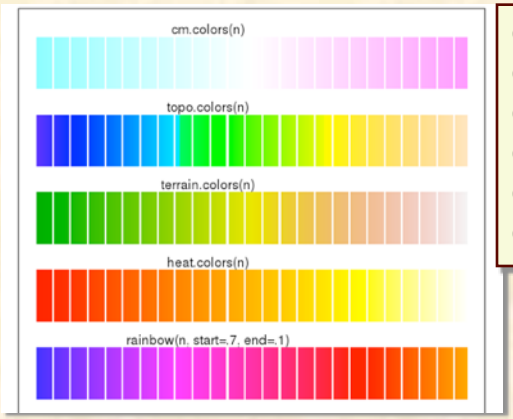
Especificar un color:

```
> col = n #de 1 a 657
> col = "colorname"
> col = rgb(n1,n2,n3)
> col = hsv(n1,n2,n3)
> plot(x,y,col = ...)
```




ver: http://research.stowers-institute.org/efg/R/Color/Chart/index.htm

Paletas de color con *n* colores contiguos:




```
col=rainbow(n)
col=heat.colors(n)
col=terrain.colors(n)
col=topo.colors(n)
col=cm.colors(n)
col=gray(1:n/n)
```

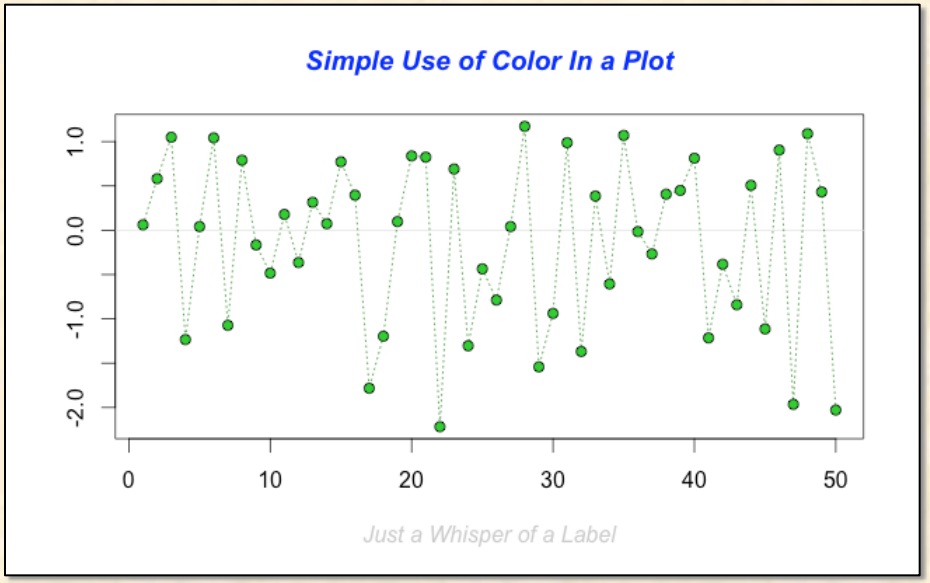
Se puede interpolar entre colores con: `colorRamp` y `colorRampPalette`

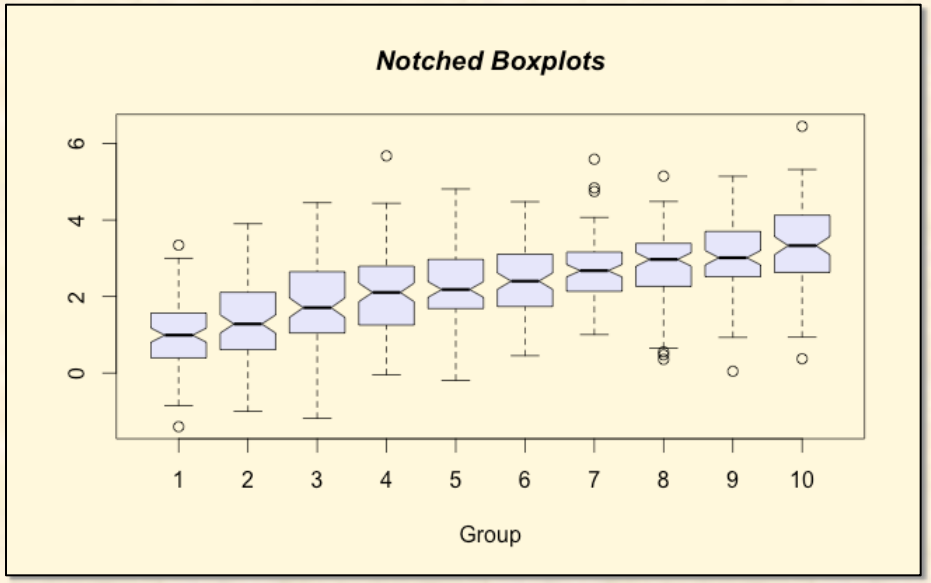
Cambiando parámetros de parte del gráfico:

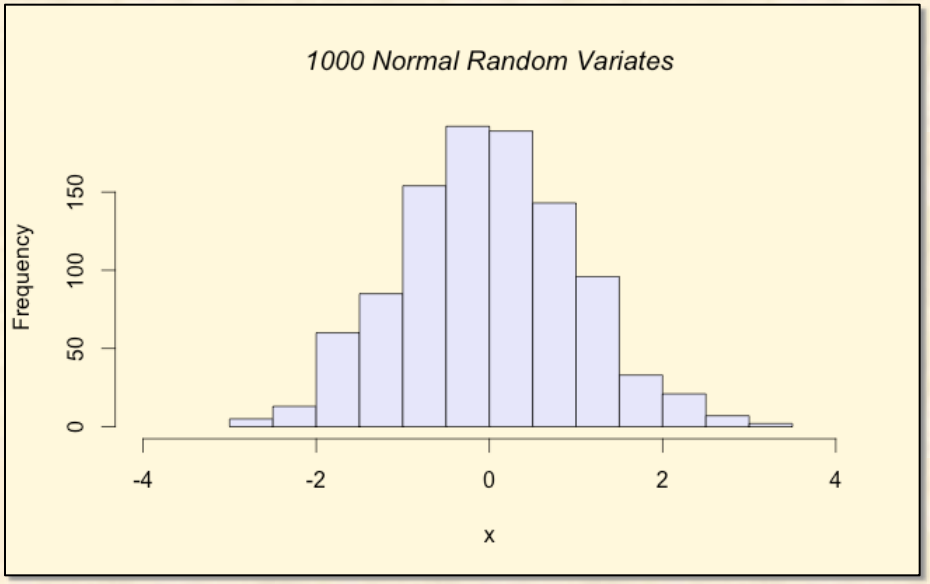
```
col=
col.axis=
col.lab=
col.main=
```

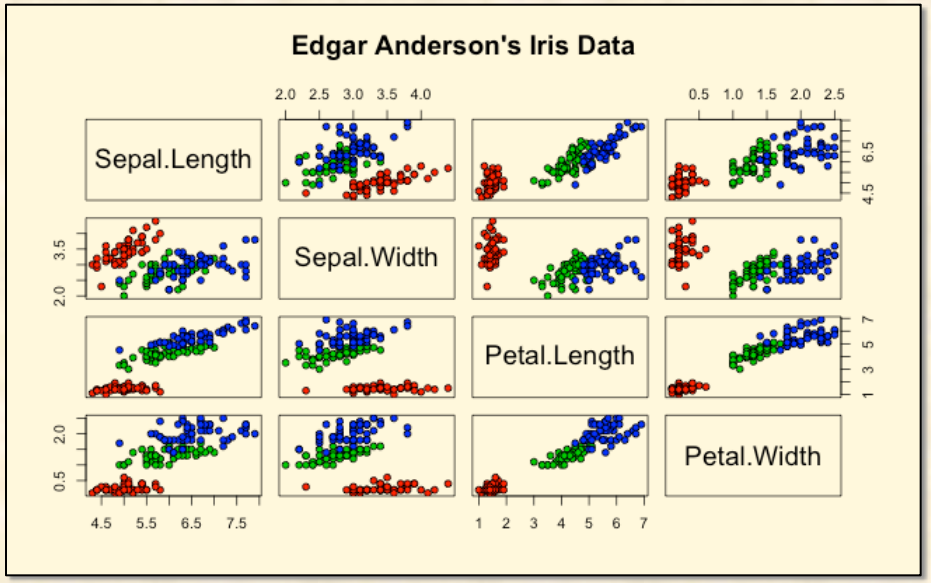
# Gráficas - *demos*

```
> demo(graphics)
```



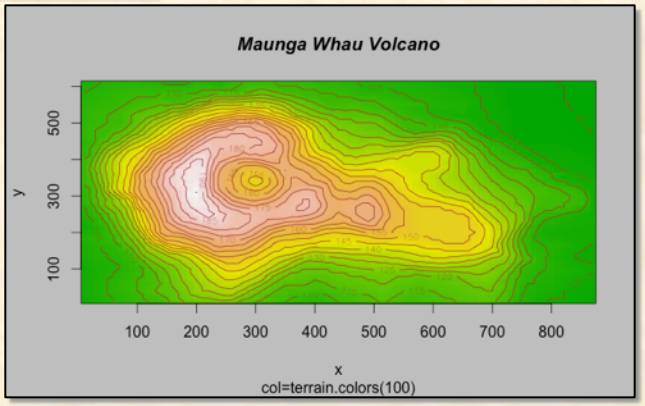



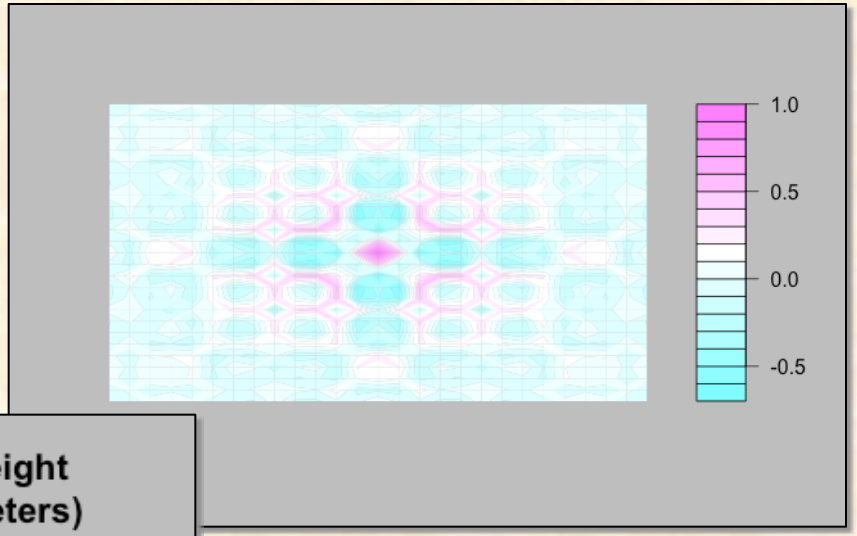



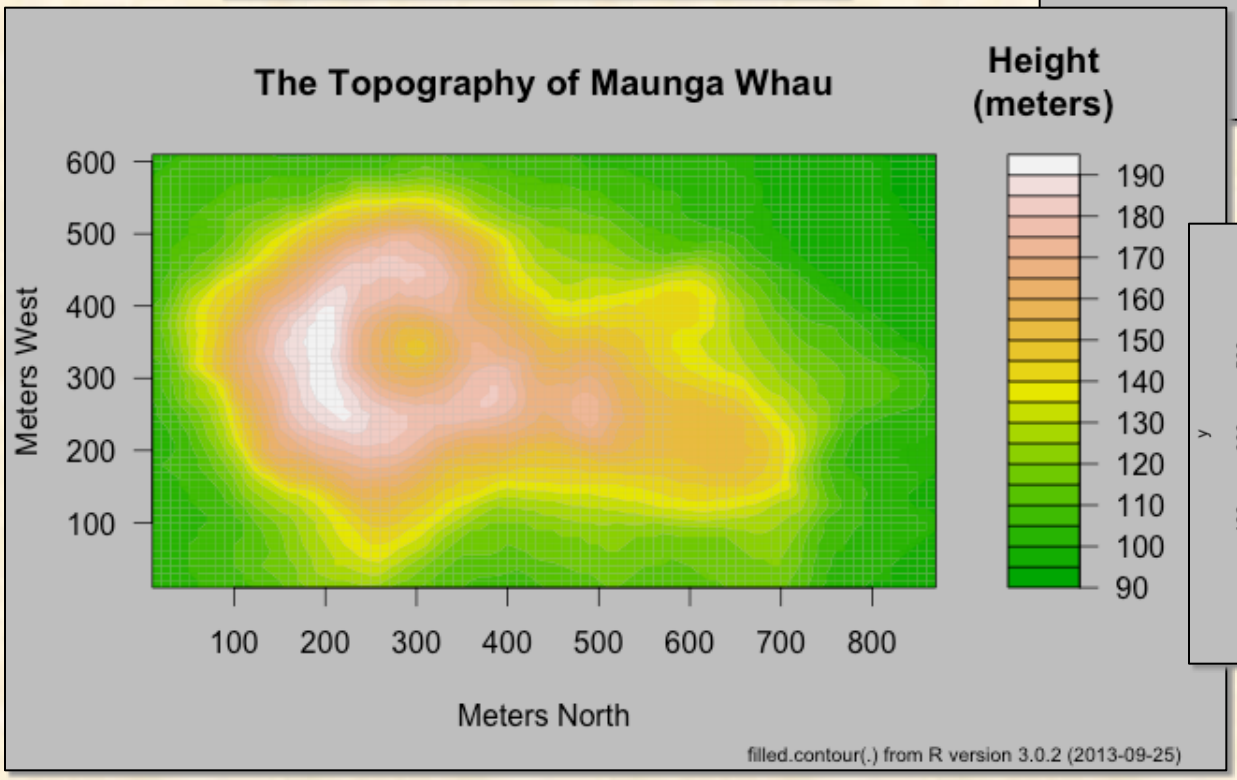



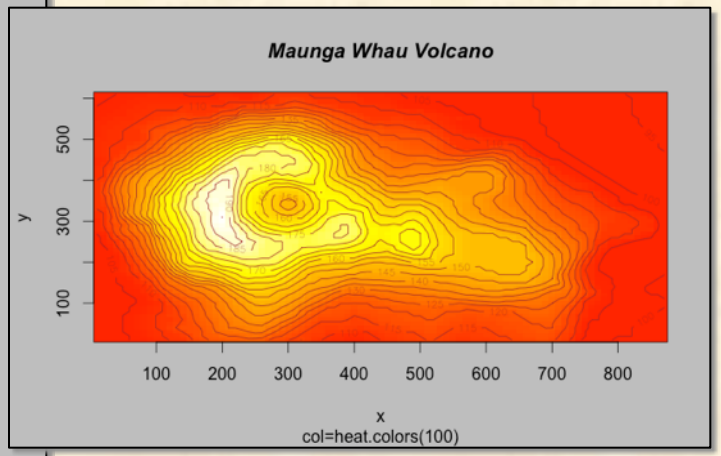
# Gráficas - *demos*

```
> demo(image)
```



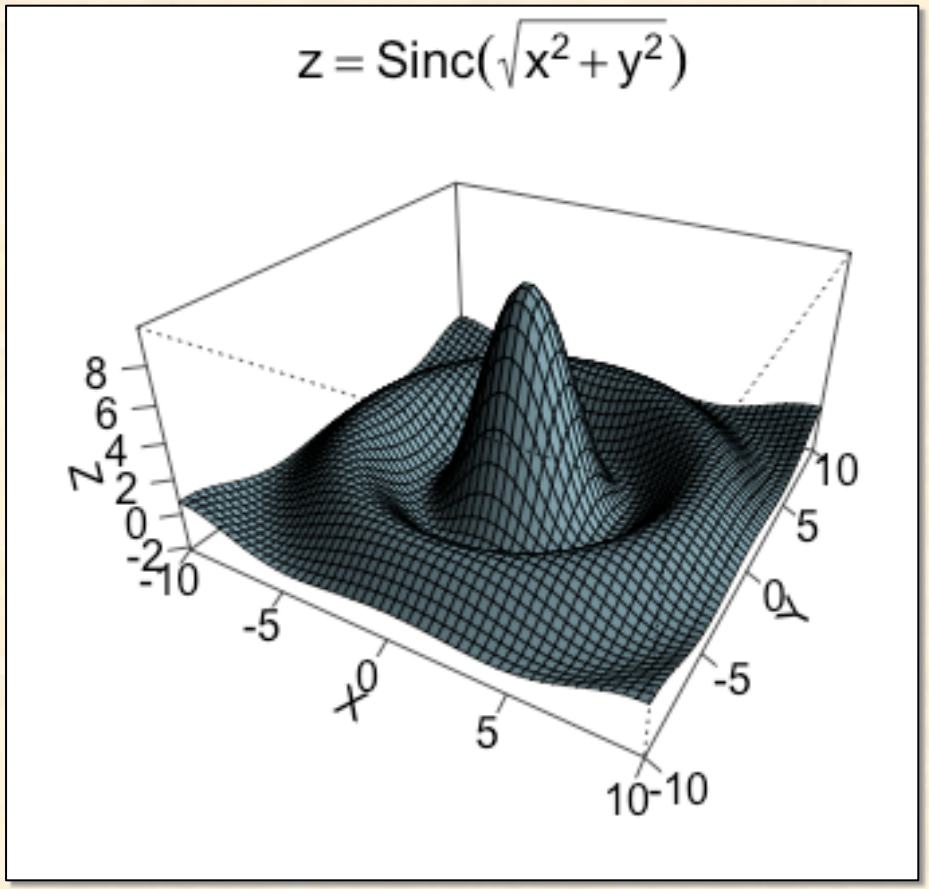



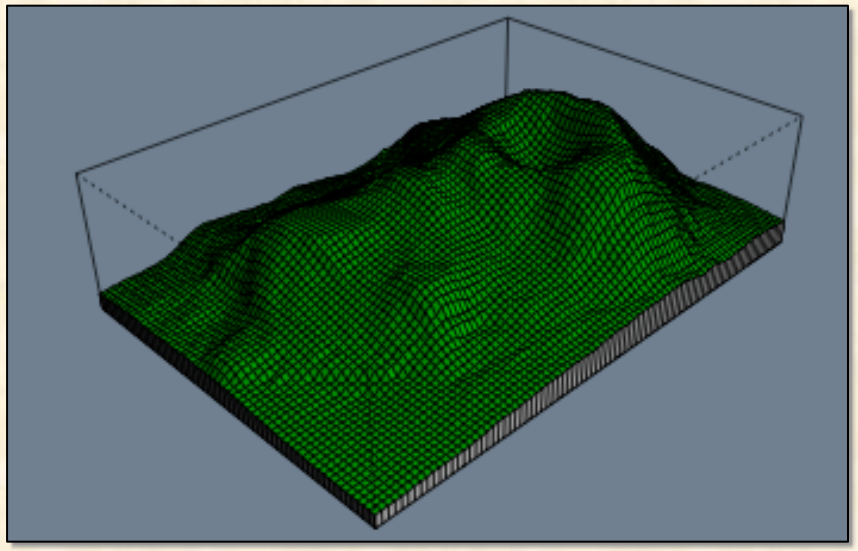



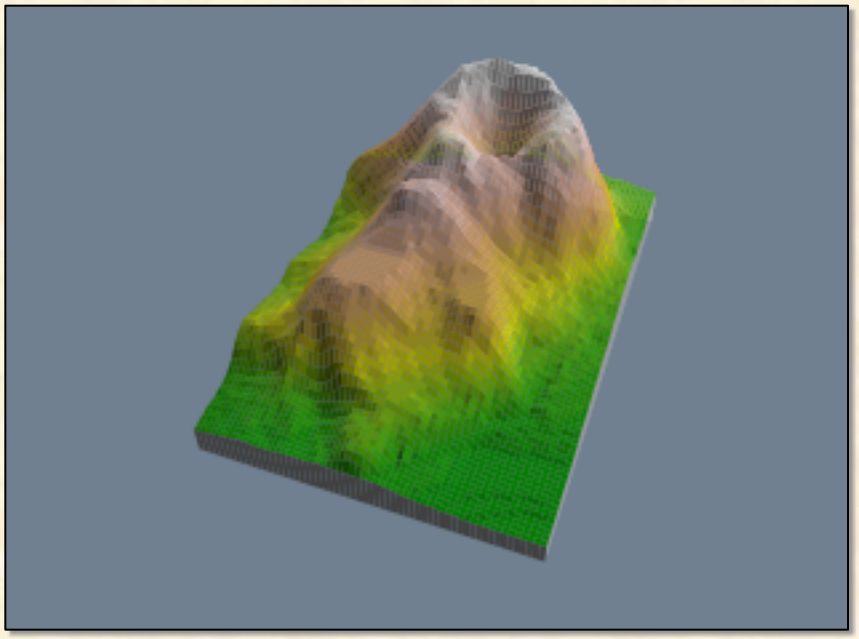



# Gráficas - *demos*

```
> demo(persp)
```



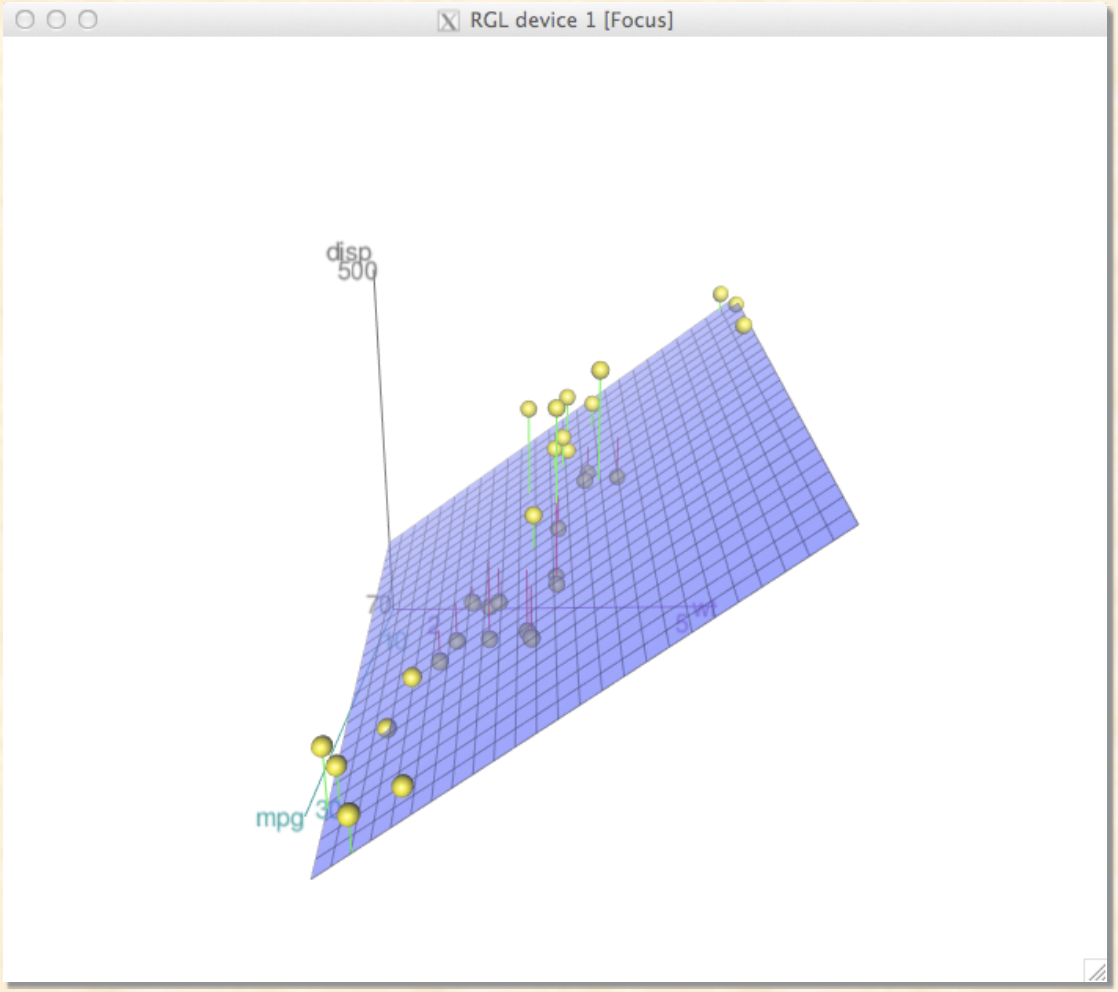



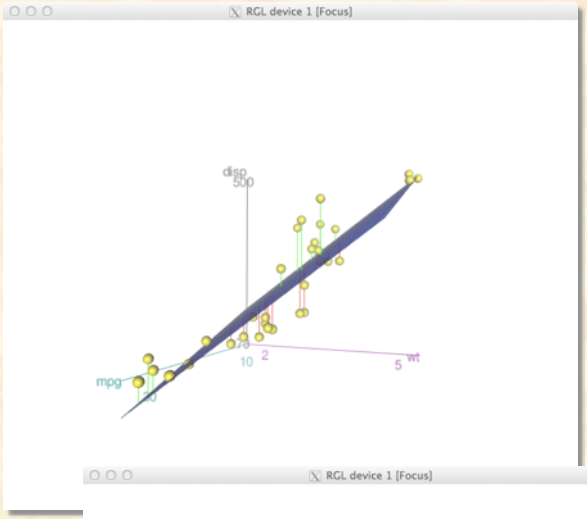


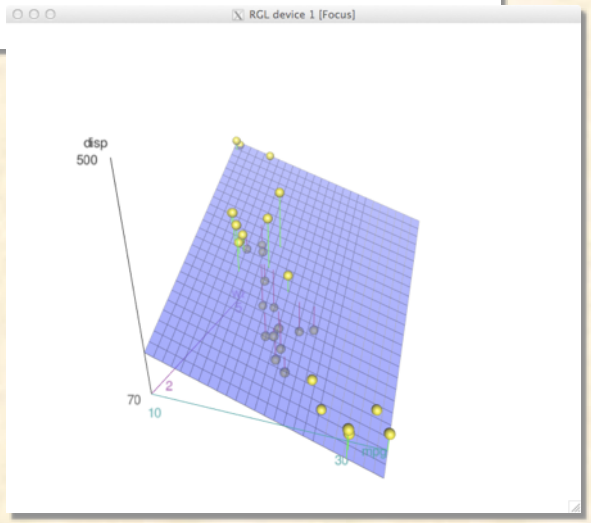
# Gráficas - *demos*

***R*** permite incluso representaciones tridimensionales interactivas

```
> install.packages("car")
> library(car)
> attach(mtcars)
> scatter3d(wt,disp,mpg)
```

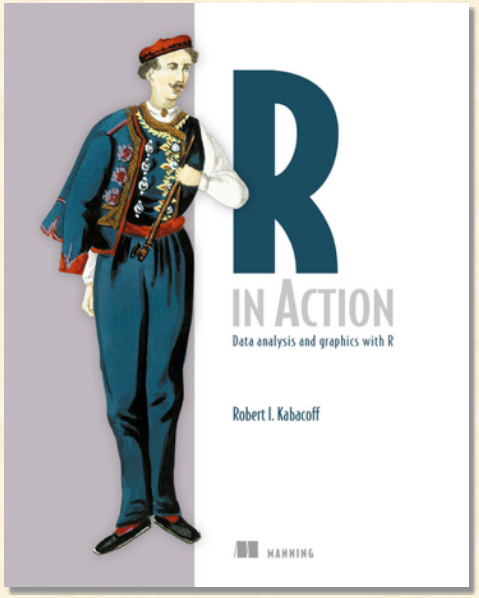


# Referencias

- **R in action**, Robert I. Kabacoff, Manning Publications
- **Introductory Statistics with R**, Peter Dalgaard, Springer
- **Data Analysis and Graphics using R** , John Maindonald & W. John Braun, Cambridge University Press
- **The R Book**, Michael J. Crawley, Ed. John Wiley & Sons
- **R for dummies**, Joris Meys, Andrie de Vries
- **Beginning R: An Introduction to Statistical Programming**, Larry Pace, Apress
- **Beginning R: The Statistical Programming Language**, Mark Gardener, Wrox




## + Manuales y cursos en internet

- R manuals: https://cran.r-project.org/manuals.html
- An Introduction to R: https://cran.r-project.org/doc/manuals/r-release/R-intro.pdf
- R tutorial: http://www.cyclismo.org/tutorial/R/
- Code School. Try R: http://tryr.codeschool.com
- Curso Univ. Rochester: http://www.rochester.edu/college/psc/thestarlab/help/rcourse/R-Course.pdf
- Center for Astrostatistics: http://astrostatistics.psu.edu
- Gráficos simples con R: http://www.harding.edu/fmccown/r/
- R Reference Card: http://statmaster.sdu.dk/bent/courses/ST501-2011/Rcard.pdf
- Comparación de comandos en MATLAB,etc: http://mathesaurus.sourceforge.net/matlab-python-xref.pdf

Introduction to R for Data Science

Microsoft - DAT204x

Inicio: 19 Sep 2016

Ver curso

https://courses.edx.org/courses/course-v1:Microsoft+DAT204x+6T2016/info

Foundations of Data Analysis - Part 1

UTAustinX - UT.7.11x

Inicio: 06 Sep 2016

Ver curso

https://courses.edx.org/courses/course-v1:UTAustinX+UT.7.11x+3T2016/info

R Programming

Universidad Johns Hopkins

https://www.coursera.org/learn/r-programming

Introducción a Data Science: Programación Estadística con R

Universidad Nacional Autónoma de México

https://www.coursera.org/learn/intro-data-science-programacion-estadistica-r

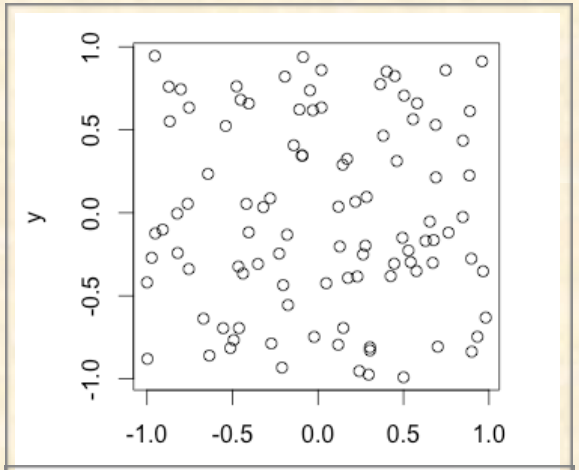
# Ejemplo de *script*: cálculo de $\pi$

Estimamos el valor de $\pi$ utilizando puntos aleatorios generados en el interior de un cuadrado. Se calcula la fracción de puntos dentro de un círculo de radio unidad frente al número de puntos en un cuadrado de lado 2.

$$\text{Fraccion puntos} = \frac{\text{area circulo radio 1}}{\text{area total}} = \frac{\pi}{4}$$

Creamos un *script* `pirandom.R` y se graba en el directorio de trabajo (menu `File/Save As` en *Rstudio*)

```
> pirandom <- function(n) # Definimos una función
+ {
+   x <- runif(n,-1,1)    # n nº aleatorios entre -1 y 1
+   y <- runif(n,-1,1)
+   plot(x,y)
+   r <- sqrt(x*x+y*y)    # distancia al centro
+   rinside <- r[r<1]     # dentro de un círculo de radio 1?
+   print(4*length(rinside)/n)  # estimación de pi
+ }
```




Se carga el script y se corre para dos números de puntos:

```
> source("pirandom.R")
> pirandom(100)
[1] 3.08
> pirandom(1000)
[1] 3.144
```