Análisis de datos y Estadística Avanzada

Máster Interuniversitario de Astrofísica UCM+UAM

Introducción a R

Javier Gorgas y Nicolás Cardiel

Departamento de Astrofísica y Ciencias de la Atmósfera

Facultad de Ciencias Físicas

Universidad Complutense de Madrid

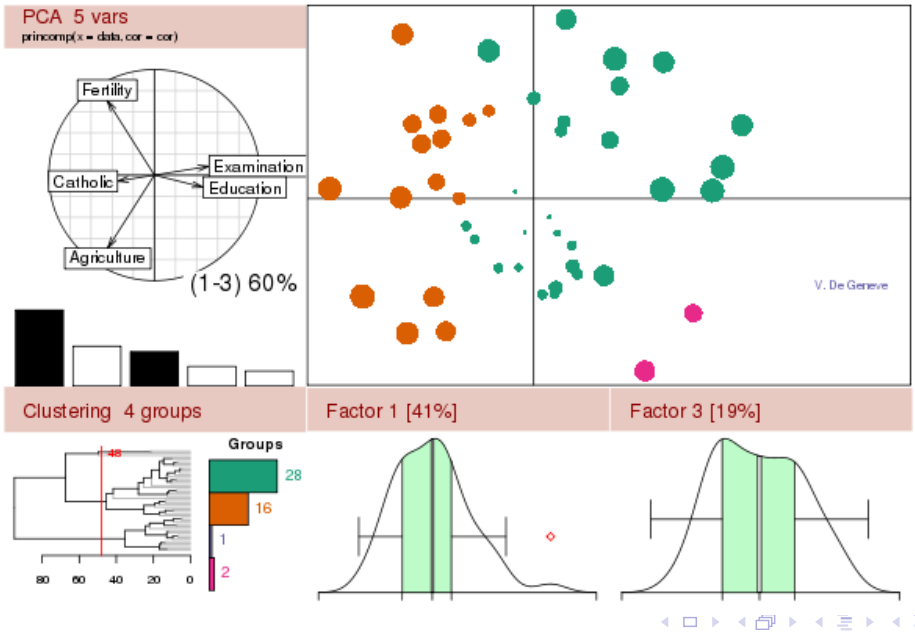
Esquema

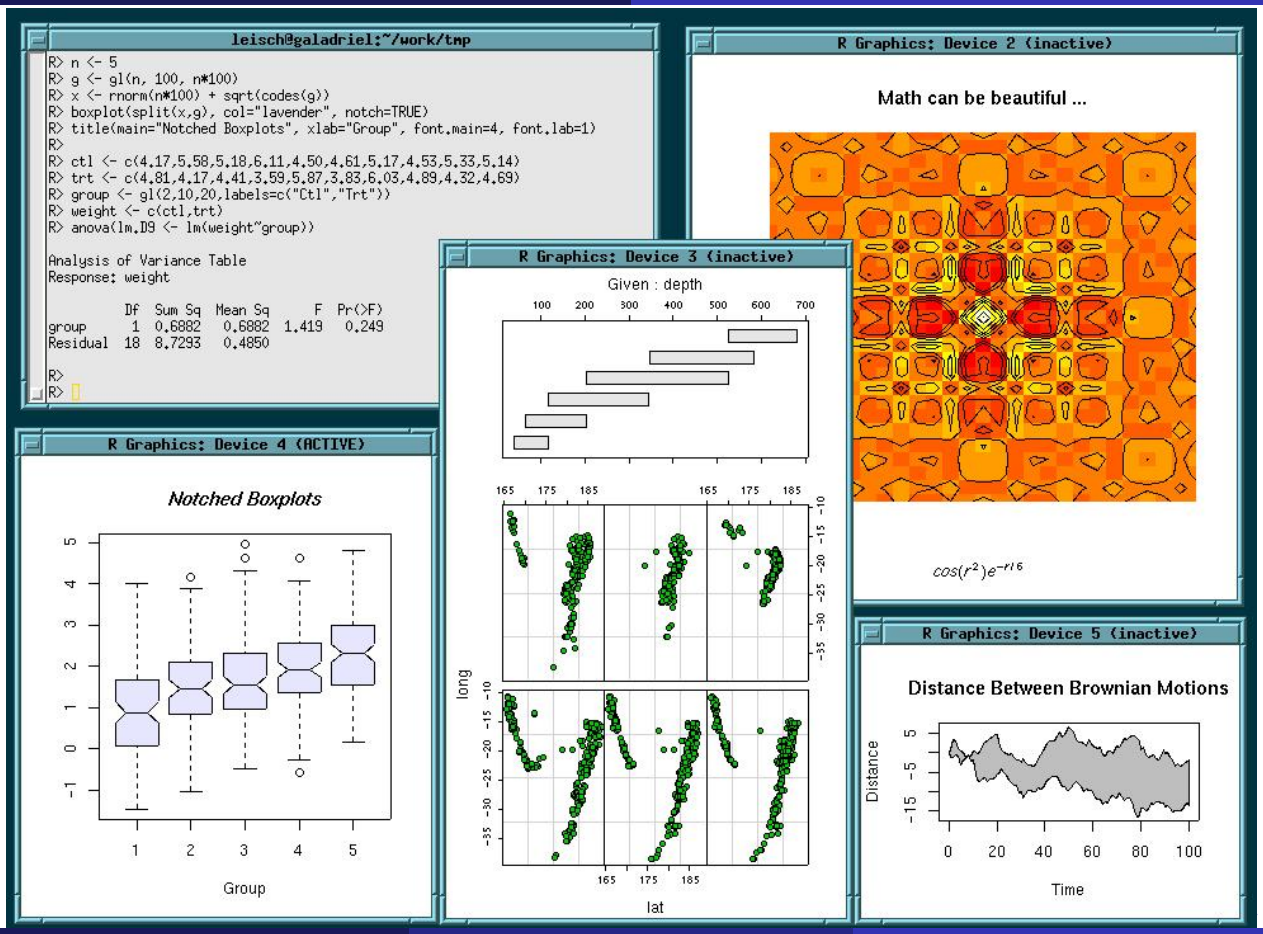
Esquema

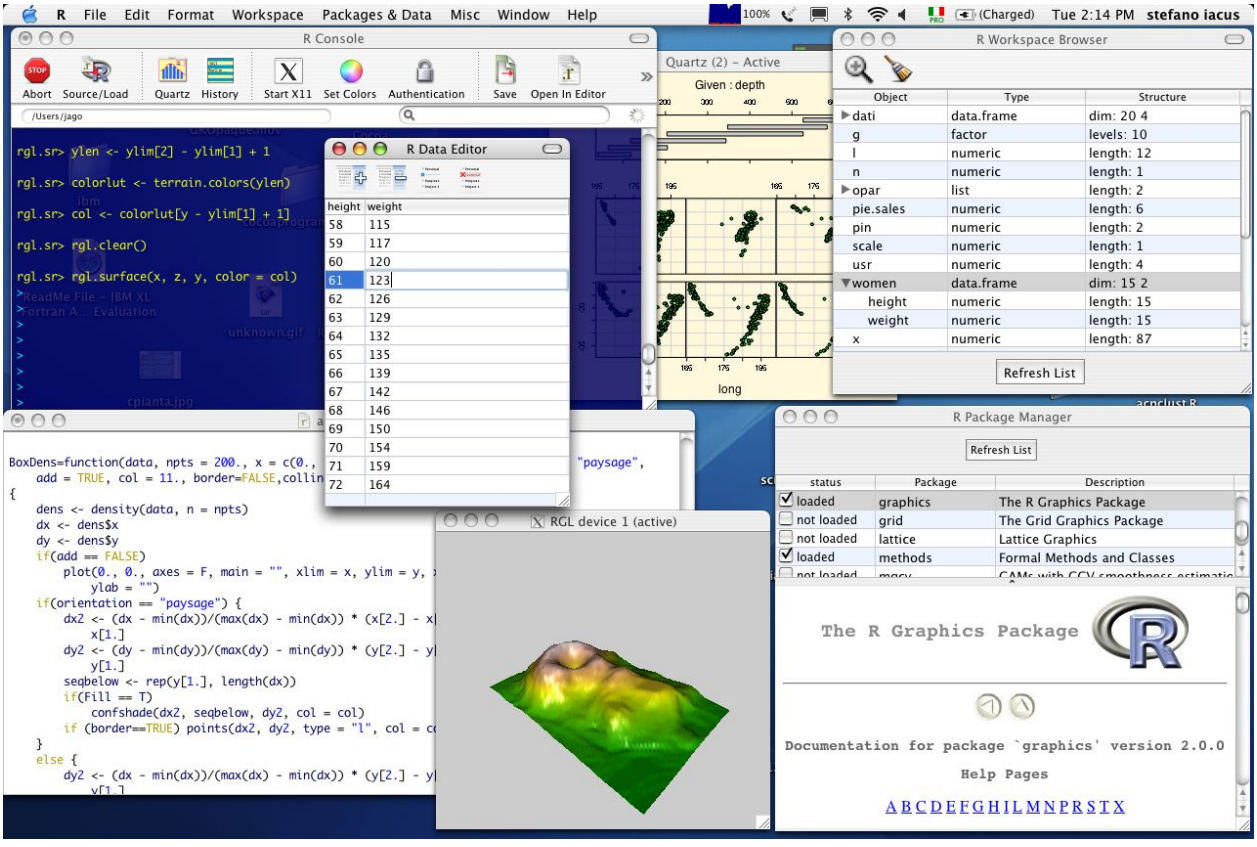
El proyecto R

http://www.r-project.org

es un lenguaje y un entorno especialmente desarrollado para el trabajo estadístico y la representación gráfica de datos.




leisch@galadriel:~/work/tmp
R> n <- 5
R> g <- gl(n, 100, n*100)
R> x <- rnorm(n*100) + sqrt(codes(g))
R> boxplot(split(x,g), col="lavender", notch=TRUE)
R> title(main="Notched Boxplots", xlab="Group", font.main=4, font.lab=1)
R> ctl <- c(4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14)
R> trt <- c(4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69)
R> group <- gl(2,10,20,labels=c("Ctl","Trt"))
R> weight <- c(ctl,trt)
R> anova(lm.D9 <- lm(weight~group))
Analysis of Variance Table
Response: weight
Df Sum Sq Mean Sq F Pr(>F)
group 1 0.6882 0.6882 1.419 0.249
Residual 18 8.7293 0.4850
R Graphics: Device 2 (inactive)
Math can be beautiful ...
R Graphics: Device 3 (inactive)
Given : depth
long
lat
R Graphics: Device 4 (ACTIVE)
Notched Boxplots
Group
R Graphics: Device 5 (inactive)
Distance Between Brownian Motions
Distance
Time


R File Edit Format Workspace Packages & Data Misc Window Help
R Console
R Data Editor
R Workspace Browser
R Package Manager
RGL device 1 (active)
Refresh List
The R Graphics Package
Documentation for package `graphics' version 2.0.0
Help Pages

Esquema

Características generales

- Es un paquete de software similar a otros programas tipo MATLAB, especialmente desarrollado para el tratamiento estadístico.
- Es un entorno integrado: se ha desarrollado como un todo y no es simplemente una colección de herramientas. Incluye:
  - Un sistema eficiente de almacenaje y manipulación de datos.
  - Una colección de operaciones para el manejo de arreglos (*arrays*), en particular matrices.
  - Herramientas integradas para el análisis de datos.
  - Generación de gráficos en pantalla y en formatos transportables.
  - Un simple y efectivo lenguaje de programación.
- ¡Es software libre! Disponible a través de su página WEB `http://www.r-project.org` o a través de CRAN (*Comprehensive R Archive Network*) `http://cran.r-project.org`
- Está disponible para diferentes plataformas (código fuente y binarios pre-compilados): UNIX, MacOS, Windows.

Esquema

Consejo

Antes de empezar a trabajar con R conviene crear un subdirectorio específico donde se realizará el trabajo deseado. De hecho, si se trabaja en varios proyectos simulatáneamente, es aconsejable utilizar un subdirectorio propio para cada uno.

```
user@maquina> mkdir work
user@maquina> cd work
user@maquina/work> cd work
user@maquina/work> R
```

Aviso

R es *case sensitive*, es decir, distingue mayúsculas de minúsculas:

```
> a = 1
> A = 2
```

En el ejemplo anterior, `a` y `A` son variables diferentes:

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

Arrancar el programa (linux):

```
user@maquina> R

R version 2.8.1 (2008-12-22)
Copyright (C) 2008 The R Foundation for Statistical Computing
ISBN 3-900051-07-0

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>                                          ←—prompt (línea de comandos) de R
```


Arrancar el programa (linux):

```
user@maquina> R --silent                  ←—evita el mensaje de bienvenida
user@maquina> R --help                    ←—muestra ayuda de cómo invocar R

Usage: R [options] [< infile] [> outfile]
   or: R CMD command [arguments]

Start R, a system for statistical computation and graphics, with the
specified options, or invoke an R tool via the 'R CMD' interface.

Options:
  -h, -help             Print short help message and exit
  -version              Print version info and exit
  -encoding=ENC         Specify encoding to be used for stdin
...
...
```

Salir del programa

```
> quit()
Save workspace image?  [y/n/c]: n   ←—permite guardar valores para siguiente sesión
> quit(save="no")                   ←—termina la ejecución de R sin preguntar
```

El uso de los paréntesis en `quit()` ayuda a R a saber que lo que estamos escribiendo es una función y no una variable que se llama `quit`.

Obteniendo ayuda dentro de R:

```
> ?comando                          ←—muestra ayuda sobre un comando
> help("comando")                   ←—muestra ayuda sobre un comando
```

Si no sabemos el nombre del comando podemos intentar:

```
> help.search("cadena de texto")    ←—busca y muestra ayuda sobre un comando/concepto
```

Para una ayuda mucho más completa:

```
> help.start()                      ←—lanza ayuda completa en ventana independiente
```

El tabulador completa o ayuda a completar los comandos:

```
> Sys.⟨tabulador⟩                   ←—pulsamos tabulador para ayudar a completar
```

```
Sys.chmod       Sys.glob        Sys.setlocale   Sys.unsetenv
Sys.Date        Sys.info        Sys.sleep       Sys.which
Sys.getenv      Sys.localeconv  Sys.time
Sys.getlocale   Sys.putenv      Sys.timezone
Sys.getpid      Sys.setenv      Sys.umask
> Sys.time()
[1] "2009-03-06 17:28:04 CET"
```


Información básica del sistema:

```
> Sys.info()
                                sysname                                 release
                                "Linux"     "2.6.27.19-78.2.30.fc9.x86_64"
                                version                                nodename
"#1 SMP Tue Feb 24 19:44:45 EST 2009"                  "xblax.fis.ucm.es"
                                machine                                   login
                              "x86_64"                                "cardiel"
                                   user
                              "cardiel"
> R.version.string
[1] "R version 2.8.1 (2008-12-22)"
> capabilities()
    jpeg      png     tiff    tcltk      X11     aqua http/ftp  sockets
    TRUE     TRUE     TRUE    FALSE     TRUE    FALSE     TRUE     TRUE
  libxml     fifo   cledit    iconv      NLS  profmem    cairo
    TRUE     TRUE     TRUE     TRUE     TRUE    FALSE     TRUE
> citation()
To cite R in publications use:

  R Development Core Team (2008). R: A language and environment for
  statistical computing. R Foundation for Statistical Computing,
  Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.
...
```

Moverse por los subdirectorios y ver su contenido:

```
> R.home()                                    ←—retorna el directorio home de R
[1] "/home/cardiel/s/R-2.8.1"
> getwd()                                     ←—retorna el directorio actual
[1] "/home/cardiel/w/clases/2008-2009/analisis_datos/R"
> setwd("/home/cardiel")                      ←—establece un nuevo directorio actual
> dir()                                       ←—muestra los ficheros existentes
...
...
```

Es importante no confundir `dir()` con `ls()`. Este último muestra los objetos definidos.

Ejecutar comandos almacenados en un fichero externo (externo):

```
> source("comandos.R")    ←—ejecuta los comandos guardados en el fichero comandos.R
```

Enviar el resultado de la ejecución a un fichero externo:

```
> sink("registro.txt")          ←—guarda los resultados en el fichero registro.txt
```

A partir del momento de ejecutar la instrucción superior, la salida en pantalla queda deshabilitada.

```
> sink()                                      ←—restablece salida de resultados por consola
```


Esquema

Tipos de estructuras datos

- **Vectores**: son la estructura más simple. Sirven para almacenar colecciones de variables del mismo tipo.
  - Vectores numéricos
  - Vectores de valores lógicos
  - Vectores de caracteres (cadenas de texto)
- **Factores**: vectores que sirven para realizar un agrupamiento de los componentes de otro vector del mismo tamaño.
- **Matrices**: colección de datos a los que se accede por varios índices enteros (dimensiones).
- **Listas**: colección ordenada de objetos, en la que los elementos pueden ser de distinto tipo.
- **Data frames**: tipo particular de listas de gran utilidad para el trabajo estadístico.
- **Funciones**: objetos que pueden ser creados por el usuario y reutilizados para realizar operaciones específicas.

Vectores numéricos:

Varias formas de asignar valores a una variable:

```
> a <- 1.7          ←—asignamos un valor a un vector de un único elemento (escalar)
> 1.7 -> a          ←—asignamos un valor a un vector de un único elemento (escalar)
> a = 1.7           ←—asignamos un valor a un vector de un único elemento (escalar)
> assign("a",1.7)   ←—asignamos un valor a un vector de un único elemento (escalar)
```

Mostramos valores:

```
> a                 ←—mostramos el valor en pantalla (no funciona en scripts)
[1] 1.7
> print(a)          ←—mostramos el valor en pantalla (sí funciona en scripts)
[1] 1.7
```

Vamos a generar un vector con varios elementos numéricos:

```
> a <- c(1,2,5,7,9)     ←—asignamos varios número usando el comando c()
> a                     ←—mostramos el valor en pantalla
[1] 1 2 5 7 9
```

Las operaciones se realizan sobre todos los elementos del vector numérico:

```
> a*a                   ←—evaluamos el cuadrado de cada elemento del vector
[1] 1 4 25 49 81
> 1/a                   ←—evaluamos el inverso de cada elemento del vector
[1] 1.0000000 0.5000000 0.2000000 0.1428571 0.1111111
> b <- a-1              ←—restamos 1 a cada elemento y lo asignamos a la variable b
> b
[1] 0 1 4 6 8
```

Vectores numéricos:

Generando secuencias:

```
> 1:10                                  ←—generamos la secuencia n1 a n2 con n1:n2
[1] 1 2 3 4 5 6 7 8 9 10
> 20:60                                 ←—generamos la secuencia n1 a n2 con n1:n2
 [1] 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41
[23] 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
> 5:1                                   ←—generamos una secuencia inversa
[1] 5 4 3 2 1
```

Una forma más general de generar secuencias:

```
> seq(1,10,3)                           ←—generamos la secuencia n1 a n2 con paso n3
[1] 1 4 7 10
> seq(from=1, to=10, by=3)              ←—generamos la misma secuencia
[1] 1 4 7 10
> seq(length=10, from=1, by=3)          ←—generamos otra secuencia fijando su longitud
[1] 1 4 7 10 13 16 19 22 25 28
```

Para saber más sobre este comando, es el momento de consultar la ayuda:

```
> help(seq)                             ←—invocamos la ayuda del comando seq
...
```


Vectores numéricos:

Generamos repeticiones:

```
> a <- 1:3; b <- rep(a,times=3); c <- rep(a,each=3)      ←— comando rep()
```

En el ejemplo anterior hemos ejecutado tres comandos en una misma línea. Los tres comandos aparecen separados por el símbolo punto y coma.

Tras ejecutar el comando anterior, las tres variables contienen ahora:

```
> 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
```

Si necesitamos conocer qué objetos hemos definido hasta el momento, podemos listarlos:

```
> ls()
[1] "a" "b" "c"
```

Podemos borrar los objetos que deseemos mediante la función `rm()`:

```
> rm(a,c)                               ←— borramos los objetos a y c
> ls()                                  ←— comprobamos qué objetos quedan
[1] "b"
```

Vectores de valores lógicos:

```
> a <- seq(1:10)                    ←—generamos una secuencia
> a                                 ←—mostramos su valor en pantalla
[1] 1 2 3 4 5 6 7 8 9 10
> b <- (a>5)                        ←—asignamos valores de una desigualdad
> b                                 ←—mostramos su valor en pantalla
[1] FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE
```

Vectores de caracteres (cadenas de texto):

```
> a <- "Ejemplo de cadena"          ←—generamos un vector de caracteres
> a                                 ←—mostramos contenido del vector en pantalla
[1] "Ejemplo de cadena"
```

Podemos concatenar vectores tras haberlos convertido en cadenas de caracteres previamente:

```
> x <- 1.5
> y <- -2.7
> paste("El punto es (",x,",",y,")",sep="")     ←—concatenamos los vectores
[1] "El punto es (1.5,-2.7)"
```


Accediendo a elementos concretos de un vector

Vectores numéricos:

```
> a <- 1:15                         ←—generamos una secuencia
> a <- a*a                          ←—elevamos al cuadrado
> a                                 ←—mostramos en pantalla
[1] 1 4 9 16 25 36 49 64 81 100 121 144 169 196 225
> a[3]                              ←—accedemos a un valor particular
[1] 9
> a[3:5]                            ←—accedemos a una secuencia continua de valores
[1] 9 16 25
> a[c(1,3,10)]                      ←—accedemos a una secuencia cualquiera de valores
[1] 1 9 100
> a[a > 100]                        ←—accedemos a una secuencia basada en una condición
[1] 121 144 169 196 225
```

Con cadenas de texto se accede a los elementos de forma diferente:

```
> a <- "Ejemplo de cadena de texto"     ←—definimos una cadena de texto
> substr(a,5,10)                        ←—mostramos un subconjunto de la cadena
[1] "plo de"
```

Valores especiales:

A veces resuta útil definir algunos valores como **no disponibles** (*not available*):

```
> a <- c(0:2,NA,NA,5:7)
> a
[1] 0 1 2 NA NA 5 6 7
```

←—definimos vector con un valor NA
←—mostramos contenido en pantalla

Podemos realizar operaciones:

```
> a*a
[1] 0 1 4 NA NA 25 36 49
```

←—calculamos el cuadrado

Podemos averiguar si algún valor no está definido:

```
> nodisponible <- is.na(a)
> nodisponible
[1] FALSE FALSE FALSE TRUE TRUE FALSE FALSE FALSE
```

A veces los cálculos conduce a valores sin sentido matemático:

```
> a <- log(-1)
> a
[1] NaN
```

←—NaN: *Not a Number*

```
> a <- 1/0; b <- 0/0; c <- log(0); d <- c(a,b,c)
> d
[1] Inf NaN -Inf
```

←—±Inf: ±*infinito*

```
> 1/Inf
[1] 0
```

←—¡cuando tiene sentido, se puede operar con infinitos!

Valores especiales:

A veces resuta útil definir algunos valores como **no disponibles** (*not available*):

```
> a <- c(0:2,NA,NA,5:7)
> a
[1] 0 1 2 NA NA 5 6 7
```

←—definimos vector con un valor NA
←—mostramos contenido en pantalla

```
> b <- 1/a
> b
[1] Inf 1.0000000 0.5000000 NA NA 0.2000000 0.1666667
[8] 0.1428571
```

```
> is.infinite(b)
[1] TRUE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

←—miramos si tenemos valores ±Inf

```
> c <- sqrt(-b)
Warning message:
In sqrt(-b) :  NaNs produced
> c
[1] NaN NaN NaN NA NA NaN NaN NaN
```

```
> is.nan(c)
[1] TRUE TRUE TRUE FALSE FALSE TRUE TRUE TRUE
```

←—miramos si tenemos valores NaN

Podemos eliminar los datos no disponibles de una forma trivial usando:

```
> aa <- a[!is.na(a)]
> aa
[1] 0 1 2 5 6 7
```

Valores especiales:

La mayor parte de las funciones habituales de R (`mean`, `var`, `sum`, `min`, `max`,...) aceptan un argumento llamado `na.rm=` que puede establecerse en `TRUE` o `FALSE` para eliminar o no los datos que faltan en los cálculos.

```
> a <- c(0:2,NA,NA,5:7)                    ←—definimos vector a con un valor NA
> a
[1] 0 1 2 NA NA 5 6 7
> aa <- a[!is.na(a)]                       ←—definimos vector aa elimando valores NA
> aa
[1] 0 1 2 5 6 7
> mean(a)
[1] NA
```

porque hay valores no disponibles. Sin embargo podemos calcular la media ignorando los valores no disponibles ejecutando

```
> mean(a,na.rm=TRUE)
[1] 3.5
```

que es el mismo resultado que obtenemos calculando

```
> mean(aa)
[1] 3.5
```


Factores:

Los factores son vectores que contienen información que permite realizar un agrupamiento de los valores de otros vectores del mismo tamaño. Veamos un ejemplo.

```
> 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) de una muestra de 13 galaxias
> mean(bv)                                          ←—calculamos la media
[1] 0.9238462
```

Si ahora disponemos de información adicional, como por ejemplo el tipo morfológico de las galaxias, podemos utilizar dicha información para realizar un análisis estadístico segregando los datos por dicho tipo. Para ello generamos primero un **factor** como un vector que contiene dicha información:

```
> morfo <- c("Sab","E","Sab","S0","E","E","S0","S0","E","Sab",
+ "E","Sab","S0")                  ←—información morfológica en el mismo orden
> length(morfo)                        ←—el vector ha de ser del mismo tamaño
[1] 13
> morfo                                ←—mostramos el contenido del nuevo vector
[1] "Sab" "E" "Sab" "S0" "E" "E" "S0" "S0" "E" "Sab" "E"
[12] "Sab" "S0"
> fmorfo <- factor(morfo)                ←—generamos el factor con factor()
> fmorfo                                  ←—mostramos el contenido del factor
[1] Sab E Sab S0 E E S0 S0 E Sab E Sab S0
Levels:  E S0 Sab                  ←—nos dice los diferentes niveles que tenemos
```

Factores:

```
> levels(fmorfo)                    ←—nos permite averiguar los niveles que tenemos
+ [1] "E" "S0" "Sab"
```

Ahora podemos calcular los colores promedio para cada tipo morfológico de galaxia. Para ello usamos la función especial `tapply()`, a la cual indicamos el vector sobre el cual queremos calcular los colores `bv`, el factor asociado `fmorfo` y la función que queremos evaluar, en nuestro caso la media `mean`:

```
> meanbv <- tapply(bv, fmorfo, mean)
> meanbv
     E     S0    Sab
0.9700 0.9025 0.8875
```

De la misma forma se puede evaluar cualquier otra función (intrínseca de R o definida por el usuario) segregando los datos usando la información del factor. Por ejemplo, el cálculo de la desviación típica podría realizarse como:

```
> stbv <- tapply(bv,fmorfo,sd)
> stbv
         E          S0         Sab
0.04358899 0.03774917 0.02753785
```


Factores y Tablas:

Muchas veces resulta útil (e.g. para la construcción de tablas) poder generar factores a partir de una variable numérica continua. Para ello podemos utilizar el comando `cut`, cuyo parámetro `breaks=` indica cómo se subdivide el recorrido de los datos. Si `breaks` es igual a un número, éste se utiliza como el número de intervalos (de igual longitud):

```
> 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) de una muestra de 13 galaxias
> fbv <- cut(bv,3)                   ←—dividimos el recorrido en 3 intervalos iguales
> fbv              ←—indica en qué intervalo se encuentra el valor (B-V) de 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)                              ←—generamos una tabla con los 3 intervalos
fbv
 (0.85,0.913] (0.913,0.977]   (0.977,1.04]
            6             6             1
```

Si `breaks` es un vector, sus valores se utilizan como límites de los intervalos:

```
> ffbv <- cut(bv,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
```

Factores y Tablas:

Si queremos un número simplemente aproximado de intervalos, pero cuyos valores sean números *redondos*, se puede utilizar la función `pretty` (¡que no siempre retorna el número de intervalos indicado!):

```
> fffbv <- cut(bv,pretty(bv,3))                    ←—pedimos 3 intervalos "bonitos"
> table(fffbv)                                     ←—aunque en realidad retorna 4 intervalos

fffbv
(0.85,0.9] (0.9,0.95]   (0.95,1]   (1,1.05]
         3          5          3          1
```

Podemos incluso utilizar una división en cuartiles:

```
> ffffbv <- cut(bv,quantile(bv,(0:4)/4))           ←—pedimos los 4 cuartiles
> table(ffffbv)

ffffbv
 (0.85,0.9]  (0.9,0.92] (0.92,0.96] (0.96,1.04]
          3           4           3           2
```

¡Cuidado! Los dos agrupamientos anteriores han dejado fuera el valor 0.85, que es uno de los datos.

Factores y Tablas:

Los factores pueden utilizarse para construir tablas multidimensionales. Veamos cómo.

Primero vamos a definir los datos (en un caso más realista leeríamos los datos de un fichero):

```
> alturas <- 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)
> pesos <- (c(64,77,82,62,71,72,85,68,72,75,81,88,72,
+ 71,74,69,81,67,65,73)
> edades <- c(12,34,23,53,23,12,53,38,83,28,28,58,38,
+ 63,72,44,33,27,32,38)
```

Para cada una de estas variables continuas generamos unos factores:

```
> falturas <- cut(alturas,c(1.50,1.60,1.70,1.80,1.90))   ←—factor en alturas
> fpesos <- cut(pesos,c(60,70,80,90))                    ←—factor en pesos
> fedades <- cut(edades,seq(10,90,10))                   ←—factor en edades
```

La generación de las tablas es inmediata a partir de los factores. Podemos, por ejemplo, construir tablas bidimensionales:

```
> ta <- table(falturas,fpesos)                     ←—generamos una tabla alturas vs. pesos
> ta
           fpesos
falturas    (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
```

Factores y Tablas:

Se pueden incluir las frecuencias marginales:

```
> addmargins(ta)
            fpesos
falturas     (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
```

También se puede trabajar con las frecuencias relativas

```
> tta <- prop.table(ta)
> addmargins((tta)
            fpesos
falturas     (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
```


Factores y Tablas:

También podemos generar una tabla tridimensional. Siguiendo con el ejemplo anterior, podemos examinar la misma tabla bidimensional para cada intervalo de edad:

```
> table(falturas,fpesos,fedades)
, , fedades = (10,20]                          # primer intervalo en edad

            fpesos
falturas     (60,70] (70,80] (80,90]
  (1.5,1.6]       0       0       0
  (1.6,1.7]       1       1       0
  (1.7,1.8]       0       0       0
  (1.8,1.9]       0       0       0

, , fedades = (20,30]                         # segundo intervalo en edad

            fpesos
falturas     (60,70] (70,80] (80,90]
  (1.5,1.6]       0       0       1
  (1.6,1.7]       0       1       0
  (1.7,1.8]       1       1       1
  (1.8,1.9]       0       0       0

...
```

Factores y Tablas:

```
...

, , fedades = (70,80]                          # penúltimo intervalo en edad

           fpesos
falturas    (60,70] (70,80] (80,90]
  (1.5,1.6]       0       0       0
  (1.6,1.7]       0       1       0
  (1.7,1.8]       0       0       0
  (1.8,1.9]       0       0       0

, , fedades = (80,90]                            # Último intervalo en edad

           fpesos
falturas    (60,70] (70,80] (80,90]
  (1.5,1.6]       0       0       0
  (1.6,1.7]       0       0       0
  (1.7,1.8]       0       1       0
  (1.8,1.9]       0       0       0

> sum(table(falturas,fpesos,fedades))
[1] 20                          # Comprobación: número total de individuos
```


Matrices:

Una matriz es una colección de datos a los que se accede por varios índices enteros:

```
> a <- array(1:12,dim=c(3,4))          ←—definimos una matriz de 3 filas y 4 columnas

> a
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

> dim(a)                          ←—devuelve las dimensiones de la matriz (filas, columnas)
[1] 3 4

> a[2,3]                                  ←—devuelve un valor concreto de la matriz
[1] 8

> a[2,]                                ←—devuelve los valores de una fila de la matriz
[1] 2 5 8 11

> a[,3]                             ←—devuelve los valores de una columna de la matriz
[1] 7 8 9
```

Conviene resaltar que el convenio para el orden de los índices en una matriz `a[i,j]` es el mismo que el que se utiliza en matemáticas para los coeficientes matriciales $a_{i,j}$.

Matrices:

El acceso a los elementos de las matrices podemos hacerlos con índices almacenados en otras matrices auxiliares:

```
> a
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12

> ind <- array(c(1:3,3:1),dim=c(3,2))   ←matriz auxiliar con valores de índices i, j

> ind
     [,1] [,2]
[1,]    1    3
[2,]    2    2
[3,]    3    1

> a[ind] <- 0
> a
     [,1] [,2] [,3] [,4]
[1,]    1    4    0   10
[2,]    2    0    8   11
[3,]    0    6    9   12
```


Matrices:

Los elementos de los vectores y matrices **se reciclan** cuando las dimensiones involucradas así lo requieren:

```
> x <- c(1,1,1)
> x
[1] 1 1 1                                          ←len(x) = 3
> y <- c(x,7,7,7,x)
> y
[1] 1 1 1 7 7 7 1 1 1                              ←len(y) = 9
```

Vamos a realizar una operación con vectores de tamaño diferente:

```
> x+y+0.5
[1] 2.5 2.5 2.5 8.5 8.5 8.5 2.5 2.5 2.5
```

En este ejemplo el vector `x` se reutiliza 3 veces, el vector `y` solo una vez, mientras que el escalar 0.5 se reutiliza 9 veces.

Otro ejemplo:

```
> a <- array(1:7,dim=c(4,4))
> a
     [,1] [,2] [,3] [,4]
[1,]    1    5    2    6
[2,]    2    6    3    7
[3,]    3    7    4    1
[4,]    4    1    5    2
```

Matrices:

Generar tablas 2D a partir de matrices es muy sencillo:

```
> 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
```

Sin embargo, la tabla así generada no es una tabla real de R. Para transformarla en una tabla, podemos utilizar:

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

```
> mode(mtab);mode(rtab)
```
←—no se distinguen en `mode`

```
[1] "numeric"
[1] "numeric"
```

```
> typeof(mtab);typeof(rtab)
```
←—no se distinguen en `typeof`

```
[1] "double"
[1] "double"
```

```
> class(mtab);class(rtab)
```
←—¡pero sí se distinguen en `class`!

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


Matrices:

A las tablas se pueden aplicar funciones específicas para calcular distribuciones marginales (`margin.table`), frecuencias (`prop.table`),... incluso el comando `summary` proporciona inmediatamente un test χ^2 de independencia de factores:

```
>  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
```

El mismo comando produce un resultado muy distinto aplicado a un objeto de tipo matriz:

```
>  summary(mtab)
  ellipticals      spirals
 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
```

Listas:

Colección ordenada de objetos, en la que los elementos pueden ser de distinto tipo. Se definen con la función `list()`:

```
> 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$          ←—al pulsar el tabulador tras escribir $ se muestran los elementos de gal
gal$T_RC3 gal$colours gal$morf gal$name
> length(gal)                    ←—averiguamos cuántos elementos contiene gal
[1] 4
> names(gal)                     ←—mostramos los nombres de los diversos elementos
[1] "name" "morf" "T_RC3" "colours"
```


Listas:

Las diferentes componentes pueden accederse indicando la etiqueta utilizada para identificarlas:

```
> gal$morf
[1] "E"
> gal$colours
[1] 0.53 0.96

> gal[["morf"]]
[1] "E"
> gal[["colours"]]
[1] 0.53 0.96

> gal$colours[1]
[1] 0.53
> gal$colours[2]
[1] 0.96
```

Se pueden añadir nuevos elementos de forma trivial, simplemente definiéndolos:

```
> gal$radio <- TRUE                  ←—añadimos un elemento booleano
> gal$redshift <- 0.002922           ←—añadimos un elemento real
> names(gal)                         ←—mostramos todos los elementos definidos
[1] "name" "morf" "T_RC3" "colours" "radio" "redshift"
```

Listas:

Las listas se pueden concatenar para generar listas más grandes. Si por ejemplo tenemos tres listas `lista1, lista2, lista3`, podemos genera una lista única que sea la unión de esas tres listas mediante:

```
> lista123 <- c(lista1,lista2,lista3)
```

Como cada elemento de una lista puede ser un objeto de R de tipo diferente:

- Las listas son extremadamente versátiles para almacenar cualquier tipo de información (algo bueno)
- Las listas pueden convertirse en objetos de estructura bastante complicada (algo malo). Por ejemplo, una lista puede contener varios elementos que sean vectores de diferentes longitudes, lo cual se asemejaría a una tabla que tuviera columnas con distinto número de filas.

Lo ideal es beneficiarse de la versatilidad de las listas pero evitando que crezcan con una estructura excesivamente compleja. Por ello en R se ha definido un tipo de dato que posee ambas características y que se conoce como **Data Frame**.

Data frames:

Un *data frame* es un tipo particular de lista de gran utilidad para el trabajo estadístico. Debe cumplir algunas restricciones que garantizan que pueda ser utilizada para dicho cálculo estadístico.

Entre otras restricciones, un *data frame* debe verificar:

- Los componentes de la lista deben ser vectores (numéricos, de caracteres o de valores lógicos), factores, matrices numéricas, listas u otros data frames.
- Los vectores que constituyen variables en el data frame deben tener la misma longitud.

En un data frame, los vectores de caracteres se convierten automáticamente en factores, y el número de niveles se determina como el número de valores distintos que aparecen en dicho vector.

De forma básica, en un *data frame* toda la información se despliega como en una tabla en la que las columnas dan lugar a un mismo número de filas, y en el que las columnas pueden contener objetos de tipos distintos (números, caracteres,. . .).

Data frames:

Los *data frames* pueden definirse con la ayuda de la función `data.frame()`. En el siguiente ejemplo vamos a definir un *data frame* con dos elementos, un vector numérico y un vector de caracteres (nótese que tienen la misma longitud):

```
> df <- data.frame(numeros=c(10,20,30,40),texto=c("a","b","c","a"))
> df$numeros
[1] 10 20 30 40                                  ←—vector numérico de longitud 4
> df$texto
[1] a b c a                                      ←—vector de caracteres de longitud 4
Levels:  a b c                                   ←—¡se ha convertido en un factor de 3 niveles!
> tapply(df$numeros,df$texto,mean)               ←—calculamos la media usando los factores
a b c
25 20 30
> mode(df)
[1] "list"
> typeof(df)
[1] "list"
> class(df)
[1] "data.frame"
```

Sin embargo, la forma más habitual de definir un *data frame* es construirlo leyendo los datos almacenados en un fichero. Veremos más adelante cómo hacerlo mediante el uso de la función `read.table()`.

Funciones:

Objetos que pueden ser creados por el usuario y reutilizados para realizar operaciones específicas.

Por ejemplo, podemos definir una función para calcular la desviación típica:

```
> f <- function(x)
+ sqrt(sum((x-mean(x))^2)/(length(x)-1))          ←—definimos la función
```

Vamos a aplicar la función así definida al ejemplo visto anteriormente sobre los colores de diferentes galaxias:

```
> 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) de una muestra de 13 galaxias
> morfo <- c("Sab","E","Sab","S0","E","E","S0","S0","E","Sab",
+ "E","Sab","S0")                                 ←—información morfológica en el mismo orden
> fmorfo <- factor(morfo)                         ←—generamos el factor con factor()

> tapply(bv,fmorfo,f)
         E         S0        Sab
0.04358899 0.03774917 0.02753785          # probamos con nuestra función

> tapply(bv,fmorfo,sd)
         E         S0        Sab
0.04358899 0.03774917 0.02753785                  # ¡sale lo mismo!
```

Esquema


```
> a <- c(7+4,7-4,7*4,7/4)
> a
[1] 11.00 3.00 28.00 1.75
```
←operaciones aritméticas elementales $+, -, \times, \div$

```
> length(a)
[1] 4
```
←devuelve el tamaño del vector

```
> c(min(a),max(a))
[1] 1.75 28.00
```
←cálculo del mínimo y el máximo

```
> range(a)
[1] 1.75 28.00
```
←cálculo del mínimo y el máximo simultáneamente

```
> which.min(a)
[1] 4
```
←indica qué elemento contiene el mínimo

```
> which.max(a)
[1] 3
```
←indica qué elemento contiene el máximo

```
> sort(a)
[1] 1.75 3.00 11.00 28.00
```
←ordena los valores del vector

```
> sum(a)
[1] 43.75
```
←cálculo del sumatorio

```
> mean(a)
[1] 10.9375
```
←cálculo de la media

```
> median(a)
[1] 7
```
←cálculo de la mediana

```
> a <- c(7+4,7-4,7*4,7/4)
> a
[1] 11.00 3.00 28.00 1.75
```
←operaciones aritméticas elementales $+, -, \times, \div$

```
> var(a)
[1] 146.1823
```
←calcula la varianza

```
> sd(a)
[1] 12.09059
```
←calcula la desviación típica

```
> quantile(a,0.25)
   25%
2.6875
```
←calcula el primer cuartil (o percentil 25)

Existe un comando que de forma sencilla proporciona información estadística básica:

```
> summary(a)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.750   2.688   7.000  10.940  15.250  28.000
```

Funciones matemáticas importantes: `exp()`, `sin()`, `cos()`, `tan()`, `log()`, `log10()`,... Ejecutar `?Trig`, `?exp` para ver detalles.

Funciones matemáticas especiales: `beta(a,b)`, `gamma(x)`,... Ejecutar `?Special` para ver una lista.

Esquema

R permite importar datos desde ficheros ASCII.

Supongamos que el contenido del fichero `"galaxias.dat"` es el siguiente:

```
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
...
```

Dicho fichero puede leerse con la instrucción:

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

donde `header=TRUE` indica que la primera línea no contiene datos sino una etiqueta identificando el contenido de cada columna.

```
> gal                                    ←−−mostramos el contenido de gal por pantalla

    GALAXY morf T_RC3  U.B  B.V  # ¡las etiquetas U-B y B-V han cambiado!
1  NGC1357  Sab     2  0.25 0.87
2  NGC1832   Sb     4 -0.01 0.63
3  NGC2276   Sc     5 -0.09 0.52
4  NGC3245   S0    -2  0.47 0.91
5  NGC3379    E    -5  0.53 0.96
...
```


El fichero es leído como una lista:

```
> names(gal)
[1] "GALAXY" "morf"   "T_RC3"  "U.B"    "B.V"

> gal[["morf"]]            # las cadenas de texto se consideran factores
 [1] Sab Sb  Sc  S0  E   Sab Sb  S0  E   Im  S0  Sc
Levels: E Im S0 Sab Sb Sc

> tapply(gal$U.B,gal$morf,mean)
         E          Im          S0         Sab          Sb          Sc
 0.5400000 -0.2200000  0.4533333  0.3500000  0.0950000 -0.0600000
```

Puede evitarse tener que especificar continuamente el nombre de la lista y acceder al nombre de los diferentes campos de la lista a través de su nombre utilizando:

```
> attach(gal)                ←−−permite el acceso directo a los elementos de la lista gal
> morf
[1] Sab Sb Sc S0 E Sab Sb S0 E Im S0 Sc
Levels:  E Im S0 Sab Sb Sc

> detach(gal)                ←−−elimina el acceso directo a los elementos de la lista gal
```

Si el fichero que se va a leer sólo contiene números, también podemos leer la información y asignarla a una matriz en lugar de a una lista. Por ejemplo, supongamos que queremos leer un fichero que contiene 3 columnas:

```
> a <- matrix(scan("numeros.dat",0),ncol=3,byrow=TRUE)

> a
      [,1]  [,2] [,3]
 [1,]    2  0.25 0.87
 [2,]    4 -0.01 0.63
 [3,]    5 -0.09 0.52
 [4,]   -2  0.47 0.91
 [5,]   -5  0.53 0.96
 [6,]    1  0.45 0.92
 [7,]    3  0.20 0.73
 [8,]   -3  0.51 0.94
 [9,]   -5  0.55 0.96
[10,]   10 -0.22 0.39
[11,]   -1  0.38 0.85
[12,]    5 -0.03 0.63
```

Si no se especifica `ncol` se genera un array unidimensional con todos los elementos del fichero.

Si no se especifica `byrow=TRUE` la asignación de los elementos no conserva la información de las columnas.

R contiene muchos datos de ejemplo. Todas las funciones y conjuntos de datos se almacenan en paquetes.

Para ver qué paquetes están instalados:

```
> library()
Packages in library '/home/cardiel/s/R-2.8.1/library':

base                    The R Base Package
boot                    Bootstrap R (S-Plus) Functions (Canty)
...
```

Para solicitar información sobre un paquete dado:

```
> library(help = splines)
```
←—muestra ayuda sobre el paquete `splines`

Para cargar un paquete y poder utilizar su funcionalidad:

```
> library(splines)
```
←—carga el paquete `splines`

Para ver qué listas de datos están disponibles en un momento dado:

```
>  data()
Data sets in package 'datasets':

AirPassengers           Monthly Airline Passenger Numbers 1949-1960
BJsales                 Sales Data with Leading Indicator
...
```

Para ver qué listas están disponibles en un paquete concreto:

```
> data(package="cluster")          ←muestra listas de datos en el paquete cluster
> data(animals,package="cluster")  ←carga la lista animals del paquete cluster
```


Esquema

R permite realizar gráficos de forma sencilla:

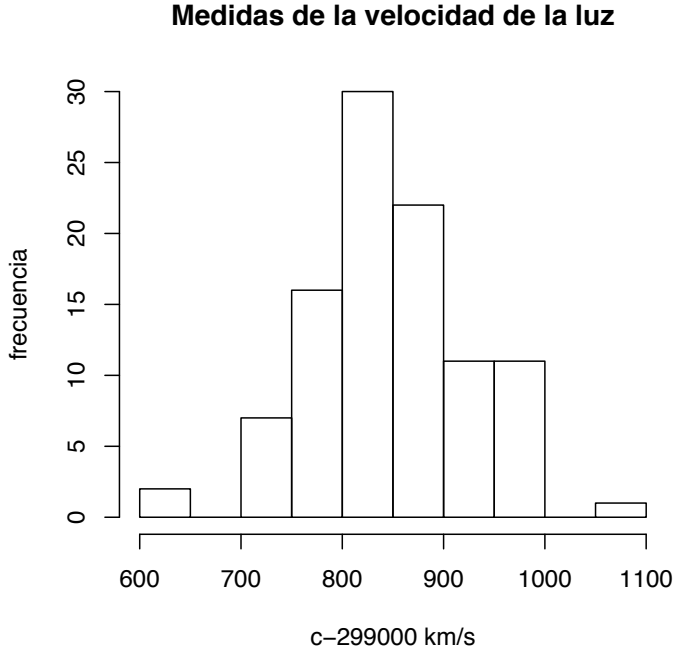
Tomamos como ejemplo las medidas de la velocidad de la luz realizadas por Michelson y Morley en su famoso experimento. Realizaron 5 series (`Expt`) de 20 medidas cada una (`Run`):

```
> data(morley)
> morley
    Expt Run Speed
001    1   1   850
002    1   2   740
003    1   3   900
004    1   4  1070
...  ...  ..  ....
100    5  20   870
```

⟵los datos están en el paquete `base`, cargado por defecto

Generamos un histograma:

```
> hist(morley$Speed,
+ main="Medidas de la...",
+ xlab="c-299000 km/s",
+ ylab="frecuencia")
```




```
> hip <- read.table("http://www.iiap.res.in/astrostat/tuts/HIP.dat",
+ header=TRUE)
```

⟵leemos fichero WEB

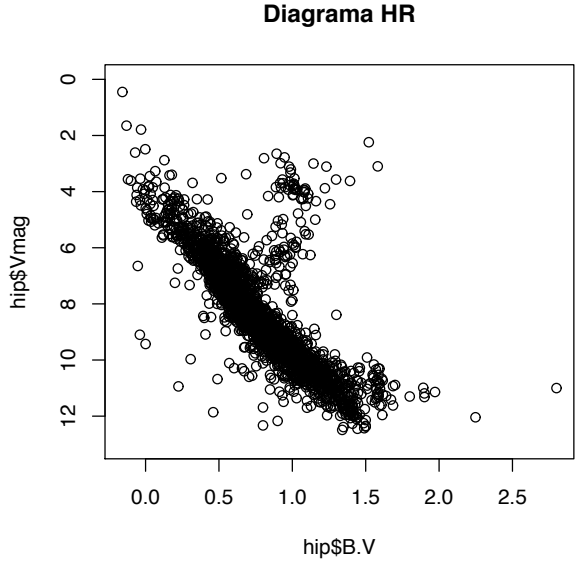
```
> names(hip)
```

⟵mostramos componentes de `hip`

```
[1] "HIP" "Vmag" "RA" "DE" "Plx" "pmRA" "pmDE" "e_Plx" "B.V"
```

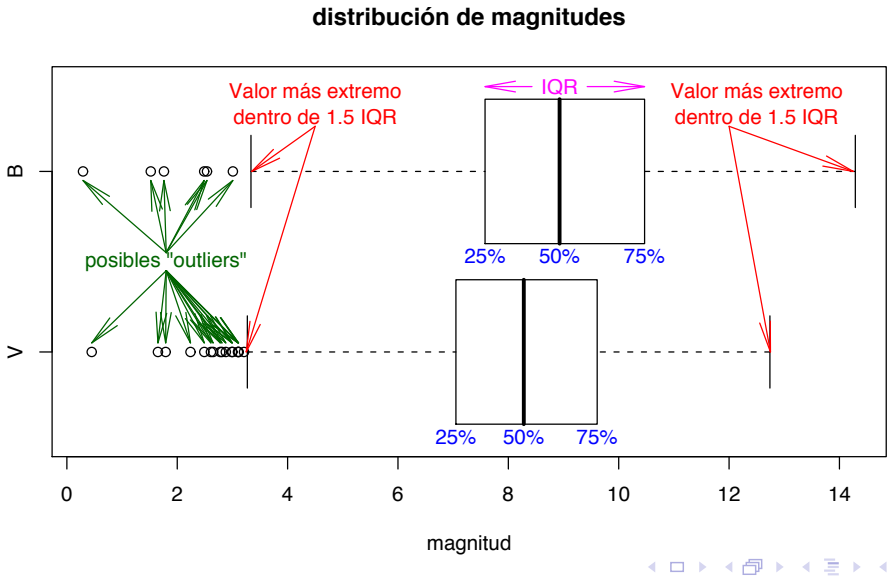
Generamos un diagrama de dispersión:

```
> plot(hip$B.V,hip$Vmag,ylim=c(13,0),main="Diagrama HR")
```



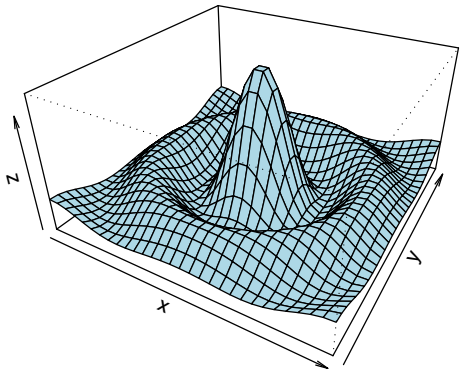
Vamos a analizar rápidamente la distribución magnitudes de las estrellas cuyos datos acabamos de leer. Para ello vamos a generar una nueva lista conteniendo sólo dos componentes:

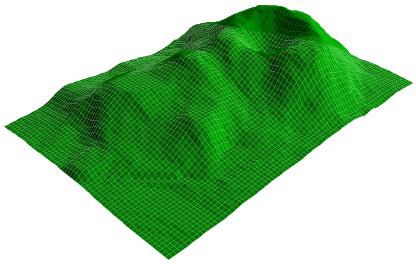
```
> hipBmag <- hip$B.V + hip$Vmag          ←—calculamos magnitudes en la banda B
> newlist = list(V = hip$Vmag, B = hipBmag)      ←—generamos una nueva lista
> names(newlist)                          ←—mostramos componentes de la nueva lista
[1] "V" "B"
> boxplot(newlist,                        ←—generamos un box-and-whiskers plot
+ horizontal=TRUE,main="distribución de magnitudes",xlab="magnitud")
```



R permite representaciones de datos tridimensionales. Ver, por ejemplo:

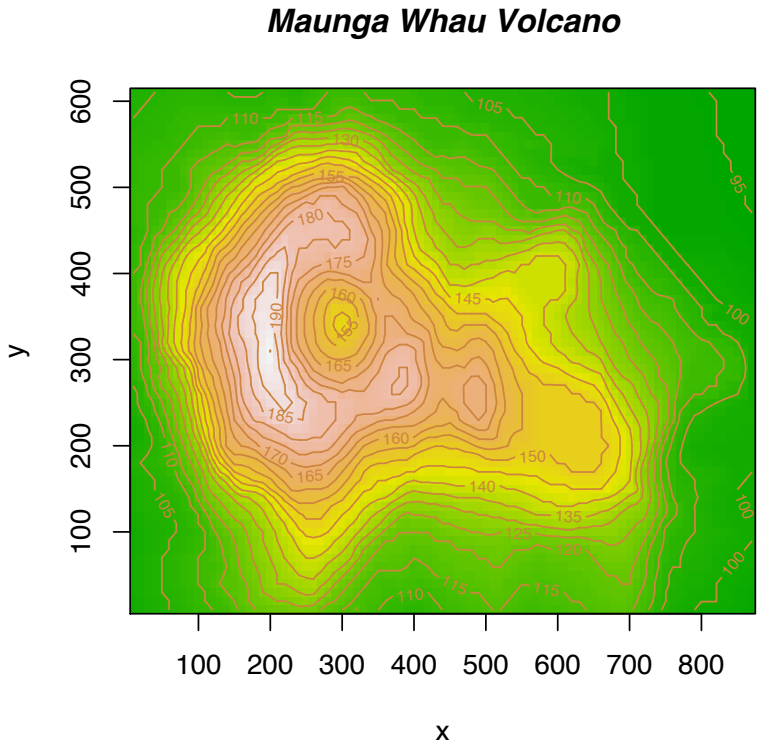
```
> demo(persp)
...
```





R permite representaciones de imágenes. Ver, por ejemplo:

```
> demo(image)
...
```



Esquema

R posee una biblioteca muy completa de funciones estadísicas, incluyendo las distribuciones de probabilidad más usuales:

Distribution	R name	additional arguments
beta	`beta`	`shape1, shape2, ncp`
binomial	`binom`	`size, prob`
Cauchy	`cauchy`	`location, scale`
chi-squared	`chisq`	`df, ncp`
exponential	`exp`	`rate`
F	`f`	`df1, df2, ncp`
gamma	`gamma`	`shape, scale`
geometric	`geom`	`prob`
hypergeometric	`hyper`	`m, n, k`
log-normal	`lnorm`	`meanlog, sdlog`
logistic	`logis`	`location, scale`
negative binomial	`nbinom`	`size, prob`
normal	`norm`	`mean, sd`
Poisson	`pois`	`lambda`
Student's t	`t`	`df, ncp`
uniform	`unif`	`min, max`
Weibull	`weibull`	`shape, scale`
Wilcoxon	`wilcox`	`m, n`

Evaluando las distribuciones de probabilidad

Para cada distribución de probabilidad tenemos varias funciones asociadas, a las cuales se accede añadiendo un prefijo al nombre de la distribución:

Prefijo	Significado
`d`	distribución de probabilidad
`p`	función de distribución
`q`	inversa de la función de distribución
`r`	generación aleatoria de números que siguen la distribución de probabilidad

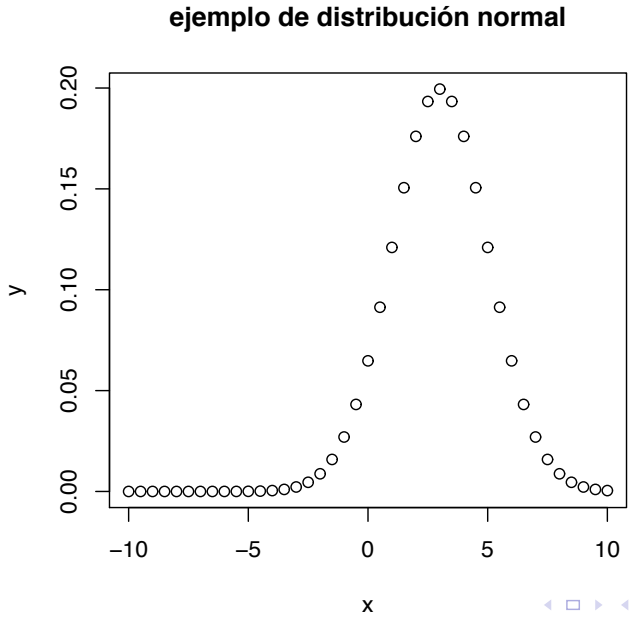
Los argumentos de cada una de las funciones asociadas serán, lógicamente, diferentes. Por ejemplo, para la distribución normal:

```
> ?Normal
...
     dnorm(x, mean = 0, sd = 1, log = FALSE)
     pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
     qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
     rnorm(n, mean = 0, sd = 1)
...
```

1. `dnorm(x, mean = 0, sd = 1, log = FALSE)`

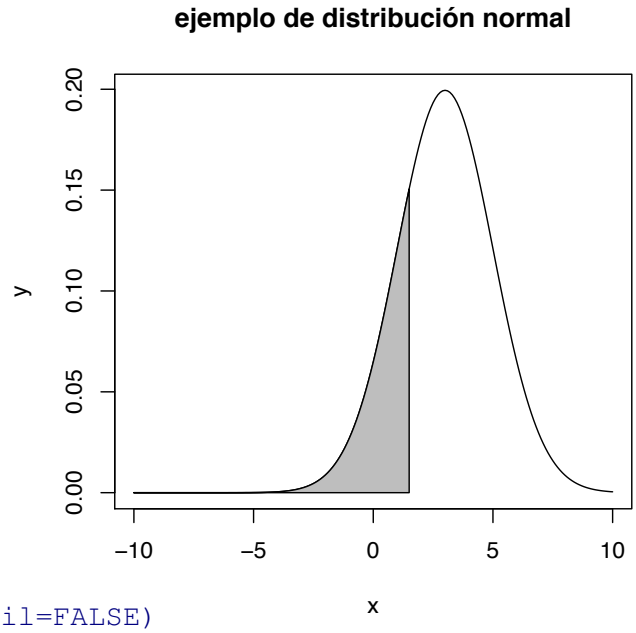
Evalúa la distribución de probabilidad de una distribución normal con media `mean` y desviación típica `sd` en la abcisa `x`:

```
> x <- seq(-10,10,by=.5)                    ←—generamos secuencia de números
> y <- dnorm(x, mean=3, sd=2)      ←—calculamos la función normal con μ = 3 y σ = 2
> plot(x,y,main="ejemplo de distribución normal")   ←—dibujamos el resultado
```



2. `pnorm(q, mean = 0, sd = 1, lower.tail = FALSE, log.p = FALSE)`

Evalúa la función de distribución (área bajo la distribución de probabilidad) de una distribución normal con media `mean` y desviación típica `sd`. Por defecto `lower.tail = TRUE` devuelve el área de la cola de la izquierda. Si `lower.tail = FALSE` devuelve la cola de la derecha.



Cola de la izquierda (defecto):

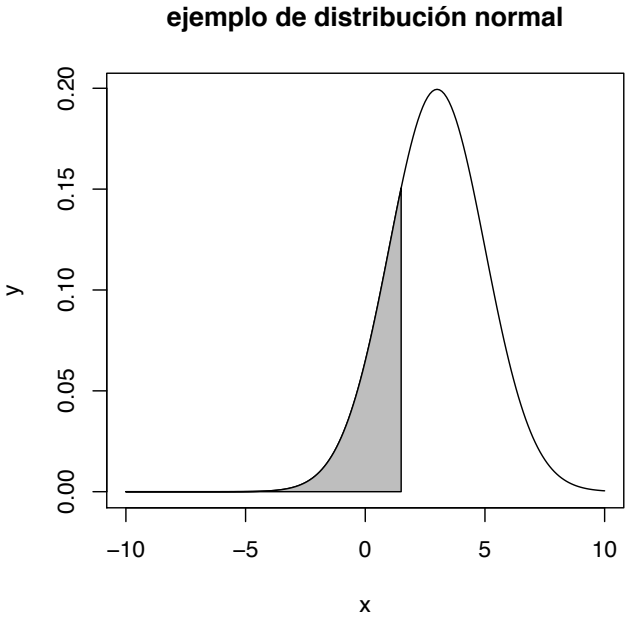
```
> pnorm(1.5,mean=3,sd=2)
[1] 0.2266274
```

Cola de la derecha:

```
> pnorm(1.5,mean=3,sd=2,lower.tail=FALSE)
[1] 0.7733726
```

3. `qnorm(p, mean = 0, sd = 1, lower.tail = FALSE, log.p = FALSE)`

Evalúa la función de inversa de la función de distribución (abcisa que de una área `p` bajo la distribución de probabilidad) de una distribución normal con media `mean` y desviación típica `sd`. Por defecto `lower.tail = TRUE` asume que el área de entrada corresponde a la cola de la izquierda. Si `lower.tail = FALSE` asume que el área corresponde a la cola de la derecha.



Cola de la izquierda (defecto):

```
> qnorm(0.2266274,mean=3,sd=2)
[1] 1.500000
```

Cola de la derecha:

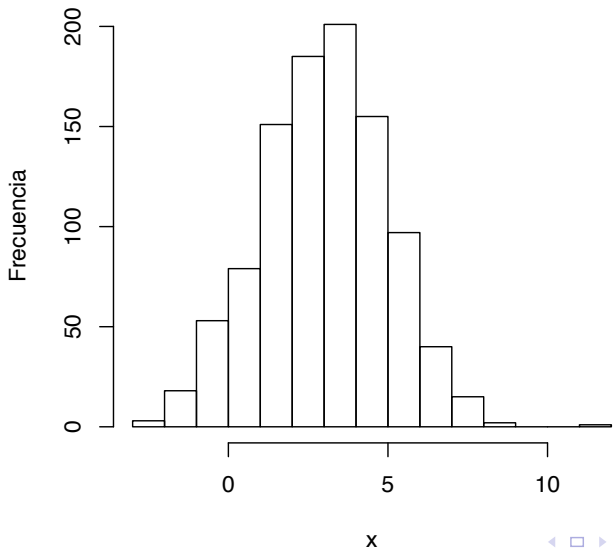
```
> qnorm(0.7733726,mean=3,sd=2,lower.tail=FALSE)
[1] 1.500000
```


4. `rnorm(n, mean = 0, sd = 1)`

Genera una secuencia aleatoria de `n` números siguiendo una distribución normal con media `mean` y desviación típica `sd`.

```
> x <- rnorm(1000,mean=3,sd=2)
> hist(x,main="simulación de distribución normal",
+ ylab="Frecuencia")
```

←−generamos 1000 números con $\mu = 3$ y $\sigma = 2$



Distribuciones de probabilidad más comunes

distribución	funciones asociadas
Uniforme	`dunif, punif, qunif, runif`
Binomial	`dbinom, pbinom, qbinom, rbinom`
Poisson	`dpois, ppois, qpois, rpois`
...	`d..., p..., q..., r...`
Normal	`dnorm, pnorm, qnorm, rnorm`
t de Student	`dt, pt, qt, rt`
χ^2	`dchisq, pchisq, qchisq, rchisq`
F de Fisher	`dt, pt, qt, rt`
...	`d..., p..., q..., r...`

Esquema

Revolviendo un algoritmo

El trabajo de Böhm and Jacopini[a] demostró que los programas de ordenador pueden crearse utilizando exclusivamente tres estructuras de control:

1. **Estructura de secuencia**: salvo que se indique lo contrario, las instrucciones se ejecutan en el orden secuencial en que están escritas. En R esto es inherente al ejecutar comandos de forma interactiva a través del intérprete, y es también el modo en el que se ejecutan las instrucciones de un *script*.
2. **Estructura de selección:** permite ejecutar diferentes instrucciones en función de una condición. En R este tipo de estructura viene implementada a través de la instrucción:
   ```
   > if (expr_1) expr_2 else expr_3
   ```
3. **Estructura de repetición:** permite repetir la ejecución de un conjunto de instrucciones en un bucle. En R este tipo de estructura puede realizarse con la instrucciones:
   ```
   > for (name in expr_1) expr_2
   > while (condition) expr
   > repeat expr
   ```

[a] Böhm C., and Jacopini G,. *Flow Diagrams, Turing Machines, and Languages with Only Two Formation Rules*, Communications of the ACM, Vol 9., No. 5, 1966, 336–371.

Uso práctico de las estructuras de control

Cualquier algoritmo puede resolverse combinando las estructuras de control descritas. Dichas estructuras pueden incluirse unas dentro de otras. Para ello resulta útil utilizar las llaves `{...}` para agrupar bloques de instrucciones y “sangrar” el texto a la hora de escribir scripts.

```
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")
  }
}
```

Esquema

Ejercicio del Tema 1

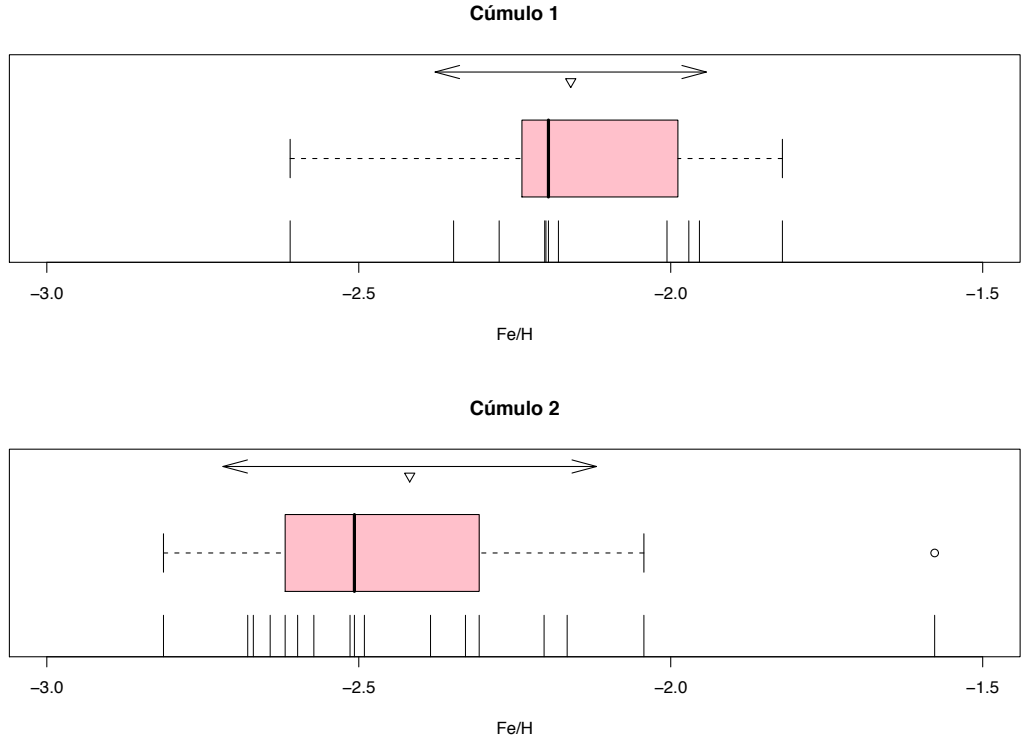
A partir de observaciones espectroscópicas se ha determinado la metalicidad de varias estrellas pertenecientes a dos cúmulos globulares. Dicha metalicidad se calcula como

$$[\text{Fe/H}] = \log_{10}\left(\frac{N_{\text{Fe}}}{N_{\text{H}}}\right)_{\text{estrella}} - \log_{10}\left(\frac{N_{\text{Fe}}}{N_{\text{H}}}\right)_{\text{sol}}$$

Se asume que en cada cúmulo las estrellas se formaron en un único brote a partir de una nube de gas con una composición química homogénea. Las diferencias entre las metalicidades deducidas para las distintas estrellas dentro de cada cúmulo son debidas a errores de medida aleatorios. A partir de los datos disponibles, ¿se puede afirmar que los dos cúmulos globulares tienen la misma metalicidad? Utilizar como niveles de significación $\alpha = 0.05$ y $\alpha = 0.01$.

	Metalicidades de las estrellas individuales					
Cúmulo 1	−2.348	−2.006	−1.821	−2.610	−1.971	−1.954
	−2.202	−2.196	−2.200	−2.275	−2.180	
Cúmulo 2	−2.669	−2.598	−2.329	−2.043	−2.385	−2.514
	−2.642	−2.618	−1.577	−2.307	−2.507	−2.491
	−2.203	−2.572	−2.813	−2.166	−2.678	

Box-and-whiskers plot:



Inicializamos la semilla del generador de número aleatorios a un valor pre-definido. De esta forma podremos volver a repetir la misma secuencia de valores:

```
> set.seed(1234)
```
←—fijamos la semilla

Generamos las metalicidades para el primer cúmulo:

```
> n1 <- 11
```
←—número de estrellas

```
> feh1 <- rnorm(n1,mean=-2.070,sd=0.230)
```
←—dist. normal $\mu = -2.070$ y $\sigma = 0.230$

```
> feh1 <- round(feh1,3)
```
←—redondeamos a tres decimales

```
> feh1
```
←—mostramos los valores en pantalla

```
 [1] -2.348 -2.006 -1.821 -2.610 -1.971 -1.954 -2.202 -2.196 -2.200
[10] -2.275 -2.180
```

Calculamos valores muestrales para la media, desviación típica y varianza:

```
> mean_feh1 <- mean(feh1)
```
←—media muestral

```
> stdv_feh1<-sd(feh1)
```
←—desviación típica muestral

```
> var_feh1 <- var(feh1)
```
←—varianza muestral

Hacemos lo mismo para el segundo cúmulo:

```
> n2 <- 17
```
←—número de estrellas

```
> feh2 <- rnorm(n2,mean=-2.350,sd=0.320)
```
←—dist. normal $\mu = -2.350$ y $\sigma = 0.320$

```
> feh2 <- round(feh2,3)
```
←—redondeamos a tres decimales

```
> mean_feh2 <- mean(feh2)
```
←—media muestral

```
> stdv_feh2<-sd(feh2)
```
←—desviación típica muestral

```
> var_feh2 <- var(feh2)
```
←—varianza muestral

Establecemos los niveles de significación que vamos a utilizar:

```
> alpha <- c(0.01,0.05)
```
←—usaremos $\alpha = 0.01$ y $\alpha = 0.05$

Test de igualdad de varianzas:

La hipótesis nula $H_0 : \sigma_1^2 = \sigma_2^2$ se acepta si se verifica

$$F \equiv \frac{s_1^2}{s_2^2} \in [F_{1-\alpha/2,n_1-1,n_2-1}, F_{\alpha/2,n_1-1,n_2-1}]$$

```
Ftest<-round(var_feh1/var_feh2,3)
for (i in 1:length(alpha)) {
  lim1<-round(qf(1-alpha[i]/2,n1-1,n2-1,lower.tail=FALSE),3)
  lim2<-round(qf(alpha[i]/2,n1-1,n2-1,lower.tail=FALSE),3)
  cat("alpha=",alpha[i],": [",lim1,",",lim2,"]")
  if((lim1 < Ftest) && (lim2>Ftest)) {
    cat(" => se acepta H0\n")
  } else {
    cat(" => se rechaza H0\n")
  }
}
```

El código anterior calcula en `Ftest` el cociente entre las varianzas muestrales. Los límites del intervalo se calculan utilizando la función `qf(...)` que devuelve la abcisa de la distribución F de Fisher que deja a su derecha (`lower.tail=FALSE`) un área determinada.

Test de igualdad de medias (caso $n_1 + n_2 \leq 30$ y $\sigma_1 = \sigma_2$):

La hipótesis nula $H_0 : \mu_1 = \mu_2$ se acepta si se verifica

$$t \equiv \frac{|\overline{x_1} - \overline{x_2}|}{s_p \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \leq t_{\alpha/2,n_1+n_2-2}$$

donde

$$s_p^2 = \frac{(n_1 - 1)s_1^2 + (n_2 - 1)s_2^2}{n_1 + n_2 - 2}$$

```
sp2<-((n1-1)*var_feh1+(n2-1)*var_feh2)/(n1+n2-2)
ttest<-abs(mean_feh1-mean_feh2)/(sqrt(sp2)*sqrt(1/n1+1/n2))
for (i in 1:length(alpha)) {
  tlim<-qt(alpha[i]/2,n1+n2-2,lower.tail=FALSE)
  cat("alpha=",alpha[i],": t_alpha/2,n1+n2-2=",round(tlim,3))
  if(ttest <= tlim) {
    cat(" => se acepta H0\n")
  } else {
    cat(" => se rechaza H0\n")
  }
}
```

`tlim` es el estadístico de prueba que comparamos con el valor que retorna la función `qt(...)` que calcula $t_{\alpha/2,n1+n2-2}$.

Test de igualdad de medias (caso $n_1 + n_2 \leq 30$ y $\sigma_1 \neq \sigma_2$):

La hipótesis nula $H_0 : \mu_1 = \mu_2$ se acepta si se verifica

$$t \equiv \frac{|\overline{x_1} - \overline{x_2}|}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \leq t_{\alpha/2,f}$$

donde

$$f = \frac{\left(\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}\right)^2}{\frac{(s_1^2/n_1)^2}{n_1+1} + \frac{(s_2^2/n_2)^2}{n_2+1}} - 2$$

```
ttest<-abs(mean_feh1-mean_feh2)/sqrt(var_feh1/n1+var_feh2/n2)
fwelch<-((var_feh1/n1+var_feh2/n2)^2)/(((var_feh1/n1)^2)/(n1+1)+
                                       ((var_feh2/n2)^2)/(n2+1))-2
fwelch<-round(fwelch)
for (i in 1:length(alpha)) {
  tlim<-qt(alpha[i]/2,fwelch,lower.tail=FALSE)
  cat("alpha=",alpha[i],": t_alpha/2,f=",round(tlim,3))
  if(ttest <= tlim)
  cat(" => se acepta H0\n") else cat(" => se rechaza H0\n")
}
```

`ttest` es el estadístico de prueba que comparamos con el valor que retorna la función `qt(...)` que calcula $t_{\alpha/2,f}$, y `fwelch` es el número de grados de libertad.

Proyecto

Web del proyecto: `http://www.r-project.org`
Descarga del software: `http://cran.r-project.org`

Enlaces de interés

Centro de Astroestadística: `http://astrostatistics.psu.edu`

Tutoriales

- Tutorial 1: `http://cran.r-project.org/doc/manuals/R-intro.html`
- Tutorial 2: `http://www.cyclismo.org/tutorial/R`
- Tutorial 3: `http://zoonek2.free.fr/UNIX/48_R/all.html`