Curso de Introducción a la sintaxis de R mediante RStudio
Sesión 1: Introducción a RStudio

Servei d'Estadística Aplicada
Universitat Autònoma de Barcelona

17, 19, 24 y 26 de enero, 2023

Contenido de la Sesión

Motivación

R

- Entorno interactivo para el análisis estadístico y gráfico.
- Agilidad para implementar y compartir nuevas técnicas.
- Permite generar gráficos de alta calidad.
- Múltiples recursos de soporte y ayuda.
- Elevada presencia en determinados ámbitos de investigación.
- Libre distribución (sin garantías).
- Multiplataforma (Windows, Linux, Mac).

El lenguaje R

- R es un lenguaje de programación de alto nivel con funciones orientadas a objetos.
- Es un lenguaje interpretado (como Java) y no compilado (como C, Fortran, Pascal, etc.), los comandos escritos con el teclado son ejecutados directamente sin necesidad de construir ejecutables.
- La sintaxis es simple e intuitiva.
- La estructura y facilidad de uso de R permite implementar funciones y rutinas propias a medida que aparecen nuevas necesidades.
- Sinergia con otros lenguajes: C++, Python, Latex.
- Existen programas complementarios tipo GUI (graphical user interface: R Commander, DeduceR, Rattle, RExcel) o IDE (integrated development environment: RStudio, TinnR, emacs, ...).

Instalación de R

- Para instalar R es necesario descargar los ficheros necesarios de la página web del proyecto R: `http://cran.r-project.org`.
- Actualmente (enero 2023), la versión más reciente de R para el entorno Windows es la R-4.2.2.
- Una vez bajado este fichero, hay que ejecutarlo y seguir las instrucciones del programa de instalación.
- Posteriormente se pueden instalar librerías y paquetes complementarios (actualmente, más de 10000 paquetes disponibles).
- Una vez instalado podremos abrirlo desde el listado de programas en el menú inicio. Al abrirse aparece un mensaje de bienvenida indicando la versión instalada.
- No es necesario visitar constantemente la página del proyecto para comprobar si existen actualizaciones disponibles.
- Si fuera necesario instalar una nueva versión, no es necesario desinstalar las versiones antiguas, se pueden tener instaladas varias versiones del programa.

Inicio de la sesión

- La ventana principal corresponde a la consola de R (ventana de comandos) en la cual se pueden escribir comandos y en la que aparecerán los resultados de los análisis o tareas.
- Los comandos se escriben en el margen izquierdo tras el símbolo ">" o PROMPT.
- Para ejecutar un comando escrito utilizamos la tecla de "intro".
- Algunos comandos iniciales de interés son:
  - `demo()` permite realizar demostraciones del uso de R.
  - `license()`: R es un software libre y viene sin GARANTÍA ALGUNA, aunque es posible redistribuirlo bajo ciertas circunstancias.
  - `citation()`: referencia para la citación de R o paquetes en publicaciones científicas.

Inicio de la sesión

- Es recomendable indicar el directorio de trabajo asociado a cada proyecto.
- Se puede cambiar de directorio desde el menú **Archivo** → **Cambiar dir...** o con el comando `setwd("C:/Directorio")`.
- A partir de ahora, si se omite la ruta, R entenderá que el fichero referido se encuentra en el directorio de trabajo.
- El comando `getwd()` permite recuperar qué directorio es el directorio de trabajo actual.

Inicio de la sesión

- En la consola, las flechas ↑ y ↓ permiten recuperar comandos ejecutados anteriormente.
- Para separar instrucciones en una misma línea se emplea ";".
- Si un comando contiene errores, aparecerán los mensajes pertinentes.
- Si un comando está incompleto, el software quedará a la espera. La tecla ESC restablece el PROMPT.
- La combinación Ctrl+C interrumpe la edición o ejecución en curso.
- La combinación Ctrl+Z elimina la ejecución en curso.
- `q()` es el comando para salir de R (alternativamente, cerrar la ventana).

R como calculadora

- Las órdenes elementales de sintaxis consisten en expresiones o en asignaciones.
- Si una orden consiste en una expresión o cálculo, esta se evalúa, se imprime y su valor se pierde. Así, el uso más básico de R sería como calculadora científica. Por ejemplo:

```
> 2+4
[1] 6
```

- Ejemplo de operadores implementados en R: +, -, * , /, ^ o **, %% (módulo), %/% (división entera).
- Ejemplos de funciones implementadas en R: `log(x)`, `exp(x)`, `log10(x)`, `sqrt(x)`, `factorial(x)`, `floor(x)`, `round(x,digits=0)`, `abs(x)`, ...

Sintaxis elemental en R

- Al contrario de las evaluaciones, una asignación, evalúa una expresión, no la imprime y guarda su valor en un objeto.
- Las asignaciones se realizan mediante los signos "<-", "->" o "=":

```
> a <- 6*4+3
> sqrt(a) -> b
> B=log(b)
```

- Para visualizar el contenido de un objeto el comando corresponde al nombre del objeto (o bien poniendo la asignación entre paréntesis):

```
> a
[1] 27
> (sqrt(a) -> b)
[1] 5.196152
```

Fin de la sesión

- La colección de objetos almacenados en cada momento se denomina espacio de trabajo (workspace).
- Desde el menú **Archivo** → **Guardar área de trabajo** o con el comando `save.image()` se puede guardar una imagen del espacio de trabajo realizado hasta ese momento.
- El área de trabajo (Workspace) corresponde a todos los objetos creados durante la sesión. La extensión para estos archivos es .RData.
- Podemos retomar el área de trabajo en posteriores sesiones abriendo desde el menú **Archivo** → **Cargar área de trabajo** la imagen anteriormente guardada.
- También es posible guardar todos los comandos ejecutados de la sesión desde el menú **Archivo** → **Guardar Histórico**.

RStudio

- RStudio es un entorno de desarrollo integrado (IDE) para R. Incluye una consola, editor de sintaxis que apoya la ejecución de código, así como herramientas para el trazado, la depuración y la gestión del espacio de trabajo.
- Para instalar RStudio, descargar el programa desde la página web: https://www.rstudio.com.
- Actualmente (enero 2022), la versión más reciente de RStudio para el entorno Windows es 2022.12.0.
- Una vez bajado este fichero, hay que ejecutarlo y seguir las instrucciones del programa de instalación.
- Es necesario tener instalado R para poder utilizar RStudio.

La ayuda de R

- R dispone de una ayuda muy completa sobre todas las funciones, procedimientos y elementos que configuran el lenguaje.
- Además de las opciones de menú propias de R, desde la ventana de comandos se puede acceder a información específica sobre las funciones de R con el comando `help()` o mediante `?`.

```
> help(log)
> ?log
```

- Para acceder al código de una función de R basta con introducir el nombre de la función sin los argumentos.

```
> log
```

- Cuando no sepamos el nombre exacto del comando o de la función que necesitamos, pero sí el tema sobre el que queremos ayuda (en inglés), podemos utilizar la función `help.search()`.

```
> help.search("data input")
```

y posteriormente:

```
> ?read.table
```

La ayuda de R

- Es posible acceder a manuales desde la página web de CRAN: `http://cran.r-project.org/`.
- Algunos de los manuales disponibles son:
  - An Introduction to R: introduce el lenguaje, y presenta R como herramienta para el análisis estadístico y gráfico.
  - R Reference: contiene todos los ficheros de ayuda de la versión estándar de R junto con los de las librerías recomendadas.
  - R Data Import/Export: describe las utilidades para importar y exportar datos en R incluyendo librerías disponibles en CRAN.
  - R Language Definition: documenta el lenguaje R. De gran utilidad cuando se van a programar funciones.
  - Writing R Extensions: cubre cómo crear nuestras propias librerías, escribir los archivos de ayuda y conectar con lenguajes externos (C, C++, Fortran, etc.).
  - R Internals: es una guía para las estructuras internas de R.
  - R Installation and Administration.

Soporte complementario de R

- Desde el propio programa, a través del menú Ayuda se puede acceder a las FAQ.
- Bibliografía de R: En la página web del proyecto existe "Contributed Documentation" que además es de libre distribución.
  - John M. Chambers (2008), Software for Data Analysis: Programming with R. Springer, New York.
  - Peter Dalgaard (2008), Introductory Statistics with R, 2nd edition. Springer.
  - Michael J. Crawley (2007), The R book. John Wiley and Sons, Ltd.
- Congresos: useR y R-Hispano.
- The R Journal.
- R-bloggers.

Programación orientada a Objetos

- R es mucho más que una calculadora científica en la que aplicar unos comandos o instrucciones.
- R es un lenguaje Orientado a Objetos (Object-oriented programming, OOP). Bajo este término se esconde la simplicidad y flexibilidad de R.
- R almacena los resultados en objetos para ser observados o analizados posteriormente, produciendo unas salidas mínimas.
- El usuario puede extraer sólo aquella parte de los resultados que le interesa.
- Otros programas como SAS o SPSS proporcionan sin solicitarlo una salida copiosa para cualquier análisis.

Programación orientada a Objetos

- Cuando en el mundo real nos referimos a un objeto significa que hablamos de algo más o menos abstracto que puede ser cualquier cosa.
- En OOP la generalización (o definición) de un objeto se llama Clase.
- Todo objeto tiene atributos o propiedades. Se puede acceder a las propiedades de un objeto mediante distintas funciones.
- Se pueden asociar acciones o métodos a los objetos. Cualquier proceso que implica una acción o pauta de comportamiento por parte de un objeto se define en su clase para que luego pueda manifestarse en cualquiera de sus objetos.

Programación orientada a Objetos

Tipos de objetos

- Vectores: son el tipo básico de objeto en R.
- Matricies: vectores indexados por dos o más índices. Generalizaciones multidimensionales de los vectores.
- Factores: sirven para representar datos categóricos.
- Listas: son una forma generalizada de vector cuyos elementos no tienen por qué ser del mismo tipo y a menudo son a su vez vectores o listas.
- Dataframes: las bases de datos (o data frames) son estructuras similares a una matriz, en que cada columna puede ser de un tipo distinto a las otras.
- Funciones: son también objetos de R que pueden almacenarse en el espacio de trabajo.

Programación orientada a Objetos

- Consecuente con sus orígenes en UNIX, R distingue entre mayúsculas y minúsculas, de tal modo que "A" y "a" son símbolos distintos y se referirán a objetos distintos.
- Los nombres de los objetos pueden contener sólo letras mayúsculas o minúsculas (sin acentos), junto con números y puntos (sin blancos, _ , %, $, etc.).
- Los corchetes o dobles corchetes se utilizan para seleccionar partes de un objeto así como el dólar $ o la arroba .
- 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()`.
- Es posible eliminar objetos con el comando `rm()`.

Programación orientada a Objetos

Vectores

- La función principal para definir un vector es a través de sus componentes, con la función concatenar: `c()`.

```
> v <- c(1,2,3,b)
> v
[1] 1.000000 2.000000 3.000000 5.196152
```

- La función `length()` muestra el número de componentes de un vector y `NA` identfica los elementos faltantes (missings).
- Para referirnos a la componente enésima del vector "v" escribiremos `v[n]`.

```
> v[4]
[1] 5.196152
```

Vectores

- Se pueden modificar los elementos de vectores, ampliarlos o borrarlos:

```
> v[2] <- 22
> v[5] <- a
> v
[1]  1.000000 22.000000  3.000000  5.196152 27.000000
```

- Lo que convierte a R en una potente herramienta de trabajo es que está diseñado de forma que la mayoría de operaciones y de funciones están definidas con carácter vectorial, es decir para operar componente a componente.
- Es conveniente explotar esta posibilidad ya que agiliza mucho el tiempo de computación.
- Las funciones estadísticas como la suma, el producto o la media aritmética devuelven un solo valor para cada vector.

Matrices

- La función `matrix(vector,...)` organiza los componentes de un vector en forma de matriz.
- La opción nrow/ncol permite especificar el número de filas/columnas. El número de filas/columnas se determina mediante el redondeo por exceso de la longitud del vector entre el número de filas/columnas:

```
> A <- matrix(1:9,ncol=3)
```

- La función `dim(matriz)` muestra el número de filas y columnas de la matriz:

```
> dim(A)
[1] 3 3
```

Matrices

- Los operadores "+", "-", "*" y "/" se pueden aplicar a dos o más matrices de la misma dimensión. La ejecución se realiza componente a componente (el resultado es otra matriz de la misma dimensión):

```
> A*A
     [,1] [,2] [,3]
[1,]    1   16   49
[2,]    4   25   64
[3,]    9   36   81
```

- El producto matricial se denota por `%*%`:

```
> A%*%A
     [,1] [,2] [,3]
[1,]   30   66  102
[2,]   36   81  126
[3,]   42   96  150
```

Matrices

- Con `A[i,j]`, `A[i,]` y `A[,j]` nos referimos a un elemento, a una fila o a una columna de la matriz A, respectivamente. Si se utiliza un vector como subíndice obtenemos la submatriz correspondiente:

```
> A[2,3];A[1,]
[1] 8
[1] 1 4 7
```

- Es posible poner nombres tanto a las filas como a las columnas de una matriz y utilizarlos para extraer elementos de la matriz:

```
> rownames(A) <- c("Fila 1","Fila 2","Fila 3")
> colnames(A) <- c("Col 1","Col 2","Col 3")
> A["Fila 1","Col 3"]
[1] 7
```

Paquetes y librerías

- Múltiples usuarios desarrollan técnicas y las comparten generando paquetes de funciones adicionales.
- Para acceder a tales paquetes, es necesario descargarlos de la web del proyecto, aunque esto se realiza directamente desde `RStudio`. Para instalarlos es necesario seguir los pasos:
  - Desde el menú **Tools** $\rightarrow$ **Install Packages...**
  - Seleccionar el CRAN mirror (recomendable 0-Cloud).
  - Seleccionar el paquete deseado. Si el paquete necesita otros paquetes, se instalarán automáticamente.
  - Alternativamente, utilizar la función: `install.packages("nombrepaquete")`.
  - La instalación no implica que los paquetes ya puedan ser utilizados. Es necesario cargarlos desde la ventana **Packages** o con la función: `library("nombrepaquete")`.

Scripts

- Para operaciones que requieran varias instrucciones consecutivas, resulta especialmente útil trabajar con un fichero de comandos editable (script).
- R proporciona por defecto la posibilidad de trabajar con scripts como ventanas del propio programa: **File** → **New File** → **R Script**.
- En las ventanas script se pueden entrar varios comandos, separados por líneas o por ";". Pueden ser ejecutados conjuntamente o por separado desde la pestaña **Run line or selection** o bien con Crtl + R (en Windows).
- El signo # como prefijo indica la introducción de un comentario.
- Los scripts se pueden guardar y utilizar posteriormente. La extensión de este tipo de ficheros es .R.
- Mediante la función `source()` se puede cargar un script de R entero (también desde el menú **Code** → **Source File...**):

```
> source("F:/Curso R/script.R")
```

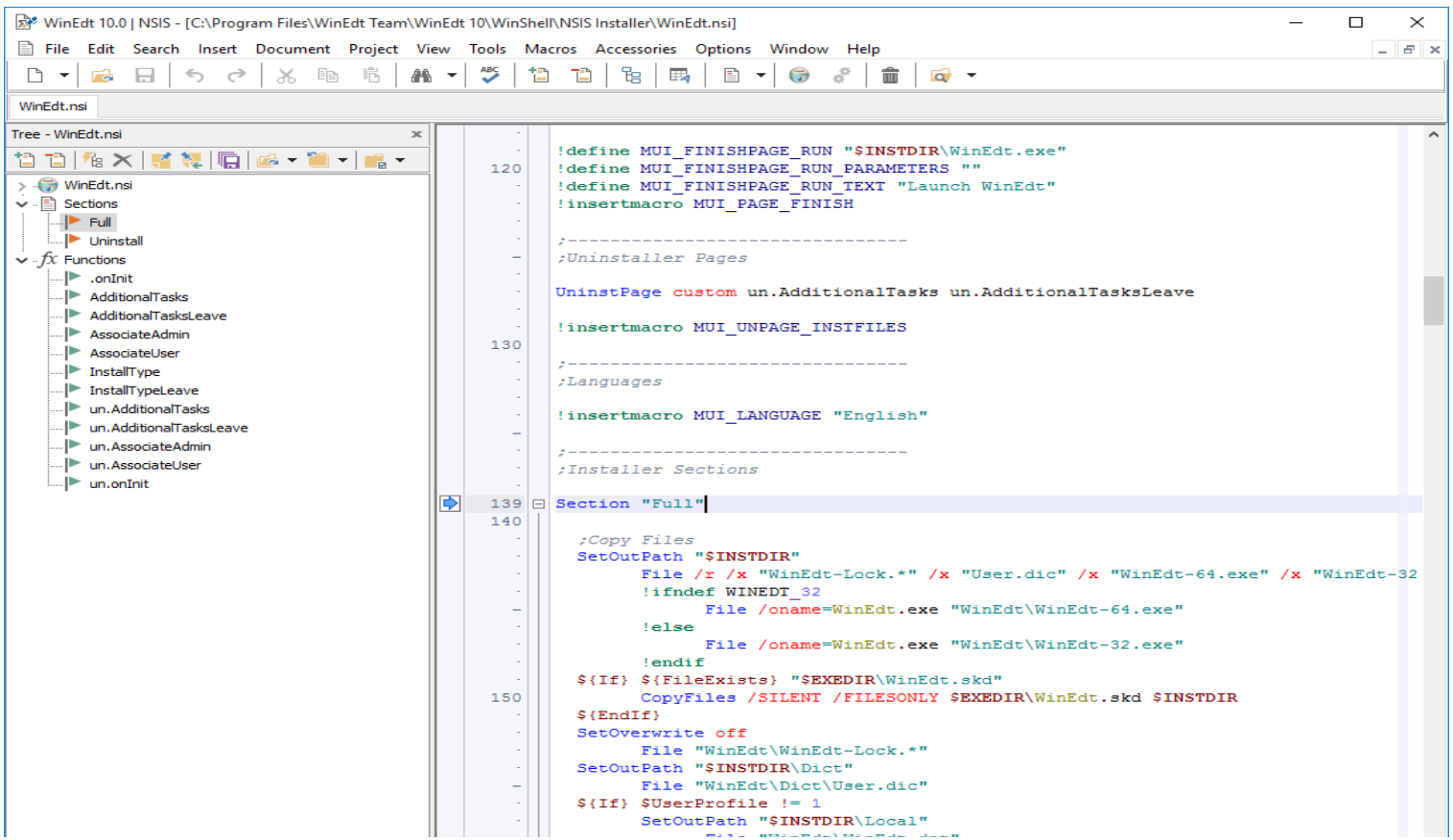
Scripts

- Existen diferentes editores que pueden facilitar el trabajo con R como RStudio o TinnR. Estos editores permiten abrir y editar diferentes scripts y ejecutarlos en R.
  - La ventaja de estos editores sobre la ventana de los scripts en R es que ofrecen una serie de opciones no existentes en R.
- También existen varias Interfaces Gráficas de Usuario (GUI en inglés) que permiten trabajar mediante menús sin necesidad de conocer el lenguaje de programación de R como R Commander o Deducer.
  - Estos programas permiten trabajar en un entorno más amigable para el usuario, no obstante, son mucho menos flexibles ya que limitan el uso de opciones de muchas funciones.
- Finalmente, Shiny es una interfaz de presentación de resultados que permite construir aplicaciones web interactivas.

Otras interfaces de usuario IDE

WinEdt

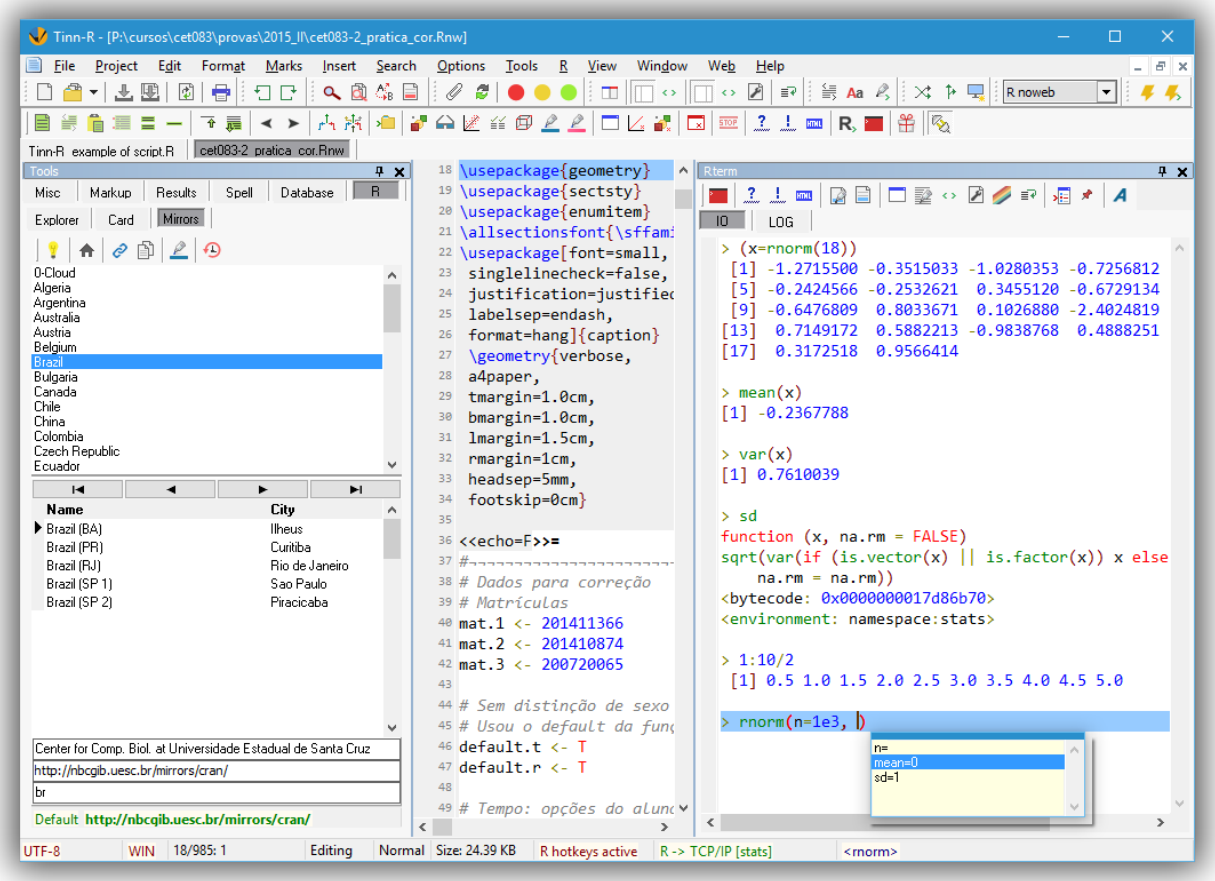
- Editor para scripts bajo licencia, disponible en el sistema operativo Windows `www.winedt.com`.
- Clásicamente empleado para LaTeX.



Otras interfaces de usuario IDE

Tinn-R

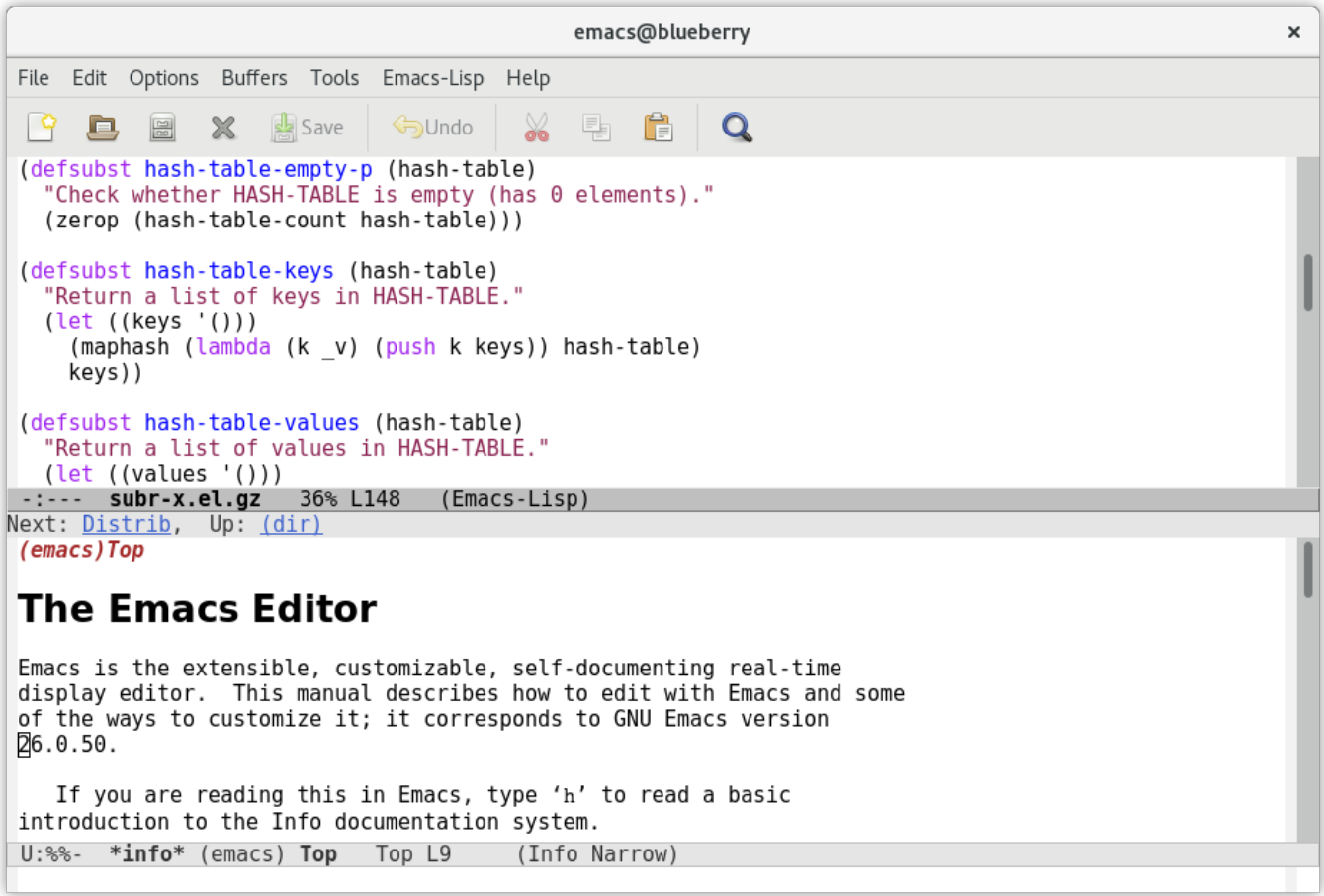
- Editor de libre distribución, similar a WinEdt. Se puede descargar desde `https://sourceforge.net/projects/tinn-r`.



Otras interfaces de usuario IDE

Emacs

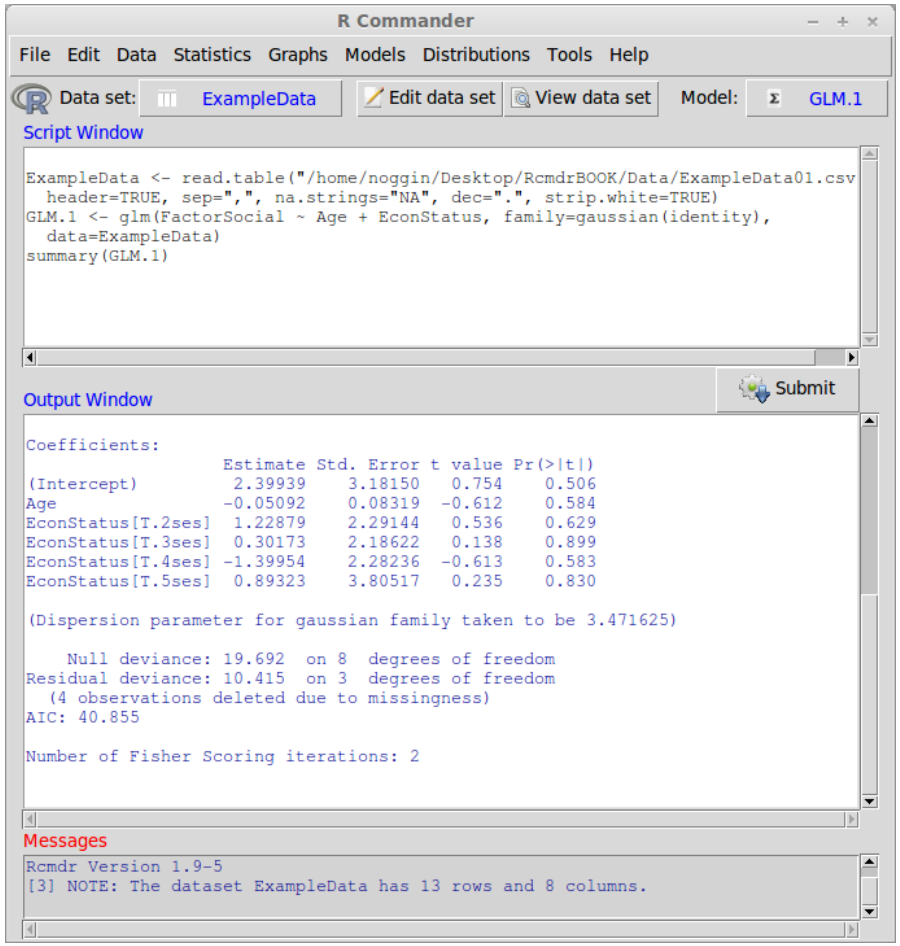
- Editor de libre distribución muy potente. Puede descargarse desde `www.gnu.org/software/emacs`.



Interfaces gráficas GUI

R Commander

- GUI muy potente y abierto a personalizaciones.



Interfaces gráficas GUI

R Commander

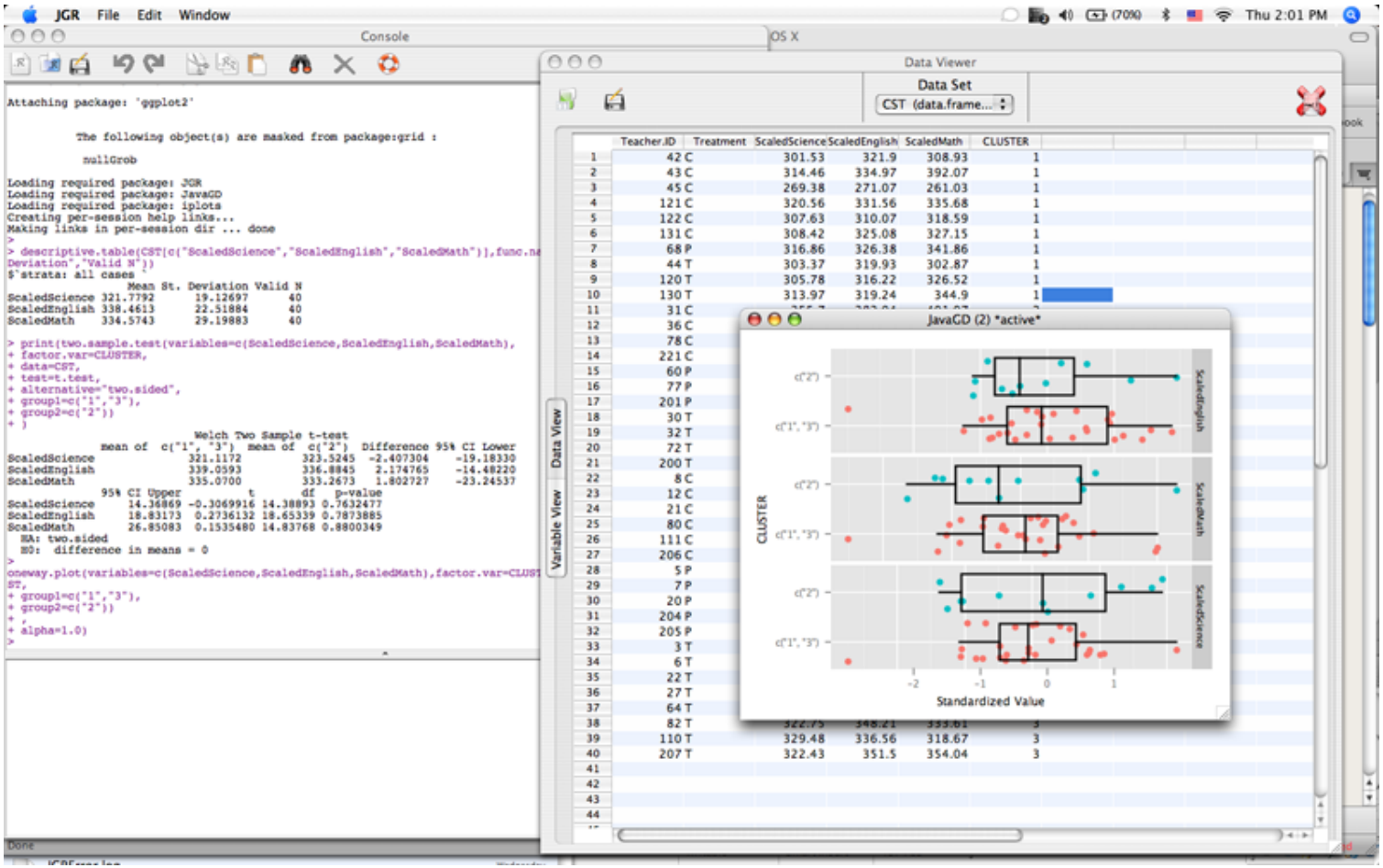
- Permite acceder a muchas capacidades del entorno estadístico R sin que el usuario tenga que conocer el lenguaje de programación propio de este entorno.
- Para abrir R Commander debemos ejecutar la siguiente instrucción desde el programa R:

```
> install.packages("Rcmdr")
> library("Rcmdr")
```

Interfaces gráficas GUI

DeduceR

- GUI muy visual y ágil.



Interfaces gráficas GUI

DeduceR

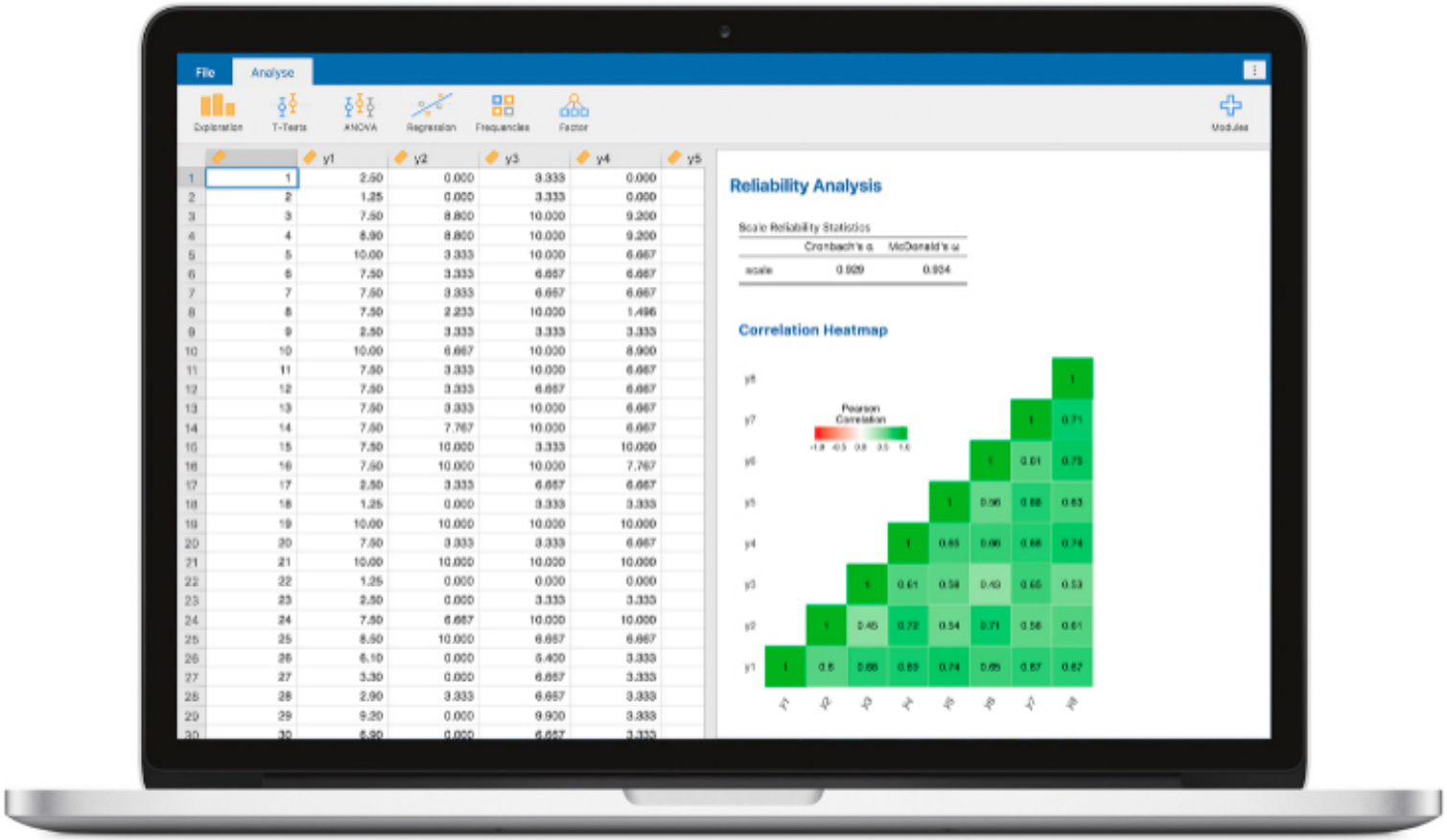
- Está diseñado para ser usado con la consola de R basada en Java: JGR.
- Para abrir Deducer debemos ejecutar la siguiente instrucción desde el programa R:

```
> install.packages(c("JGR","Deducer"))
> library(JGR)
> JGR()
```

Interfaces gráficas GUI

jamovi

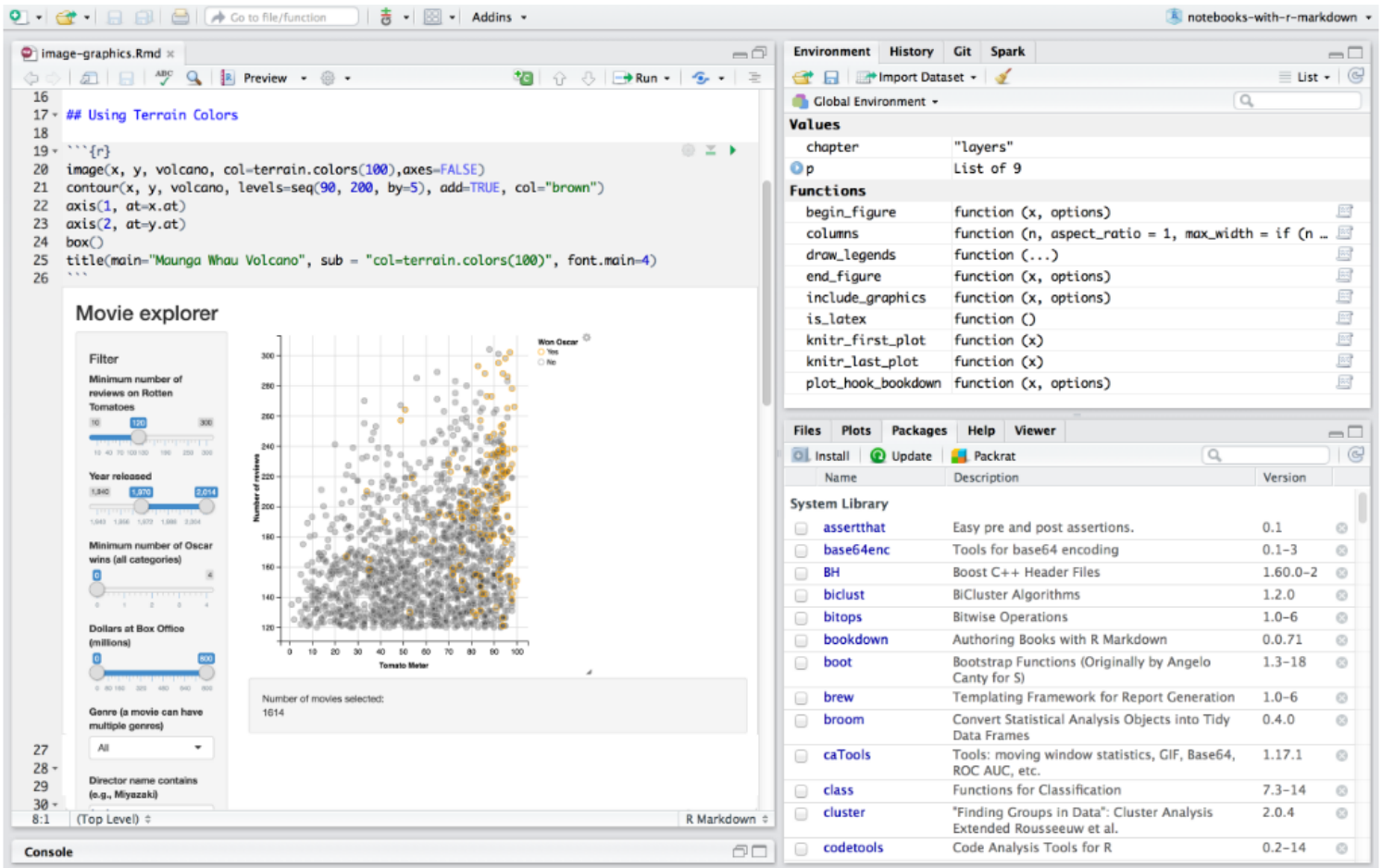
- No necesita instarlar R, una alternativa a SPSS. Puede descargarse desde www.jamovi.org.



Interfaces de presentación de resultados

Shiny

- Permite la construcción de aplicaciones web interactivas: https://shiny.rstudio.com.



Curso de Introducción a la sintaxis de R mediante RStudio
Sesión 2: Manejo de datos

Servei d'Estadística Aplicada
Universitat Autònoma de Barcelona

17, 19, 24 y 26 de enero, 2023

Contenido de la Sesión

Características de los objetos en R

- R es un lenguaje orientado a objetos: las variables, datos, funciones, resultados, etc. se guardan en la memoria activa del ordenador en forma de objetos con un nombre específico.
- Se pueden modificar o manipular estos objetos con operadores (aritméticos, lógicos y comparativos) y funciones (que a su vez son objetos).
- Las estructuras de datos más usuales son: variables, vectores, matrices, factores, variables indexadas, cadenas de caracteres, listas y dataframes.
- Existen otros como series temporales y es posible crear nuevas estructuras de datos.
- Recordar el funcionamiento de `objects()`, `ls()` y `rm()`.

Características de los objetos en R

- Los objetos están compuestos de elementos. Los elementos más simples pueden ser de los siguientes tipos:

  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. Pueden ser TRUE o FALSE.

Ejemplo

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

Datos especiales: NA's

- En algunos casos las componentes de un objeto pueden no ser completamente conocidas.
- Cuando un elemento sqrt(-17+0i)es "not available" se le asigna el valor especial NA.
- En general una operación con elementos NA resulta NA, a no ser que mediante una opción de la función, se puedan 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.

Datos especiales: NA's

Ejemplo

```
> x<-NA
> x+1
[1] NA
> y<-c(x,3,5,x)
> mean(y)
[1] NA
> mean(y,na.rm=TRUE)
[1] 4
```

Datos especiales: Inf y NaN's

- En la mayoría de los casos, no hay que preocuparse de 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.
- Para trabajar con números complejos, hay que 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, del inglés "not a number").

Datos especiales: Inf y NaN's

Ejemplo

```
> sqrt(-17)
[1] NaN
> sqrt(-17+0i)
[1] 0+4.123106i
> x<-5/0
> exp(-x)
[1] 0
> exp(x)-exp(x)
[1] NaN
```

Clasificación de los objetos en R

- 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, las matrices o las series temporales.

  Recursivos Pueden combinar una colección de otros objetos de diferente tipo (o modo), como son los dataframes y las listas.

- Existen otras estructuras recursivas, por ejemplo:

  Modo function Está formado por las funciones que constituyen R, unidas a las funciones escritas por cada usuario.

  Modo expression Corresponde a una parte avanzada de R.

- Además, es posible crear nuevas estructuras.
- La función `str(objeto)` devuelte el tipo (o modo) y el valor de un objeto.

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 se designa el tipo básico de sus elementos básicos.
- 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 objeto.

Ejemplo

```
> x<-c(1,3)
> mode(x);length(x)
[1] "numeric"
[1] 2
```

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	Tipo
mode, length	Todos
storage.mode	Todos
names	Vectores y listas
dim, 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()
character(0)
```

- 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

- 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, " ", etc. mediante la función que indica el tipo de dato (que corresponde a su modo) y entre paréntesis el numero de elementos a crear.

Ejemplo

```
> x1<-logical(4)
> #Inicializa un vector de longitud 4
> 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
```

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.
- Las comparaciones que dan un resultado lógico son: "<", "<=", ">", ">=", "==" y "!=".
- 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:

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

Operadores lógicos

Ejemplo

```
> a <- c(5,3,6,8); b <- c(4,9,3,8)
> (x <- a<=5); (y <- b<=5)
[1]  TRUE  TRUE FALSE FALSE
[1]  TRUE FALSE  TRUE FALSE
> x&y; x|y
[1]  TRUE FALSE FALSE FALSE
[1]  TRUE  TRUE  TRUE FALSE
> xor(x,y); any(x)
[1] FALSE  TRUE  TRUE FALSE
[1] TRUE
> all(x)
[1] 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.
- Existen dos posibilidades:
  - Comprobar el tipo de datos utilizando funciones `is.tipo()`, que responde con un valor lógico.
  - Forzar al tipo de datos deseados **coercionando**, para lo cual se pueden utilizar funciones del tipo `as.tipo()`, 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. Para ello existe la función `is.na()`, que responde con un valor lógico: TRUE para los elementos del vector que son NA.
- De manera similar, se puede utilizar `is.finite()`, `is.nan()`, etc.

Coerción de tipos

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

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

Coerción de tipos

Ejemplo

```
> x<-c(1:10)
> is.numeric(x)
[1] TRUE
> is.vector(x)
[1] TRUE
> is.complex(x)
[1] FALSE
> is.character(x)
[1] FALSE
> x<-as.character(x)
> #fuerza el vector x a tomar los valores
> #("1" "2" "3" "4" "5" "6" "7" "8" "9" "10")
```

Vectores de índices

Características de los vectores

- Extienden los tipos básicos.
- Concatenación de elementos.
- Todos los elementos del vector han de ser del mismo tipo.
- Se crean con la función `c()`.
- Se puede usar vector[5] para acceder al quinto elemento.

Ejemplo

```
> nombre<-c("Luis","María")
> edad<-c(23,24)
> varon<-c(TRUE,FALSE);adult<-edad>18
> estatura<-c(1.77,1.64)
> estatura[2]
[1] 1.64
```

Variables categóricas o factores

- Un factor es un vector utilizado para especificar una clasificación discreta de los elementos de otro vector de igual longitud.
- En R existen dos tipos de factores:
  - **Nominales** No ordenados. No existe jerarquía entre ellos (p.e., colores).
  - **Ordinales** Ordenados. Existe jerarquía entre ellos (p.e., grupos de edad).
- Se pueden crear a partir de un vector numérico con las funciones `as.factor()`, `as.ordered()` o con la función `gl()`.
- También a partir de un vector de caracteres utilizando `factor()`.
- Las etiquetas se asignan con `levels()`.

Ejemplo

```
> f<-as.factor(c(1,2,3,1,2,1,1,3,2)) #Factor Nominal
> levels(f)<-c("Bajo","Medio","Alto")
> ford<-as.ordered(f) #Factor Ordinal
```

Vectores de caracteres

- 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 para gráficos.
- 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"),1:9,sep="")
> labs
[1] "X1" "Y2" "X3" "Y4" "X5" "Y6" "X7" "Y8" "X9"
```

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).
- 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 el vector resultante.

Generación de secuencias regulares

Comprobar las siguientes expresiones:

```
> pi:1
[1] 3.141593 2.141593 1.141593
> seq(0,1,length=10)
 [1] 0.0000000 0.1111111 0.2222222 0.3333333 0.4444444
 [6] 0.5555556 0.6666667 0.7777778 0.8888889 1.0000000
> seq(3)
[1] 1 2 3
> seq(1,5,by=0.5)
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
> -1:1/0
[1] -Inf  NaN  Inf
```

Generación de secuencias regulares

Comprobar las siguientes expresiones:

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

```
> rep(1:4,c(2,2))
> #error, tamaños no coinciden
```

```
> sequence(c(2,3))
[1] 1 2 1 2 3
```

Extrayendo muestras de un vector

- R 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.
- Para que las muestras extraidas sean siempre iguales, es necesario definir una semilla, mediante la instrucción `set.seed`

Ejemplo

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

Vectores de índices

- R permite seleccionar subconjuntos de elementos de un vector añadiendo al nombre del vector un vector índice entre corchetes [].
- El vector índice puede ser de tres tipos distintos:
  1. Un vector lógico. El vector índice debe ser de la misma longitud que el vector del cual se quieren seleccionar elementos. Los valores correspondientes a TRUE en el vector índice son seleccionados, los correspondientes a FALSE omitidos.
  2. Un vector de números enteros. En caso de ser positivos, los correspondientes elementos indicados en el índice son concatenados, mientras que si son negativos los valores especificados son omitidos. No hace falta que sea de la misma longitud que el vector de origen.
  3. Un vector de nombres. Esta posibilidad se aplica cuando el objeto tiene el atributo names que identifica sus componentes. La función `names` permite añadir etiquetas (o nombres) a un vector numérico.
- La función `which()` genera un vector numérico con las posiciones seleccionadas.

Vectores de índices

- Vectores índice lógicos:

```
> x<-c(0,NA,1,2)
> y<-x[!is.na(x)]
> (x+1)[!is.na(x) & x>0] -> z
```

- Vectores de números enteros:

```
> x<-letters[c(13,1,18,9,15)]
> paste(x,collapse="")
[1] "mario"
> x[-(4:5)]
[1] "m" "a" "r"
```

- Vector de nombres:

```
> frutos<-c(5,10,1,20)
> names(frutos)<-c("pera","banana","melon","naranja")
> postre<-frutos[c("pera","melon")]
```

Vectores de índices

Ejemplo

```
> x<-10:20;sum(x)
[1] 165
> length(x)
[1] 11
> sum(x<15)
[1] 5
> x[x<15]
[1] 10 11 12 13 14
> sum(x[x<15])
[1] 60
> length(x[x<15])
[1] 5
> sum(x*(x<15))
[1] 60
> sum(which(x<15))
[1] 15
```

Variables indexadas (arrays)

- Una variable indexada (array) es un objeto con elementos todos del mismo tipo con un atributo adicional (dim) el cual a su vez es un vector numérico de dimensiones (números enteros positivos) formado por varios índices. Vectores y matrices son casos particulares.
- Los elementos del vector de dimensiones indican los límites superiores de los índices. Los límites inferiores siempre valen 1.
- Existen dos maneras de crear una variable indexada:
  1. Un vector puede transformarse en una variable indexada cuando se asigna un vector de dimensiones al atributo dim.

     ```
     > z<-numeric(1500);dim(z)<-c(3,5,100)
     ```

  2. Utilizando la función `array(vector de datos,vector de dimensiones)`:

     ```
     > h<-numeric(24);Z<-array(h,dim=c(3,4,2))
     ```

Variables indexadas (arrays)

- Al igual que en Fortran, la creación de la variable indexada sigue la regla de que el primer índice es el que se mueve más rápido y el último es el más lento.

Ejemplo

Si se define una variable indexada, a, con vector de dimensiones c(3,4,2), la variable indexada tendrá 3x4x2 = 24 elementos que se formarán a partir de los elementos originales en el orden a[1,1,1], a[2,1,1], ..., a[2,4,2], a[3,4,2].

- Para referenciar a un elemento de un array se debe indicar el nombre de la variable y, entre corchetes, los índices que lo refieren separados por comas.

Ejemplo

a[1,2,1], ..., a[2,4,2], etc.

Variables indexadas (arrays)

- Se puede referenciar a una parte de un array mediante una sucesión de vectores índices, teniendo en cuenta que si un vector índice es vacío equivale a utilizar todo el rango de valores para dicho índice.

Ejemplo

a[2,,] es una variable indexada 4x2, con vector de dimensión c(4,2) cuyos elementos son (a[2,1,1], a[2,2,1], a[2,3,1], a[2,4,1], a[2,1,2], a[2,2,2], a[2,3,2], a[2,4,2]) en ese orden. a[,,] equivale a la variable completa; coincide con a.

- Si se especifica una variable indexada con un solo índice o vector índice, sólo se utilizan los elementos correspondientes del vector de datos (el vector de dimensión se ignora).

Ejemplo

a[2] sería el elemento 2 del vector a, es decir el a[2,1,1] del array.

Operaciones con matrices

- Las matrices son un caso particular de array con dos dimensiones: un objeto con un atributo adicional el cual a su vez es un vector numérico de longitud 2, que define el número de filas y columnas de la matriz.
- Todos los elementos deben ser del mismo tipo.
- Una matriz se define con el comando `matrix()` especificando el número de filas y columnas o asignando la dim a un vector.
- Recordar que la matriz se crea por columnas, aunque con la opción byrow=TRUE lo hace por filas.
- Se pueden asignar nombres a las filas y columnas con el atributo `dimnames`.
- Las funciones `is.matrix()` y `as.matrix()` comprueban o fuerzan el carácter de matriz de un objeto.

Operaciones con matrices

- La situación más sencilla es la de seleccionar unas determinadas filas con un vector y unas columnas con otro vector, siguiendo las reglas comentadas en los arrays.
- R tiende a reducir complejidad: al seleccionar una columna, el objeto resultante es un vector y no una matriz. Para mantener las dimensiones, añadir drop=F.
- Se pueden seleccionar elementos de una matriz poniendo entre corchetes otra matriz, de las mismas dimensiones que la original, con valores lógicos que indicarán con TRUE los elementos a considerar.
- Cuando se realizan operaciones que mezclan variables indexadas (matrices, p.e.) y vectores conviene tener en cuenta que si se asigna un vector más corto a una matriz, se extiende repitiendo sus elementos hasta alcanzar el tamaño deseado.

Operaciones con matrices

Ejemplo

```
> x<-c(1:6);dim(x)<-c(2,3)
> dimnames(x)<-list(c("Fila1","Fila2"),c("Col1","Col2","Col3"))
> ejema<-matrix(1:12,ncol=3,byrow=T,dimnames=list(letters[1:4],LETTERS[1:3]))
> ejema[1,1]
[1] 1
> ejema[,c(2,3)]
   B  C
a  2  3
b  5  6
c  8  9
d 11 12
```

Operaciones con matrices

Ejemplo

```
> ejema[,c(-1,-3),drop=F]
   B
a  2
b  5
c  8
d 11
> sel<-matrix(rep(c(T,F),6),4,3)
> ejema[sel]
[1] 1 7 2 8 3 9
```

Operaciones con matrices

- Funciones útiles para trabajar con matrices:

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

Operaciones con matrices

- Algunas operaciones con matrices

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

Operaciones con matrices

Ejemplo

```
> x<-matrix(1:6,2,3) ; x[,2]
[1] 3 4
> x[1,1:2]
[1] 1 3
> y<-matrix(1:6,3,2) ; y[3,] ; y[3]
[1] 3 6
[1] 3
> ncol(x); nrow(y)
[1] 3
[1] 3
> t(x)
     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

Operaciones con matrices

Ejemplo

```
> cbind(1,x)
     [,1] [,2] [,3] [,4]
[1,]    1    1    3    5
[2,]    1    2    4    6
> cbind(1:3,1:6)
     [,1] [,2]
[1,]    1    1
[2,]    2    2
[3,]    3    3
[4,]    1    4
[5,]    2    5
[6,]    3    6
> diag(x) # no es necesario que la matriz sea cuadrada
[1] 1 4
> apply(x,1,sum)
[1]  9 12
```

Listas

- Una lista es un objeto consistente en una colección ordenada de objetos que se suelen llamar componentes.
- No es necesario que los componentes sean del mismo tipo, ni de la misma longitud: una lista puede estar compuesta de, por ejemplo, un vector numérico de tamaño 2, un valor lógico, un vector de tamaño 3, una matriz y una función.
- Se construyen con la función `list()` o concatenando otras listas.
- Son una parte importante de la programación de funciones en R.
- Los componentes siempre están numerados y pueden ser referidos por dicho número, o por su nombre (si lo tiene, por defecto no lo tiene).
- Aspectos a tener en cuenta al seleccionar partes de una lista:
  - La selección de elementos se hace con doble corchete o con el nombre del elemento precedido del símbolo del dólar.
  - Si se utiliza el corchete simple se está considerando una sublista (de menos componentes) de la lista.

Listas

Ejemplo

```
> ejemplolista<-list(nombre="Pedro",casado=T,esposa="María",
 no.hijos=3,edad.hijos=c(4,7,9))
> ejemplolista
$nombre
[1] "Pedro"

$casado
[1] TRUE

$esposa
[1] "María"

$no.hijos
[1] 3

$edad.hijos
[1] 4 7 9
```

Listas

Ejemplo

```
> ejemplolista[5]
$edad.hijos
[1] 4 7 9
> is.vector(ejemplolista[5]); is.list(ejemplolista[5])
[1] TRUE
[1] TRUE
> is.vector(ejemplolista[[5]]); is.list(ejemplolista[[5]])
[1] TRUE
[1] FALSE
> ejemplolista[[5]][2]
[1] 7
> ejemplolista[[5]]
[1] 4 7 9
```

Listas

Ejemplo

```
> ejemplolista$casado
[1] TRUE
> ejemplolista$nombre
[1] "Pedro"
> is.recursive(ejemplolista) # vemos el tipo de objeto
[1] TRUE
> is.atomic(ejemplolista)
[1] FALSE
> listamasgrande<-c(ejemplolista,list(edad=40))
```

Curso de Introducción a la sintaxis de R mediante RStudio
Sesión 3: Base de datos en R

Servei d'Estadística Aplicada
Universitat Autònoma de Barcelona

17, 19, 24 y 26 de enero, 2023

Contenido de la Sesión

Bases de datos o data frames

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

Covariables				
individuo	anyos	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 frames son apropiados para describir "matrices de datos" donde cada fila representa a un individuo y cada columna una variable (característica) que puede ser numérica o categórica.
- Al tratarse de una "matriz de datos" muchas de las funciones vistas para las matrices numéricas son de aplicación a los objectos de tipo data frame.

Bases de datos o data frames

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

Ejemplo

```
> datos<-data.frame(individuo=c(1,2,3,4,5,6),
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"))
```

- Los elementos de esta función pueden ser vectores (numéricos, carácteres o lógicos), factores, matrices numéricas u otros data frames.
- Puesto que un vector representa una variable de la base de datos, las columnas de una matriz representarán varias variables. La longitud de los vectores debe ser la misma y coincidir con el número de filas de las matrices.
- Los datos que no son numéricos, la función `data.frame()` los considera factores, con tantos niveles como valores distintos encuentre.

Las funciones attach() y detach()

- Para trabajar con las variables de una base de datos, se puede 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 de sus variables. Para poderlas utilizar por su nombre como vectores, hay que utilizar la función `attach(nombre de data frame)`. La operación inversa se realiza con la función `detach(df)`.

Ejemplo

```
> datos$anyos
> anyos
> attach(datos)
> anyos
> detach(datos)
> anyos
```

Explorar ficheros de datos

- Al igual que las matrices, los data frames tienen un atributo de dimensión. Además, también pueden tener atributos adicionales como nombres de filas, nombres de columnas y comentarios.
- Las funciones `dim(df)` y `attributes(df)` permiten acceder a los atributos del data frame.

Ejemplo

```
> dim(datos)
[1] 6 5
> attributes(datos)
$names
[1] "individuo" "anyos"     "tipo"      "edad"
[5] "sexo"

$class
[1] "data.frame"

$row.names
[1] 1 2 3 4 5 6
```

Explorar ficheros de datos

- La función `rownames(df)` permiten asignar un nombre a las filas del data frame.
- También podemos cambiar los nombres de las columnas existentes utilizando las funciones `colnames(df)` o `names(df)`.
- Por último, al igual que los vectores, las listas y las matrices, podemos añadir un comentario a un data frame sin que ello afecte a su funcionamiento con la función `comment(df)`.

Ejemplo

```
> rownames(datos) <- c("fila 1","fila 2","fila 3","fila 4","fila 5","fila 6")
> names(datos) <- c("Sujeto", "Años", "Tipo", "Edad", "Género")
> comment(datos) <- c("Esta base de datos contiene....")
> attributes(datos)
$names
[1] "Sujeto" "Años"   "Tipo"   "Edad"   "Género"

$class
[1] "data.frame"

$row.names
[1] "fila 1" "fila 2" "fila 3" "fila 4" "fila 5" "fila 6"

$comment
```

Explorar ficheros de datos

- Para acceder a las componentes de un data frame podemos hacer servir los operadores de listas [, [[o $.
- Acceder con [[o $ es similar. Sin embargo, difiere para [en que, indexar con [nos devolverá un data frame pero los otros dos lo reducirán a un vector.

Ejemplo

```
> datos["Sujeto"]
       Sujeto
fila 1      1
fila 2      2
fila 3      3
fila 4      4
fila 5      5
fila 6      6
> datos[["Sujeto"]]
[1] 1 2 3 4 5 6
> datos$'Años'
[1] 1.3 0.4 1.1 2.3 3.1 1.3
```

Explorar ficheros de datos

- También se puede acceder al data frame como a una matriz proporcionando un índice para la fila y la columna.

Ejemplo

```
> datos[c(2,3),]
       Sujeto Años Tipo Edad Género
fila 2      2  0.4    3   21      M
fila 3      3  1.1    3   34      H
```

- Finalmente, un data frame puede examinarse utilizando funciones como `str(df)` y `head(df)`.

Ejemplo

```
> str(datos)
'data.frame':        6 obs. of  5 variables:
 $ Sujeto: num  1 2 3 4 5 6
 $ Años  : num  1.3 0.4 1.1 2.3 3.1 1.3
 $ Tipo  : num  2 3 3 1 3 1
 $ Edad  : num  22 21 34 42 17 43
 $ Género: chr  "H" "M" "H" "H" ...
 - attr(*, "comment")= chr "Esta base de datos contiene...."
> #head(datos)
```

Cargar bases de datos de R

- R proporciona dos formatos de archivo propios para almacenar datos, `.RDS` (o `.Rds`) y `.RData` (o `.Rda`). Los archivos RDS pueden almacenar un único objeto R, y los archivos RData pueden almacenar múltiples objetos R.
- Para abrir un archivo RDS se utiliza la función `readRDS()` y para abrir un archivo RData la función `load()`.
- No es necesario asignar la salida a un objeto. Los objetos R de un archivo RData se cargarán en la sesión R con sus nombres originales.
- Un truco útil es poner paréntesis alrededor del comando de carga (`load()`) para que R imprima los nombres de cada objeto que se cargará en la sesión.
- Tanto `readRDS()` como `load()` toman como primer argumento la ruta de archivo (al igual que otras funciones de lectura y escritura de R). Si el archivo está en el directorio de trabajo, la ruta del archivo será el nombre mismo del archivo.

Ejemplo

```
> (load("ADL2.RData"))
[1] ".Random.seed" "ADL"
```

Lectura de ficheros de texto

- Para leer datos de un fichero en formato ASCII existen dos funciones, `scan()` y `read.table()`.
- Como los requisitos de lectura de `R` son bastante estrictos, muchas veces conviene modificar el archivo de datos externo previamente.
- Para asignar una base de datos a un data frame se utiliza la función `read.table()`. La forma más sencilla es con un fichero en el que:
  - La primera línea del archivo contiene los nombres de las variables.
  - En cada una de las siguientes líneas, el primer elemento es la etiqueta de la fila, y a continuación aparecen los valores de cada variable (sino, `R` asigna automáticamente etiquetas a las filas).

Ejemplo

```
> read.table(file="BD1.txt",header=T)
```

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()
```

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

La función scan()

Ejemplo

Supongamos que el fichero "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
```

Importar ficheros de Excel

- La lectura de datos desde un fichero de Excel se puede realizar con la función `read.xlsx()` de la libreria `xlsx`:

```
> install.packages("xlsx")
> library(xlsx)
> read.xlsx(file="BD2.xls",sheetIndex=1,header=T)
```

- También se puede leer un fichero de Excel directamente a partir de la función `readWorksheet()`. Para utilizar esta función es necesario instalar previamente el paquete `XLConnect`:

```
> library(XLConnect)
> readWorksheet(loadWorkbook("BD2.xls"),sheet=1)
```

- Otra alternativa es puede guardar los datos (desde Excel) en formato .csv y posteriormente utilizar la función `csv.get()` de la libreria `Hmisc`:

```
> library(Hmisc)
> csv.get(file="BD3.csv",header=T)
```

Importar ficheros con otras extensiones

- El paquete `Hmisc` también permite importar datos de SPSS, SAS y STATA (funciones `spss.get()`, `sas.get()` y `stata.get()` respectivamente).
- Otro paquete útil para importar bases de datos es `foreign`. Este paquete permite importar y exportar datos en formato .arff, .dbf y .dta, así como importar ficheros de SPSS y otras extensiones.

```
> library(foreign)
> read.dbf("BD4.dbf")
```

Exportar bases de datos

- Para exportar ficheros en formato .txt se utiliza la función `write.table()`.

```
> write.table(datos, file="c:/datos.txt", sep="\t")
```

- Para exportar a formato Excel se utiliza la función `write.xlsx()` de la libreria `xlsx`.

```
> library(xlsx)
> write.xlsx(datos, file="c:/datos.xlsx")
```

- Para exportar a otros formatos como SPSS, SAS o Stata se utilizan las funciones `write.foreign()` y `write.dta()` de la libreria `foreign`.

```
> library(foreign)
> #Exportat a SPSS
> write.foreign(datos, "c:/datos.txt", "c:/datos.sps",   package="SPSS")
> #Exportat a SAS
> write.foreign(datos, "c:/datos.txt", "c:/datos.sas",   package="SAS")
```

  En ambos casos escribe un archivo de datos de texto y un programa SPSS/SAS para leerlo.

```
> #Exportar a Stata
> write.dta(datos, "c:/datos.dta")
```

Guardar datos (Workspace)

- La función `save(objetos, list = character(0), file = "nomfich.RData", ascii = FALSE)` nos permite guardar los objetos que queramos en un archivo .RData.

Ejemplo

```
> #Guardar datos sesión
> save(datos,file="datos.RData")
```

- Podemos grabar toda la imagen `R` con la función `save.image()`, es un caso particular de la función `save(list=ls(all.names = TRUE),file=".Rdata")`.
- Al cargar los datos desde el menú **Archivo** → **Cargar área de trabajo** de `RStudio` o hacer doble click sobre el fichero .RData en el fondo estamos utilizando la función `load()`.
- Finalmente, `RStudio` incorpora diferentes menús para abrir/importar datos y para guardar/exportar datos sin necesidad de recurrir a las funciones de `R`.

Curso de Introducción a la sintaxis de R mediante RStudio
Sesión 4: Gestión de bases de datos

Servei d'Estadística Aplicada
Universitat Autònoma de Barcelona

17, 19, 24 y 26 de enero, 2023

Contenido de la Sesión

1. Filtrar bases de datos
2. Ordenar bases de datos
3. Reestructurar bases de datos
   - Funciones de la familia `apply`
4. Fusionar bases de datos
5. Transformar y Recodificar variables

Crear un subconjunto de datos

- En cualquier momento es posible llevar a cabo un filtrado de datos utilizando expresiones lógicas y vectores índices.
- En el caso de los data frames es posible seleccionar filas y columnas deseadas y guardarlas en una nueva base de datos. También se puede crear un subconjunto de datos con la función `subset()`.

Ejemplo

```
> #Cargar datos sesión
> load("datos.RData")
> datos.filtrados<-datos[datos$sexo=="H",]
> mas.peq<-subset(datos,anyos<1,select=c(edad,sexo))
```

Eliminar filas o columnas

- R no tiene una función que elimine las Filas o columnas de un data frame sin embargo, podemos crear un subconjunto que solo contenga las filas o columnas deseadas como se ha visto antes.
- No obstante, hay formas más sencillas de eliminar filas y columnas. Por ejemplo:

Ejemplo

```
> #Eliminar la 4a fila
> datos2 <- datos[-4,]
> #Eliminar la 4a, 5a y 1a filas
> datos3 <- datos[-c(4,5,1),]
> #Eliminar la 4a fila y la segunda columna
> datos4 <- datos[-4, -2]
> #Eliminar columnas por nombre
> datos5 <- datos[!names(datos) %in% c("anyos","tipo")]
```

Ordenar bases de datos

- Para ordenar un data frame en R, se utiliza la función `order(df)`. Por defecto, la ordenación es ASCENDENTE. Si se antepongo a la variable de ordenación un signo menos el orden será DESCENDENTE.

Ejemplo

```
> #Orden descentente por individuo
> datos_ord <- datos[order(-datos$individuo),]
> #Alternativa
> attach(datos)
> datos_ord <- datos[order(-individuo),]
> detach(datos)
```

Reestructurar bases de datos

En `R` exiten diversas funciones que nos ayudan a reestructurar y visualizar la base de datos de diferentes maneras.

- La función `t(df)` permite transponer un data frame entero.
- En el paquete `reshape` se encuentran múltiples funciones para llevar a cabo la reestructuración. Las más habituales son:

  melt : se "funden" los datos para que cada fila sea una combinación única de las varaibles indicadas como identificadoras.

  cast : a partir de un objeto de tipo `melt` reestructura o agrega los datos.

  recast Pasos melt y cast en uno solo.

  La documentación del paquete `reshape` incorpora un video demostrativo.

Reestructurar bases de datos

Ejemplo

```
> library(reshape)
> datos2<-data.frame(id=c(1,1,1,2,2,2),
 visita=c(1,2,3,1,2,3),
 h1=c(143, 128, 122, 145, 139, 137),
 h2=c(82, 81, 79, 100, 102, 99))
> datos2.melt <- melt(datos2, id=c("id","visita"), na.rm=TRUE)
> #Medias por individuos
> id.medias <- cast(datos2.melt, id ~ variable, mean)
> #o bien
> recast(datos2, id ~ variable, id.var=1:2, fun.aggregate="mean")
> #Medias por visitas
> visita.medias <- cast(datos2.melt, visita ~ variable, mean)
> #o bien
> recast(datos2, visita ~ variable, id.var=1:2, fun.aggregate="mean")
```

Funciones de la familia `apply`

Alternativamente, se pueden utilizar las funciones de la familia `apply` para agregar/resumir/filtrar la información de un data frame.
Las funciones de la família apply son las siguientes:

- lapply: Devuelve una lista de resultados al aplicar una función a un vector/es.
- sapply: Devuelve un vector de resultados al aplicar una función a un vector/es.
- apply: Devuelve el vector o la lista de valores obtenidos aplicando una función a un vector o matriz por grupos.
- tapply: Devuelve una lista con los resultados obtenidos al aplicar una función a un vector por grupos.
- ddply: Devuelve un data.frame de resultados al aplicar una función a un data.frame por grupos.

Funciones de la familia `apply`

- Para aplicar la función `apply()` se utilizan tres parámetros:
  - El objeto array o matriz
  - La dimensión sobre la que actuaremos
  - La función que se aplicará
- En el caso de las matrices, si el segundo argumento es 1, significa que la función se aplicará por filas, mientras que si es un 2, se aplicará a las columnas.
- En muchos casos se puede utilizar una instrucción `apply()` en lugar de un bucle.
- El resultado es un vector o una matriz con nombres (si los había), y es más eficiente que un bucle.

Funciones de la familia apply

Ejemplo

```
> medias.por.fila <- apply(datos.todo, 1, mean)
> medias.por.columnas <- apply(datos.todo[,3:6],2,mean)
```

Funciones de la familia apply

- Las funciones `lapply()` y `sapply()` son como la función `apply()` pero no hace falta especificar que el segundo argumento es 2, es decir, estas dos funciones realizan los cálculos por columnas.

Ejemplo

```
> lapply(datos.todo[,3:6], mean)
> lapply(datos.todo[,3:6], mean, na.rm=TRUE)
> sapply(datos.todo[,3:6], mean)
```

Funciones de la familia apply

- La función `tapply(x,y,function)` realiza los calculos especificados en `function` a la variable `x` según la variable `y`.

Ejemplo

```
> x.media <- tapply(datos.todo$h1,datos.todo$visita,mean)
> x.var <- tapply(datos.todo$h1,datos.todo$visita,var)
> x.mediana <- tapply(datos.todo$h1,datos.todo$visita,
                      median)
```

Funciones de la familia apply

- La función `ddply()` permite resumir los datos para cada uno de los subconjuntos definidos.
- Esta función se encuentra dentro de la libreria `plyr`.

Ejemplo

```
> library(plyr)
> ddply(datos, .(sexo, ec), summarize,
  mean = round(mean(edad),2), sd = round(sd(edad),2))
```

Funciones estadísticas básicas

Función	Utilidad
sum(...,na.rm=FALSE)	Suma
max(...,na.rm=FALSE)	Máximo
min(...,na.rm=FALSE)	Mínimo
which.min(...)	Posición del mínimo
which.max(...)	Posición del máximo
mean(...,na.rm=FALSE)	Media
weighted.mean(...,w,na.rm=FALSE)	Media ponderada
median(...,na.rm=FALSE)	Mediana
quantile(...,prob=C(0,0.25,0.5,0.75,1),na.rm=FALSE)	Cuantiles
IQR(...,na.rm=FALSE)	Rango inter-cuatílico
range(...,na.rm=FALSE,finite=FALSE)	Rango
var(...,na.rm=FALSE)	Varianza
sd(...,na.rm=FALSE)	Desviación típica
summary(...,na.rm=FALSE)	Min, 1c, mediana, media, 3c, max
table(...,na.rm=FALSE)	Recuento casos

Fusionar bases de datos

A la hora de fusionar bases de datos podemos estar interesados en añadir nuevas variables (columnas) o añadir nuevos individuos (filas).

- Añadir columnas: para combinar dos base de datos se utiliza la función `merge()`. Se fusionan en base a una o más variables clave.

Ejemplo

```
> datos3<-data.frame(id=c(1,1,1,2,2,2),
 visita=c(1,2,3,1,2,3),
 col1=c(273, 248, 259, 210, 201, 197),
 col2=c(222, 217, 208, 197, 200, 189))
> datos23 = merge (datos2, datos3, by=c("id","visita"))
```

Fusionar bases de datos

- Añadir filas: para fusionar dos bases de datos verticalmente se utiliza la función `rbind()`. Para ello es necesario que ambas bases de datos tengan exactamente las mismas variables pero no es necesario que esten en el mismo orden.

Ejemplo

```
> masdatos<-data.frame(id=c(3,3,3),
 visita=c(1,2,3), col1=c(301, 289, 291),
 col2=c(250, 241, 256), h1=c(123, 128, 125),
 h2=c(74, 75, 69))
> datos.todo = rbind(datos23, masdatos)
```

Transformaciones y Recodificaciones

- En cualquier momento se puede modificar o manipular cualquier objeto con operadores (aritméticos, lógicos y comparativos) y funciones. Así, por ejemplo, si una variable no es normal, puede transformarse para conseguir dicha normalidad.
- ¡Cuidado! Para que la transformación/recodificación forme parte del data frame, es necesario indicarlo específicamente (no con attach).
- Para modificar el contenido interno de los data frames se puede utilizar la función `transform()`. Esta función permite:
  - Transformar variables existentes.
  - Crear nuevas variables mediante la transformación de variables existentes.

Transformaciones y Recodificaciones

Ejemplo

```
> attach(datos)
> edad.final<-edad+anyos
> # no afecta al data.frame
> datos.1<-transform(datos,edad.final=edad+anyos)
> # si afecta
> datos.2<-transform(datos.1,edad=edad+1)
> # altera la variable
```

Transformaciones y Recodificaciones

Mediante código puede realizarse la recodificación de cualquier variable.

Ejemplo

```
> datos$ec[datos$edad<=25]<-"joven"
> datos$ec[datos$edad>25]<-"adulto"
> datos$sexo.cod[datos$sexo=="H"]<-0
> datos$sexo.cod[datos$sexo=="M"]<-1
```

Por defecto, la variable resultante es carácter o numérica, pero no factor. Para transformarlas a variables tipo Factor, se realiza:

Ejemplo

```
> datos$ec<-as.factor(datos$ec)
> datos$sexo.cod<-as.factor(datos$sexo.cod)
```

Transformaciones y Recodificaciones

- Mediante la función `recode()` de la librería `car` es posible llevar a cabo recodificaciones de otras variables.
- Se debe especificar cuál es la variable a recodificar, cuáles son los criterios de recodificación y si la nueva variable resultante de la recodificación será un factor o no.
- ¡Cuidado! Las recodificaciones tampoco afectarán al propio data frame si no se indica específicamente.

Licencia de este material