Datos: lectura, manejo de observaciones y variables. Funciones

000R Team

Métodos estadísticos de investigación: introducción a R y Rstudio

Antes de empezar

Pepara la sesión de trabajo

Para ello:

- Establece un directorio de trabajo (*working directory*)
- Crea un script de R con un nombre identificativo de la sesión y guárdalo en tu directorio de trabajo
- Haz un comentario en dicho script (usando la almohadilla #) en el que reflejes la fecha de hoy y otras anotaciones que consideres explicativas.

Las ayudas de R

help()

La función `help()` muestra información sobre la función que pasemos como argumentos

```
help(rep)
```

Es equivalente a escribir

```
?rep
```

apropos()

La función `apropos()` lista todas las funciones que contienen el string que se le pasa como argumento

```
apropos("rep")
```

```
##  [1] "agrep"                          "agrepl"
##  [3] "bug.report"                     ".__C__classRepresentation"
##  [5] ".__C__ClassUnionRepresentation" ".__C__MethodSelectionReport"
##  [7] ".__C__refClassRepresentation"   ".__C__repeat"
##  [9] "grep"                           "grepl"
## [11] "grepRaw"                        "makeClassRepresentation"
## [13] "newClassRepresentation"         "preplot"
## [15] "rep"                            "rep.Date"
## [17] "repeat"                         "rep.factor"
## [19] "rep.int"                        "replace"
## [21] "replayPlot"                     "rep_len"
## [23] "replicate"                      "replications"
## [25] "rep.numeric_version"            "rep.POSIXct"
## [27] "rep.POSIXlt"                    "representation"
## [29] "setReplaceMethod"               "setRepositories"
```

example()

La funcion `example()` muestra diferentes casos (ejemplos) de uso de la función que se le pasa como argumento

```
example(rep)
```

```
##
## rep> rep(1:4, 2)
## [1] 1 2 3 4 1 2 3 4
##
## rep> rep(1:4, each = 2)       # not the same.
## [1] 1 1 2 2 3 3 4 4
##
## rep> rep(1:4, c(2,2,2,2))     # same as second.
## [1] 1 1 2 2 3 3 4 4
##
## rep> rep(1:4, c(2,1,2,1))
## [1] 1 1 2 3 3 4
##
## rep> rep(1:4, each = 2, len = 4)    # first 4 only.
## [1] 1 1 2 2
##
## rep> rep(1:4, each = 2, len = 10)   # 8 integers plus two recycled 1's.
##  [1] 1 1 2 2 3 3 4 4 1 1
##
## rep> rep(1:4, each = 2, times = 3)  # length 24, 3 complete replications
##  [1] 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4 1 1 2 2 3 3 4 4
##
## rep> rep(1, 40*(1-.8)) # length 7 on most platforms
```

RsiteSearch()

Para buscar en manuales de ayuda y listas de correo archivadas información sobre una función se puede usar `RsiteSearch()`

```
RSiteSearch("gbm")
```

Devuelve un conjunto de resultados (con un índice o puntuación para cada uno de ellos) que coinciden con la consulta.

demo()

`demo` es una interfaz de uso amigable para ejecutar algunos scripts de R a modo de demostración

Escribiendo

```
demo()
```

aparece en pantalla las funciones que tiene disponible esta utilidad.

demo() 2

```
demo( graphics )
demo( persp )
demo( intervals )
```

La chuleta (R Reference Card 2.0 de Baggott)

Compendio de funciones básicas (y no tanto) de R.

`https://cran.r-project.org/doc/contrib/Baggott-refcard-v2.pdf`

R: los objetos

Vectores

Vectores

- Los vectores son un conjunto ordenado de valores
- Para calcular con todo el vector se emplea el nombre del objeto
- Para utilizar un subconjunto valores se emplea subíndices
- Los subíndices se incluyen entre corchetes (`x[ 3 ]`)
- Los subíndices están en el rango: 1 — *número de elementos del vector*
- Los subíndices pueden ser expresiones

Ejemplo

```
x <- c( 8, 5, 2, 4, 1, 6, 3 )
length( x )
```

```
## [1] 7
```

```
x
```

```
## [1] 8 5 2 4 1 6 3
```

```
x[]
```

```
## [1] 8 5 2 4 1 6 3
```

Ejemplo

```
x[ 1 ]
```

```
## [1] 8
```

```
x[ 2:4 ]
```

```
## [1] 5 2 4
```

```
x[ c( 3, 5 ) ]
```

```
## [1] 2 1
```

Ejemplo

```
x[ -1 ]
```

```
## [1] 5 2 4 1 6 3
```

```
x[ x > 4 ]
```

```
## [1] 8 5 6
```

```
which( x > 4 )
```

```
## [1] 1 2 6
```

Ejercicio 1

- Crea un vector de números enteros (mínimo 5 elementos)
- Comprueba el tipo de tu vector con la función `str()`
- Aplica algunas funciones a tu vector para calcular: su media, la suma de sus componentes...
- Sustituye el tercer componente del vector por la suma de los dos anteriores
- Haz que tu vector (que ya está creado) sea de tipo `cadena de caracteres`.
- Guárdalo en una nueva variable
- Comprueba su tipo con la función `str()`
- Intenta aplicar las funciones anteriores a ese vector. ¿Qué sucede?

Matrices

Matrices

- Una matriz es un conjunto ordenado de vectores
- Los elementos de la matriz están ordenados por filas y columnas
- Todos los vectores son del mismo tipo: enteros, caracteres, ...
- Los elementos de una matriz se identifican por dos subíndices
- El uso de los subíndices sigue las mismas reglas que en el caso de los vectores
- Se puede crear uniendo vectores o mediante la función `matrix()`

Ejemplo

```
m <- matrix( 1:12, 4, 3 )
m
```

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

```
m[ 1, ]
```

```
## [1] 1 5 9
```

Data frames

Data frames

- Son semejantes a las matrices
- Se organizan por filas y columnas
- Las columnas no tienen por que ser homogéneas
- Las columnas tienen nombre
- Habitualmente los *data frames* se obtienen de la lectura de un fichero de datos

Ejemplo PIB

	Valor		Estructura porcentual	
	2008	2009	2008	2009
Andalucía				
A. Agricultura, ganadería, silvicultura y pesca	6.467.357	6.025.496	4,3	4,2
F. Construcción	21.477.597	19.223.889	14,4	13,5
Hostelería	10.076.699	10.005.749	6,8	7,0
Aragón				
A. Agricultura, ganadería, silvicultura y pesca	1.197.806	1.188.230	3,5	3,6
F. Construcción	4.678.884	4.447.108	13,5	13,4
Hostelería	1.905.278	1.919.271	5,5	5,8
Galicia				
A. Agricultura, ganadería, silvicultura y pesca	2.278.151	2.244.530	3,9	4,0
F. Construcción	7.901.498	7.507.649	13,7	13,4
Hostelería	2.710.660	2.927.641	4,7	5,2
Madrid, Comunidad de				
A. Agricultura, ganadería, silvicultura y pesca	224.709	195.667	0,1	0,1
F. Construcción	20.320.981	19.171.700	10,5	10,1
Hostelería	10.418.601	10.861.179	5,4	5,7
Murcia, Región de				
A. Agricultura, ganadería, silvicultura y pesca	1.377.749	1.233.257	4,7	4,5
F. Construcción	4.110.927	3.526.090	14,1	12,8
Hostelería	1.505.280	1.489.960	5,2	5,4

Figure 1:

Leer datos de un fichero

La función más importante para *importar* datos: `read.table()` - automáticamente convierte los datos en un *dataframe*.

```
read.table(file, header = FALSE, sep = "",
           dec = ".",
           row.names, col.names,
           na.strings = "NA",
           skip = 0,
           comment.char = "#",
           stringsAsFactors = default.stringsAsFactors(),
           )
```

read.table()

```
df <- read.table(file,
                 header = valor_logico,
                 sep = "delimitador",
                 dec = "signo del decimal")
```

- `header` TRUE/FALSE según la primera fila del fichero - `sep` es el delimitador: el separador de campos + `sep = ";"`, `sep="."`, `sep=","`, `sep="\t"` (tabulación)... - `dec` es el indicador del signo decimal + `dec="."`, `dec=";"`...

Ejercicio 2.1

Leed el fichero `pibCcAaEj.dat` teneindo en cuenta que el separador, el símbolo de decimales y la cabecera

```
##    ciudad actividad anho     valor
## 1 Andaluc     Agric 2008  6467.357
## 2 Andaluc     Const 2008 21477.597
## 3 Andaluc      Host 2008 10076.699
## 4    Arag     Agric 2008  1197.806
## 5    Arag     Const 2008  4678.884
## 6    Arag      Host 2008  1905.278
```

Seleccionar subconjuntos de un dataframe

Forma general

$$nuevo\_df \leftarrow df[\,\text{índices}Filas,\ \text{índices}Columnas\,]$$

- **Ojo:** la regla es *"filas por columnas"*.

Seleccionar variables (columnas. . .)

$$nuevo\_df \leftarrow df[\quad, índices]$$

Observar que dejamos en blanco la primera posición entre los corchetes para referirnos a la columnas

Acceder a las variables de un data frame

```
options( width = 80)
df[ , 4 ]
```

```
##  [1]  6467.357 21477.597 10076.699  1197.806  4678.884  1905.278  2278.151
##  [8]  7901.498  2710.660   224.709 20320.981 10418.601  1377.749  4110.927
## [15]  1505.280  6025.496 19223.889  1005.749  1188.230  4447.108  1919.271
## [22]  2244.530  7507.649  2927.641   195.667 19171.700 10861.179  1233.257
## [29]  3526.090  1489.960
```

```
df$valor
```

```
##  [1]  6467.357 21477.597 10076.699  1197.806  4678.884  1905.278  2278.151
##  [8]  7901.498  2710.660   224.709 20320.981 10418.601  1377.749  4110.927
## [15]  1505.280  6025.496 19223.889  1005.749  1188.230  4447.108  1919.271
## [22]  2244.530  7507.649  2927.641   195.667 19171.700 10861.179  1233.257
## [29]  3526.090  1489.960
```

```
df [, "valor" ]
```

```
##  [1]  6467.357 21477.597 10076.699  1197.806  4678.884  1905.278  2278.151
##  [8]  7901.498  2710.660   224.709 20320.981 10418.601  1377.749  4110.927
## [15]  1505.280  6025.496 19223.889  1005.749  1188.230  4447.108  1919.271
## [22]  2244.530  7507.649  2927.641   195.667 19171.700 10861.179  1233.257
## [29]  3526.090  1489.960
```

Acceder a las variables de un data frame

```
df[ 4 ]
```

```
##         valor
## 1    6467.357
## 2   21477.597
## 3   10076.699
## 4    1197.806
## 5    4678.884
## 6    1905.278
## 7    2278.151
## 8    7901.498
## 9    2710.660
## 10    224.709
## 11 20320.981
## 12 10418.601
## 13  1377.749
## 14  4110.927
## 15  1505.280
## 16  6025.496
## 17 19223.889
## 18  1005.749
## 19  1188.230
## 20  4447.108
## 21  1919.271
## 22  2244.530
## 23  7507.649
## 24  2927.641
```

Ejercicio 2.2

del cto de datos: `pibCcAaEj.dat`

- Selecciona el valor del PIB para Andalucía
- Selecciona el valor del PIB en Andalucía correspondiente a la agricultura
- Haz un boxplot para valor del PIB en general
- Haz un boxplot para valor del PIB para el año 2008 y otro para el año 2009
- Intenta hacer los dos boxplots en una mismo *device* gráfico

Factores

Factores

Hay variables codificadas,_ *e.g.* `sexo`, donde 1 es *'Masculino'* y 2 *'Femenino'*, que en realidad tiene un carácter "cualitativo" o "categórico".

```
sexo     <- c(1, 1, 1, 1, 2, 2, 2)
peso     <- c(60, 65, 70, 66, 80, 60, 76)
df       <- data.frame(sexo, peso)
df$sexo <- factor(df$sexo)
```

Factores: etiquetas de valores

Creamos las etiquetas

```
df$sexo <- factor( df$sexo,
                   levels = c(1, 2),
                   labels = c("masculino", "femenino")
                   )
```

¿Hay algún *factor* en nuestro conjunto de datos?

```
head( df )
```

```
##        sexo peso
## 1 masculino   60
## 2 masculino   65
## 3 masculino   70
## 4 masculino   66
## 5  femenino   80
## 6  femenino   60
```

read.table() ii

```
df <- read.table(
  "./materiales-sesion2/pibCcAaEj.dat",
  sep=";" ,  header=T,
 stringsAsFactors = T)
head(df)
```

```
##    ciudad actividad anho     valor
## 1 Andaluc     Agric 2008  6467.357
## 2 Andaluc     Const 2008 21477.597
## 3 Andaluc      Host 2008 10076.699
## 4    Arag     Agric 2008  1197.806
## 5    Arag     Const 2008  4678.884
## 6    Arag      Host 2008  1905.278
```

read.table() iii

- variables de tipo carácter se convierten a factores
  - `stringsAsFactor = FALSE`
- `help(read.table)`
  - `na.strings` = "NA"
  - `dec` = "."

Algunas funciones útiles

Función	Acción
`length(obj)`	Número de componentes, elementos
`dim(obj)`	Dimensión de un objeto
`str(obj)`	Estructura de un objeto
`class(obj)`	Clase (class) o tipo de objeto
`names(obj)`	Nombres de los componentes de un objeto
`c(obj,obj,...)`	Combina objetos en un vector
`head(obj)`	Lista la primera parte de un objeto
`tail(obj)`	Lista la última parte (cola) de un objeto
`ls()`	Lista los objetos actuales
`rm(obj)`	Borra un objeto
`newobj \<- edit(obj)`	Edita un objeto y lo guarda
`fix(obj)`	Edita sobre un objeto ya creado

Manipulación de objetos

Datos de ejemplo para trabajar. data.frame()

- Trabajaremos con un *dataset* sencillo: `introducimos los datos a mano`

```
# getwd()
id      <- c( 1:5 )
date    <- c( "10/07/08", "10/08/08", "10/09/08", "10/10/08", "10/11/08" )
country <- c( "US", "US", "UK", "UK", "UK" )
gender  <- c( "M", "F", "F", NA, "F" )
age     <- c( NA, 45, 25, 39, 99 )
q1      <- c( 5, 3, 3, 3, 2 )
q2      <- c( 5, 5, 5, NA, 2 )
q3      <- c( 5, 5, 2, NA, 1 )
df      <- data.frame( id, date, country, gender, age, q1, q2, q3,
                       stringsAsFactors = FALSE )
df
```

```
##   id     date country gender age q1 q2 q3
## 1  1 10/07/08      US      M  NA  5  5  5
## 2  2 10/08/08      US      F  45  3  5  5
## 3  3 10/09/08      UK      F  25  3  5  2
## 4  4 10/10/08      UK   <NA>  39  3 NA NA
## 5  5 10/11/08      UK      F  99  2  2  1
```

Ejercicio 3

Con el objeto creado anteriormente `df`

- Crea un nuevo data frame que incluya únicamente las variables q1, q2 y q3.
- Crea de dos formas un nuevo data frame que incluya el país al que pertenecen los sujetos (con el nombre y con el índice)
- Crea un nuevo data frame eliminando las variables q2 y q3
- Selecciona las 3 primeras filas
- ¿Cómo escribirías la condición de ser mujer Y de UK?
- Selecciona las observaciones que cumplan dicha condición

Transformaciones de variables

transform()

```
# creamos variables con la función transform()
df <- transform( df,
                 sumx  = q1 + q2,
                 meanx = ( q1 + q2 ) / 2 )
df
```

```
##    id     date country gender age q1 q2 q3 sumx meanx
## 1   1 10/07/08      US      M  NA  5  5  5   10     5
## 2   2 10/08/08      US      F  45  3  5  5    8     4
## 3   3 10/09/08      UK      F  25  3  5  2    8     4
## 4   4 10/10/08      UK   <NA>  39  3 NA NA   NA    NA
## 5   5 10/11/08      UK      F  99  2  2  1    4     2
```

eliminar variables

```
ncol(df)
```

```
## [1] 10
```

```
df <- df[  ,-c(ncol(df) - 1 , ncol(df))]
df
```

```
##    id     date country gender age q1 q2 q3
## 1   1 10/07/08      US      M  NA  5  5  5
## 2   2 10/08/08      US      F  45  3  5  5
## 3   3 10/09/08      UK      F  25  3  5  2
## 4   4 10/10/08      UK   <NA>  39  3 NA NA
## 5   5 10/11/08      UK      F  99  2  2  1
```

crear variables al vuelo

```
df$nueva <- df$q1 * 2
df
```

```
##    id     date country gender age q1 q2 q3 nueva
## 1   1 10/07/08      US      M  NA  5  5  5     10
## 2   2 10/08/08      US      F  45  3  5  5      6
## 3   3 10/09/08      UK      F  25  3  5  2      6
## 4   4 10/10/08      UK   <NA>  39  3 NA NA      6
## 5   5 10/11/08      UK      F  99  2  2  1      4
```

Recodificación de variables (recordando operadores)

Table 2: Operadores lógicos

Operador	Descripción
`<`	Menor que
`<=`	Menor o igual que
`>`	Mayor que
`>=`	Mayor o igual que
`==`	Exactamente igual a
`!=`	No igual a/que
`!x`	Diferente de x
`x | y`	x o y
`x & y`	x e y
`isTRUE(x)`	Prueba si x es TRUE

Recodificación de variables (2)

```
df$agecat[ df$age > 75 ]                  <- "anciano" # crea la v
df$agecat[ df$age <= 75 & df$age > 44 ] <- "maduro"
df$agecat[ df$age <= 44 ]                 <- "joven"
df
```

```
##   id     date country gender age q1 q2 q3 nueva  agecat
## 1  1 10/07/08      US      M  NA  5  5  5     10    <NA>
## 2  2 10/08/08      US      F  45  3  5  5      6  maduro
## 3  3 10/09/08      UK      F  25  3  5  2      6   joven
## 4  4 10/10/08      UK   <NA>  39  3 NA NA      6   joven
## 5  5 10/11/08      UK      F  99  2  2  1      4 anciano
```

Renombrar variables

```
names( df )
```

```
##  [1] "id"      "date"    "country" "gender"  "age"     "q1"      "q2"
##  [8] "q3"      "nueva"   "agecat"
```

```
names( df )[ 1 ] <- "ID"
names( df )[ 3 ] <- "pais"
names( df )[ 4 ] <- "G"

names( df )[ 6:8 ] <- c( "it1", "it2", "it3" )
df
```

```
##   ID     date pais    G age it1 it2 it3 nueva  agecat
## 1  1 10/07/08   US    M  NA   5   5   5    10    <NA>
## 2  2 10/08/08   US    F  45   3   5   5     6  maduro
## 3  3 10/09/08   UK    F  25   3   5   2     6   joven
## 4  4 10/10/08   UK <NA>  39   3  NA  NA     6   joven
## 5  5 10/11/08   UK    F  99   2   2   1     4 anciano
```

Valores perdidos (missing values)

NaN y NA

- Valores imposibles: `NaN` (*Not a Number*)
- Los valores perdidos: `NA` (*Not available*)

Hay muchas funciones para identificar estos valores: `is.na()`

is.na()

```
# missings
y <- c( 1, 2, 3, NA )
is.na( y )  # ¿cuáles son NA?
```

```
## [1] FALSE FALSE FALSE  TRUE
```

```
# devuelve un objeto del mismo tamaño que el que recibe
is.na( df[ , 4:10 ] )
```

```
##          G   age   it1   it2   it3 nueva agecat
## [1,] FALSE  TRUE FALSE FALSE FALSE FALSE   TRUE
## [2,] FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
## [3,] FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
## [4,]  TRUE FALSE FALSE  TRUE  TRUE FALSE  FALSE
## [5,] FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
```

Recodificar valores a *missing*

```
df$age[ is.na( df$age ) ] <- 99
```

```
# observa qué ocurre
is.na( df[ , 1:10 ] )
```

```
##         ID  date  pais     G   age   it1   it2   it3 nueva agecat
## [1,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE   TRUE
## [2,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
## [3,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
## [4,] FALSE FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE  FALSE
## [5,] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE  FALSE
```

```
df$age[ df$age == 99 ] <- NA
```

Excluir valores *missing* del análisis

Emplearemos la opción `na.rm=TRUE` para que **no** considere los valores faltantes

```
# excluir los missings del análisis
x <- c( 1, 2, NA, 3 )
y <- x[ 1 ] + x[ 2 ] + x[ 3 ] + x[ 4 ]
z <- sum( x )
y
```

```
## [1] NA
```

```
z
```

```
## [1] NA
```

```
# ¡¡¡¡Ambos son NA!!!!
sum( x, na.rm = T )  # ¡Ahora no!
```

```
## [1] 6
```

na.omit()

`na.omit()` que elimina cualquier *fila* de un fataframe que tenga valores faltantes.

```
df <- na.omit( df )
df
```

```
##   ID     date pais G age it1 it2 it3 nueva agecat
## 2  2 10/08/08   US F  45   3   5   5     6 maduro
## 3  3 10/09/08   UK F  25   3   5   2     6  joven
```

Conversión de tipos

dos tipos de funciones

Table 3: Conversión de tipos

lógico	convierte
`is.numeric()`	`as.numeric()`
`is.character()`	`as.character()`
`is.vector()`	`as.vector()`
`is.matrix()`	`as.matrix()`
`is.data.frame()`	`as.data.frame()`

Conversión de tipos (2)

```
# conversión de tipos.
a <- c( 1, 2, 3 );a
```

```
## [1] 1 2 3
```

```
is.numeric( a )
```

```
## [1] TRUE
```

```
is.vector( a )
```

```
## [1] TRUE
```

```
a <- as.character( a );a
```

```
## [1] "1" "2" "3"
```

Conversión de tipos (2)

```
a <- as.character( a );a
```

```
## [1] "1" "2" "3"
```

```
is.numeric( a )
```

```
## [1] FALSE
```

```
is.vector( a )
```

```
## [1] TRUE
```

Ordenar datos

order()

```
df
```

```
##   ID     date pais G age it1 it2 it3 nueva agecat
## 2  2 10/08/08   US F  45   3   5   5     6 maduro
## 3  3 10/09/08   UK F  25   3   5   2     6  joven
```

ascendente por age

```
df_ordenado <- df[ order( df$age ),  ]
df_ordenado
```

```
##   ID     date pais G age it1 it2 it3 nueva agecat
## 3  3 10/09/08   UK F  25   3   5   2     6  joven
## 2  2 10/08/08   US F  45   3   5   5     6 maduro
```

order() (2)

Ascendente por gender y descendente segun age

```
df_ordenado2 <- df[ order( df$G , -df$age ),  ]
df_ordenado2
```

```
##    ID      date pais G age it1 it2 it3 nueva agecat
## 2   2 10/08/08   US F  45   3   5   5     6 maduro
## 3   3 10/09/08   UK F  25   3   5   2     6  joven
```

Conjuntos de datos incorporados en R

Hay muchos conjuntos de datos incorporados en la libería base de R

`https://stat.ethz.ch/R-manual/R-devel/library/datasets/html/00Index.html`

Vamos a usar brevemente el conjunto `iris`

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
?iris
```

Conjuntos de datos incorporados en R: iris

- Usando la función `names()` traduce sus nombres al español (y guárdalos). Utiliza nombres correctos (recuerda qué sucede con los espacios y caracteres extraÑos)
- Ordena los datos por longitud del pétalo de forma descendente

Listas

Listas

- Son objetos que pueden contener conjuntos heterogéneos de objetos
  - valores
  - vectores
  - matrices
  - *data frames*
  - listas
- Se suelen encontrar como resultado de funciones

Funciones

Funciones Predefinidas

Funciones Predefinidas

Hay miles y miles, agrupadas en paquetes, y cada día hay más

`http://cran.es.r-project.org/`

Funciones matemáticas

Función	lo que hace
`abs(x)`	Valor absoluto de x, abs(-4) devuelve 4
`sqrt(x)`	Raíz cuadrada de x, sqrt(25) devuelve 5
`ceiling(x)`	Entero más pequeño mayor que x
`floor(x)`	Entero más grande no mayor que x
`trunc(x)`	Truncamiento de x
`round(x, digits=n)`	Redondea x a un número específico de decimales
`log(x,base=n)`	Logaritmos

Funciones estadísticas

Función	lo que hace
`mean(x)`	Media
`median(x)`	Mediana
`sd(x)`	Desviación estándar
`var(x)`	Varianza
`sum(x)`	Suma
`range(x)`	Rango
`min(x)`	Mínimo
`max(x)`	Máximo
`scale(x,center=TRUE,scale=TRUE)`	Estandarizar

Miscelánea

cat()

```
# concatenar
cat( "hola", "amigo" )
```

```
## hola amigo
```

```
cat( "hola",
     "amigo",
     "\n",
     "¿cómo estas?",
     file = "fichero-cocatenado.txt" ) # escribimos en un fichero
```

a objetos

podemos aplicar funciones a una gran cantidad de objetos: vectores, arrays, matrices, dataframes...

```
# aplicar funciones a objetos 'complejos'
y <- c( 1.23, 4.56, 7.89 )
round( y )
```

```
## [1] 1 5 8
```

a objetos (ii)

```
z <- matrix( rnorm( 12 ), 3 )
z
```

```
##            [,1]       [,2]      [,3]       [,4]
## [1,]  0.8204273  1.1741923 0.3393213 -0.1023994
## [2,] -1.0555518  0.9007617 1.0562711  0.5109410
## [3,]  0.6526495 -1.5315953 0.7910393  1.6377127
```

```
log( z )
```

```
## Warning in log(z): Se han producido NaNs
```

```
##            [,1]       [,2]        [,3]       [,4]
## [1,] -0.1979300  0.1605805 -1.08080792        NaN
## [2,]        NaN -0.1045145  0.05474491 -0.6715012
## [3,] -0.4267151        NaN -0.23440768  0.4933006
```

```
mean( z )  # devuelve un escalar
```

```
## [1] 0.4328141
```

```
sd ( z )
```

```
## [1] 0.9200508
```

Sentencias IF-ELSE

Ejecutan sentencias si se da una condición `if`, si no se da ejecutan la sentencia `else`. Se pueden anidar

```
if (condición) sentencia
ó
if (condición) sentencia1 else sentencia2
```

Sentencias IF-ELSE (2)

```
# IF-ELSE
if ( TRUE ) {
    print( "esta siempre se ejecuta" )
}

if ( FALSE ) {
    print( "esta nunca se ejecuta" )
} else {
    print( "¿lo ves?" )
}
```

funciones ad hoc

Funciones escritas por el usuario

- escribir funciones *ad hoc*
- se pueden empaquetar en grupos
- . . . . tenerlas siempre disponibles para todas las sesiones

La sintaxis es sencilla:

```
mi_funcion <- function( arg1, arg2, ... ) {
    sentencias
    return( objeto )
}
```

mi primera función

```
f.potencia <- function( num, exp = 2 ) {
    return( num ** exp )
}
f.potencia( 5, 2 )
```

```
## [1] 25
```

```
f.potencia( 5, 5 )
```

```
## [1] 3125
```

```
f.potencia( 5 )
```

```
## [1] 25
```

Ejemplo

Una función que acepte dos argumentos:

- El primer argumento, uno que sea un char con dos posibilidades "param" o "noparam", que por defecto sea el valor "param".
- El segundo argumento que sea un vector numérico. Y que nos devuelva para "param", la media, la desviación típica y la varianza del vector.

Y para "noparam", la mediana, máximo y mínimo del vector.

Ejercicio 4

Crea una función que calcule el área de un circulo, dado un radio

Recuerda que $A = \pi \cdot r^2$

Ejercicio 5

Crea una función que calcule el área de un trianguno , conocida la longitud de la base (b) y la altura (b)

Recuerda que $A = \frac{b \cdot h}{2}$