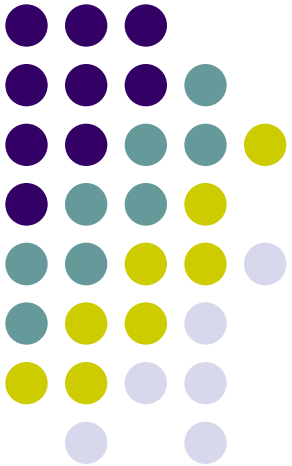
R: Entorno de análisis y programación estadística

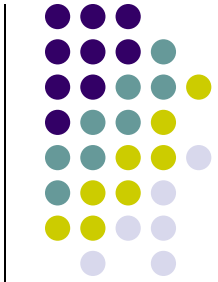


- Introducción: Qué es R
- Objetos en R

http://www.r-project.org/

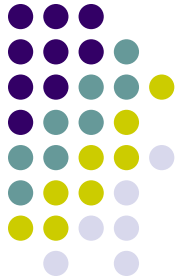
GNU Project

Objetivos



- Conocer qué es R y valorar sus capacidades frente a otros programas estadísticos.
- Facilitar una sesión inicial de trabajo con R bajo Windows.
- Conocer y manipular los tipos de objetos habituales en R.
- Realizar operaciones básicas con vectores y matrices.
- Generar y manipular marcos de datos (data.frame).
- Aprender a incorporar datos externos a R para su análisis estadístico.

Metodología



A continuación se recoge de modo esquemáticos algunos de los conceptos fundamentales introducidos en los temas 1 y 2 del bloque dedicado a R.

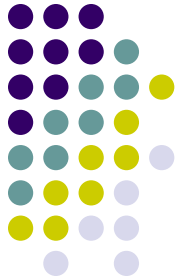
Se recomienda al estudiante que utilice estos materiales junto con los propios del tema, escribiendo en un fichero de órdenes (script) los ejemplos allí recogidos.

La autoevaluación y puesta en práctica de los contenidos vendrá dada por la resolución de los ejercicios propuestos a lo largo de esta presentación. Las soluciones están disponibles en ficheros auxiliares para su autocorrección, no obstante sugerimos al estudiante que intente hacerlos antes de comprobar la solución.

El foro del curso permitirá discutir y resolver los problemas que vayan surgiendo.

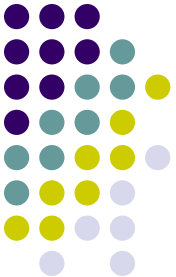
¿Qué es R?

= proyecto + lenguaje + programa de análisis estadístico
= *entorno de análisis y programación estadística*



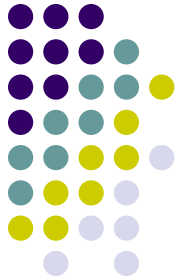
- Es parte del proyecto de software libre GNU (General Public Licence, www.gnu.org): *absolutely no warranty*.
- El proyecto R comenzó en 1995 por un grupo de estadísticos de la universidad de Auckland, dirigidos por Ross Ihaka y Robert Gentleman.
- R está basado en el lenguaje de programación S, diseñado específicamente para la programación de tareas estadísticas en los años 80 por los Laboratorios Bell AT&T. El lenguaje S se considera un lenguaje de programación estadística orientado a objetos de alto nivel.
- Frente a otros lenguajes de programación, R es sencillo, intuitivo y eficiente ya que se trata de un lenguaje interpretado (a diferencia de otros como Fortran, C++, Visual Basic, etc.).
- Como programa de análisis estadístico, R-base permite realizar tareas estadísticas habituales (análisis descriptivos, cálculo de probabilidades, inferencia estadística básica, etc.). Además permite extensiones que implementan técnicas estadísticas avanzadas, de modo que cubre las necesidades de cualquier analista, tanto en el ámbito de la estadística profesional como en el de la investigación estadística.

¿Qué diferencia a R de otros lenguajes/programas estadísticos?



- En R todo es un objeto y el trabajo en R consiste básicamente en crear y manipular objetos.
- De forma muy general, distinguimos dos tipos de objetos básicos: las **funciones** y los **objetos de datos**:
  - Los objetos de datos son lo que habitualmente denominamos variables.
  - Las funciones son objetos que realizan tareas sobre objetos de datos. Habitualmente consisten en cálculos, representaciones gráficas o análisis de datos en general. Estas funciones se obtienen como parte del sistema base de R, mediante extensiones de dicho sistema base (*packages* adicionales) o bien mediante su creación durante una sesión de trabajo.
- En este sentido podemos decir que *R es el sistema o entorno que nos va a permitir crear objetos de datos y manipular dichos objetos usando funciones.*

Libros (packages) en R



R consta de un sistema base que se extiende incorporando distintos ***packages*** adicionales (ver listado actual en http://cran.es.r-project.org/):

- **base** El libro que contiene las funciones y objetos base
- **boot** Métodos bootstrap
- **cluster** Funciones y datos para análisis clúster
- **ctest** Tests clásicos
- **eda** Análisis exploratorio de datos
- **foreign** Incorporar datos desde formatos Minitab, S, SAS, SPSS, Stata, ...
- **maxLik** Estimación por máxima verosimilitud
- **distr** Distribuciones de probabilidad
- **mva** Análisis Multivariante clásico
- **survival** Análisis de supervivencia
- **Rcmdr** R Commander: interfaz gráfico (GUI) para realizar análisis estadísticos básicos con R

Documentación, manuales y ayuda

Accesible desde http://cran.es.r-project.org/

- Documentación general (en español):

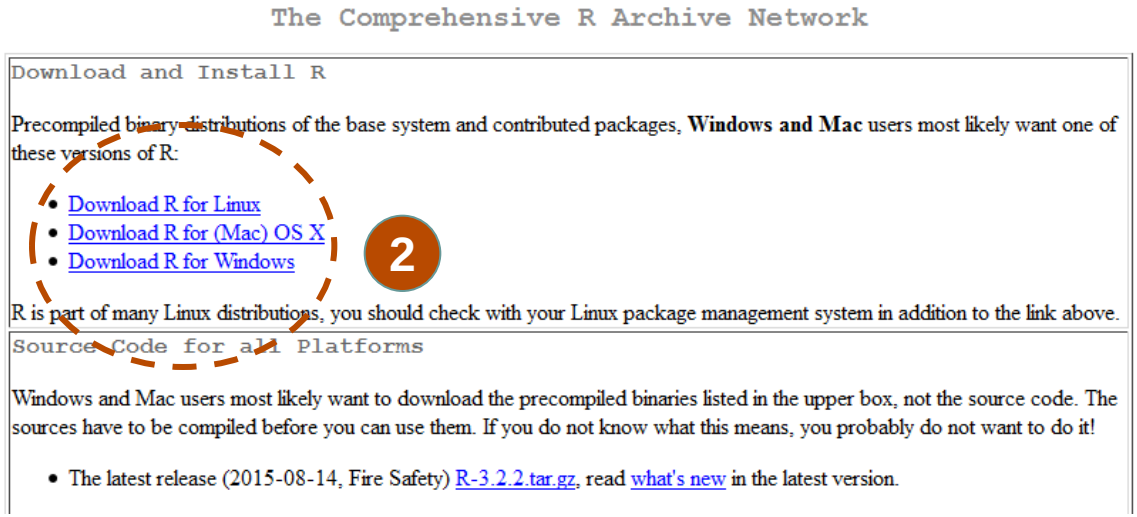
Spanish

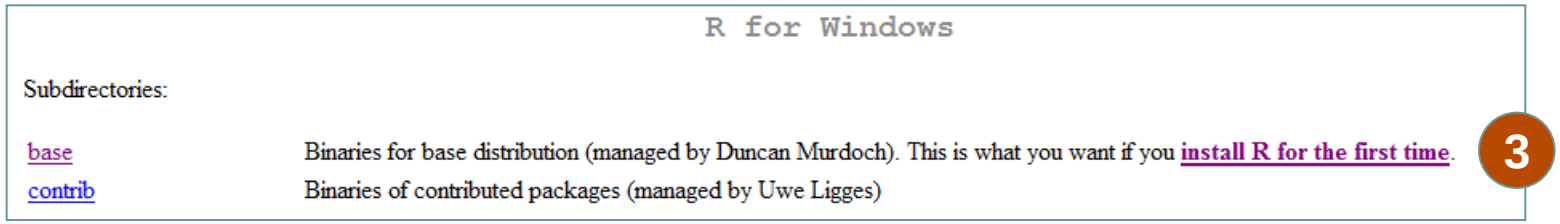
- **"R para Principiantes"**, the Spanish version of "R for Beginners", translated by Jorge A. Ahumada (PDF).
- A Spanish translation of **"An Introduction to R"** by Andrés González and Silvia González (PDF, Texinfo sources).
- **"Gráficos Estadísticos con R"** by Juan Carlos Correa and Nelfi González (PDF).
- **"Cartas sobre Estadística de la Revista Argentina de Bioingeniería"** by Marcelo R. Risk (PDF).
- **"Introducción al uso y programación del sistema estadístico R"** by Ramón Díaz-Uriarte, transparencies prepared for a 16-hours course on R, addressed mainly to biologists and bioinformaticians (PDF).
- **"Generacion automatica de reportes con R y LaTeX"** by Mario Alfonso Morales Rivera (PDF).
- **"Metodos Estadisticos con R y R Commander"** by Antonio Jose Saez Castillo (PDF, ZIP, 2010-07-08).
- **"Optimización Matemática con R: Volumen I"** by Enrique Gabriel Baquela and Andrés Redchuk (PDF, 161 pages).
  Data sets and complementary information are available at http://www.modelizandosistemas.com.ar/p/optimizacion-con-r.html.
- **"Introducción al uso de R y R Commander para el análisis estadístico de datos en ciencias sociales"** by Rosario Collatón Chicana (PDF, 128 pages, 2014-05-11).
- **"El arte de programar en R"** by Julio Sergio Santana and Efraín Mateos Farfán (PDF, 197 pages, 2014-12-15; online).

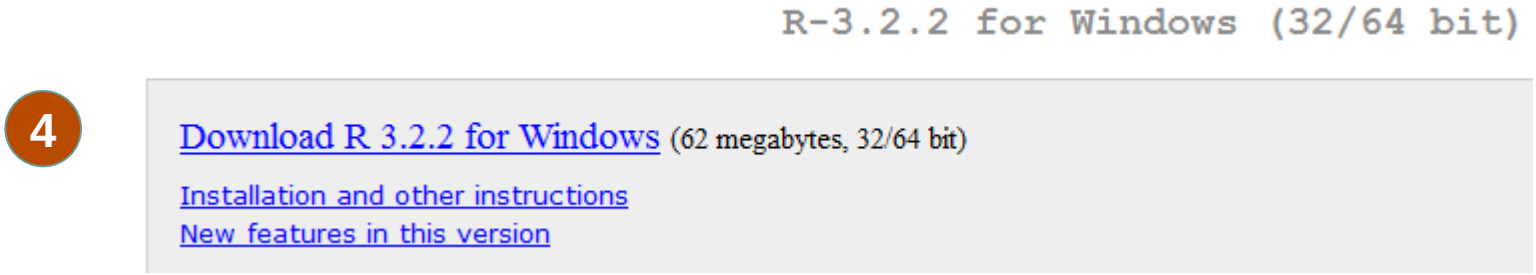
- Documentación específica sobre los *contributed packages*
- Ayuda a través de:
  - El propio programa (funciones help, Help.Start,…)
  - FAQ, Mailing Lists, Wiki.

Instalación de R bajo Windows

- Se puede descargar la última versión en CRAN



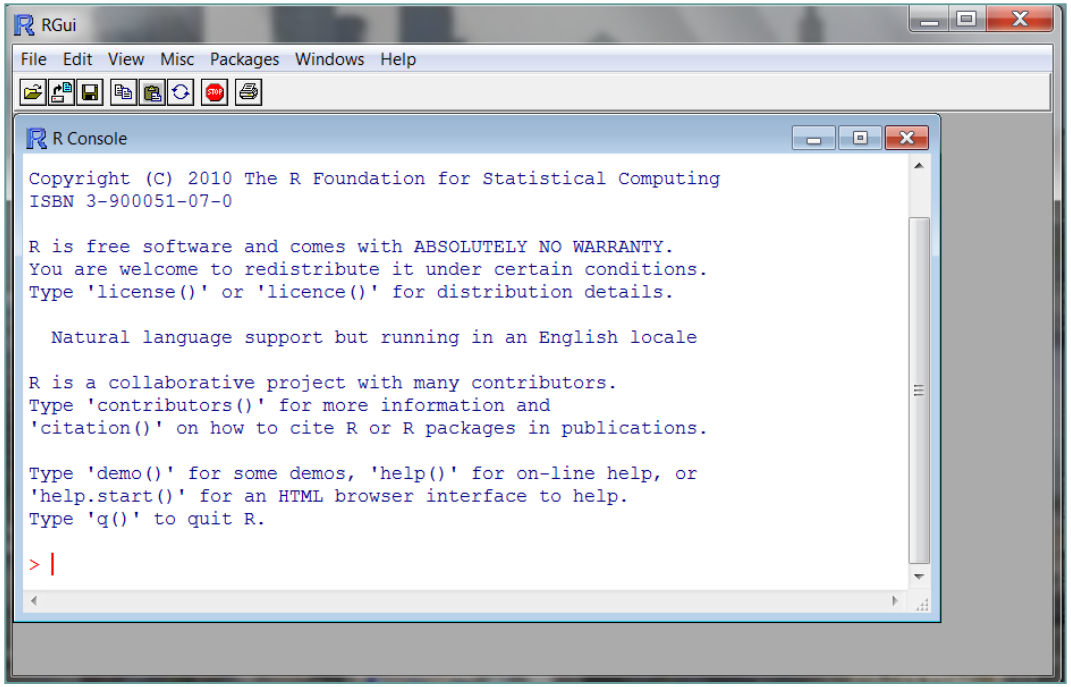




- Ejecutar el fichero de instalación (R-3.2.2.exe), con opciones estándar

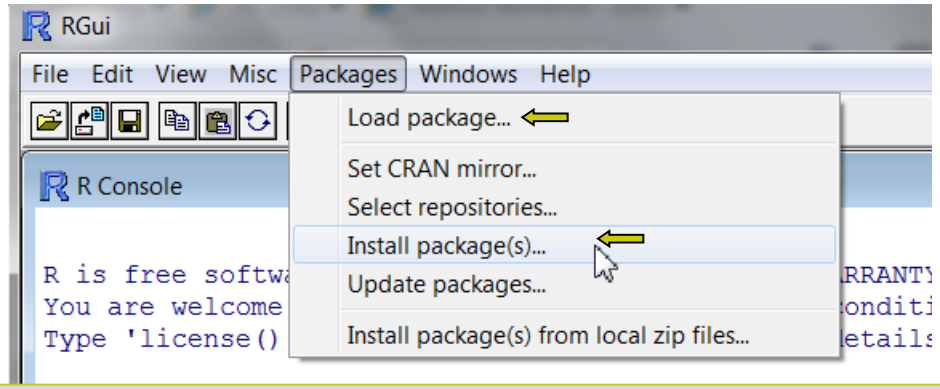
Ejecutando R bajo Windows

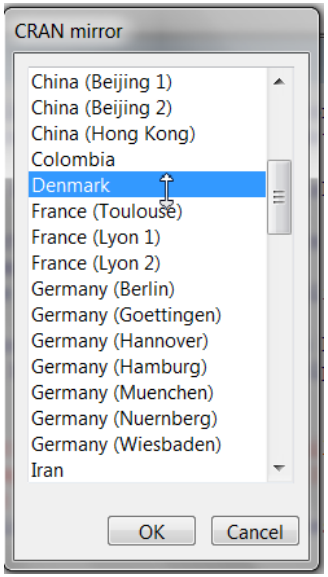
- Por defecto el programa funciona bajo el interfaz RGui

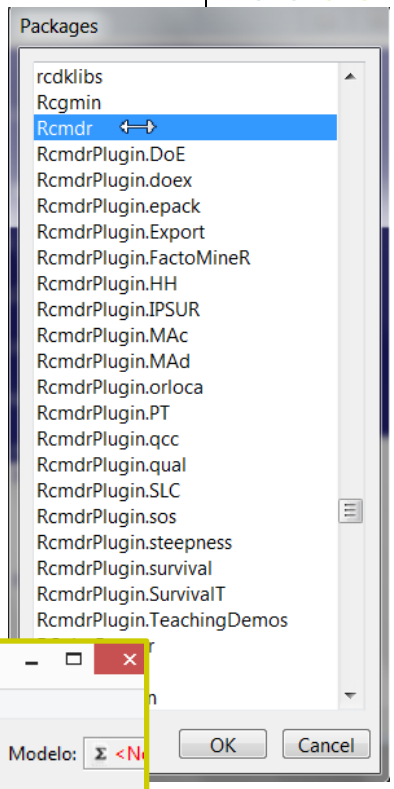


- **RGui**: R Graphical User Interface.
- El símbolo '>' seguido por el cursor indica que R está esperando que se escriban órdenes.
- Es posible usar otros interfaces: **R-studio** (http://www.rstudio.org/) ,Tinn-R (http://sourceforge.net/projects/tinn-r/), XEmacs, R-WinEdt, R-commander,…

Instalación y carga de contributed packages a través de RGui

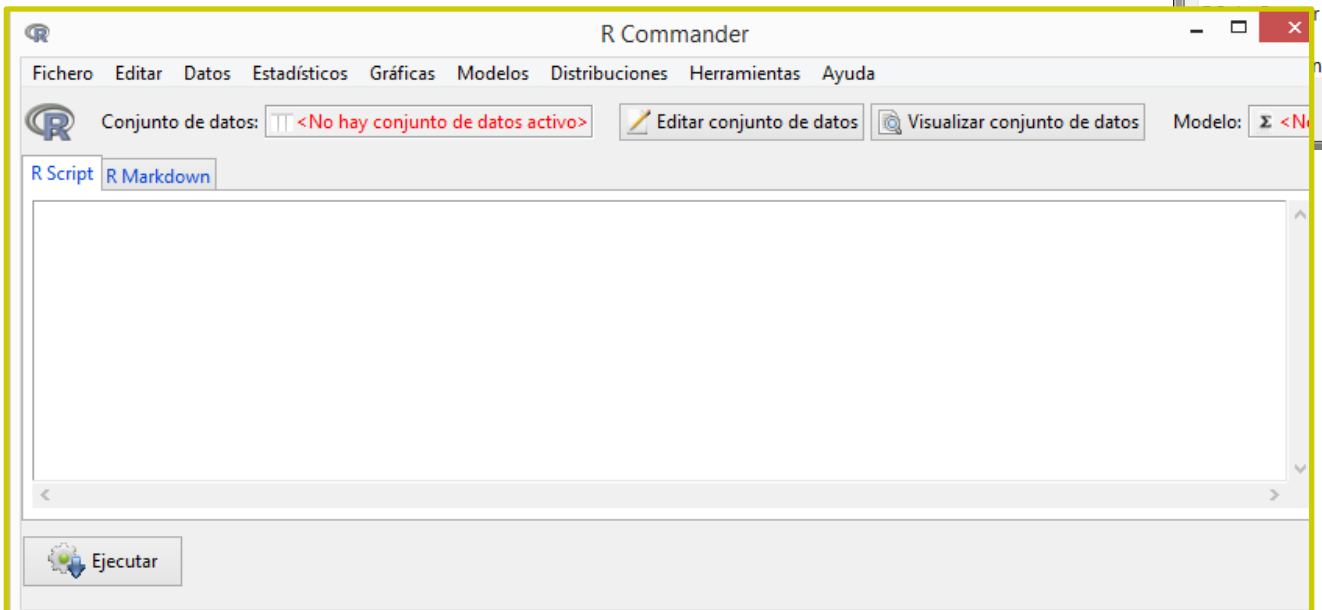




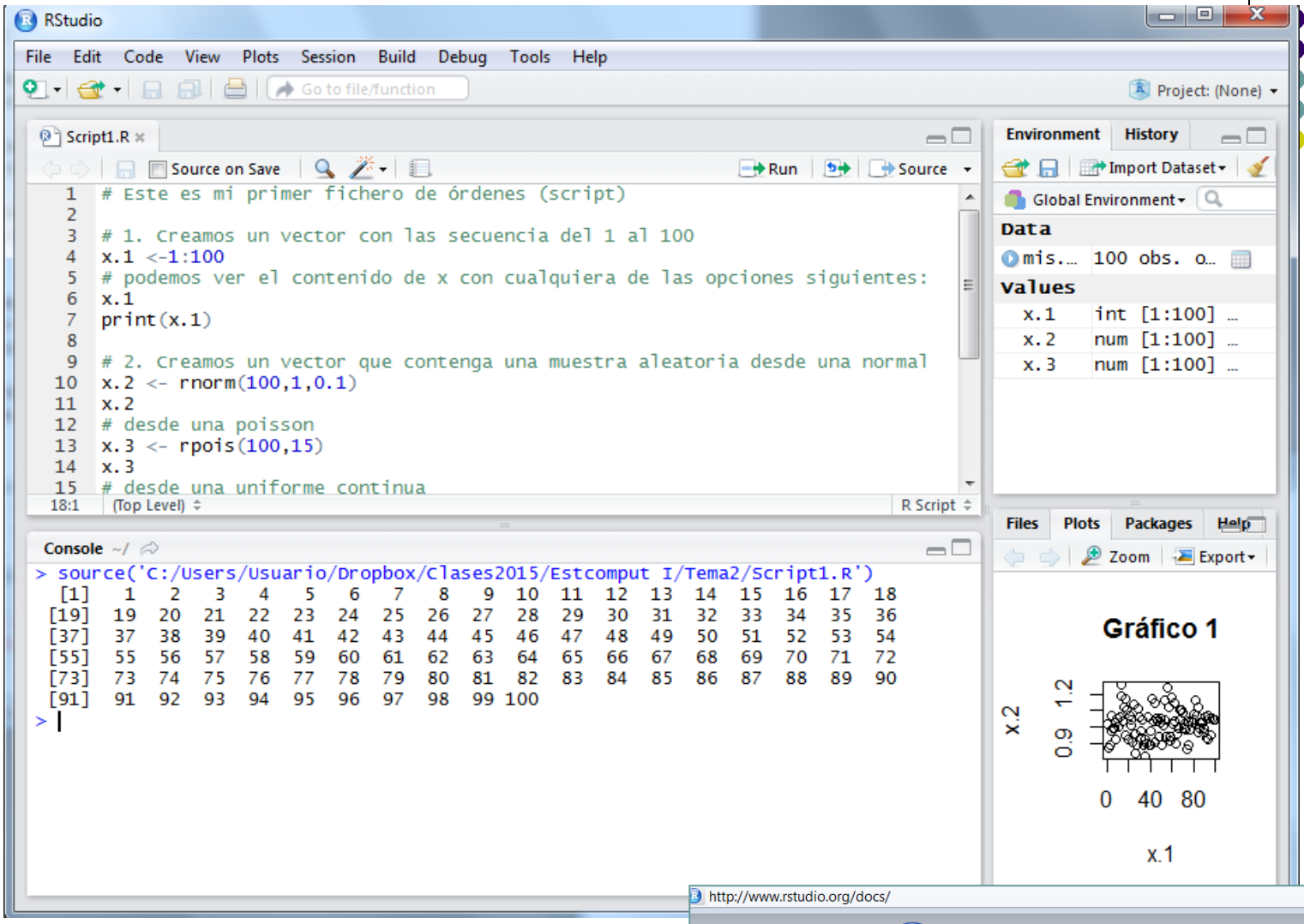


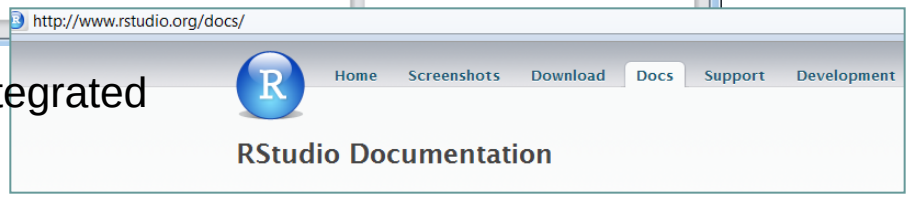
Tarea:

1) **Instalar** y **cargar** **Rcmdr**
2) Observar que se muestra un nuevo interfaz con opciones para el tratamiento de los datos y su análisis estadístico



El IDE RStudio



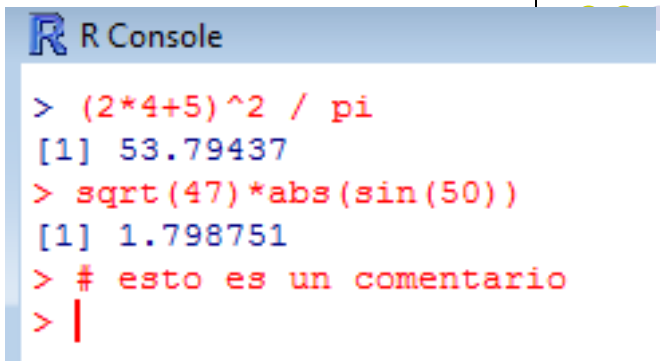


- RStudio es entorno de desarrollo integrado (integrated development environment, IDE).
- Instalar RStudio y observar en entorno que proporciona y las diferencias respecto a RGui.

Trabajando directamente en R-console

1. **Usar R como una calculadora**:

```
(2*4+5)^2 / pi
sqrt(47)*abs(sin(50))
# esto es un comentario
```



2. **Ejecutar órdenes**:
   - Solicitar ayuda

```
help.start()
help(base)  # o bien ?base
help.search('mean')
demo(graphics)
```

   - Cargar un package

```
library(boot)
```

   - Obtener la definición de una función ya creada

```
library
```

   - Cerrar el programa

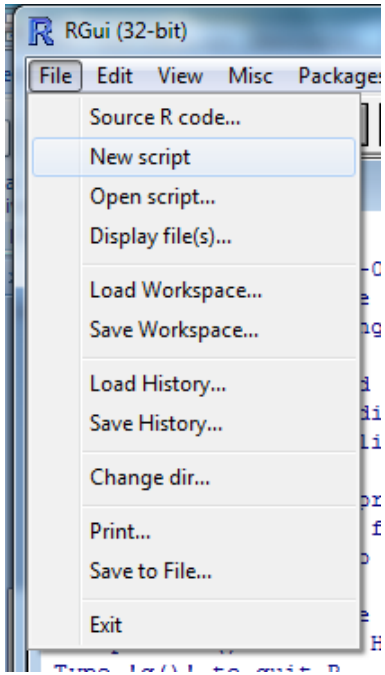
```
q()
```

   - Algunas de estas opciones se pueden obtener desde el menú

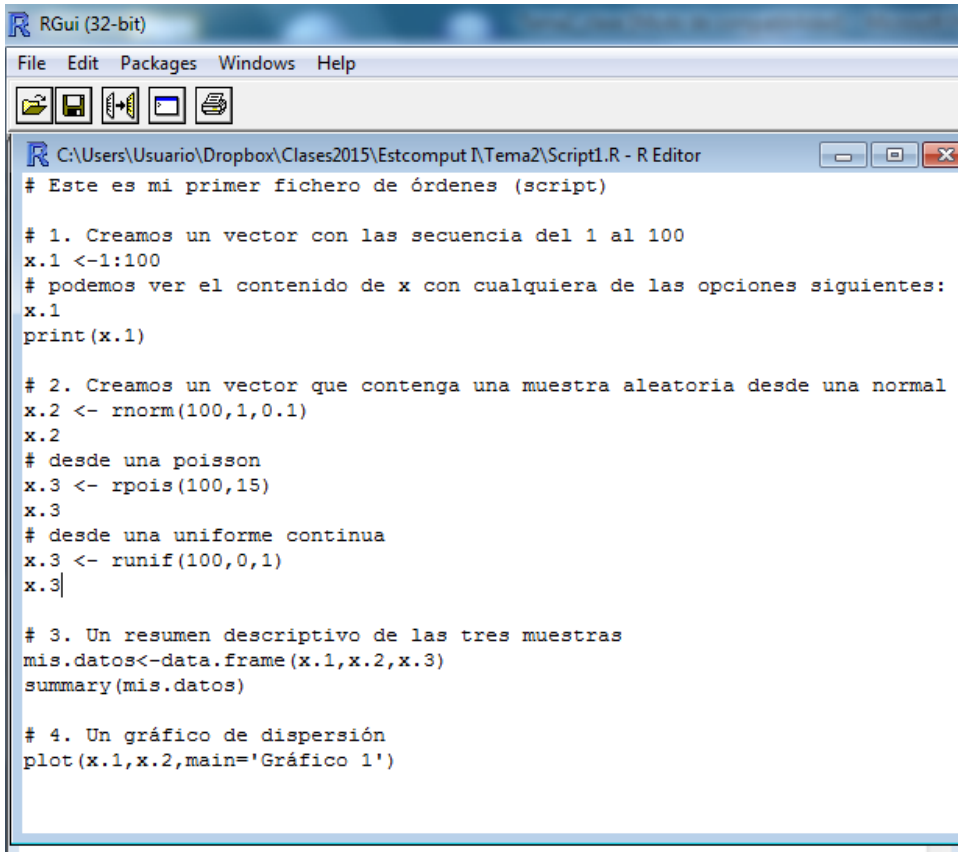
Trabajando con ficheros de órdenes (scripts)

- **Escribir directamente en R-console tiene varios inconvenientes.**
- **Scripts**: son ficheros de texto que continen secuencias de órdenes escritas utilizando el lenguaje de R.
- Los scripts tienen extensión R. Se pueden editar en la ventana que facilita Rgui y ejecutar el código usando el botón derecho del ratón o desde el menú.

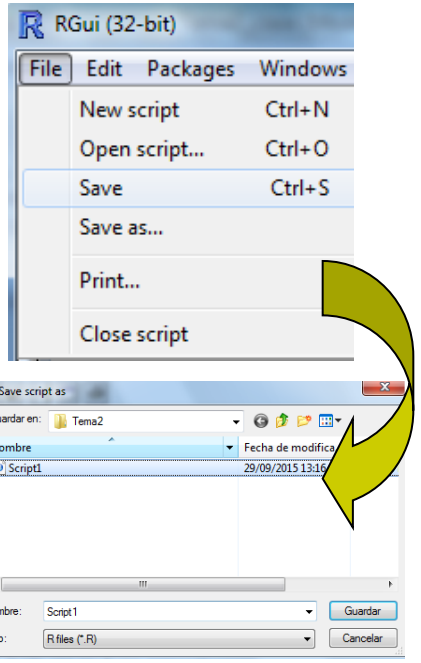
1. Crear el script



2. Escribir la secuencia de órdenes



3. Guardar el fichero

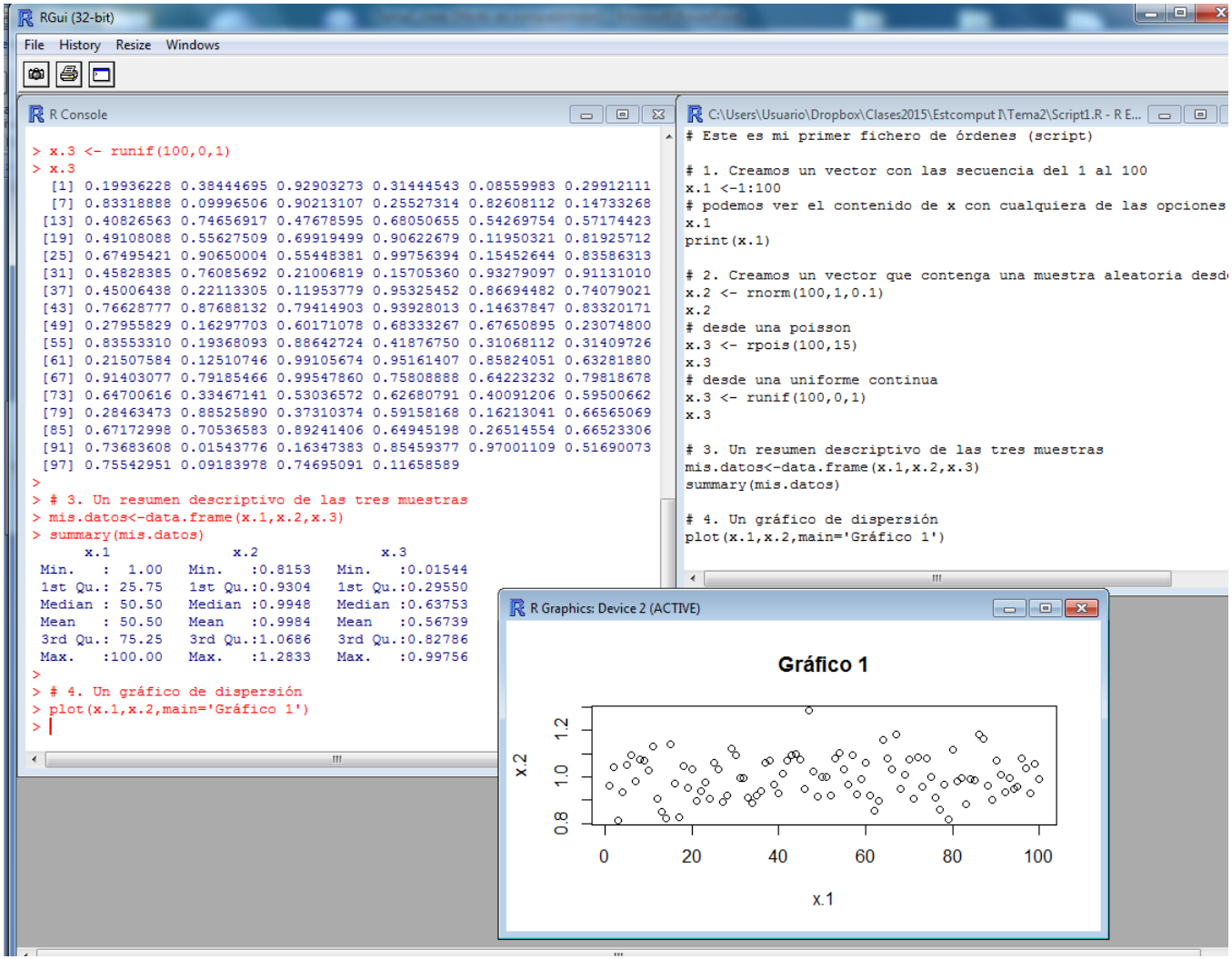


Reglas de sintaxis básicas

- ✓ Técnicamente el lenguaje R posee una sintaxis muy sencilla. Ojo que distingue entre mayúsculas y minúsculas.
- ✓ Las líneas de código elementales consisten en evaluación de expresiones o asignaciones.
- ✓ Si se escribe una expresión de evaluación pero no se asigna a un objeto, el resultado se imprime en la R-console pero no se guarda.
- ✓ El operador de asignación "**<-**" permite evaluar una expresión y a la vez guardarla en un objeto de R. Para ver el resultado es necesario imprimir/visualizar dicho objeto.
- ✓ Dos sentencias de órdenes se pueden escribir en la misma línea usando como separador punto y coma (";").
- ✓ Varias sentencias se pueden agrupar construyendo un bloque de sentencias usando como delimitadores corchetes del tipo "{" y "}".
- ✓ Se pueden escribir comentarios comenzando con el símbolo '**#**'.
- ✓ Si una sentencia de código es muy larga se puede dividir entre varias líneas. En R-console el símbolo "**+**" indica que la sentencia no se ejecutará hasta que se complete.

Trabajando con scripts: un ejemplo

Ejemplo: Crear desde el menú Archivo un script como el de la diapositiva anterior y ejecutarlo.



- Repetir este ejemplo usando RStudio y observar las diferencias.

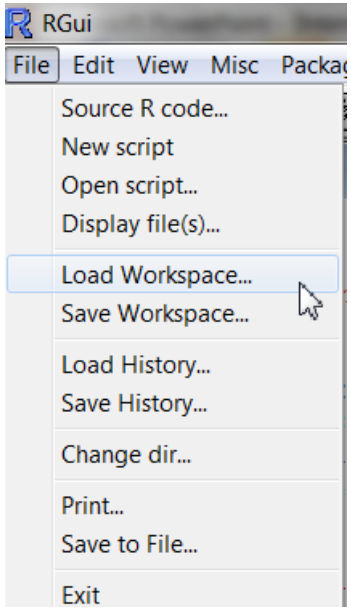
Guardar los resultados

- Todo lo que se muestra en R-console es texto que se puede **copiar y pegar** en otra aplicación de Windows.
- Además se pueden usar funciones especiales en R para escribir el contenido de los objetos en ficheros de tipo texto (write, write.table,…).
- Algunos entornos de desarrollo integrado como RStudio (incluso R-commander) permite hacer esto desde el menú.
- Para guardar gráficos generados en una sesión con R se puede hacer uso de las opciones del menú accesible pulsando con el botón derecho del ratón sobre la venta que muestra el gráfico.
- El posible guardar el área de trabajo (objetos creados durante la sesión) así como el historial de sentencias con las opciones del menú: “Save Workspace”, “Save History”.

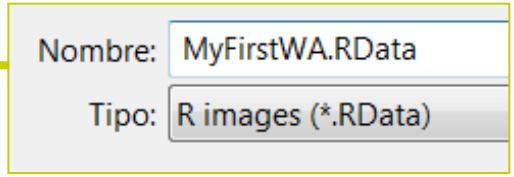
Listar y borrar objetos:

- **ls()**, **objects()** para listar los objetos en memoria (Workspace)
- **rm(**objeto**)** borra el objeto indicado o **rm()** borra todos los objetos

Guardar/cargar el Workspace (objetos generados en una sesión de trabajo)

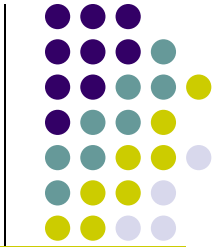


Ejemplo: guardar el Workspace y finalizar la sesión con R. Después volver a ejecutar R y cargar el Workspace que se guardó. Comprobar que están todos los objetos.



Repetir usando RStudio y observar las diferencias

Objetos de datos



Tipo de datos **logical, integer, single, double, numeric, complex, character**

vector objetos básicos. Todos sus elementos deben ser del mismo tipo (**mode**)

```
x <- c(1, 2, 3); y <- c("a", "b", "c"); z <- c(TRUE, TRUE, FALSE)
x[1] # el primer elemento del vector x
```

array, matrix colección de datos bidimensional, tridimensional, etc. (todos del mismo tipo)

```
A <- matrix(rnorm(20), nrow=10, ncol = 2)
A[1,1]  # el elemento en primera fila y primera columna
A[1,]  # la primera fila de la matriz A (es objeto de tipo vector)
A[1:3,1]  # los tres primeros elementos de la primera columna
B<-A[,-1]  # B es una copia de A eliminado la primera columna
```

factor vector especial habitualmente utilizado para datos categóricos, los datos se representan mediante códigos numéricos y etiquetas

```
ff <- factor(c(1, 2, 2, 1, 1),levels=c(1,2),labels=c("M", "H"))
```

Más detalles: utilizar la ayuda y observar la sintaxis y uso

data.frame Es similar a una matriz pero puede almacenar datos de distinto tipo.

Típica hoja de datos en estadística

```
Mi.dataframe <- data.frame(ID=c("C1", "C2", "C3"), var1 = c(10, 25, 33), var2 = c(10, 34, 15), var.logic = c(TRUE, TRUE, FALSE))
Mi.dataframe$ID # devuelve las columnas del data.frame
Mi.dataframe[1,1] # elemento en posición (1,1)
```

list Lista de objetos de datos con (posiblemente) distinta estructura y tipo de datos

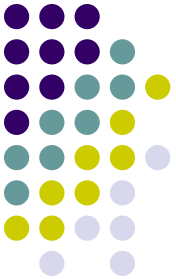
```
Lista.1 <- list( x = 1:10, y = "Hello",
A <- matrix(rnorm(20), ncol = 5),
Lista.2 = list(a = 5, b = factor(c("a", "b"))) )
```

ts Serie de tiempo

```
ts(1:47, frequency = 12, start = c(2000, 2))
ts(1:10, frequency = 4, start = c(2000, 2))
```

Más detalles: utilizar la ayuda y observar la sintaxis y uso

Valores perdidos o no disponibles



- Los datos perdidos se representan por **NA** (Not Available)

  Triang <-matrix(c(1,2,3,4,5,NA,6,NA,NA),ncol=3,byrow=T)

- R representa los valores numéricos no-finitos mediante **Inf** y los valores numéricos resultantes de operaciones no válidas mediante **NaN** (Not A Number)

  x<-1/0

  exp(-x)

  x-x

Atributos de los objetos de datos

- Todos los objetos tienen dos atributos intrínsecos: **mode** y **length**
  - mode se refiere al tipo de datos en el objeto (numeric, character, complex, logical)
  - length al número de elementos (longitud) del objeto

    x <- seq(from=1 , to=10 , by=2)

    mode(x) ; length(x)

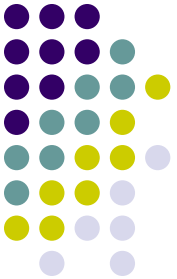
    A <- “a word"; B <- TRUE; z <- 1i

    mode(A) ; mode(B) ; mode(z)
- **attributes**(objeto) devuelve una lista con los atributos del objeto

  A<-matrix(1:6,2) ; attributes(A)

  ll<- list(x=x,A=A) ; attributes(ll)
- Algunos atributos se pueden obtener usando funciones específicas

  dim(A) ; names(ll)
- Para nombrar objetos se pueden combinar letras, números y símbolos como “.”. Un nombre nunca debe comenzar por un número. Recordar que R es sensible al uso de mayúsculas o minúsculas.



Función de concatenación: c

```
x<-c(1,3,NA,8,0)
y<-c(x,1/x)
```

Secuencias regulares:

```
x<-1:10 ; y<-10:1
1:10-1 ; 1:(10-1)
```

seq permite especificar el incremento o la longitud

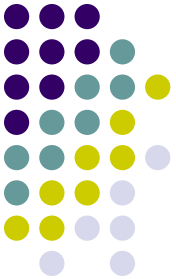
```
seq(from=1,to=10,by=3)
seq(1,10,length.out=10)
```

Valores repetidos

```
x<-rep(1,10) ; rep(x,2)
```

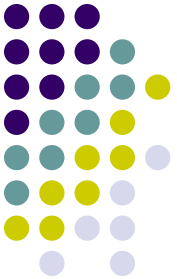
Creación de una rejilla de índices: expand.grid

```
expand.grid(1:10,1:5)
```



Generación de valores de variables aleatorias

law	function
Gaussian (normal)	`rnorm(n, mean=0, sd=1)`
exponential	`rexp(n, rate=1)`
gamma	`rgamma(n, shape, scale=1)`
Poisson	`rpois(n, lambda)`
Weibull	`rweibull(n, shape, scale=1)`
Cauchy	`rcauchy(n, location=0, scale=1)`
beta	`rbeta(n, shape1, shape2)`
'Student' (t)	`rt(n, df)`
Fisher–Snedecor (F)	`rf(n, df1, df2)`
Pearson (χ^2)	`rchisq(n, df)`
binomial	`rbinom(n, size, prob)`
multinomial	`rmultinom(n, size, prob)`
geometric	`rgeom(n, prob)`
hypergeometric	`rhyper(nn, m, n, k)`
logistic	`rlogis(n, location=0, scale=1)`
lognormal	`rlnorm(n, meanlog=0, sdlog=1)`
negative binomial	`rnbinom(n, size, prob)`
uniform	`runif(n, min=0, max=1)`
Wilcoxon's statistics	`rwilcox(nn, m, n)`, `rsignrank(nn, n)`



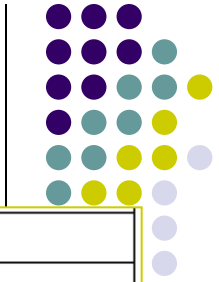
Operadores

Operators					
Arithmetic		Comparison		Logical	
`+`	addition	`<`	lesser than	`! x`	logical NOT
`-`	subtraction	`>`	greater than	`x & y`	logical AND
`*`	multiplication	`<=`	lesser than or equal to	`x && y`	id.
`/`	division	`>=`	greater than or equal to	`x | y`	logical OR
`^`	power	`==`	equal	`x || y`	id.
`%%`	modulo	`!=`	different	`xor(x, y)`	exclusive OR
`%/%`	integer division				

Funciones básicas

`sum(x)`	sum of the elements of `x`
`prod(x)`	product of the elements of `x`
`max(x)`	maximum of the elements of `x`
`min(x)`	minimum of the elements of `x`
`which.max(x)`	returns the index of the greatest element of `x`
`which.min(x)`	returns the index of the smallest element of `x`
`range(x)`	id. than `c(min(x), max(x))`
`length(x)`	number of elements in `x`
`mean(x)`	mean of the elements of `x`
`median(x)`	median of the elements of `x`

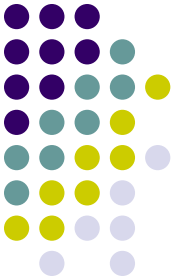
Otras funciones útiles



`cumsum(x)`	a vector which *i*th element is the sum from `x[1]` to `x[i]`
`cumprod(x)`	id. for the product
`cummin(x)`	id. for the minimum
`cummax(x)`	id. for the maximum
`match(x, y)`	returns a vector of the same length than `x` with the elements of `x` which are in `y` (`NA` otherwise)
`which(x == a)`	returns a vector of the indices of `x` if the comparison operation is true (`TRUE`), in this example the values of `i` for which `x[i] == a` (the argument of this function must be a variable of mode logical)

Ejemplos:

```
x<-1:10
cumsum(x) ; cumprod(x)
which.max(x)
ind<-which(x<=3) ; ind
x[ind]
match(x,1:15)
range(x)
```



Función *cut*: Divide el rango de un vector x en intervalos y codifica los valores de x de acuerdo con el intervalo en el que caen.

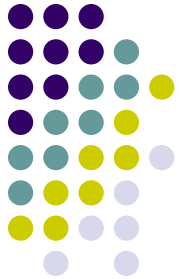
cut(x, breaks, labels)

- ***x*:** vector de datos
- ***breaks*:** vector de puntos de corte para determinar los intervalos
- ***labels*:** etiquetas de los niveles

Ejemplos:

```
x<-rnorm(100,mean=10,sd=2)
x.d1<-cut(x,breaks=seq(3,16,length.out=5))   # 4 intervalos
table(x.d1)
x.d2<-cut(x,breaks=seq(3,16,length.out=10)) # 9 intervalos
table(x.d2)
```

Comprobación y cambio de tipo de datos u objeto



Existen funciones que permiten comprobar si un objeto es de un tipo determinado (is.vector, is.list, is.factor, is.data.frame etc.) o cambiarlo a un tipo concreto (as.vector, as.matrix etc.)

Ejemplos:

```
x<-rnorm(100,mean=10,sd=2)
is.vector(x)
is.numeric(x)
is.matrix(x)
x.m<-as.matrix(x)
is.matrix(x.m)
x.d1<-cut(x,breaks=seq(3,16,length.out=5))
is.numeric(x.d1)
is.factor(x.d1)
as.numeric(x.d1)
```

Crear un script (ejercicios1.R) que contenga las sentencias de órdenes que permitan resolver las siguientes cuestiones:

1. Crear un vector **v.1** que contenga los números impares entre 10 and 50. Calcular la suma de **v.1**.
2. Mostrar los cuatro primeros elementos del vector **v.1**. Calcular el producto de dichos elementos. Repetir ahora con los elementos en posición 4, 9 y 11 del mismo vector **v.1**.
3. Crear un vector **v.2** con 100 números aleatorios de una uniforme continua en el intervalo [-3,3]. Calcular la media y la varianza de dicho vector. Crear un vector **s.v.2** que contenga los valores tipificados de **v.2**.
4. Crear un vector **c.v.2** con las sumas acumuladas de **v.2**
5. Calcular el número de elementos positivos **v.2** y almacenarlos en un nuevo vector **v.3**
6. Reemplazar los valores negativos en **v.2** con zeros.
7. Crear un vector **v.4** con 100 números aleatorios de una normal con media 1 y varianza 0.25. Representa la densidad de estos valores con "**plot(density(v.4))**".
8. Crear un vector **v.5** con 100 números aleatorios de una gamma con media 1500 y varianza 100. Representa la densidad de estos valores.

Guardar el script, ejecutarlo y observar los nuevos objetos creados en el área de trabajo

Operaciones con matrices y arrays

R dispone de funciones my útiles para trabajar con matrices y arrays, además de los operadores básicos:

- **rbind** y **cbind** permite pegar matrices (o añadir vectores) por filas o columnas, respectivamente.

```
m1 <- matrix(1:4, 2, 2)
m2 <- matrix(2, 2, 2)
m3<-rbind(m1, m2) ; dim(m3)
m4<-cbind(m1, m2) ; dim(m4)
```

- **%*%** corresponde al producto matricial

```
m1 %*% m2
m1*m2  # sería el producto elemento a elemento
```

- **diag** crea una matriz diagonal o devuelve la diagonal de una matriz ya creada.

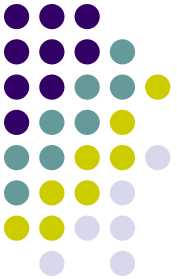
```
m5 <- diag(1:10,10)
diag(m1)
```

- **rowSums, colSums** devuelve las sumas por filas o columnas, respectivamente.

```
rowSums(m1) ; colSums(m1)
```

- **rowMeans, colMeans** devuelve las medias aritméticas por filas o columnas, respectivamente.

```
rowMeans(m1) ; colMeans(m1)
```



El cálculo vectorial es el modo más eficiente de trabajar en R.

Las funciones que trabajan con el objeto completo son siempre preferibles al uso de estructuras de tipo bucle que trabajan elemento a elemento sobre un vector o matriz.

Antes se han definido algunas de estas funciones (rowSums, colSums etc.). El siguiente ejercicio servirá de ilustración sobre este tipo de cálculos.

Ejercicio: Consideremos la matriz triangular superior siguiente:

225	417	536	594	610
240	432	539	589	
255	469	599		
195	349			
220				

$$\{C_{ij}, 1 \leq i+j-1 \leq n\}$$

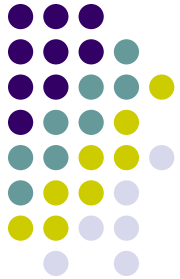
Denotemos a sus elementos por .

Aqui el índice i denota las filas, j las columnas y la dimensión es n=5.

1. Crear una matriz de dimensión 5 con los datos anteriores.
2. Crear el vector de dimensión n-1 que contenga los factores F_j (j=2,3,4,5) definidos como:

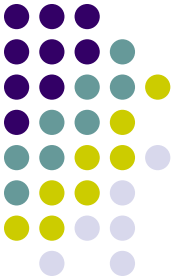
$$F_j = \frac{\sum_{i=1}^{n+1-j} C_{ij}}{\sum_{i=1}^{n+1-j} C_{i,j-1}}$$

Vectorización (solución al ejercicio)



```
C<-matrix(NA,5,5)
C[1,]<-c(225,417,536,594,610)
C[2,1:4]<-c(240,432,539,589)
C[3,1:3]<-c(255,469,599)
C[4,1:2]<-c(195,349)
C[5,1]<-c(220)
F.j <- numeric(4)
F.j[1]<- sum(C[1:4,2]) / sum(C[1:4,1])
F.j[2]<- sum(C[1:3,3]) / sum(C[1:3,2])
F.j[3]<- sum(C[1:2,4]) / sum(C[1:2,3])
F.j[4]<- C[1,5] / C[1,4]
```

Data frames



El objeto de tipo data.frame es por lo general el modo más adecuado de almacenar datos para posteriores análisis estadísticos.

- Los vectores (columnas) en un data.frame deben tener la misma longitud.
- Algunas funciones para leer datos desde ficheros de texto como **read.table** devuelve este tipo de objeto.

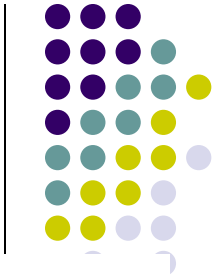
data.frame(data, row.names=NULL, checkrows = FALSE, checknames = TRUE)

- *row.names*: permite especificar un vector de caracteres con los nombres de las filas
- *check.rows*: si TRUE, se comprueba la longitud y los nombres
- *check.names*: si TRUE, se comprueba que los nombres de las variables son correctos

Ejemplos:

```
Data <- data.frame(1:10,rnorm(10),paste("id_",1:10))
names(Data) <- c("Row", "x", "ID")
x<-Data$x  # o también x<-Data[,2]
Data.2<-data.frame(Data , z=2*x+12)
```

Leer datos desde un fichero de texto: read.table



El modo más común de leer datos de tipo texto es mediante la función read.table cuyo uso es como sigue:

read.table (file, header = FALSE, sep = "", quote = "\"" , dec = ".", row.names, col.names, as.is = FALSE, na.strings = "NA", colClasses = "NA", nrows = -1, skip =0, check.names = TRUE, fill =!blank.lines.skip, strip.white = FALSE, blank.lines.skip = TRUE, comment.char = "#")

- **file** nombre del fichero (si el fichero no está en el directorio de trabajo entonces hay que escribir el camino completo)
- **header** si TRUE leerá los nombres de las variables de la primera fila
- **sep** el separador utilizado entre datos dentro de las filas (**sep="\t"** si están separados por un tabulador)
- **row.names** permite especificar un vector con los nombres de las filas
- **col.names** permite especificar un vector con los nombres de las columnas
- **nrows** máximo de filas que se deben leer
- **skip** permite especificar cuántas filas se debe saltar al leer datos desde el principio

Escribir help(read.table) para más detalles.

Ejemplo:

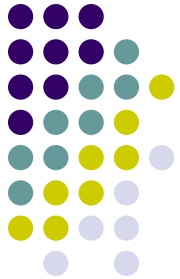
Crear un fichero de texto que contenga la matriz triangular del ejercicio anterior. Para ello utilizar por ejemplo el bloc de notas y denominar el fichero “Triangulo.txt” (usar el tabulador como separación entre los datos de cada fila).

Dicho fichero se puede leer, almacenado su contenido en un data.frame con las siguientes sentencias:

```
Mi.directorio<-”C:/clase”  # especificar el directorio donde se ha creado
                           # el fichero con la matriz

setwd(Mi.directorio)  # define el directorio de trabajo

A<-read.table(file="Triangulo.txt",header=F,sep="\t")

is.data.frame(A)    # comprobamos que la función devuelve
                    # un objeto de tipo data.frame

edit(A)  # mostramos A en la ventana de edición de datos de R

A<-edit(A)  # permitiría editar A dentro de la ventana de edición
```

3	225	417	536	594	610
4	240	432	539	589	
5	255	469	599		
6	195	349			
7	220				

Leer datos de un fichero de texto: scan



Para leer datos y almacenarlos en un vector se puede usar la función scan. Su uso es como sigue:

```
scan (file= "", what = double(0), nmax = -1, n = -1, sep = "", quote = if (sep=="\n") "" else "\"", dec = ".", skip = 0, nlines = 0, na.strings = "NA", flush = FALSE, fill = FALSE, strip.white = FALSE, quiet = FALSE, blank.lines.skip = TRUE, multi.line = TRUE, comment.char = "", allowEscapes = TRUE)
```

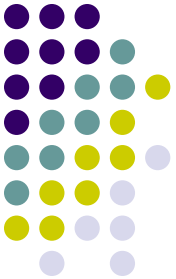
- **file** nombre del fichero de datos. Si no se especifica entonces los datos se leen desde R-console

Escribir help(scan) para más detalles

Ejemplo:

```
Vector.0<-scan(file="Triangulo.txt")
Vector.0    # se ignoran los NA's (devuelve sólo 15 valores)
Vector.1<-scan(file="Triangulo.txt",sep="\t")
Vector.1   # ahora lee 25 valores (10 son NA's)
matrix(Vector.1 ,5 ,byrow=T)   # para escribir el vector en forma
                               # de matriz de dimensión 5
```

Leer datos en otros formatos



- R también puede leer datos desde otros formatos (por ejemplo Excel, SAS, SPSS), así como bases de datos. Para ello se pueden usar funciones desde packages como: foreign, Rcmdr etc.
- Una forma sencilla de leer datos desde Excel es copiar y pegar a un fichero de texto y usar **read.table** o **scan** tal y como se describió antes, o bien desde el portapapeles mediante:

```
read.table(file= "clipboard", sep="\t")
```

- Otra opción sencilla consiste en utilizar interfaces como Rcommander que facilitan este tipo de tareas.

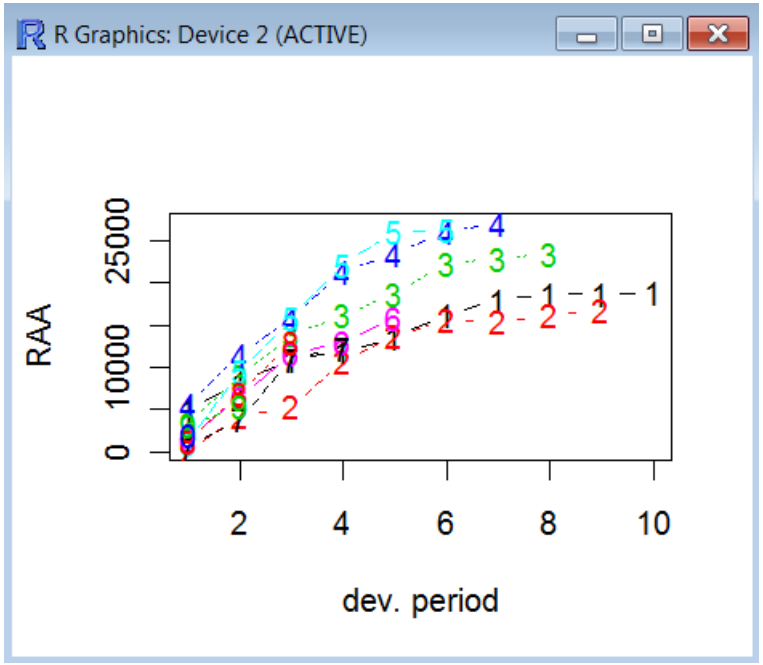
Cargar datos desde packages de R

En los libros de R habitualmente hay disponible un gran número de conjuntos de datos que sus creadores han puesto a disposición de los usuarios para ilustrar los procedimientos implementados en el libro.

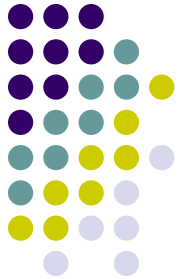
Para cargar uno de estos conjuntos de datos primero es necesario tener instalado y cargado el libro y entonces se puede usar la función **data()**

Ejemplo:

```
library(ChainLadder)
data(RAA)
RAA
class(RAA)  # se trata de un objeto especial
plot(RAA)    # el método plot permite
             # representar los datos
```



Escribir datos: write.table, write



write.table() permite escribir datos de tipo matriz o vector, que estén en el espacio de trabajo durante una sesión con R, en un fichero de texto. Para un objeto de datos dado (lo escribimos como objeto.datos), su uso sería como sigue:

```
write.table(objeto.datos, file = “”, append = FALSE, quote = TRUE, sep = " ",
  eol = "\n", na = "NA", dec = ".", row.names = TRUE, col.names = TRUE,
  qmethod = c("escape", "double"))
```

write () es similar aunque permite especificar el número de columnas a escribir. Consultar la ayuda para más detalles así como su uso.

Ejemplos:

```
write.table(Triangulo,file=“PT.txt”)
# abre el fichero que has escrito y comprueba su contenido.
# escribe las sentencias siguientes y comprueba una a una cómo se van
# escribiendo (añadiendo) los datos en el fichero PT.txt
write.table(Triangulo,file=“PT.txt”,append=T, row.names=F)
write(as.matrix(Triangulo),file=“PT.txt”,append=T)
write(t(as.matrix(Triangulo)),file=“PT.txt”,append=T)
```

Crear un script (ejercicios2.R) con las sentencias que permitan realizar en R las siguientes tareas:

1. Crear una matriz cuadrada de dimensión tres que tenga los elementos (por filas) 1,1/2,...,1/9. Denominar dicha matriz como A.
2. Calcular la traspuesta de A y multiplicar A por su traspuesta.
3. Crear una matriz B como A pero eliminando la primera columna.
4. Crear una matriz C como A pero acumulando los valores de A por filas.
5. Crear una matriz cuadrada D de dimensión cuatro. El cuadrante superior izquierdo de dimensión tres debe ser la matriz A, y la última fila y la columna deben ser vectores de ceros.
6. Crear una matriz E de dimensión cuatro por cinco. Debe contener D en las cuatro primeras columnas y las medias por filas de D en última columna
7. Convertir la matriz A en un data.frame (con nombre DA1) añadiendo nombres a las columnas (por ejemplo “A1”, “A2” y “A3”). Escribir dicho data.frame en un fichero de texto. Una vez hecho esto, leer el fichero y cargar su contenido en un nuevo data.frame (con nombre DA2) usando una función adecuada. Después visualizar DA2 y ver que es igual que DA1.
8. Cargar el objeto de datos volcano (del package datasets), mostrarlo y calcular un resumen descriptivo usando la función summary().
9. Cargar el objeto de datos Nile (package datasets), mostrarlo y representarlo usando la función plot().