

Análisis de Datos con el Sistema Estadístico R

Lic. Patricia Vásquez Sotero



Sesión 2

Definiciones básicas Archivos de datos

CONTENIDO

- ☐ Definiciones básicas
 - Objetos para el manejo de datos
 - Características de los objetos en **R**: modos y atributos Datos especiales
 - Asignación, Operadores lógicos y Coerción de tipos
 - Paquetes (bibliotecas o librerías) de **R**
- ☐ Archivos de datos
- ☐ Interfaz gráfica de R en perspectiva - RCommander

ESTABLECER EL DIRECTORIO DE TRABAJO

- Como hay nombres que tendemos a repetir, es recomendable utilizar un directorio de trabajo diferente para cada problema que analicemos con R. Es posible cambiar de directorio desde el menú Archivo o con el comando `setwd("C:/midir")`
- Si no sabemos en qué directorio estamos podemos averiguarlo con el comando `getwd()`
- A veces nos interesa ejecutar varias órdenes que tengamos almacenadas en un archivo, p.e. `ordenes.R`. Si el archivo está en el directorio de trabajo, es posible ejecutarlas dentro de una sesión de R con la orden `source("ordenes.R")`. Esta función también está disponible en el menú Archivo.

COMANDOS, FUNCIONES Y OBJETOS

DEFINICIONES BÁSICAS

☐ Objetos

○ Funciones

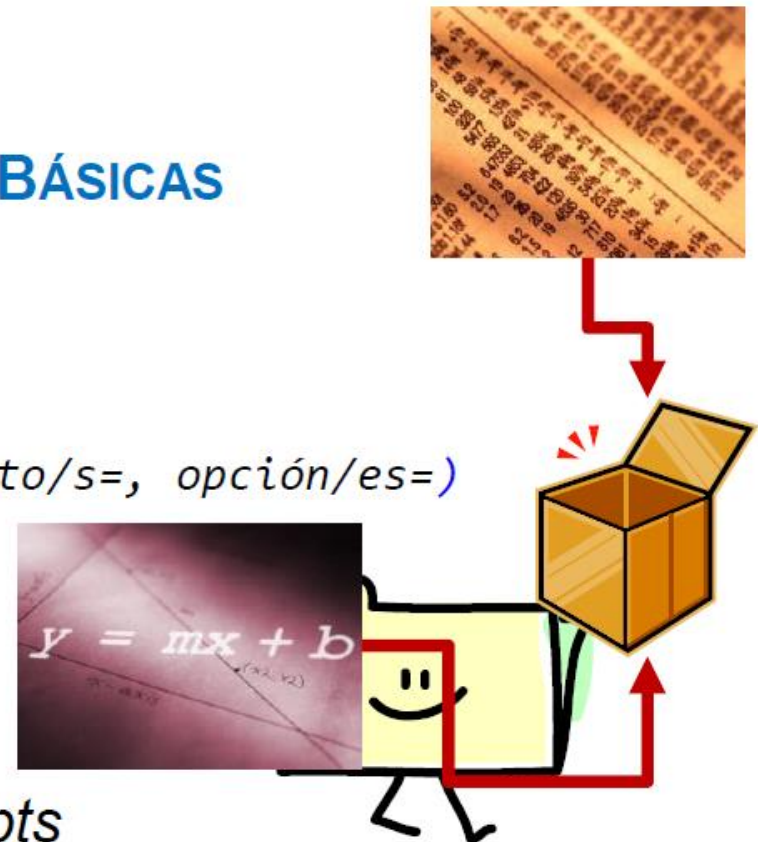
▪ *nombre.de.la.función(argumento/s=, opción/es=)*

☐ Espacio o área de trabajo

☐ Directorio de trabajo

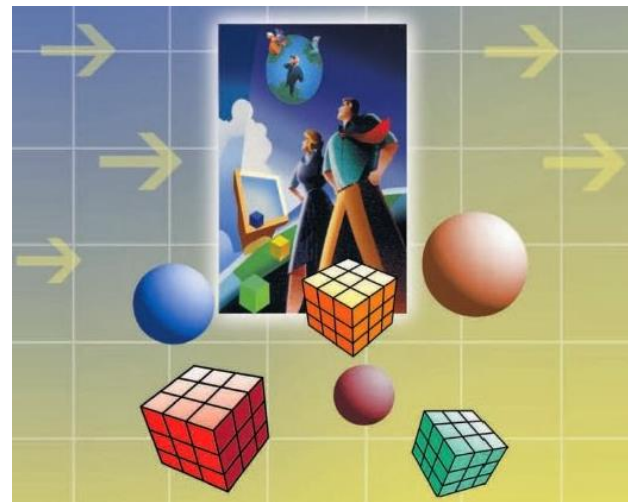
☐ Paquetes

☐ Archivos de comandos o *scripts*



COMANDOS, FUNCIONES Y OBJETOS

- **R es un lenguaje Orientado a Objetos.** Bajo este término se esconde la simplicidad y flexibilidad de R.
- Mientras que programas más clásicos muestran directamente los resultados de un análisis, **R almacena los resultados en objetos**, para ser observados o analizados posteriormente, produciendo unas salidas mínimas.
- Esto puede ser un poco extraño para el usuario, pero esta característica suele ser muy útil. De hecho, el usuario puede extraer sólo aquella parte de los resultados que le interesa.



PERMANENCIA Y ELIMINACIÓN DE OBJETOS

- La colección de objetos almacenados en cada momento se denomina **espacio de trabajo (workspace)**.
- Los objetos creados durante una sesión de R pueden almacenarse en un archivo para su uso posterior (archivos **.RData**).
- Es posible eliminar objetos con el comando **rm()**
- Desde el menú **Misc>remove todos los objetos...** se pueden eliminar todos los objetos a la vez.

CARACTERÍSTICAS DE LOS OBJETOS

Los objetos están compuestos de elementos. Los elementos más simples, las **variables**, pueden ser:

Tipos o modos de datos

- `numeric`: número real con doble precisión. Los podemos escribir como enteros (3, -2), con fracción decimal (3.27) o con notación científica (3.12e-47).
- `complex`: números complejos de la forma $a+bi$.
- `character`: Cadenas alfanuméricas de texto.
- `logical`: variables lógicas. Puede ser TRUE o FALSE.

Ejemplo.

```
nombre<-"Luis"  
varon<-TRUE  
edad<-23  
estatura<-1.77
```


CARACTERÍSTICAS DE LOS OBJETOS

Valores especiales: NA

En algunos casos las componentes de un objeto pueden no ser completamente conocidas. Por ejemplo, las coordenadas de los vectores pueden tomar alguno de los siguientes valores especiales:

NA, NaN, Inf

- **NA**: Significa “**Not Available**”. Es útil cuando nos falta algún dato.
- En general una operación con elementos NA resulta NA, a no ser que mediante una opción de la función, podamos omitir o tratar los datos faltantes de forma especial.
- La opción por defecto en cualquier función es **na.rm=FALSE** (que indica que NO elimina los NA), que da como resultado NA cuando existe al menos un dato faltante.
- Con la opción **na.rm=TRUE**, la operación se efectúa con los datos válidos.

CARACTERÍSTICAS DE LOS OBJETOS

Valores especiales: NA

Ejemplo

```
x<-NA
```

Asignar NA a la variable x

```
x+1
```

Observar que el resultado es también un NA.

```
y<-c(x,3,5,x)
```

Asignamos al vector y dos NA y dos números.

```
mean(y)
```

Calcula la media teniendo en cuenta los NA's.

```
y[3]
```

Calcula la media SIN tener en cuenta los NA's.

CARACTERÍSTICAS DE LOS OBJETOS

Valores especiales: Inf y NaN's

En la mayoría de los casos, no debemos preocuparnos si los elementos de un objeto numérico son enteros, reales o incluso complejos. Los cálculos se realizarán internamente como números de doble precisión, reales o complejos según el caso.

- **NaN:** Significa “**Not a Number**”. Aparece como resultado de operaciones cuyo resultado es ambiguo o inexistente en el conjunto de números reales.
- **Inf:** Significa “**Infinity**” (infinito).
- Para trabajar con números complejos, deberemos indicar explícitamente la parte compleja. En determinadas ocasiones los cálculos realizados pueden llevar a respuestas con valor infinito positivo (representado por **R** como Inf) o infinito negativo (-Inf).
- Es posible realizar y evaluar cálculos que involucren Inf.
- Sin embargo, a veces, determinados cálculos llevan a expresiones que no son números (representados por **R** como NaN's).

Las funciones **is.na(x)** y **is.nan(x)** detectan qué coordenadas de un vector **x** son NA o NaN.

CARACTERÍSTICAS DE LOS OBJETOS

Valores especiales: Inf y NaN's

Ejemplo

`sqrt(-17)`

Produce un “Warning message” avisando que es un NaN.

`sqrt(-17+0i)`

Calcula el resultado y devuelve un número complejo.

`x<-5/0`

Asignamos a x un valor infinito.

`exp(-x)`

Devuelve el valor 0.

`exp(x)-exp(x)`

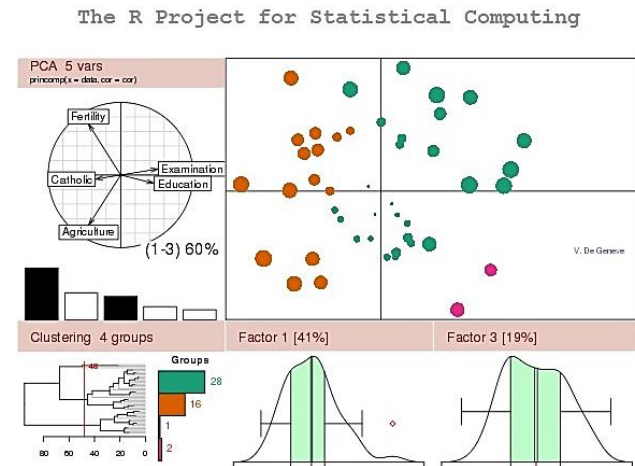
Devuelve NaN: es una indeterminación.

CLASIFICACIÓN DE LOS OBJETOS

- Los objetos se pueden clasificar en dos grandes grupos:
 - **atómicos**: todos los elementos que los componen son del mismo tipo (o modo), como por ejemplo los vectores, matrices, series temporales.
 - **recursivos**: pueden combinar una colección de otros objetos de diferente tipo (o modo), como son los data.frame, listas.
- Existen otras estructuras recursivas, p.e.:
 - El **modo function** está formado por las funciones que constituyen **R**, unidas a las funciones escritas por cada usuario. Las trataremos en una sesión posterior.
 - El **modo expression** corresponde a una parte avanzada de **R** que trataremos más adelante al presentar las **fórmulas** en la sesión de modelos estadísticos.
- Recordar además que es posible crear nuevas estructuras.
- Utilizando la **función str(objeto)** podemos obtener información sobre su estructura.

TIPOS DE OBJETOS

- **Los vectores** son el tipo básico de objeto en R, pero existen más tipos.
- **Las matrices** o, más generalmente, variables indexadas (Arrays) son generalizaciones multidimensionales de los vectores. De hecho, son vectores indexados por dos o más índices.
- **Los factores** sirven para representar datos categóricos.
- **Las listas** son una forma generalizada de vector en las cuales los elementos no tienen por qué ser del mismo tipo y a menudo son a su vez vectores o listas. Las listas permiten devolver los resultados de los cálculos estadísticos de un modo conveniente.
- **Las hojas de datos** (data frames) son estructuras similares a una matriz, en que cada columna puede ser de un tipo distinto a las otras.
- **Las funciones** son también objetos de R que pueden almacenarse en el espacio de trabajo, lo que permite extender las capacidades de R fácilmente.



TIPOS DE OBJETOS

i) Vectores

- Un vector en R puede contener una colección de números o de caracteres no numéricos. Para definir un vector, por ejemplo, el vector $x = (1, 3, 5)$, usaríamos la orden

`x<-c(1,3,5)`

- Así podremos llamar al vector x en el futuro. Observemos también que es la **función de concatenación** `c()` la que construye el vector.
- También es posible definir un vector de números consecutivos, por ejemplo, el vector $(1, 2, 3, 4, 5)$ mediante

`1:5`

- De forma más general, la **función** `seq()` permite definir secuencias desde un inicio hasta un fin con una determinada separación entre ellos. Por ejemplo,

`y<-seq(-3,3,0.5)`

`y`

que proporcionará `[1] -3.0 -2.5 -2.0 -1.5 -1.0 -0.5 0.0 0.5 1.0 1.5 2.0 2.5 3.0`

TIPOS DE OBJETOS

i) Vectores

- También es útil la función `rep()` para definir vectores como repetición de otros vectores. Por ejemplo, `rep(0,100)` devolvería un vector de 100 ceros.

O también, `rep(1:3,3)` que devolvería, `[1] 1 2 3 1 2 3 1 2 3`

- De forma más general, la función `seq()` permite definir secuencias desde un inicio hasta un fin con una determinada separación entre ellos. Por ejemplo,

`y<-seq(-3,3,0.5)`

- Si queremos saber la longitud de un vector, usaremos `length()`. Por ejemplo, `length(y)` nos devolvería el valor 13.

- Decir, por último, que no hay problema en que un vector, en vez de incluir números, incluya caracteres, siempre que éstos estén entre comillas. Por ejemplo, podríamos definir el vector

`genero<-c("Mujer","Hombre")`
`genero`

TIPOS DE OBJETOS

i) **Vectores** de caracteres: la función `paste()`

- R tiene algunos vectores de caracteres predefinidos: `LETTERS`, `letters`, `month.name` y `month.abb`.
- **La función `paste()`** une todos los vectores de caracteres que se le suministran y construye una sola cadena de caracteres. Muy útil en gráficas.
- También admite argumentos numéricos, que convierte inmediatamente en cadenas de caracteres.
- En su forma predeterminada, en la cadena final, cada argumento original se separa del siguiente por un espacio en blanco, aunque ello puede cambiarse utilizando el argumento `sep="cadena"`, que sustituye el espacio en blanco por cadena, la cual podría ser incluso vacía.

Ejemplo.

```
labs <- paste(c("X","Y"),1:10,sep="")  
almacena, en labs,  
c("X1","Y2","X3","Y4","X5","Y6","X7","Y8","X9","Y10")
```

TIPOS DE OBJETOS

i) **Vectores:** Generación de secuencias regulares

- R tiene grandes facilidades para generar vectores de secuencias de números.
- El **operador** más usual es: que genera una **secuencia desde:hasta** en incrementos (o decremento si hasta es menor que desde) de uno.
- Este operador tiene la prioridad más alta en una expresión.
- La función **seq()** permite generar sucesiones más complejas. Dispone de varios argumentos (**from, to, by, length**). Existen funciones similares que realizan los cálculos más rápidamente (muy útiles a la hora de programar). Chequear la ayuda **?seq**.
- Una función relacionada es **rep()**, que permite duplicar un objeto de formas diversas. Su forma más sencilla es **rep(x, times=5)** que produce cinco copias de **x**, una tras otra.
- La función **sequence(nvec)** genera secuencias de vectores concatenados. **nvec** es un vector de enteros que especifica los límites superiores de las secuencias que se generan. Todas comienzan en uno y se van concatenando en un vector resultante.

TIPOS DE OBJETOS

i) Vectores: Extrayendo muestras

- R nos permite seleccionar muestras de un vector utilizando la función `sample(x,size,replace=FALSE,prob)`:
- Obtiene una muestra de tamaño `size` de `x`, con o sin reemplazamiento, pudiendo tener los elementos de `x` probabilidades distintas a la `uniforme` (por defecto). Si `x` tiene longitud 1, se considera el `vector 1:x`.

Ejemplo.

```
x<-c(1:10)
sample(x,3)
sample(x)
# Permutaciones de los elementos de x
y<-sample(5:15,5)
y
```

```
> x<-c(1:10)
> x
[1] 1 2 3 4 5 6 7 8 9 10
> sample(x,3)
[1] 10 5 8
> y<-sample(5:15,5)
> y
[1] 5 10 13 12 15
> |
```

TIPOS DE OBJETOS

ii) Factores

- Los factores son un tipo especial de vectores que se usan para especificar los grupos a los que pertenecen elementos de otros vectores.

Por ejemplo, si deseamos analizar datos de una muestra de 6 municipios de la región Lima según su provincia, cada elemento del objeto municipio está entrecomillado. Para convertirlo tenemos que utilizar la **función factor()** de la siguiente manera:

```
> municipio<-c("Sayán","Paramonga","Santa Eulalia","Canta","Chilca","Churín")  
> provincia<-factor(municipio)  
> Provincia
```

```
[1] Sayán   Paramonga   Santa Eulalia   Canta       Chilca
```

```
[6] Churín
```

```
Levels: Canta Chilca Churín Paramonga Santa Eulalia Sayán
```



TIPOS DE OBJETOS

iii) Matrices

- Una matriz se define mediante la función `matrix()` a la que hay que especificar sus elementos y su dimensión.
- El comando `matrix(x,nrow=m,ncol=n)` permite crear una matriz de n columnas y m filas con los elementos de x que en este caso serán números consecutivos del 1 al 10:

```
matrix(1:10,ncol=5,nrow=2)
```

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

- Por ejemplo, para definir la matriz

$$\begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$
 lo haríamos usando el comando:

```
matriz<-matrix(c(1,2,3,4,5,6,7,8,9),3,3)
```

TIPOS DE OBJETOS

iii) Matrices

Funciones útiles para trabajar con matrices

Función	Utilidad
<code>ncol(x)</code>	Número de columnas de <code>x</code> .
<code>nrow(x)</code>	Número de filas de <code>x</code> .
<code>t(x)</code>	Transpuesta de <code>x</code>
<code>cbind(...)</code>	Combina secuencias de vectores/matrices por col's.
<code>rbind(...)</code>	Combina secuencias de vectores/matrices por filas.
<code>diag(x)</code>	Extrae diagonal de matriz o crea matriz diagonal.
<code>col(x)</code>	Crea una matriz con elemento <code>ij</code> igual al valor <code>j</code>
<code>row(x)</code>	Crea una matriz con elemento <code>ij</code> igual al valor <code>i</code>
<code>apply(x,margin,FUN,)</code>	Aplica la función <code>FUN</code> a la dimensión especificada en <code>margin</code> 1 indica filas, 2 indica columnas. NB.
<code>outer(x,y,fun="*")</code> otra forma <code>x%o%y</code>	Para dos vectores <code>x</code> e <code>y</code> , crea una matriz $A[i,j]=FUN(x[i],y[j])$. Por defecto crea el producto externo.

TIPOS DE OBJETOS

iii) Matrices

Algunas operaciones con matrices

Función	Utilidad
<code>x%*%y</code> <code>crossprod(x,y=x)</code> <code>cov(x,y=x,use="all.obs")</code> <code>cor(x,y=x,use="all.obs")</code>	Multiplicación de matrices Idem que <code>t(x)%*%y</code> , pero más rápida Matriz de varianzas-covarianzas Matriz de correlaciones
<code>scale(x,center=,scale=)</code>	Resta a las columnas la media si <code>center=TRUE</code> , Si <code>center</code> es un vector, resta dichos valores. Idem para <code>scale</code> , luego de centrar, pero divide por la desviación típica si <code>scale=TRUE</code> o los valores asignados si es un vector.
<code>chol(x)</code>	Descomposición de Choleski.
<code>solve(a,b,tol=1e-7)</code>	Resolución de la ecuación <code>a%*%x=b</code> . <code>tol</code> =tolerancia para detectar dependencias lineales en las columnas de <code>a</code>
<code>eigen(x)</code> <code>sdv(x)</code>	Cálculo de valores y vectores propios. Descomposición en valores singulares.

TIPOS DE OBJETOS

iv) Hojas de datos - Listas

- Una lista es una colección de objetos de R que, por la razón que sea conviene agrupar. Los objetos pueden tener distintos tipos y características. Se construye con la **función list()**.

Para definir una lista:

```
milista = list(nombre='Pepe',no.hijos=3,edades.hijos=c(4,7,9))
```

- El comando anterior define una lista llamada **milista** con tres elementos: nombre, no.hijos y edades.hijos. Para ver el contenido de un elemento escribe, por ejemplo,

```
milista$nombre
```

- Los elementos también están numerados. El comando **milista[[1]]** (con doble corchete) equivale al anterior.

TIPOS DE OBJETOS

iv) Hojas de datos - Listas

- Podemos también extraer elementos de los elementos de una lista. Toma como ejemplo los comandos siguientes:

`milista$edades.hijos[2], milista[[3]][2]`

¿Cuál es la diferencia entre `milista[1]` y `milista[[1]]`?

¿Cuál es el resultado de `milista[2:3]`?

¿Y el resultado de `milista[[2:3]]`?

TIPOS DE OBJETOS

iv) Hojas de datos - Listas

- **Más ejemplos.**

```
ejemplolista <- list(nombre="Pedro", casado=T,  
  esposa="María",no.hijos=3, edad.hijos=c(4,7,9))  
ejemplolista  
ejemplolista[5]  
is.vector(ejemplolista[5]); is.list(ejemplolista[5])  
ejemplolista[[5]]  
is.vector(ejemplolista[[5]]); is.list(ejemplolista[[5]])  
ejemplolista[[5]][2]  
ejemplolista[[5]]  
ejemplolista$casado  
ejemplolista$nombre  
is.recursive(ejemplolista)  
is.atomic(ejemplolista) # vemos el tipo de objeto  
listamasgrande <- c(ejemplolista,list(edad=40))
```

TIPOS DE OBJETOS

iv) Hojas de datos - Listas

Los data.frame

- Las bases de datos en estadística son, habitualmente de la forma:

individuo	Variables			
	años	implante	edad	sexo
1	1.3	2	22	H
2	0.4	2	21	M
3	1.1	2	34	H
4	2.3	1	42	H
5	3.1	3	17	M
6	1.3	1	43	H

- R organiza este tipo de información en objetos del tipo `data.frame`, un caso particular de lista.
- Los `data.frame` son apropiados para describir “matrices de datos” donde cada fila representa a un individuo y cada columna una variable, cuyas variables pueden ser numéricas o categóricas.

TIPOS DE OBJETOS

iv) Hojas de datos – Listas Los data.frame

- Los data.frame se crean con la función `data.frame()`:

Ejemplo.

```
datosimp <- data.frame(anyos=c(1.3,0.4,1.1,2.3,3.1,1.3),  
  tipo=c(2,3,3,1,3,1),edad=c(22,21,34,42,17,43),  
  sexo=c("H","M","H","H","M","H"))
```

```
> datosimp  
  años tipo edad sexo  
1  1.3    2   22    H  
2  0.4    3   21    M  
3  1.1    3   34    H  
4  2.3    1   42    H  
5  3.1    3   17    M  
6  1.3    1   43    H
```

- Las componentes deben ser vectores (numéricos, carácter o lógicos), factores, matrices numéricas u otros data.frames.
- Puesto que un **vector** representará una variable del banco de datos y las columnas de una **matriz** representarán varias variables de ese mismo banco, la longitud de los vectores debe ser la misma y coincidir con el número de las de las matrices.
- Los datos que no son numéricos, la estructura de data.frame los considera factores, con tantos niveles como valores distintos encuentre, tal y como ocurre en el ejemplo que estamos considerando.

TIPOS DE OBJETOS

iv) Hojas de datos - Listas

Las funciones `attach()` y `detach()`

- Para trabajar con las variables del banco de datos, podemos utilizar la notación estándar de las listas, `$nombre` o `[[]]`, pero resulta más natural emplear simplemente el nombre de la columna.
- El problema es que R guarda el nombre del data.frame pero no sus variables. Para poderlas utilizar con su nombre como vectores, tenemos que utilizar la función `attach(nombre de data.frame)`. La operación inversa se realiza con la función `detach()`.
- En cualquier momento es posible llevar a cabo un filtrado de datos utilizando expresiones lógicas (por ejemplo, para seleccionar de un banco de datos sólo los que son mayores de 65 años) y vectores índices.
- En el caso de los data.frames es posible seleccionar filas y columnas deseadas, y crear con ellas otros data.frame con las características deseadas con la función `subset()`.

TIPOS DE OBJETOS

iv) Hojas de datos - Listas

Las funciones attach() y detach()

Ejemplo

```
datosimp <- data.frame(anyos=c(1.3,0.4,1.1,2.3,3.1,1.3),  
                      tipo=c(2,3,3,1,3,1),edad=c(22,21,34,42,17,43),  
                      sexo=c("H","M","H","H","M","H"))
```


anyos

```
attach(datosimp); anyos
```

```
detach(datosimp); anyos
```

```
datos.hombre.filtrados <- datosimp[datosimp$sexo=='H',]
```

```
mas.peq <- subset(datosimp,anyos<1,select=c(edad,sexo))
```



> datos.hombre.filtrados				
	anyos	tipo	edad	sexo
1	1.3	2	22	H
3	1.1	3	34	H
4	2.3	1	42	H
6	1.3	1	43	H

TIPOS DE OBJETOS

Notas

- Los nombres de los objetos pueden contener sólo letras mayúsculas o minúsculas (son distintas), junto con números y puntos
(NO blancos, NO __ , NO%, NO \$, etc.).
- Los corchetes o dobles corchetes se utilizan para seleccionar partes de un objeto así como el dólar.
- Durante una sesión de trabajo con **R** los objetos que se crean se van almacenando por su nombre.
- La función `objects()` se puede utilizar para obtener los nombres de los objetos almacenados en **R**. Es equivalente a la función `ls()`.

ATRIBUTOS DE LOS OBJETOS

- Los atributos de un objeto suministran información específica sobre el propio objeto.
- El modo o tipo de un objeto es un caso especial de un atributo de un objeto. Con el modo de un objeto designamos el tipo básico de sus constituyentes fundamentales.
- Los atributos de un objeto suministran información específica sobre el propio objeto. Todos los objetos tienen dos atributos intrínsecos: el modo y su longitud.
- Las funciones `mode(objeto)` y `length(objeto)` se pueden utilizar para obtener el modo y longitud de cualquier estructura.

Ejemplo.

```
x<-c(1,3)
```

```
mode(x) # Devuelve el tipo numérico
```

```
length(x) # Devuelve la longitud
```

ATRIBUTOS DE LOS OBJETOS

- Mediante `attributes(objeto)` podemos obtener una lista de los atributos no **intrínsecos** y con `attr(objeto, atributo)` podemos usar el atributo seleccionado (p.e. para asignarle un valor).
- Los atributos son distintos según el tipo de objeto. Una pequeña lista de atributos es la siguiente:

Atributos de cada tipo

Atributo	Tipos
mode	Todos
storage.mode	Todos los datos de modo numérico
length	Todos
names	Vectores y listas
dim	Matrices y arrays
dimnames	Matrices y arrays
tsp	Series temporales
levels	Factores
class	Cualquier clase

MODIFICACIÓN DE LA LONGITUD DE UN OBJETO

- Un objeto, aunque esté vacío, tiene modo.

Ejemplo.

`v<-numeric()` almacena en `v` una estructura vacía de vector numérico.

`character()` es un vector de caracteres vacío, y lo mismo ocurre con otros tipos.

- Una vez creado un objeto con un tamaño cualquiera, pueden añadirse nuevos elementos sin más que asignarlos a un índice que esté fuera del rango previo.

Ejemplo.

`v[3]<-17` transforma `v` en un vector de longitud 3, (cuyas dos primeras componentes serán `NA`).

- Esta regla se aplica a cualquier estructura, siempre que los nuevos elementos sean compatibles con el modo inicial de la estructura.

ASIGNACIÓN

- Recordar que la función principal para definir un objeto es a través de sus componentes, con la **función** `c()`, mediante el comando más importante en R que es `<-` el de la asignación.
- Las asignaciones pueden realizarse también con una flecha apuntando a la derecha, realizando el cambio obvio en la asignación.

Ejemplo.

`x <- 3` es equivalente a `3 -> x` pero no a `x -> 3`

- La asignación puede realizarse también mediante la **función** `assign()`.

Ejemplo.

`x<-c(1,3)` es equivalente a `assign("x", c(1, 3))`

- En muchos contextos el **símbolo** `=` puede también utilizarse indistintamente para asignar un objeto.

ASIGNACIÓN

- Si una expresión se utiliza como una orden por sí misma, su valor se imprime y se pierde. Así pues, la orden **1/x** simplemente imprime el inverso de lo que sea el objeto **x** sin modificar su valor.
- Podemos crear vectores con valores iniciales, **FALSE**, **0,0+0i**, **" "**, mediante la función que indica el tipo de dato y entre paréntesis el numero de elementos a crear.

Ejemplo.

x1<-logical(4)

Inicializa un vector de longitud 4 cuyos elementos son FALSE

x2<-numeric(4); x3<-complex(4)

Inicializa (a 0) vectores numéricos de longitud 4

x4<-character(4)

Inicializa un vector de caracteres de longitud 4

```
> x1<-logical(4)
> x1
[1] FALSE FALSE FALSE FALSE
> x2<-numeric(4); x3<-complex(4)
> x2
[1] 0 0 0 0
> x4<-character(4)
> x4
[1] "" "" "" ""
> |
```

OPERADORES LÓGICOS

- Los elementos de un objeto de tipo lógico tienen dos posibilidades, **FALSE** o **TRUE**. Se pueden abreviar en **F** y **T** respectivamente.
- Los objetos lógicos son generalmente fruto de una comparación.

Ejemplo.

Si $x < -3$

$y <- x > 12$ produce un objeto lógico de longitud la de x con valor F o V según se cumpla o no la condición (en este caso F)

- Las comparaciones que dan un resultado lógico son:

$<; <=; >; >=; ==; !=$

- Los objetos lógicos se pueden utilizar con la aritmética ordinaria, en cuyo caso son transformados en vectores numéricos, FALSE se convierte en **0** y TRUE en **1**.

OPERADORES LÓGICOS

La aritmética entre objetos lógicos se puede llevar a cabo con los siguientes operadores lógicos:

Operadores lógicos

Operador	Operación
<code>!x</code>	Negación de x. Los T los convierte en F y viceversa.
<code>x&y</code>	Intersección, operador lógico y: T y T da T, otra comparación da F
<code>x y</code>	Unión, operador lógico o: F y F da F, otra comparación da T.
<code>xor(x,y)</code>	Exclusivo OR, <code>xor(T,F)==T</code> , otra comparación da F.
<code>all</code>	Para una secuencia de argumentos lógicos, <code>all</code> devuelve el valor lógico que indica si todos los elementos son TRUE.
<code>any</code>	Para una secuencia de argumentos lógicos, <code>any</code> devuelve el valor lógico que indica si algún elemento es TRUE.

OPERADORES LÓGICOS

Ejemplos.

```
x<-c(T,T,F,F)
```

```
y<-c(T,F,T,F)
```

```
x&y # produce TRUE FALSE FALSE FALSE
```

```
x|y # produce TRUE TRUE TRUE FALSE
```

```
xor(x,y) # produce FALSE TRUE TRUE FALSE
```

```
any(x) # produce TRUE
```

```
all(x) # produce FALSE
```

```
z<- c(NA, T, F)
```

```
?outer
```

```
# ayuda sobre el producto exterior de dos vectores
```

```
outer(z,z,"&")
```

```
outer(z,z,"|")
```

```
> z<- c(NA, T, F)
> z
[1]    NA    TRUE   FALSE
> ?outer
> outer(z,z,"&")
      [,1] [,2] [,3]
[1,]    NA    NA  FALSE
[2,]    NA    TRUE FALSE
[3,]  FALSE  FALSE  FALSE
> outer(z,z,"|")
      [,1] [,2] [,3]
[1,]    NA  TRUE    NA
[2,]  TRUE  TRUE   TRUE
[3,]    NA  TRUE  FALSE
> |
```


COERCIÓN DE TIPOS

- La mayoría de las funciones producen un error cuando el tipo de datos que esperan no coincide con los que ponemos en los argumentos.
- Tenemos dos posibilidades:
 - comprobar el tipo de datos utilizando por ejemplo funciones `is.numeric()`, que nos responde con un valor lógico,
 - o forzar al tipo de datos deseados **coercionando**, para lo cual podemos utilizar por ejemplo funciones del tipo `as.numeric()`, que fuerzan el tipo de datos.
- Para conocer si un elemento es `NA`, la comparación lógica `==`, no puede funcionar ya que `NA` es la nada. Utilizamos para ello la función `is.na()`, que nos responde con un valor lógico: `TRUE` para los elementos del vector que son `NA`.
- De manera similar, podemos utilizar `is.finite()`, `is.nan()`, etc.

COERCIÓN DE TIPOS

Ejemplo. supongamos que definimos un vector v

```
v <- c(1, 2, 3, 4, 5)
```

```
v
```

```
typeof(v)
```

```
class(v)
```

Cuando un vector lógico es convertido a un integer o double, TRUE es cambiado a 1 y FALSE a 0:

```
v <- c(FALSE, TRUE, FALSE)
```

```
as.numeric(v)
```

Los objetos pueden ser convertidos en otro tipo de forma explícita mediante el uso de la función as.()

```
as.logical(v)
```

```
as.character(v)
```

```
v <- c("a", "b", "c")
```

```
as.numeric(v)
```

COERCIÓN DE TIPOS

Lista de los tipos más importantes que se pueden comprobar o forzar

Tipo	Comprobación	Coerción
array	<code>is.array()</code>	<code>as.array()</code>
character	<code>is.character()</code>	<code>as.character()</code>
complex	<code>is.complex()</code>	<code>as.complex()</code>
double	<code>is.double()</code>	<code>as.double()</code>
factor	<code>is.factor()</code>	<code>as.factor()</code>
integer	<code>is.integer()</code>	<code>as.integer()</code>
list	<code>is.list()</code>	<code>as.list()</code>
logical	<code>is.logical()</code>	<code>as.logical()</code>
matrix	<code>is.matrix()</code>	<code>as.matrix()</code>
NA	<code>is.na()</code>	-
NaN	<code>is.nan()</code>	-
NULL	<code>is.null()</code>	<code>as.null()</code>
numeric	<code>is.numeric()</code>	<code>as.numeric()</code>
ts	<code>is.ts()</code>	<code>as.ts()</code>
vector	<code>is.vector()</code>	<code>as.vector()</code>

COERCIÓN DE TIPOS

Más ejemplos.

`x<-c(1:10)`

`is.numeric(x)`

resulta en TRUE

`is.vector(x)`

resulta en TRUE

`is.complex(x)`

resulta en FALSE

`is.character(x)`

resulta en FALSE

`x<-as.character(x)`

fuerza el vector x a tomar los valores ("1" "2" "3" "4" "5" "6" "7" "8" "9" "10")

```
> x<-c(1:10)
> x
[1] 1 2 3 4 5 6 7 8 9 10
> is.numeric(x)
[1] TRUE
> is.vector(x)
[1] TRUE
> is.complex(x)
[1] FALSE
> is.character(x)
[1] FALSE
> x<-as.character(x)
> x
[1] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10"
> |
```

PAQUETES (BIBLIOTECAS O LIBRERÍAS) DE R

- Muchas personas utilizan R como un sistema estadístico. Los autores prefieren describirlo como un entorno en el que se han implementado muchas técnicas estadísticas, tanto clásicas como modernas.
- R consta de un **sistema base** (donde están incluidas una enorme cantidad de técnicas estadísticas y numéricas) **y de paquetes** (packages) adicionales que extienden su funcionalidad.
- El hecho de distinguir entre ambos conceptos es fundamentalmente una cuestión histórica. **Junto con R se incluyen ocho paquetes** (llamados paquetes estándar) pero otros muchos están disponibles a través de Internet en CRAN:

<http://cran.es.r-project.org/>

INSTALACIÓN DE PAQUETES

Para instalar paquetes de R:

- Desde el menú **Paquetes > Instalar paquete(s)...**
 - ▶ Primero nos pide seleccionar el “**CRAN mirror**”, por comodidad podemos elegir el que hay en España: **Spain(Madrid)**.
 - ▶ Luego seleccionamos el **paquete** que queremos. Si el paquete necesita otros paquetes, los instala automáticamente.
- También podemos utilizar la **función `install.packages()`** como en GNU/Linux.
- La instalación no implica que los paquetes ya puedan ser utilizados. Es necesario cargar las **librerías** antes de empezar a usarlas. Lo mismo ocurre con las librerías existentes en la versión local de R.

INSTALACIÓN DE PAQUETES

Ejercicio. Instalar el paquete `foreign`.

- Existen dos maneras de hacerlo:
 - ▶ desde el menú **Paquetes > Cargar paquete...** seleccionamos la librería que queramos,
 - ▶ o bien desde la línea de comandos utilizando la **función `library()`**:

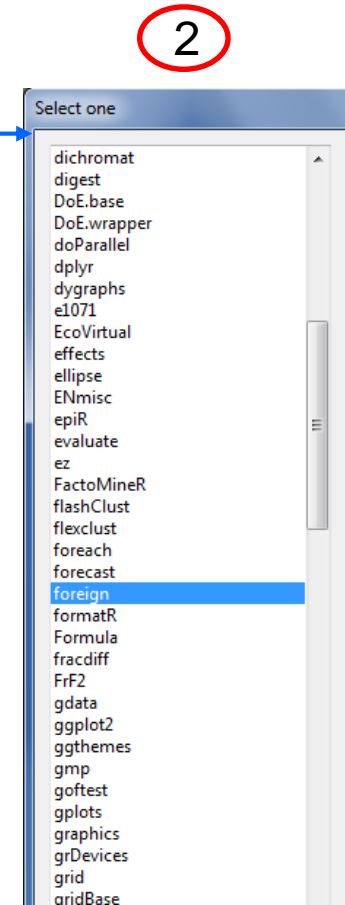
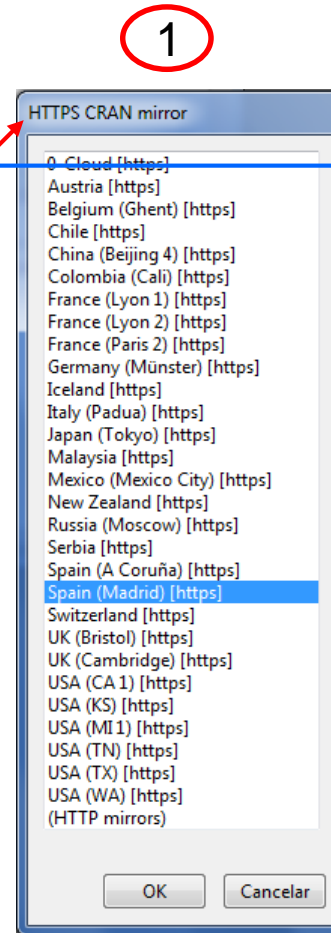
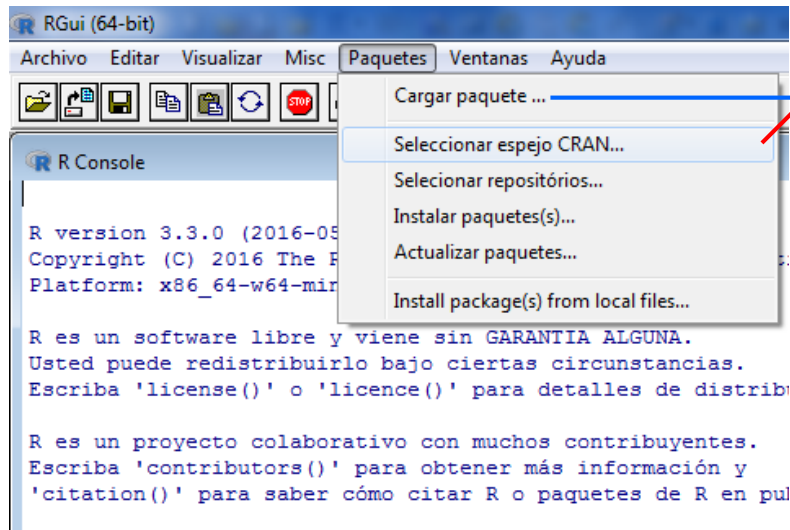
```
library(foreign)  
#Carga la librería foreign
```

- El comando `library()` abre una ventana con información sobre las librerías (paquetes) instaladas en R.
- Para obtener más información sobre estas librerías, se puede utilizar las **funciones `library` y `help`** conjuntamente:

```
library(help="foreign")  
#Abre una ventana de ayuda sobre foreign
```

INSTALACIÓN DE PAQUETES

Ejercicio. Instalar el paquete `foreign`.



ARCHIVOS DE DATOS

¿Qué es una base de datos?

- Hasta ahora hemos visto cómo introducir datos interactivamente en forma de objetos, aunque habitualmente los datos nos vienen proporcionados en archivos externos.
- En R se le llama **data frame**
- Una especie de matriz bi-dimensional:
 - **Columnas** representan variables.
 - **Filas** representan personas, registros o casos.
- Contiene diferentes tipos de datos (es un tipo de lista)
- **Ejemplo.** **Iris.sav**

LECTURA DE ARCHIVOS DE DATOS

- Como los requisitos de lectura de R son bastante estrictos, muchas veces conviene modificar el archivo de datos externo con otros editores de texto (como WinEdt, etc.).
- Para asignar un banco de datos a un data.frame se utiliza la **función `read.table()`**. La forma más sencilla es con un archivo en el que:
 - La primera línea del archivo debe contener el nombre de cada variable de la hoja de datos.
 - En cada una de las siguientes líneas, el primer elemento es la etiqueta de la variable, y a continuación deben aparecer los valores de cada variable. Si no, **R** asigna automáticamente etiquetas a las filas.

Ejemplo

```
aves<- read.table(file="F:/Datos/aves.txt", header=T)
```

ARCHIVOS DE DATOS

La función scan()

- La función `scan()` es más genérica que la anterior, y vale para asignar cualquier tipo de objetos (vectores, matrices, listas, etc.).
- Si el segundo argumento es un sólo elemento, todos los elementos del archivo deben ser del tipo indicado y se leen en un sólo vector.

Ejemplo

En su uso más básico la función `scan()` permite introducir los valores interactivamente.

```
x<- scan()  
1:
```

En el prompt nos va pidiendo valores que se van incorporando al vector hasta que pulsamos intro y acaba. Probar a introducir 1, 5, 8 y 3.

ARCHIVOS DE DATOS

La función scan()

Ejemplo

Supongamos que el archivo “entrada.txt” contiene los datos de tres vectores, de igual longitud, el primero tipo carácter y los otros dos tipo numérico, escritos de tal modo que en cada línea aparecen los valores correspondientes de cada uno de ellos:

```
entrada<- scan("entrada.txt", list("",0,0))
```

Podemos referirnos a los vectores con:

```
etiqueta <- entrada[[1]]; x <- entrada[[2]]; y <- entrada[[3]]
```

También podríamos haber utilizado:

```
entrada <- scan("entrada.txt", list(etiqueta="",x=0,y=0))
```

lo que nos permitiría utilizar la notación \$ para referirnos a los vectores:

```
etiqueta <- entrada$etiqueta; x <- entrada$x; y <- entrada$y
```

ARCHIVOS DE DATOS

Otras formas de leer y guardar datos

- La función `save(objetos, list=character(0), file="nomfich.txt", ascii=FALSE)` nos permite guardar los objetos que queramos en `nomch.txt`.
- `load()` nos permite leer datos de un archivo binario que contiene los objetos de R previamente guardados con `save()`.
- Como vimos en la primera sesión, cuando cerramos R existe la posibilidad de guardar todos los objetos de la sesión de trabajo en archivos `.RData`.
- En realidad, al grabar toda la imagen R está aplicando la función `save.image()`, un caso particular de la función `save(list=ls(),file=".Rdata")`.
- Al cargar los datos desde el menú **Archivo>cargar área de trabajo** o hacer doble click sobre el archivo `.RData` en el fondo estamos utilizando la función `load()`.

ARCHIVOS DE DATOS

Otras formas de leer y guardar datos

- Aunque no es muy elegante, siempre es posible guardar el flujo de comandos de R que contiene la definición de un `data.frame` y luego recuperarlo con la **función `source()`** que vimos en la primera sesión.
- **`data()`** lista los bancos de datos disponibles en el package `datasets`, mientras que `data(package = .packages(all.available = TRUE))` lista todos los disponibles en las librerías que hayamos cargado anteriormente.
- La **función `data(nom.banco.datos)`** carga el banco de datos para que podamos utilizarlo.
- Existen dos funciones más para guardar datos:
 - **`write()`** que guarda en un archivo texto vectores y matrices que luego se pueden leer con `scan()` y
 - **`write.table()`** que guarda también en archivo de texto los `data.frame` para leerlos luego con `read.table()`.

ARCHIVOS DE DATOS

Importar y exportar datos de otros programas

- La gran aceptación de R está haciendo cada vez más fácil el importar cheros de otros programas.
- La librería `foreign` contiene funciones para importar y exportar datos en el formato de otros programas.
- Utilizando `help(package="foreign")` podemos encontrar una lista de funciones y formatos soportados de otros programas estadísticos (SPSS, SAS, STATA, Minitab, SPlus, EpilInfo, Systat) y numéricos (Octave).

Ejemplo

```
library(foreign) # Cargamos el paquete  
datos<-read.spss(file="glucosa.sav",to.data.frame=TRUE)  
str(datos)
```

ARCHIVOS DE DATOS

Importar y exportar datos de otros programas

- Existen otras opciones (ver el manual R-data incluido en el material y la web <http://cran.r-project.org/>) para conectar R con diferentes bases de datos (Access, Excel, MySQL, dBase, etc.).
- La idea es utilizar lo que se conoce como “Open DataBase Connectivity” (ODBC), un estándar de acceso a bases de datos utilizado por la mayoría de bases de datos existentes.
- A través de ODBC, un sistema puede conectarse a cualquier base de datos (que esté local en nuestro ordenador o incluso que esté remota) que soporte ODBC sin necesidad de conocer sus características específicas. Para ello, el desarrollador de la base de datos es el encargado de crear el driver de ODBC necesario para que su base de datos pueda ser utilizada desde ODBC.
- En el caso de R, existe una librería llamada RODBC que permite conectarse mediante un ODBC y desarrollar consultas contra bases de datos.

ARCHIVOS DE DATOS

Caso particular: importar datos de Excel

- La librería RODBC permite conectarse mediante un ODBC a un archivo excel.

Comandos tipo para conectar con Excel

```
library(RODBC) # Cargamos el paquete
conexion<-odbcConnectExcel("dedos.xls")
Datos<-sqlQuery(channel=conexion,"select * from [Hoja1$]")
close(conexion)
```

- La segunda línea establece una conexión ODBC con el archivo dedos.xls utilizando la función `odbcConnectExcel()`.
- La tercera línea selecciona la **Hoja1** de ese archivo de Excel. **Datos** pasa a ser un data.frame de R.
- La cuarta línea cierra la conexión, que ya no es necesaria.

ARCHIVOS DE DATOS

La mejor opción para importar archivos de Excel es evitar conexiones externas.

Preparar datos en Excel

- La tabla a exportar debe estar en la esquina superior izquierda
- No puede haber más datos en esa hoja de Excel

Exportar a fichero ASCII

- Selección Fichero— > Salvar como
- Seleccionar “Formato CSV” (o personalizado)
- Seleccionar formato de exportación:
 - ▶ Carácter para delimitar cadenas de texto
 - ▶ Símbolo de los decimales
 - ▶ Símbolo para separar columnas

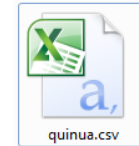
Importar datos en R

Leer con `read.table()`, `read.csv()` o `read.csv2()`

ARCHIVOS DE DATOS

Leer archivos delimitados por comas (*.csv)

Ejemplo. Leer en R el archivo `quinua.xls`, para ello es necesario convertirlo con formato `quinua.csv`



```
R Console
> datos1 = read.table(file="D:/quinua.csv", header=TRUE, sep=",")
> datos1
```

	IDENTIF	TGRANO	GERMINA	RES_MILD	ALTURA	UNIFORM	HAB_CRECC	LONG_PANJ	P_GRANO
1	1	2	2	1	60	1	3	25	55.1
2	2	2	5	1	63	1	3	21	240.0
3	3	2	5	1	71	1	3	33	205.8
4	4	2	2	2	60	1	3	21	33.7
5	5	2	5	1	75	1	3	40	275.7
6	6	2	3	2	87	1	4	39	169.0

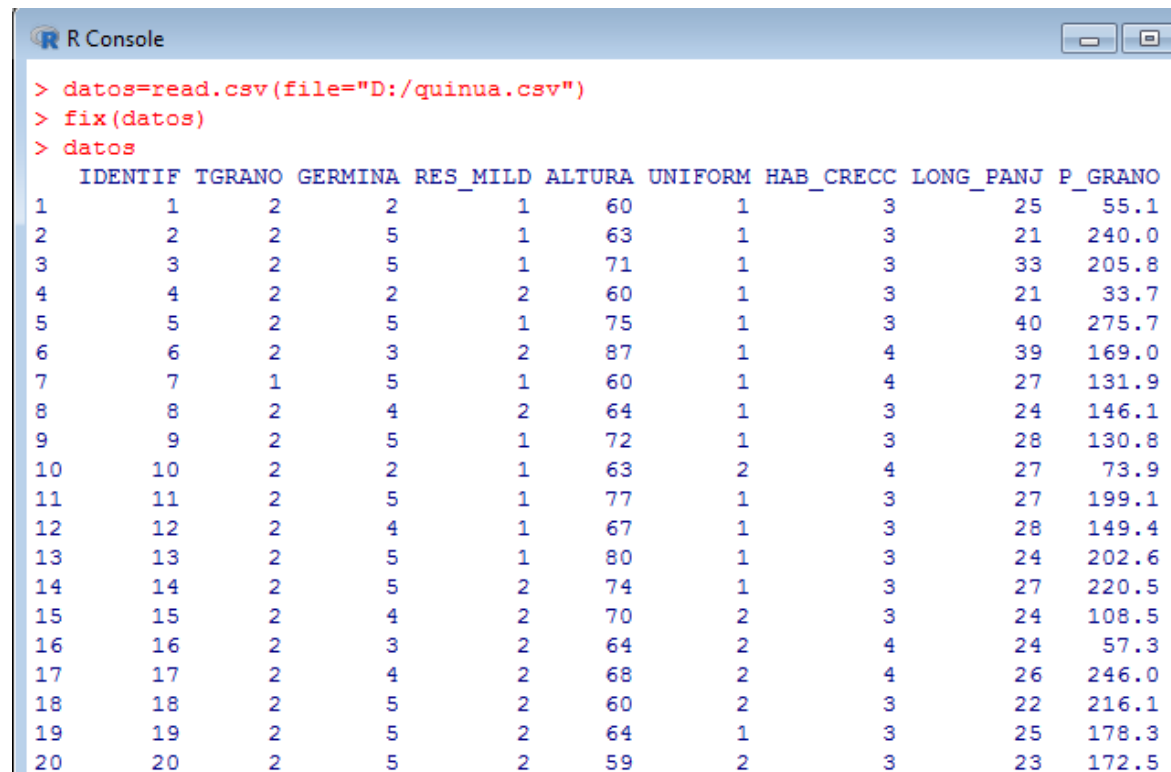
Donde:

- `datos1` es un objeto de tipo marco de datos para R.
- `read.table` es la función que nos permite leer el archivo `MMR.csv` desde R.
- `header` es un argumento de la función `read.table` que lee la primera fila del archivo `quinua.csv`, como una fila que contiene los nombres de las variables de la base de datos.
- `sep` indica el elemento que actúa como separador de los datos en este caso, la coma.

ARCHIVOS DE DATOS

Leer archivos delimitados por comas (*.csv)

Ejemplo. Cont.



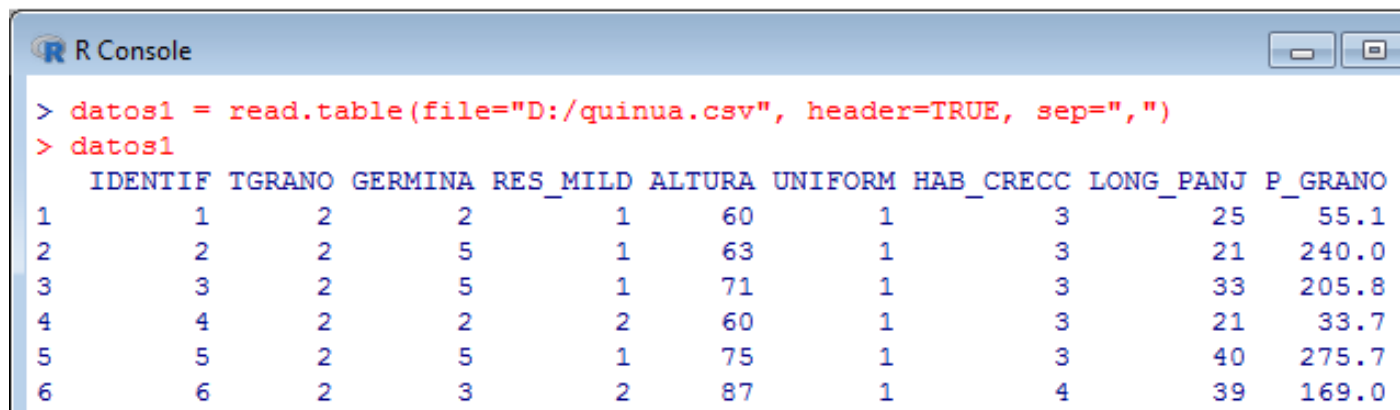
```
R Console
> datos=read.csv(file="D:/quinua.csv")
> fix(datos)
> datos
```

	IDENTIF	TGRANO	GERMINA	RES_MILD	ALTURA	UNIFORM	HAB_CRECC	LONG_PANJ	P_GRANO
1	1	2	2	1	60	1	3	25	55.1
2	2	2	5	1	63	1	3	21	240.0
3	3	2	5	1	71	1	3	33	205.8
4	4	2	2	2	60	1	3	21	33.7
5	5	2	5	1	75	1	3	40	275.7
6	6	2	3	2	87	1	4	39	169.0
7	7	1	5	1	60	1	4	27	131.9
8	8	2	4	2	64	1	3	24	146.1
9	9	2	5	1	72	1	3	28	130.8
10	10	2	2	1	63	2	4	27	73.9
11	11	2	5	1	77	1	3	27	199.1
12	12	2	4	1	67	1	3	28	149.4
13	13	2	5	1	80	1	3	24	202.6
14	14	2	5	2	74	1	3	27	220.5
15	15	2	4	2	70	2	3	24	108.5
16	16	2	3	2	64	2	4	24	57.3
17	17	2	4	2	68	2	4	26	246.0
18	18	2	5	2	60	2	3	22	216.1
19	19	2	5	2	64	1	3	25	178.3
20	20	2	5	2	59	2	3	23	172.5

ARCHIVOS DE DATOS

Leer archivos delimitados por tabulaciones (*.txt)

Ejemplo. Leer en R el archivo `datos.txt`



```
> datos1 = read.table(file="D:/quinua.csv", header=TRUE, sep=",")
> datos1
```

	IDENTIF	TGRANO	GERMINA	RES_MILD	ALTURA	UNIFORM	HAB_CRECC	LONG_PANJ	P_GRANO
1	1	2	2	1	60	1	3	25	55.1
2	2	2	5	1	63	1	3	21	240.0
3	3	2	5	1	71	1	3	33	205.8
4	4	2	2	2	60	1	3	21	33.7
5	5	2	5	1	75	1	3	40	275.7
6	6	2	3	2	87	1	4	39	169.0

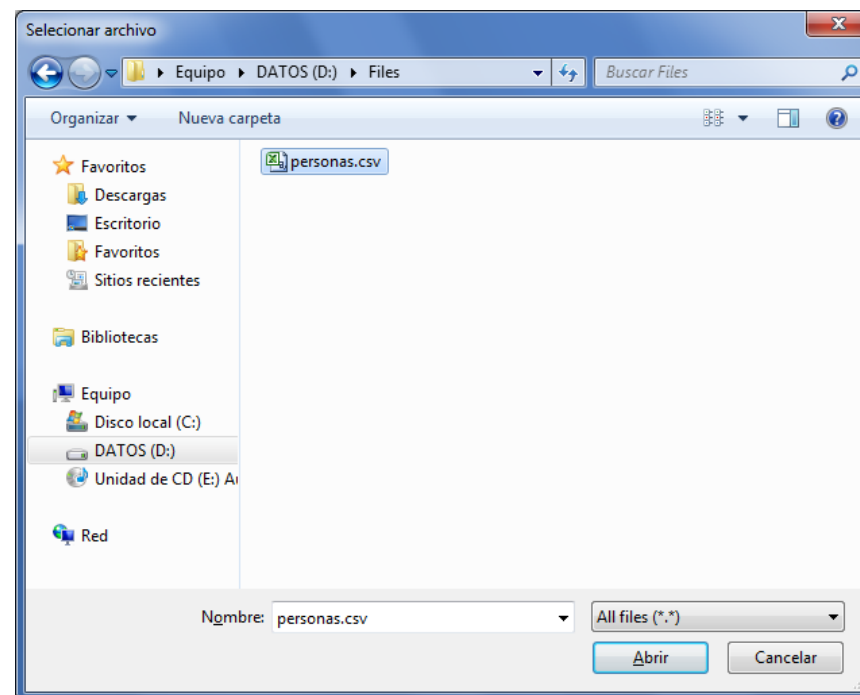
En este caso `sep="\t"`, indica que las tabulaciones son los separadores de datos.

ARCHIVOS DE DATOS

Leer archivo seleccionándolos desde una carpeta

Ejemplo.

```
> datos3 = read.table(file.choose(), header=TRUE, sep=",")
```



ARCHIVOS DE DATOS

Leer archivo seleccionándolos desde una carpeta

Ejemplo. Cont.

R Console

```
> datos3 = read.table(file.choose(), header=TRUE, sep=",")  
> datos3
```

	ID	UBIGEO	DOMINIO	AREA	SEXO	EDAD	EDAD_MES
1	1410041101	21801	2	1	1	44	1
2	1410041102	21801	2	1	2	41	1
3	1410041103	21801	2	1	1	18	1
4	1410041104	21801	4	1	2	15	1
5	1410041105	21801	4	1	1	7	1
6	1410281101	21801	4	1	1	32	1
7	1410281102	21801	2	2	2	22	0
8	4510711103	40701	3	2	1	23	0
9	4511081101	40701	3	2	1	87	1
10	4511081102	40701	3	2	2	76	0
11	4520101101	40701	3	2	1	52	1
12	14130711104	130101	1	1	2	58	0
13	14130711105	130101	4	1	1	22	1
14	14131361101	130101	1	1	1	57	1
15	14131361102	130101	1	1	2	53	0
16	14131361103	130101	1	1	1	23	0
17	14131361104	130101	1	1	2	16	1
18	24760771104	180101	3	2	1	20	1
19	24760941101	180101	3	2	1	47	0

```
> |
```


ARCHIVOS DE DATOS

Leer archivo de SPSS

Ejemplo. Leer en R la sumaria de la ENAHO-2017.

The screenshot displays the R environment. The R Console window shows the following commands and output:

```
> datos4<-read.spss(file="D:/ENAHO/ENAHO 2017/Sumaria-2017.sav", to.data.frame=TRUE)  
re-encoding from UTF-8  
> fix(datos4)
```

Below the console, the 'Editor de datos' window is open, showing a data frame with 19 rows and 8 columns. The columns are labeled: AÑO, MES, NCONGLOME, CONGLOME, VIVIENDA, HOGAR, and UBIGEO. The data represents various households from the ENAHO-2017 survey.

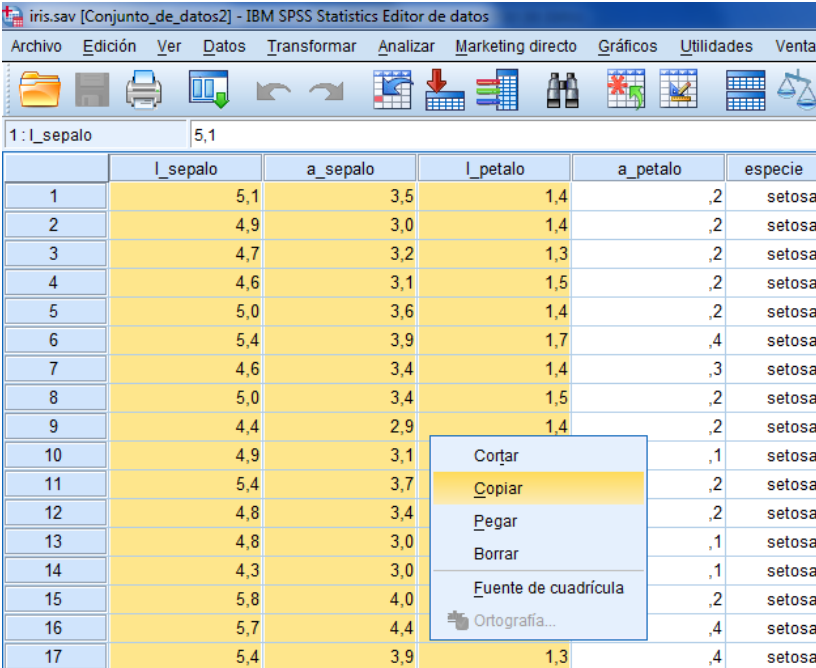
	AÑO	MES	NCONGLOME	CONGLOME	VIVIENDA	HOGAR	UBIGEO
1	2017	12	007060	005001	002	11	010101
2	2017	12	007060	005001	023	11	010101
3	2017	12	007060	005001	035	11	010101
4	2017	12	007060	005001	046	11	010101
5	2017	12	007060	005001	067	11	010101
6	2017	12	007060	005001	088	11	010101
7	2017	04	007061	005002	009	11	010101
8	2017	04	007061	005002	034	11	010101
9	2017	04	007061	005002	046	11	010101
10	2017	04	007061	005002	071	11	010101
11	2017	04	007061	005002	083	11	010101
12	2017	11	007063	005003	001	11	010101
13	2017	11	007063	005003	053	11	010101
14	2017	11	007063	005003	070	11	010101
15	2017	11	007063	005003	105	11	010101
16	2017	11	007063	005003	140	11	010101
17	2017	04	007065	005004	006	11	010101
18	2017	04	007065	005004	025	11	010101
19	2017	04	007065	005004	064	11	010101

ARCHIVOS DE DATOS

Leer un segmento de archivo de SPSS

Ejemplo. Leer en R un segmento de los datos de iris.sav.

Primero seleccionamos en **SPSS**, el segmento de datos que queremos leer, y activamos la opción copiar.



1 : l_sepalo 5,1

	l_sepalo	a_sepalo	l_petal	a_petal	especie
1	5,1	3,5	1,4	,2	setosa
2	4,9	3,0	1,4	,2	setosa
3	4,7	3,2	1,3	,2	setosa
4	4,6	3,1	1,5	,2	setosa
5	5,0	3,6	1,4	,2	setosa
6	5,4	3,9	1,7	,4	setosa
7	4,6	3,4	1,4	,3	setosa
8	5,0	3,4	1,5	,2	setosa
9	4,4	2,9	1,4	,2	setosa
10	4,9	3,1		,1	setosa
11	5,4	3,7		,2	setosa
12	4,8	3,4		,2	setosa
13	4,8	3,0		,1	setosa
14	4,3	3,0		,1	setosa
15	5,8	4,0		,2	setosa
16	5,7	4,4		,4	setosa
17	5,4	3,9	1,3	,4	setosa

ARCHIVOS DE DATOS

Leer un segmento de archivo de SPSS

Ejemplo. Leer en R un segmento de los datos de iris.sav.


R Console

```
> datos5 = read.table("clipboard")  
> datos5
```

	V1	V2	V3
1	5,1	3,5	1,4
2	4,9	3,0	1,4
3	4,7	3,2	1,3
4	4,6	3,1	1,5
5	5,0	3,6	1,4
6	5,4	3,9	1,7
7	4,6	3,4	1,4
8	5,0	3,4	1,5
9	4,4	2,9	1,4
10	4,9	3,1	1,5
11	5,4	3,7	1,5
12	4,8	3,4	1,6
13	4,8	3,0	1,4
14	4,3	3,0	1,1
15	5,8	4,0	1,2

INTERFAZ GRÁFICA EN PERSPECTIVA

- Hay variedad de interfaces para R

- *R Commander* – es un paquete de R 

- *RK Ward* – <http://rkward.sourceforge.net> 

- *Emacs* – <http://www.gnu.org/software/emacs> 

- *RStudio* – <http://www.rstudio.com> 

R COMMANDER

Introducción

- Una de las librerías de R más destacadas es **Rcommander (Rcmdr)**, que nos proporciona un interfaz gráfico, a modo de ventanas, para realizar análisis no muy complicados de una manera muy accesible.
- **R-Commander** es una Interfaz Gráfica de Usuario (GUI en inglés), creada por John Fox, que permite acceder a muchas capacidades del entorno estadístico R sin que el usuario tenga que conocer el lenguaje de comandos propio de este entorno.
- También es una magnífica herramienta de R que sustituye a paquetes tradicionales tales como SPSS, etc.
- Se instala como el resto de librerías. Al instalarla, solicita permiso para instalar un considerable número de librerías.

R COMMANDER

Introducción

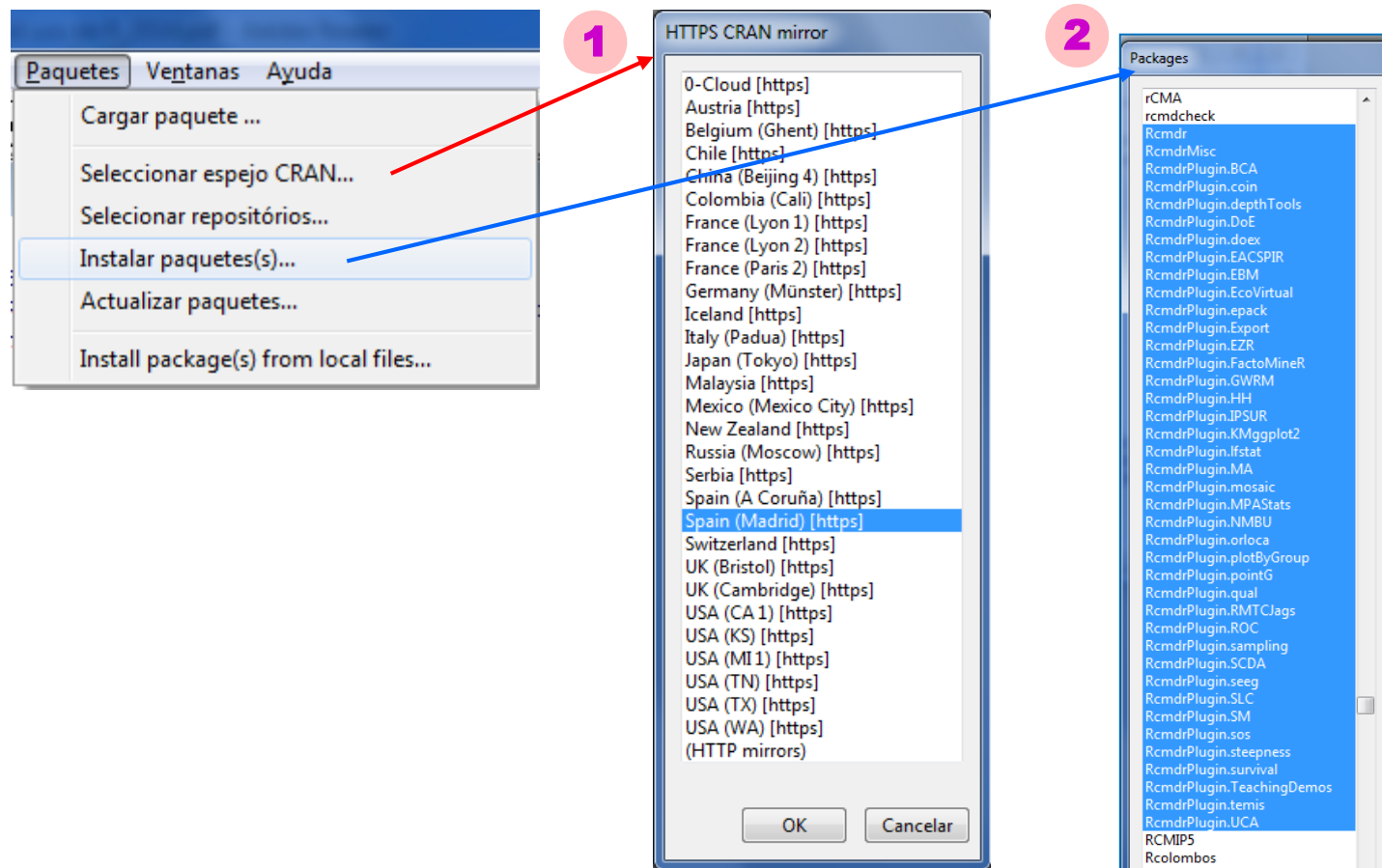
- Tras instalarla, podemos cargarla de la manera habitual para poder utilizarla. Observaremos que abre una ventana con menús similar a otros programas de estadística.
- Se le pueden incorporar una serie de complementos (o plugins) que se instalan como paquetes de R. Se cargan desde el menú

Paquetes -> Cargar paquetes -> Rcmdr

- Hay que reinicializar Rcommander para poder utilizarlos, y al hacerlo algunos menús de Rcommander cambian.

R COMMANDER

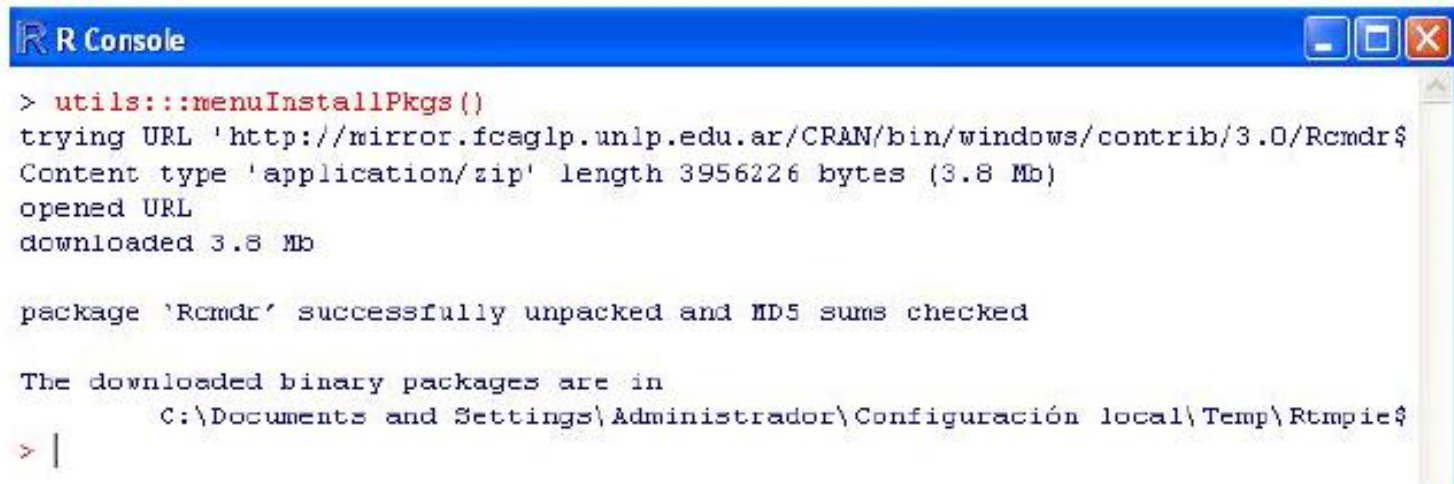
Instalación



R COMMANDER

Instalación

Entonces se mostrará el siguiente mensaje en la Consola y el cursor quedará en espera.



```
R Console
> utils::menuInstallPkgs()
trying URL 'http://mirror.fcaglp.unlp.edu.ar/CRAN/bin/windows/contrib/3.0/Rcmdr$
Content type 'application/zip' length 3956226 bytes (3.8 Mb)
opened URL
downloaded 3.8 Mb

package 'Rcmdr' successfully unpacked and MD5 sums checked

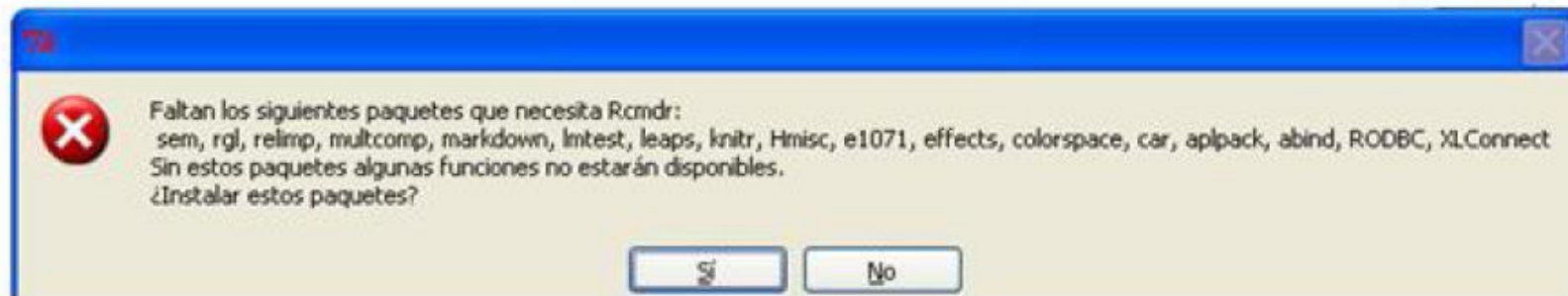
The downloaded binary packages are in
  C:\Documents and Settings\Administrador\Configuración local\Temp\Rtmpie$
> |
```

Ahí escribimos `library(Rcmdr)`. Al digitar esta instrucción por primera vez, aparecerá un mensaje que nos advierte que `Rcmdr` necesita instalar otros paquetes. Para ello, hacemos clic sobre el botón “Sí”.

R COMMANDER

Instalación

```
> library(Rcmdr)
```



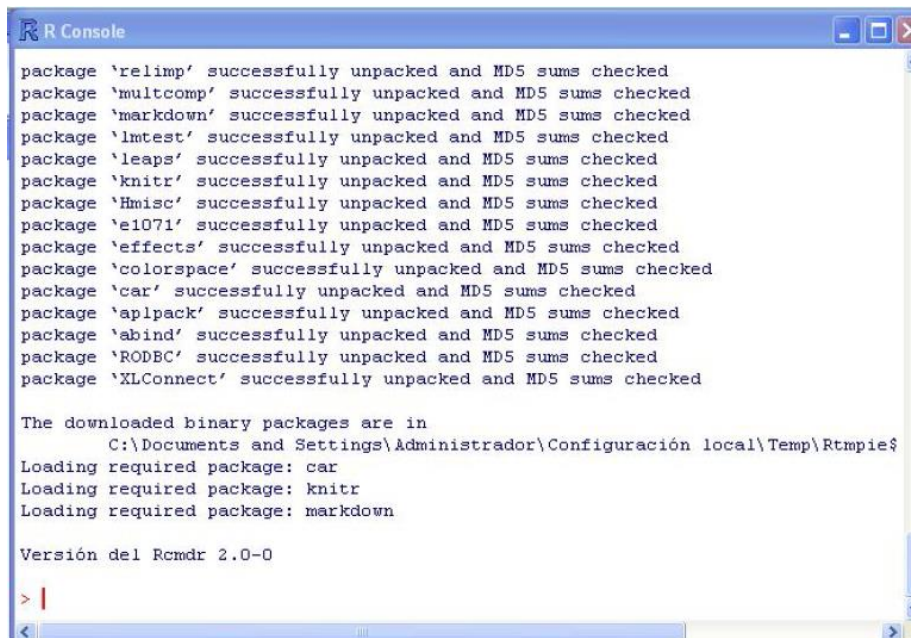
Entonces aparecerá la ventana de instalación de los paquetes restantes. Hacemos clic sobre **OK**.



R COMMANDER

Instalación

R Commander nos notificará el éxito de la instalación a través de la Consola.



```
R Console
package 'relimp' successfully unpacked and MD5 sums checked
package 'multcomp' successfully unpacked and MD5 sums checked
package 'markdown' successfully unpacked and MD5 sums checked
package 'lme4' successfully unpacked and MD5 sums checked
package 'leaps' successfully unpacked and MD5 sums checked
package 'knitr' successfully unpacked and MD5 sums checked
package 'Hmisc' successfully unpacked and MD5 sums checked
package 'e1071' successfully unpacked and MD5 sums checked
package 'effects' successfully unpacked and MD5 sums checked
package 'colorspace' successfully unpacked and MD5 sums checked
package 'car' successfully unpacked and MD5 sums checked
package 'aplpack' successfully unpacked and MD5 sums checked
package 'abind' successfully unpacked and MD5 sums checked
package 'RODBC' successfully unpacked and MD5 sums checked
package 'XLConnect' successfully unpacked and MD5 sums checked

The downloaded binary packages are in
  C:\Documents and Settings\Administrador\Configuración local\Temp\Rtmpie$
Loading required package: car
Loading required package: knitr
Loading required package: markdown

Versión del Rcmdr 2.0-0

> |
```

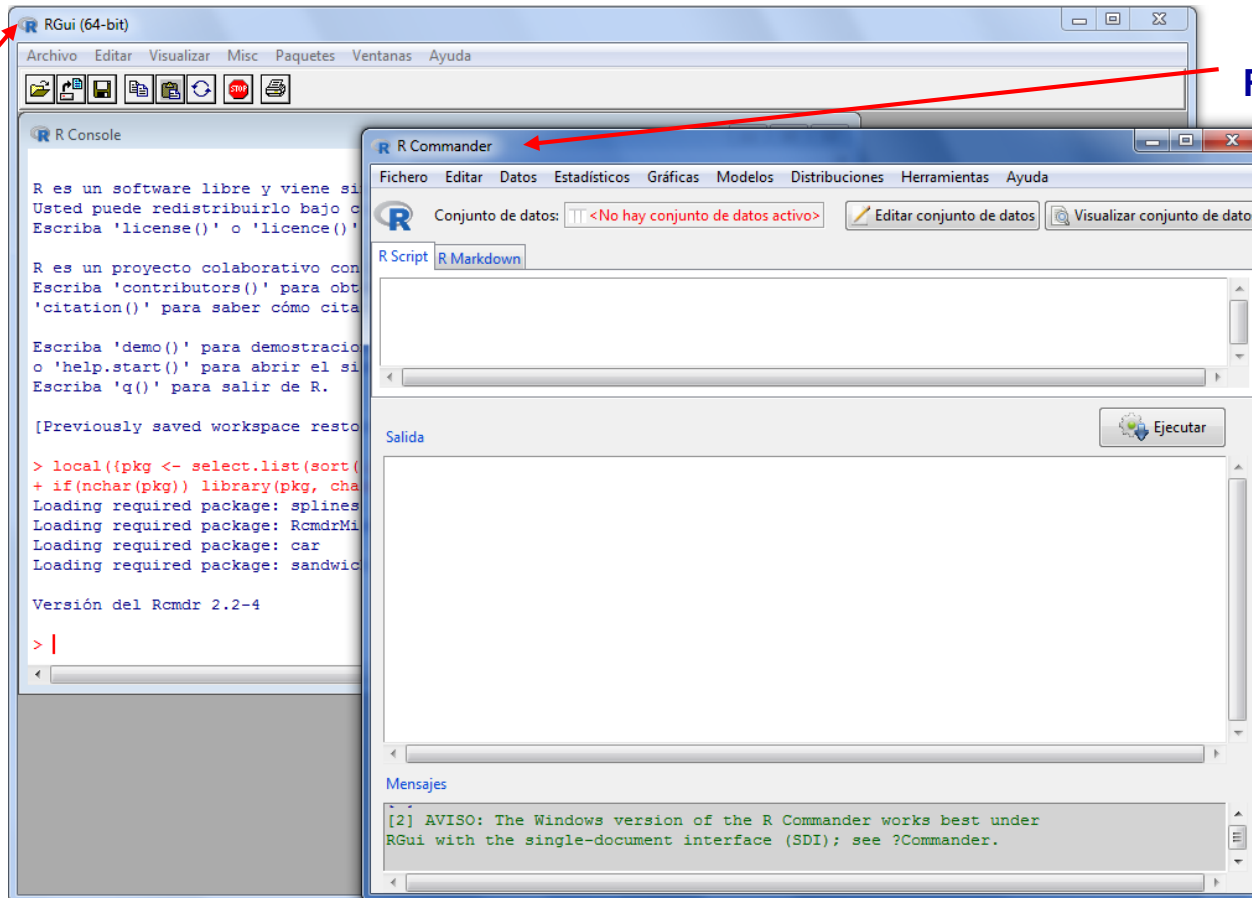
En las siguientes sesiones podemos entrar a R Commander digitando `library(Rcmdr)` en la Consola de R, o podemos cargarlo desde el menú **Packages -> Load package...**

Y se abrirá la ventana de R Commander.

R COMMANDER

Instalación

Ventana
del R

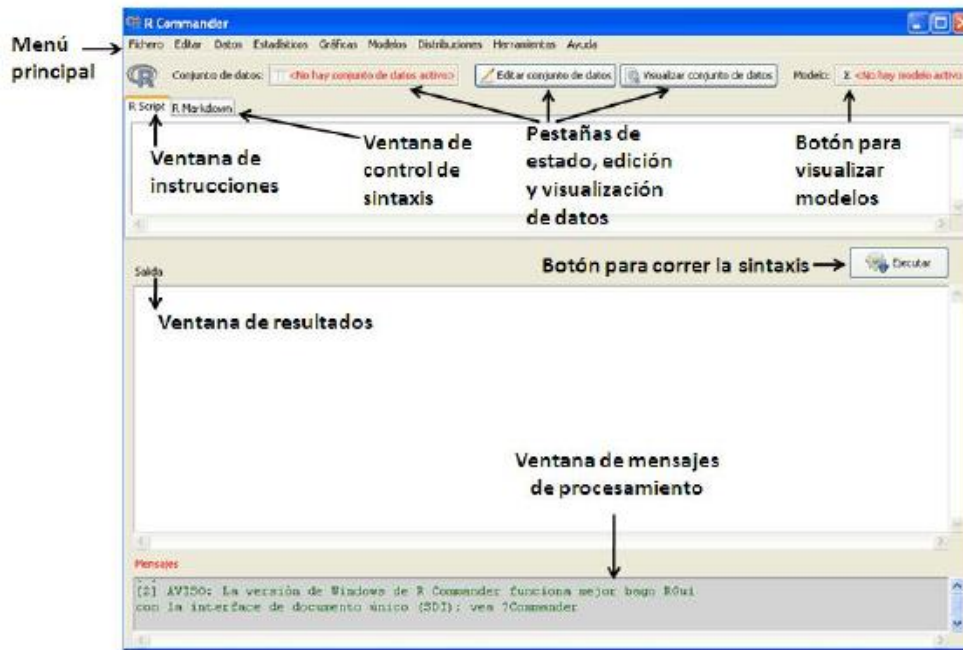


Ventana del
R Commander

R COMMANDER

Descripción del ambiente de trabajo

En el siguiente gráfico se muestran las ventanas de trabajo principales en **R Commander**



Contiene los menús: Fichero, Editar, Datos, Estadísticos, Gráficas, Modelos, Distribuciones, Herramientas y Ayuda.

Desde estos menús, enviaremos instrucciones a R para activar un conjunto de datos, realizar transformaciones de los mismos o de sus variables, realizar análisis estadísticos y gráficos con ellos, guardar los resultados o solicitar ayuda para algún procedimiento de análisis de datos.

R COMMANDER

Menús disponibles

En el Menú de la ventana de **Rcommander** aparece el acceso a las siguientes utilidades:

- ▶ **Fichero**: Abrir, Guardar instrucciones, Guardar resultados, Guardar entorno de trabajo de R, Salir.
- ▶ **Editar**: Limpiar ventana, Cortar, Copiar, Pegar, Borrar, Buscar, Seleccionar todo.
- ▶ **Datos**: Nuevo conjunto de datos, Cargar conjunto de datos, Importar datos, Conjunto de datos en paquetes, Conjunto de datos activo, Modificar variables del conjunto de datos activo.
- ▶ **Estadísticos**: Resúmenes, Tablas de contingencia, Medias, Proporciones, Varianzas, Tests no paramétricos, Análisis dimensional, Ajuste de modelos.
- ▶ **Gráficas**: diversos gráficos univariantes, bivariantes y 3D, Guardar gráfico en archivo.

R COMMANDER

Menús disponibles

Más sobre los menús disponibles en Rcommander,

- ▶ **Modelos:** Selecciona el modelo activo, Resumir el modelo, Añadir las estadísticas de las observaciones a los datos, Intervalos de confianza, AIC, BIC, Test de hipótesis, Diagnósticos numéricos, Gráficas.
- ▶ **Distribuciones:** Distribuciones continuas, Distribuciones discretas, Visualizar distribuciones.
- ▶ **Ts-Data y Ts-Models:** análisis de datos y gráficas series temporales.
- ▶ **Herramientas:** Cargar paquete(s), Cargar plugins de Rcmdr, Opciones.
- ▶ **Ayuda:** Ayuda de R Commander, Introducción de R Commander, Ayuda conjunto de datos activo (si existe), Información sobre Rcmdr.

Comunicación constante con la Escuela del INEI

Correo de la Dirección Técnica de la ENEI

Sr. Eduardo Villa Morocho (Eduardo.villa@inei.gob.pe)

Coordinación Académica

Sra. María Elena Quirós Cubillas (Maria.Quiros@inei.gob.pe)

Correo de la Escuela del INEI

enei@inei.gob.pe

Área de Educación Virtual

Sr. Gonzalo Anchante (gonzalo.anchante@inei.gob.pe)

