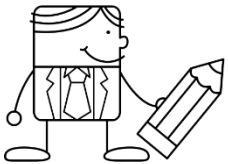
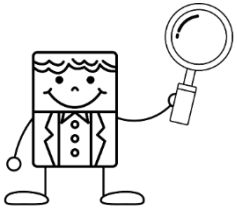
Introducción a Programación en R

- Dentro del tema encontrarás los epígrafes de contenido, con subepígrafes para una mejor comprensión.
- Todos los temas han sido elaborados siguiendo el criterio de facilidad de exposición.
- Los contenidos incluyen, en este sentido, cinco tipo de llamadas de atención que agilizan y facilitan la comprensión de los mismos:

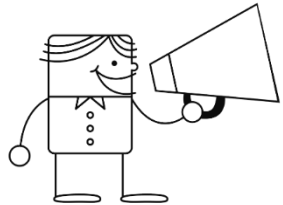


Ejemplo



Tenga en cuenta que….

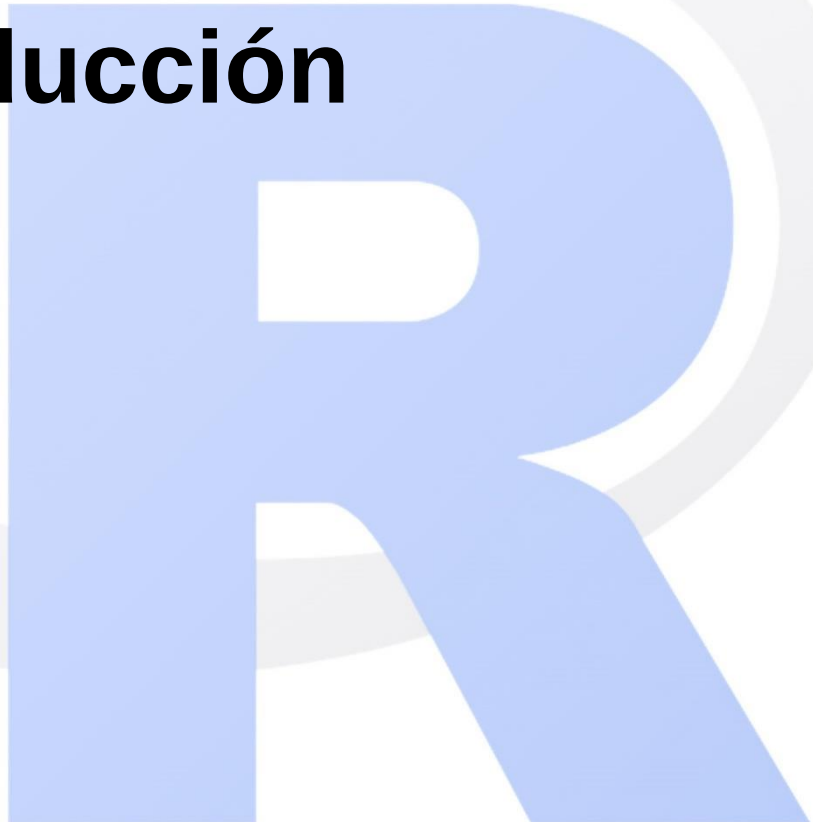
Recuerde



Definición

© Afi Escuela de Finanzas, S.A.

1 Introducción



¿Qué es R?

- R es un lenguaje estadístico *Open Source*.
- Es muy potente, flexible y extensible.
- Utilizado en muchas empresas (Google, Microsoft, Facebook, BBVA, etc…) y universidades.
- Empleado por estadísticos y *data miners* en el desarrollo de *software*.
- A diferencia de las hojas de cálculo tradicionales, en R se escriben sentencias de programación en lugar de las clásicas fórmulas. Es necesario conocer la estructura de los datos.
- Se pueden hacer prototipos con pocas líneas de código.

© Afi Escuela de Finanzas, S.A.

Historia de R

- R es una implementación del lenguaje estadístico **S** (combinado con el lenguaje de programación **Scheme**).

- S fue desarrollado en los laboratorios de AT&T por **John Chambers** a finales de los 70.

- Las dos principales implementaciones de S son:
  - R
  - S+ (S-PLUS)

- Suele haber varias *releases* al año (normalmente la más importante en Abril):
- 3.1.0 (Spring Dance) 10/04/2014
- …
  - 3.2.0 (Full of Ingredients) 16/04/2015
- …
  - 3.4.1 (Single Candle) 30/06/2017

© Afi Escuela de Finanzas, S.A.

Ventajas de R

- R es un gran *software* para resolver problemas de análisis de datos. Existen gran cantidad de paquetes para tratamiento de datos, modelización estadística, *data mining* y gráficos.

- Existe una comunidad de usuarios creando paquetes (extendiendo R). Gran parte de estos paquetes están implementados en el propio R.

- R es muy útil para hacer gráficos, analizar datos y obtener modelos estadísticos con datos que caben en la memoria RAM del PC. Existen limitaciones, desde el punto de vista de la memoria, con grandes volúmenes de datos.

- Es muy común utilizar otro lenguaje de programación para preparar los datos:
  - Volúmenes pequeños o medianos: Python, Perl…
  - Volúmenes grandes: Hadoop, Pig, Hive…

© Afi Escuela de Finanzas, S.A.

¿A qué nos referimos por R?

- Por R nos solemos referir a:
  - El lenguaje de programación.
  - El interprete que ejecuta el código escrito en R.
  - El sistema de generación de gráficos de R.
  - Al IDE de programación en R, o también conocido como **RStudio** (incluye el interprete de R, sistema de gráficos, gestor de paquetes e interfaz de usuario).

© Afi Escuela de Finanzas, S.A.

Instalación de R

- Proyecto R: https://www.r-project.org/
- Existen instaladores binarios para Windows, Linux y Mac: http://cran.es.r-project.org/
- En Linux, lo más cómodo es utilizar el gestor de paquetes:
  - sudo apt-get install r-base
  - sudo yum install r-base

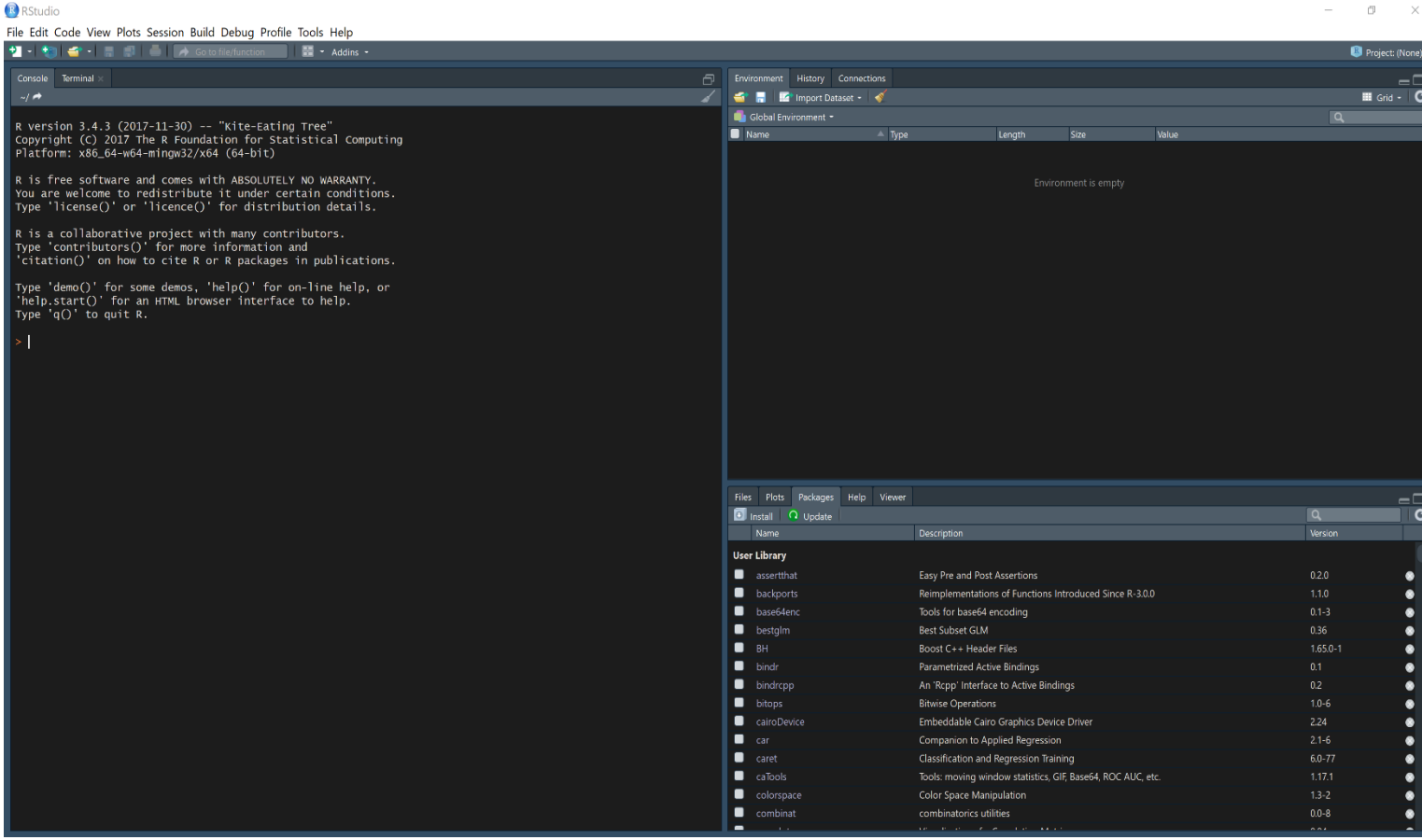
© Afi Escuela de Finanzas, S.A.

RStudio (I)

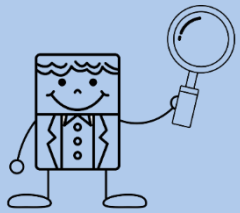
- IDE de programación para desarrollar proyectos en R: https://www.rstudio.com/
- Existen dos versiones:
  - RStudio Desktop
  - RStudio Server (interfaz de RStudio Desktop en versión web)
- Ambas versiones tienen versión open source (gratuita) y comercial (con soporte incluido).
- Permite la gestión completa de un proyecto de software:
  - Consola R
  - Gestión de ficheros
  - Ayuda
  - Gestión de paquetes (instalación, actualización, etc.)
  - Revisión del historial de comandos
  - …

© Afi Escuela de Finanzas, S.A.

RStudio (II)



© Afi Escuela de Finanzas, S.A.



Tenga en cuenta que este curso tiene un componente práctico muy fuerte. Es labor del alumno practicar todos los ejemplos y ejercicios planteados.

"A programar se aprende programando"

Llegados a este punto es imperativo tener R y RStudio instalados y funcionales.

© Afi Escuela de Finanzas, S.A.

2 R Basics

Expresiones, objetos y símbolos (I)

- El código R está compuesto de expresiones (expressions). Algunos ejemplos de expresiones:
  - Asignaciones
  - Sentencias condicionales
  - Operaciones aritméticas
  - …

- Las expresiones se componen de objetos y funciones. Cada expresión se separa de otra mediante una nueva línea o un punto y coma (;).

- El código R manipula objetos (objects). Algunos ejemplos de objetos:
  - Vectores
  - Listas
  - Funciones
  - …

© Afi Escuela de Finanzas, S.A.

Expresiones, objetos y símbolos (II)

- Formalmente los nombres de las variables en R se llaman **símbolos (*symbols*)**. Es decir, asignamos el objeto a un símbolo del entorno actual.

- El entorno está formado por el conjunto de símbolos presentes en un cierto contexto.

- Los objetos en R tienen modo, clase, y tipo:
  - **Modo (*mode*):** indica el modo de un objeto por R (como es almacenado según Becker, Chambers & Wilks, 1988).
  - **Tipo (*typeof*):** indica el tipo "interno" de un objeto R (como es almacenado). Es una versión actualizada del modo. Tipo y modo suelen coincidir pero no siempre es así.
  - **Clase (*class*):** indica la clase de un objeto R.

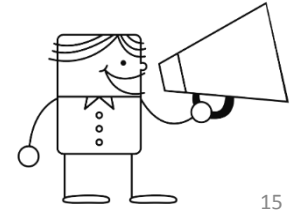
- Cuando se cambia el modo/tipo de un objeto se denomina **coerción**. Puede cambiar el modo/tipo, pero no necesariamente la clase. Es lo que usualmente se denomina *casting* en otros lenguajes de programación.

©2018, Escuela de Finanzas, S.A.

Tipos de datos básicos en R

Tipo básico	Descripción
logical	Valores booleanos TRUE/FALSE (pueden abreviarse a T/F).
integer	Números enteros (32 bit con signo).
double	Números reales (de doble precisión).
character	Cadenas de caracteres (strings). Los valores van entrecomillados.
complex	Números complejos.

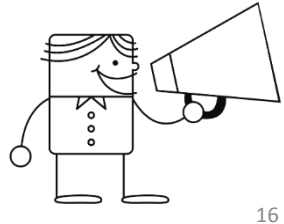
- Tanto el tipo *integer* como *double* tienen siempre modo *numeric*.



©2018, Escuela de Finanzas, S.A.

Objetos comunes en R

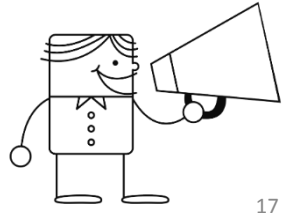
Tipo objeto	Descripción
vector	*Array* de una dimensión. Todos los elementos son del mismo tipo básico. Cada elemento puede tener nombre.
matrix	*Array* numérico de dos dimensiones. Las filas y las columnas pueden tener nombre.
factor	*Array* de datos ordinales (ordenados) o categóricos (sin orden).
data.frame	*Array* de dos dimensiones. Cada columna es un vector o factor. Las filas y las columnas pueden tener nombre.
list	*Array* de objetos. Los elementos de la lista pueden ser de diferentes tipos y tener nombre.



©2018, Escuela de Finanzas, S.A.

¿Cómo determinar la naturaleza de un objeto?

Función	Descripción
typeof(x)	El tipo del objeto x.
mode(x)	El modo del objeto x.
class(x)	La clase del objeto x.
storage.mode(x)	El modo de almacenamiento del objeto x.
attributes(x)	Los atributos de x. Por ejemplo: class y dim.
str(x)	Imprime un resumen de la estructura de x.
dput(x)	Escribe una representación ASCII del objeto x.



©2018, Escuela de Finanzas, S.A.

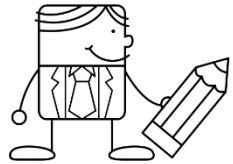
Valores no numéricos

- Existen ciertos valores especiales para representar situaciones determinadas:
  - **NULL:** se utiliza para indicar que una variable no contiene ningún objeto.
  - **NA:** valor especial para indicar que no hay datos en un determinado lugar (vector, matriz, *data frame*…)
  - **Inf:** infinito positivo.
  - **-Inf:** infinito negativo.
  - **NaN:** *Not a Number*.

- ***Tip*:** se puede tener una lista de NULLs pero no un vector de NULLs. En cambio sí se puede tener un vector de NAs.

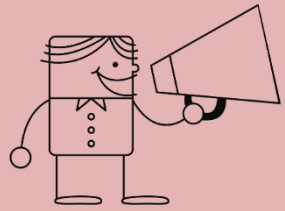
```
> x <- NULL
> is.null(x)
[1] TRUE
> length(x)
[1] 0
```

```
> y <- NA
> is.na(y)
[1] TRUE
> length(y)
[1] 1
```



©2018, Escuela de Finanzas, S.A.

Operador de Asignación



El operador más importante de R es el **operador de asignación:**

`<-`

Escrito tal cual con menor y guión,
o bien con el atajo de teclado:

`Alt + -`

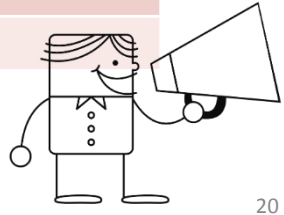
Permite asignar valores o definir relaciones entre variables.

© Afi Escuela de Finanzas, S.A.

Operadores básicos (I)

Operador aritmético	Descripción
+	Suma.
-	Resta.
*	Multiplicación.
/	División.
^ o **	Potencia.
%%	Módulo.
%/%	División entera.

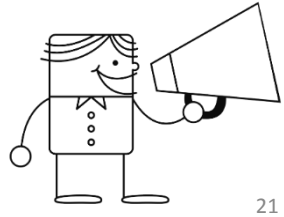
Otros operadores	Descripción
n:m	Generación de secuencias entre dos números.
%in%	Pertenencia.



©2018, Escuela de Finanzas, S.A.

Operadores básicos (II)

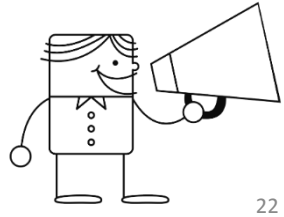
Operadores relacionales	Descripción
<	Menor que.
<=	Menor o igual que.
==	Igualdad.
>	Mayor que.
>=	Mayor o igual que.
!=	Desigualdad.



©2018, Escuela de Finanzas, S.A.

Operadores básicos (III)

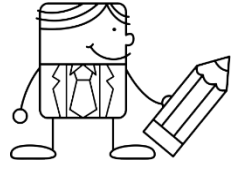
Operadores lógicos	Descripción
&	AND para vectores.
&&	AND para no vectores.
|	OR para vectores.
||	OR para no vectores.
!	NOT.



©2018, Escuela de Finanzas, S.A.

Sentencias condicionales

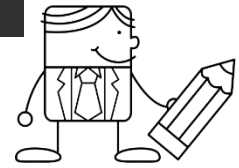
```r
# Sentencias condicionales
x <- -3
if (x < 0) {
  print("x es un número negativo")
}

x <- 5
if (x < 0) {
  print("x es un número negativo")
} else {
  print("x es un número postivo o cero")
}

x <- 5
if (x < 0) {
  print("x es un número negativo")
} else if (x == 0) {
  print("x es cero")
} else {
  print("x es un número postivo o cero")
}

ifelse(x > 0, "x es número positivo", "x es un número negativo")

x <- "size"
switch(x, red = "cloth", size = 5, name = "table")
```



©2018, Escuela de Finanzas, S.A.

Bucles

```r
# Bucles
ctr <- 1
while (ctr <= 7) {
  print(paste("ctr vale", ctr))
  ctr <- ctr + 1
}

ctr <- 1
while (ctr <= 7) {
  if (ctr %% 5 == 0)
    break
  print(paste("ctr vale", ctr))
  ctr <- ctr + 1
}

cities <- c("New York", "Paris", "London", "Tokyo", "Rio de Janeiro", "Cape Town")
for (city in cities) {
  print(city)
}

cities <- c("New York", "Paris", "London", "Tokyo", "Rio de Janeiro", "Cape Town")
for (city in cities) {
  if (nchar(city) == 6)
    next
  print(city)
}

cities <- c("New York", "Paris", "London", "Tokyo", "Rio de Janeiro", "Cape Town")
for (i in 1:length(cities)) {
  print(paste(cities[i], "está en la posición", i, "del vector de ciudades."))
}
```



©2018, Escuela de Finanzas, S.A.

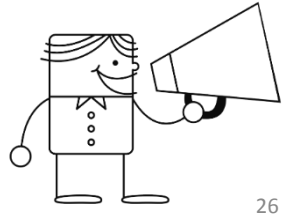
Control de errores

```
#####################
# Control de errores #
#####################
inputs = list(1, 2, 4, -5, "oops", 0, 10)

for(input in inputs) {
  print(paste("log of", input, "=", log(input)))
}

for(input in inputs) {
  try(print(paste("log of", input, "=", log(input))))
}

for(input in inputs) {
  tryCatch(print(paste("log of", input, "=", log(input))),
           warning = function(w) { print(paste("negative argument", input)); log(-input) },
           error = function(e) { print(paste("non-numeric argument", input)); NaN })
}
```

©2018, Escuela de Finanzas, S.A.

Constantes *built-in*

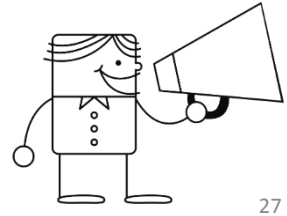
Constante	Descripción
LETTERS	26 letras del alfabeto en mayúsculas.
letters	26 letras del alfabeto en minúsculas.
month.abb	Abreviatura (3 letras) de los meses en inglés.
month.name	Meses en inglés.
pi	Ratio entre la circunferencia de un círculo y su diámetro.



©2018, Escuela de Finanzas, S.A.

Inspección de objetos (I)

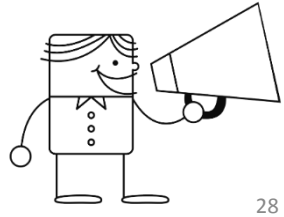
Función	Descripción
mode(x)	El modo del objeto x.
class(x)	La clase del objeto x.
typeof(x)	El tipo del objeto x.
dput(x)	Escribe una representación ASCII del objeto x.
str(x)	Imprime un resumen de la estructura de x.
summary(x)	Muestra un resumen de los estadísticos básicos de x.
head(x)	Muestra los primeros elementos del objeto x.
tail(x)	Muestra los últimos elementos del objeto x.
attributes(x)	Los atributos de x. Por ejemplo: class y dim.
is.null(x)	Comprueba si el objeto es NULL.



©2018, Escuela de Finanzas, S.A.

Inspección de objetos (II)

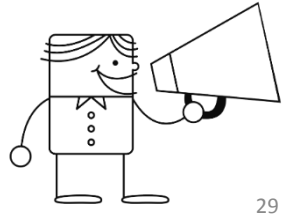
Función	Descripción
names(x)	Nombre de los elementos (para vectores y listas).
dimnames(x)	Devuelve una lista con los nombres de las filas y las columnas.
colnames(x)	Nombre de las columnas (para matrices y *data frames*).
rownames(x)	Nombre de las filas (para matrices y *data frames*).
dim(x)	Devuelve un vector con el número de filas y columnas del objeto.
nrow(x)	Número de filas del objeto.
ncol(x)	Número de columnas del objeto.



©2018, Escuela de Finanzas, S.A.

Inspección de objetos (III)

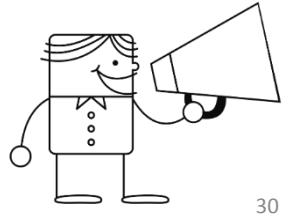
Función	Descripción
is.list(x)	TRUE o FALSE dependiendo de si es una lista o no.
is.factor(x)	TRUE o FALSE dependiendo de si es un factor o no.
is.complex(x)	TRUE o FALSE dependiendo de si es tiene el tipo básico *complex*.
is.character(x)	TRUE o FALSE dependiendo de si es tiene el tipo básico *character*.
is.matrix(x)	TRUE o FALSE dependiendo de si es un matriz o no.
is.numeric(x)	TRUE o FALSE dependiendo de si es tiene el tipo básico *numeric*.
is.integer(x)	TRUE o FALSE dependiendo de si es tiene el tipo básico *integer*.
is.vector(x)	TRUE o FALSE dependiendo de si es un vector o no.
is.data.frame(x)	TRUE o FALSE dependiendo de si es un *data frame* o no.
is.ordered(x)	TRUE o FALSE dependiendo de si está ordenado o no.



©2018, Escuela de Finanzas, S.A.

Entorno de trabajo

Función	Descripción
ls()	Devuelve la lista de variables del entorno.
rm(x)	Elimina una variable del entorno de trabajo.
help(x)/?x	Muestra la ayuda sobre una determinada función y/o paquete.
q()/quit()	Termina la sesión de trabajo.



©2018, Escuela de Finanzas, S.A.

Ejercicio 1

- Realiza el ejercicio del fichero *Ejercicio01_basics.R*

Es importante haber practicado y entendido todos los ejemplos anteriormente expuestos antes de realizar el ejercicio.

Intenta realizar los ejercicios y si te atascas tienes las soluciones a tu disposición.

©2018, Escuela de Finanzas, S.A.

3 Estructuras de datos

Vectores (*vector*)

- Los vectores son *arrays* de una única dimensión. Los índices del *array* van desde 1 hasta la longitud del vector, **length(v)**. Los vectores son también conocidos como vectores atómicos.

- Todos los elementos del vector son del mismo tipo básico:
  - *logical*
  - *integer*
  - *double*
  - *character*
  - *complex*

- Tiene un tamaño fijo que es fijado en su creación.

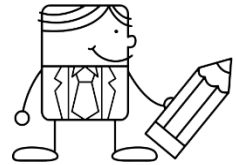
- La manera más simple de crear un vector es utilizando la función de combinación **c(v1, v2, …)**.

- Para dar nombre a los elementos de un vector con la función **names(v)**.

©2018, Escuela de Finanzas, S.A.

Creación de vectores

```
# Creación de vectores de longitud fija
v1 <- vector(mode = 'logical', length = 4)
v1
v2 <- vector(mode = 'integer', length = 4)
v2
numeric(4)
character(4)

# Usando el operador de secuencia
v3 <- 1:5
v3
v4 <- 1.4:5.4
v4
v5 <- seq(from = 0, to = 1, by = 0.1)
v5

# Usando la función de combinación
v6 <- c(TRUE, FALSE)
v6
v7 <- c(1.3, 7, 7/20)
v8
v9 <- c('black', 'white')
v9
v10 <- c(v1, v3)
v10

# Creación de vector nombres
v11 <- c(a = 1, b = 2, c = 3)
v11
```



©2018, Escuela de Finanzas, S.A.

Operaciones sobre vectores

- Cuando se realizan operaciones aritméticas entre dos vectores, R devuelve otro vector con los resultados de la operación elemento a elemento. También es posible realizar operaciones booleanas. La mayoría de las funciones y operaciones son "*vectorizadas*".

```
a_vector <- c(1, 2, 3)
b_vector <- c(4, 5, 6)

total_vector <- a_vector + b_vector
total_vector
```

- Si los vectores no tienen el mismo tamaño, R repite el menor de ellos tantas veces como sea necesario.

```
total_vector <- a_vector + 1
total_vector
```

- Podemos aplicar funciones a un vector. Por ejemplo: sum(v), max(v), mean(v)…

```
sum(total_vector)
max(total_vector)
mean(total_vector)
```

©2018, Escuela de Finanzas, S.A.

Indexación de vectores (I)

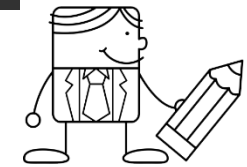
- Llamamos indexación a la selección de los elementos de un vector. Para hacer dicha selección empleamos los **corchetes [ ]**.

- Existen varias maneras de hacer selección:
  - Indexar con números positivos

```
# Indexando con números positivos
v[1] # Seleccionamos el primer elemento
v[c(1, 1, 4, 5)] # Seleccionamos el primero dos veces, el cuarto y el quinto
v[20:30] # Obtenemos los elementos entre el índice 20 y 30
v[70:100] <- 0 # Asignamos el valor cero a los elemenos entre los índices 70 y 100
v[which(v > 30)] # Seleccionamos los índices de los elementos > 30
```

  - Indexar con números negativos

```
# Indexando con números negativos
v[-1] # Seleccionamos todos menos el primer elemento
v[-c(1, 3, 4, 5)] # Seleccionamos todos menos el primero, el tercero, el cuarto y el quinto
v[-length(v)] # Todos menos el último
```



©2018, Escuela de Finanzas, S.A.

Indexación de vectores (II)

- Indexar con vectores lógicos o expresiones booleanas

```r
# Indexando con vectores lógicos o expresiones booleanas
v0 <- v[1:5]
v0[c(TRUE, FALSE, TRUE, FALSE, FALSE)] # Seleccionamos el primero y el tercero
v[v > 30] # Todos los > 30
v[v > 30 & v <= 50] # Todos los > 30 y <= 50
v[v == 0] # Todos los 0
v[v %in% c(10, 20, 30)] # Seleccionamos el 10, 20 y 30
```

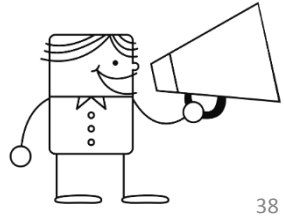
- Indexar por nombre

```r
# Indexando por nombre
names(v0) <- c("alpha", "beta", "gamma", "delta", "omega")
v0["alpha"]
v0["beta"] <- 500
v0[c("delta", "omega")]
v0[!(names(v0) %in% c("alpha", "beta"))]
```

©2018, Escuela de Finanzas, S.A.

Información sobre vectores

Función	Retorno
dim(df)	NULL.
is.atomic(v)	TRUE (si solo contiene elementos del mismo tipo).
is.vector(v)	TRUE (si es un vector).
is.list(v)	FALSE (si es una lista).
is.factor(v)	FALSE (si es un factor).
is.recursive(v)	FALSE (si contiene una estructura recursiva).
length(v)	Número de elementos en el vector.
names(v)	NULL o un vector de *characters* con los nombres de cada elemento.



©2018, Escuela de Finanzas, S.A.

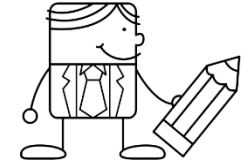
R realiza coerciones o cambios de tipo de variable de manera automática (por ejemplo de numérico a cadena de texto). Esto es cómodo pero puede inducir a error.

- Coerciones de tipos automáticas

```
x <- c(5, 'a') # Convierte 5 a '5'
x <- 1:3
x[3] <- 'a' # Convierte a '1', '2' y 'a'
typeof(1:2) == typeof(c(1, 2))
```

- Operaciones booleanas "*vectorizadas*" y no "*vectorizadas*"

```
c(T, F, T) && c(T, F, F) # TRUE !vectorizada
c(T, F, T) & c(T, F, F) # TRUE, FALSE, FALSE
```



©2018, Escuela de Finanzas, S.A.

Ejercicio 2

- Realiza el ejercicio del fichero *Ejercicio02_vectors.R*

Es importante haber practicado y entendido todos los ejemplos anteriormente expuestos antes de realizar el ejercicio.

Intenta realizar los ejercicios y si te atascas tienes las soluciones a tu disposición.

©2018, Escuela de Finanzas, S.A.

Matrices (*matrix*)

- Las matrices son *arrays* de dos dimensiones, los cuales contienen elementos del mismo tipo. Se organizan en filas y columnas.

- La manera más simple de crear una matriz es utilizando la función **matrix(data, nrow, ncol,byrow)**.
  - **data:** vector con los valores que tendrá la matriz.
  - **nrow:** número de filas deseadas.
  - **ncol:** número de columnas deseadas.
  - **byrow:** define si la matriz se llena por filas o por columnas.

- Igual que en los vectores, podremos dar nombres tanto a las filas como a las columnas con las funciones **rownames(m)** y **colnames(m)**, respectivamente.

©2018, Escuela de Finanzas, S.A.

Creación de matrices

```r
m1 <- matrix(1:9, byrow = TRUE, nrow = 3)
m1

m2 <- matrix(c(0, -1, 4)) # Crea una matriz con una columna
m2

d1 <- diag(3) # Crea una matriz diagonal 3x3
d1

d2 <- diag(c(1, 2, 3)) # Crea una matriz diagonal y asigna el vector como diagonal
d2

t_m1 <- t(m1) # Traspuesta de m1
e <- eigen(m1) # Lista con autovalores y autovectores
d <- det(m1) # Determinante de la matriz
```

©2018, Escuela de Finanzas, S.A.

Operaciones sobre matrices

- Podemos realizar operaciones aritméticas con matrices (sumas, restas, multiplicaciones…) de manera similar a como hemos hecho con los vectores.

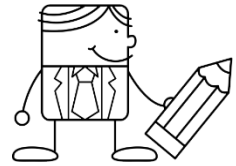
```
a_matrix <- matrix(1:9, byrow = TRUE, nrow = 3)
b_matrix <- matrix(11:19, byrow = TRUE, nrow = 3)

total_matrix <- a_matrix + b_matrix
total_matrix
```

- Igual que hicimos con los vectores, podemos realizar la misma operación sobre todos los elementos de la matriz. En este caso, R creará una matriz auxiliar con las mismas dimensiones que la matriz original.

```
total_matrix <- a_matrix + 2
total_matrix
```

- También aplicaremos funciones a una matriz. Por ejemplo: rowSums(m), colSums(m), rowMeans(m), colMeans(m)…

```
rowSums(total_matrix)
colMeans(total_matrix)
max(total_matrix)
```



©2018, Escuela de Finanzas, S.A.

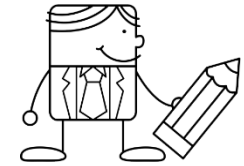
Manipulación de matrices

- Para añadir columnas a una matriz se utiliza la función **cbind(m1, m2, …)**, la cual une matrices y/o vectores por columna.

```
# Unión de matrices por columnas
big_matrix_2 <- cbind(a_matrix, b_matrix)
big_matrix_2

# Unión de matriz y vector por columnas
big_matrix_2 <- cbind(big_matrix_2, c(1, 5, 6))
big_matrix_2
```

- Para añadir filas a una matriz se utiliza la función **rbind(m1, m2, …)**, la cual une matrices y/o vectores por fila.

```
# Unión de matrices por filas
big_matrix_1 <- rbind(a_matrix, b_matrix)
big_matrix_1

# Unión de matriz y vector por filas
big_matrix_1 <- rbind(big_matrix_1, c(1, 5, 6))
big_matrix_1
```



©2018, Escuela de Finanzas, S.A.

Indexación de matrices (I)

- Al igual que en los vectores, utilizaremos los **corchetes [ ]** para indexar matrices. En el caso particular de las matrices usaremos dos números enteros: uno para la fila y otro para la columna **[row, column]**.

- Para seleccionar todos los elementos de una fila o una columna, basta con no incluir ningún número antes o después de la coma, respectivamente. Por ejemplo: matrix[row,], matrix[, col].

- Se puede hacer selección de varias maneras:
  - Indexar con números positivos

```
# Indexando con números positivos
m[1, ] # Seleccionamos la primera fila
m[1:2, ] # Seleccionamos las dos primeras filas
m[, 3] # Seleccionamos la última columna
m[, c(1, 3)] # Seleccionamos la primera y la última columna
m[1, ] <- 0 # Asigna un vector de ceros a la primera fila
```

©2018, Escuela de Finanzas, S.A.

Indexación de matrices (II)

- Indexar con números negativos

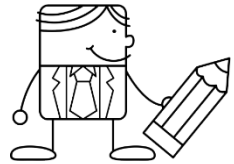
```
# Indexando con números negativos
m[-1, ] # Seleccionamos todas las filas menos la primera
m[-nrow(m), -ncol(m)] # Quitamos la última fila y la última columna
```

- Indexar con vectores lógicos o expresiones booleanas

```
# Indexando con vectores lógicos o expresiones booleanas
m_selection <- matrix(c(TRUE, FALSE, TRUE, TRUE, FALSE, TRUE, TRUE, FALSE, TRUE), byrow = TRUE, nrow = 3)
m[m_selection] # Seleccionamos en función de una matriz de booleanos
m[m > 7] # Todos los > 7
m[m == 0] # Todos los 0
```

- Indexar por nombre

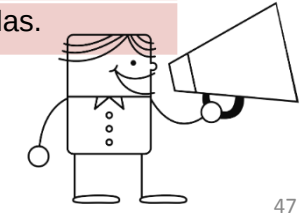
```
# Indexando por nombre
colnames(m) <- c("c1", "c2", "c3")
rownames(m) <- c("r1", "r2", "r3")
m[, c("c1", "c3")] # Selección de columnas por nombre
m[c("r2", "r3"), c("c1", "c2")] # Selección de filas y columnas por nombre
```



©2018, Escuela de Finanzas, S.A.

Información sobre matrices

Función	Retorno
dim(m)	Dimensiones de la matriz (número de filas y columnas).
is.atomic(m)	TRUE (si solo contiene elementos del mismo tipo).
is.vector(m)	FALSE (si es un vector).
is.list(m)	FALSE (si es una lista).
is.factor(m)	FALSE (si es un factor).
is.recursive(m)	FALSE (si contiene una estructura recursiva).
length(m)	Número de elementos en la matriz.
nrow(df)	Número de filas.
ncol(df)	Número de columnas.
colnames(m)	NULL o un vector de *characters* con los nombres de las columnas.
rownames(m)	NULL o un vector de *characters* con los nombres de las filas.



©2018, Escuela de Finanzas, S.A.

Ejercicio 3

- Realiza el ejercicio del fichero *Ejercicio03_matrix.R*

Es importante haber practicado y entendido todos los ejemplos anteriormente expuestos antes de realizar el ejercicio.

Intenta realizar los ejercicios y si te atascas tienes las soluciones a tu disposición.

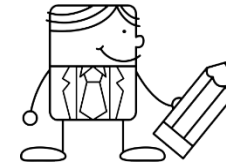
©2018, Escuela de Finanzas, S.A.

Arrays (*array*)

- Para crear un *array* de más de dos dimensiones utilizaremos la función **array(data, dim)**.
  - **data:** vector con los valores que tendrá el *array*.
  - **dim:** vector con las dimensiones del *array*.

- Los vectores y las matrices son casos especiales de *arrays*. Los vectores con una dimensión y las matrices con dos.

```
a <- array(1:8, dim=c(2, 2, 2))
a

m <- array(1:9, dim=c(3, 3))
m
```



©2018, Escuela de Finanzas, S.A.

Factores (*factor*)

- Los factores (*factors*) son un tipo de estructura de datos para almacenar variables categóricas.

- Recordatorio:
  - **Variable categórica:** aquella que puede tomar un número limitado de posibles valores (categorías). Ejemplos de variables categóricas: genero de una persona, nacionalidad…
    - Variable categórica nominal: aquella que no tiene un orden preestablecido.
    - Variable categórica ordinal: aquella que tiene un orden establecido.
  - **Variable continua:** aquella que puede tomar un número infinito de posibles valores. Ejemplos de variables continuas: peso de una persona, estatura de una persona…

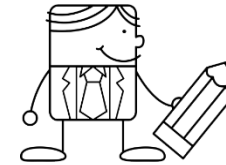
- A los distintos valores que puede tomar la variable se les denomina niveles, ***factor levels***.

- ¿Por qué usar factores?
  - Permiten establecer un orden distinto al alfabético.
  - Muchos modelos/paquetes de R los emplean.

©2018, Escuela de Finanzas, S.A.

Creación de factores

```r
# Creación de factor sin orden
gender_vector <- c('M', 'F', 'F', 'M', 'M', 'F')
gender_factor <- factor(gender_vector)
gender_factor

# Creación de factor con orden (pero sin especificar un orden)
size_vector <- c('S', 'L', 'M', 'L', 'S', 'M')
size_factor <- factor(size_vector, ordered = TRUE) # L < M < S
size_factor

# Creación de factor con orden (especificando el orden)
size_vector_2 <- c('S', 'L', 'M', 'L', 'S', 'M')
size_factor_2 <- factor(size_vector_2, ordered = TRUE, levels = c("L", "M", "S")) # L < M < S
size_factor_2

# Creación de factor especificando etiquetas
gender_levels_2 <- c('M', 'F', '-') # Como se leen los datos a la entrada
gender_labels_2 <- c('Male', 'Female', NA) # Como se etiquetan
gender_vector_2 <- c('M', 'F', 'F', 'M', 'M', '-')
gender_factor_2 <- factor(gender_vector, levels = gender_levels, labels = gender_levels)
gender_factor_2
```

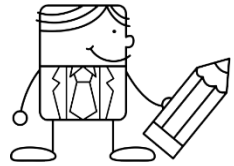


©2018, Escuela de Finanzas, S.A.

Operaciones sobre factores (I)

- No es posible realizar operaciones aritméticas con factores.
- Pero sí es posible realizar operaciones booleanas (comparaciones).

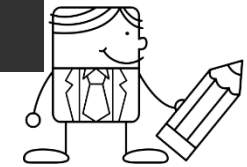
```
# Comprobaciones en factors sin orden (solo =)
gender_factor[1] == gender_factor[2]
gender_factor[1] == size_factor[2] # ERROR: solo se pueden comparar si son del mismo tipo

# Comprobaciones en factors con orden (se puede >, <...)
size_factor[1] > size_factor[2]
gender_factor[1] < gender_factor[2] # ERROR: solo se pueden comparar si son del mismo tipo
```



©2018, Escuela de Finanzas, S.A.

Operaciones sobre factores (II)

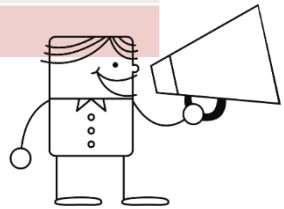
```r
# Obtener los niveles
levels(size_factor)
levels(size_factor)[1]

# Comprobar la existencia de niveles
any(levels(size_factor) %in% c('L', 'S'))

# Añadir nuevos niveles
levels(size_factor)[length(levels(size_factor)) + 1] <- 'XL'
levels(size_factor) <- c(levels(size_factor), 'XS')

# Reordenar niveles
levels(size_factor)
size_factor <- factor(size_factor, ordered = TRUE, levels(size_factor)[c(4, 1:3, 5)])

# Cambiar/re-nombrar niveles
levels(size_factor)[1] <- 'ExtraL'

# Eliminar niveles no utilizados
size_factor <- size_factor[drop = TRUE]
droplevels(size_factor)

# Unir factores
a <- factor(1:10)
b <- factor(letters[a])
union <- factor(c(as.character(a), as.character(b)))
cross <- interaction(a, b)
# ambos producen un conjunto no-ordenado de factors.
# levels: union: 20; cross: 100
# Items: union: 20; cross: 10
```



©2018, Escuela de Finanzas, S.A.

Información sobre factores

Función	Retorno
dim(f)	NULL.
is.atomic(f)	TRUE (si solo contiene elementos del mismo tipo).
is.vector(f)	FALSE (si es un vector).
is.list(f)	FALSE (si es una lista).
is.factor(f)	TRUE (si es un factor).
is.recursive(f)	FALSE (si contiene una estructura recursiva).
length(v)	Número de elementos en el factor.
is.ordered(f)	TRUE o FALSE (dependiendo de si esta ordenado o no).
mode(f)	“numeric”.
class(f)	“factor”.
typeof(f)	“integer”.



©2018, Escuela de Finanzas, S.A.

Traps sobre factores

- Al leer ficheros con *datasets* las cadenas de caracteres se convierten automáticamente a factores. Al utilizar **read.table** y **read.csv** usar el parámetro **stringsAsFactors = FALSE**.
- Si los números de un fichero se factorizan se puede revertir con: as.numeric(levels(f))[as.integer(f)]
- Evitar NA's en factores, suelen causar problemas.
- La coerción de un objeto a factor suele ser una fuente de problemas.

©2018, Escuela de Finanzas, S.A.

Ejercicio 4

- Realiza el ejercicio del fichero *Ejercicio04_factors.R*

Es importante haber practicado y entendido todos los ejemplos anteriormente expuestos antes de realizar el ejercicio.

Intenta realizar los ejercicios y si te atascas tienes las soluciones a tu disposición.

©2018, Escuela de Finanzas, S.A.

Data frame (*data.frame*)

- Como vimos antes, una matriz sólo puede contener elementos del mismo tipo. Debido a esta limitación surgen los ***data frames***.

- Normalmente un ***data frame*** contendrá un *dataset*, con las variables como columnas y las observaciones como filas.
  - Columnas: atributos, variables
  - Filas: observaciones, casos, instancias

- Internamente, R maneja los ***data frames*** como listas de vectores o factores, todas de la misma longitud.

- Podemos asimilarlos a una hoja de cálculo tradicional.

- Para crear *data frames* usaremos la función **data.frame(x, y, …)**, la cual recibe como parámetros las columnas del *dataset*.

©2018, Escuela de Finanzas, S.A.

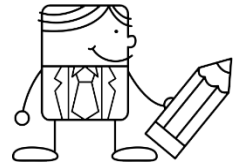
Creación de *data frames*

- Es posible crear *data frames* "a mano", pero no es la práctica habitual.

```r
# Creación de data frame vacío
empty <- data.frame()

# A partir de dos vectores
c1 <- 1:10 # vector de enteros
c2 <- letters[1:10] # vector de strings
df <- data.frame(col1 = c1, col2 = c2)
```

- Normalmente leeremos los *datasets* desde ficheros con las funciones read.csv(filename) o read.table(filename).

```r
# Lectura desde fichero
df <- read.csv('filename.csv', header = T)
```



©2018, Escuela de Finanzas, S.A.

Operaciones sobre *data frames*

- Análisis exploratorio de los datos de un *data frame*:
  - head(df): devuelve las primeras observaciones.
  - tail(df): devuelve las últimas observaciones.
  - str(df): muestra de forma rápida la estructura de la información almacenada.
    - Número total de observaciones.
    - Número total de variables.
    - Lista con todos los nombres de las variables.
    - El tipo de cada variable.
    - Las primeras observaciones de cada variable.
  - sumary(df): muestra los estadísticos básicos de cada variable.

```
head(mtcars)
head(mtcars, 10)
head(mtcars, -10)

tail(mtcars)
tail(mtcars, 10)
tail(mtcars, -10)

str(mtcars)

summary(mtcars)
```

©2018, Escuela de Finanzas, S.A.

Manipulación de *data frames*

- Para unir *data frames* la mejor manera es usar la función **rbind(df1, df2, …)**, obteniendo un *data frame* resultante con más filas.

```
# Añadir filas
df <- rbind(mtcars, data.frame(mpg = 22, cyl = 5, disp = 202, hp = 100, drat = 2.56, wt = 3.1,
                               qsec = 15, vs = 1, am = 0, gear =5, carb = 4, row.names=c("seat")))
```

- Para añadir columnas podemos emplear la función **cbind(df1, df2, …)**, pasando como parámetros otro *data frame* o un vector. También existen otras maneras de añadir columnas.

```
# Añadir columnas
df$newcolumn <- rep(1, nrow(df))
df[, 'copyofhp'] <- df$hp
df$hp.gear <- df$hp / df$gear
v <- 1:nrow(df)
df <- cbind(df, v)
```

©2018, Escuela de Finanzas, S.A.

Indexación de *data frames* (I)

- Al igual que sucedía con las matrices, utilizaremos los **corchetes [ ]** para indexar *data frames*. Emplearemos dos números enteros: uno para la fila y otro para la columna **[row, column]**.

- Podemos aplicar lo aprendido al indexar matrices para indexar *data frames*.

- Existe una manera rápida de seleccionar una columna, utilizando la expresión df$column ó df[“column”] ó df[1].

- Al indexar podemos obtener vectores o *data frames* dependiendo como lo hagamos.
  - Al seleccionar filas obtenemos siempre *data frames*.
  - Al seleccionar múltiples columnas obtenemos siempre *data frames*.
  - Al seleccionar columnas individuales podemos obtener *data frames* o vectores.

©2018, Escuela de Finanzas, S.A.

Indexación de *data frames* (II)

- Indexación de celdas

```r
# Indexando celdas
df <- data.frame(mtcars)
str(df)
df[5, 2] # Obtiene una única celda
df[1:5, 1:2] # Obtiene varias celdas
df[1:2, c('gear', 'am')]
df[1:2, c('gear', 'am')] <- 0 # Asignación de celdas
df[1:2, c('gear', 'am')]
```

- Indexación de filas

```r
# Indexando filas (siempre devuelve data frames)
df[1, ]
df[-nrow(df), ]
df[1:5, ]
df[(df$hp > 150 & df$hp < 200), ]
subset(df, hp > 150 & hp < 200)

vrow <- as.numeric(as.vector(df[1, ])) # Convertimos el resultados de la indexación en vector
```

©2018, Escuela de Finanzas, S.A.

Indexación de *data frames* (III)

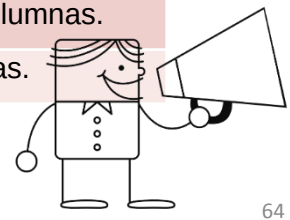
- Indexación de columnas

```r
# Indexando columnas
df$hp # Devuelve un vector
df[, "hp"] # Devuelve un vector
df[, 4] # Devuelve un vector
df["hp"] # Devuelve un data frame con una columna
df[4] # Devuelve un data frame con una columna
df[["hp"]] # Devuelve un vector
df[ , c(4, 6)] # Devuelve un data frame
df[ , c("hp", "wt")] # Devuelve un data frame
```

©2018, Escuela de Finanzas, S.A.

Información sobre *data frames*

Función	Retorno
dim(df)	Dimensiones de la matriz (número de filas y columnas).
is.atomic(df)	FALSE (si solo contiene elementos del mismo tipo).
is.vector(df)	FALSE (si es un vector).
is.list(df)	TRUE (si es una lista).
is.factor(df)	FALSE (si es un factor).
is.data.frame(df)	TRUE (si es un *data frame*).
is.recursive(f)	TRUE (si contiene una estructura recursiva).
class(df)	"data.frame".
nrow(df)	Número de filas.
ncol(df)	Número de columnas.
colnames(m)	NULL o un vector de *characters* con los nombres de las columnas.
rownames(m)	NULL o un vector de *characters* con los nombres de las filas.



©2018, Escuela de Finanzas, S.A.

Traps sobre *data frames*

- Al leer *data frames* desde ficheros normalmente se emplea el argumento **stringsAsFactors = FALSE** para evitar la coerción a factores.

- Se suele evitar emplear nombres en filas y utilizarlos sólo en columnas.

- No utilizar rbind(df1, df2, …) con factores, ya que algunas veces puede dar problemas (coerción).

©2018, Escuela de Finanzas, S.A.

Ejercicio 5

- Realiza el ejercicio del fichero *Ejercicio05_data_frames.R*

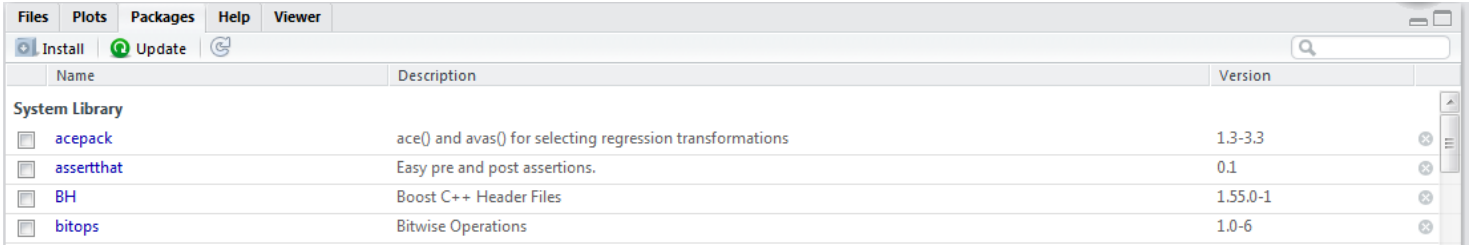
Es importante haber practicado y entendido todos los ejemplos anteriormente expuestos antes de realizar el ejercicio.

Intenta realizar los ejercicios y si te atascas tienes las soluciones a tu disposición.

©2018, Escuela de Finanzas, S.A.

Paquetes (Packages)

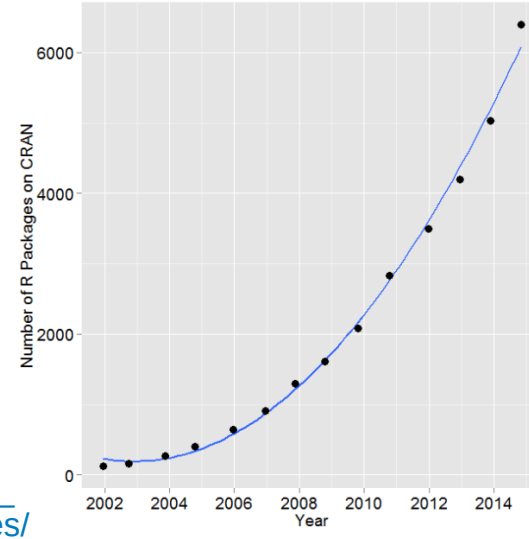
- Un paquete es una colección de funciones, datos y código compilado que han sido empaquetados juntos.
- Similares a las librerías de C/C++ o paquetes de clases de Java.
- Para utilizar un paquete hay que instalarlo y a continuación cargarlo:
  - install.packages("packageName")
  - library(packageName) o require(packageName)
- En RStudio se pueden gestionar los paquetes desde la pestaña Packages.



©2018, Escuela de Finanzas, S.A.

CRAN

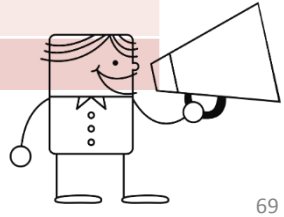
- Los paquetes de R se almacenan en un repositorio llamado **CRAN** (*Comprehensive R Archive Network*). Existen numerosas URLs para acceder al repositorio.
- Más de 6.000 paquetes disponibles.
- Paquetes por temática: https://cran.r-project.org/web/views/
- Paquetes por nombre: https://cran.r-project.org/web/packages/available_packages_by_name.html
- Paquetes por fecha de publicación: https://cran.r-project.org/web/packages/available_packages_by_date.html
- *Contributed Packages*: https://cran.r-project.org/web/packages/



©2018, Escuela de Finanzas, S.A.

Comandos sobre paquetes

Comando	Descripción funcionalidad
getOption("defaultPackages")	Lista de paquetes por defecto.
(.packages())	Lista de paquetes cargados en la sesión.
installed.packages()	Devuelve la información de los paquetes instalados actualmente.
available.packages()	Devuelve una lista de los paquetes disponibles en el repositorio.
old.packages()	Devuelve una lista de los paquetes que tienen versiones nuevas disponibles.
new.packages()	Devuelve una lista de los paquetes no instalados disponibles en el repositorio.
download.packages()	Descarga una lista de paquetes.
install.packages()	Instala una lista de paquetes desde el repositorio.
remove.packages()	Desinstala una lista de paquetes.
update.packages()	Actualiza los paquetes instalados a su última versión.
setRepositories()	Cambia la lista actual de repositorios.



©2018, Escuela de Finanzas, S.A.

Instalación de paquetes de otros repositorios

- No todos los paquetes están en CRAN.
- Con el paquete **devtools** se instalan paquetes desde otros repositorios populares (como por ejemplo: *Github*).
- Por ejemplo, **Hadley Wickman** utiliza *Github* para el desarrollo del ggplot2. Para instalar la última versión de desarrollo de ggplot2:
  - library(devtools)
  - install_github("ggplot2")

©2018, Escuela de Finanzas, S.A.

Otras funcionalidades avanzadas de R

Funciones

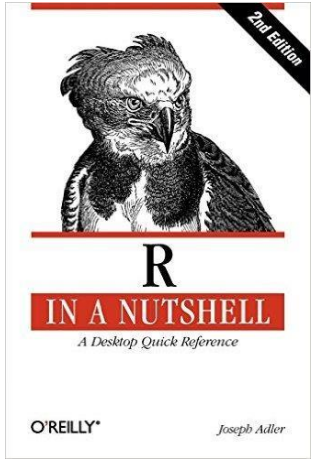
La familia apply

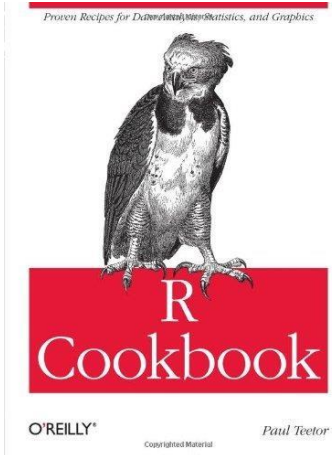
© Afi Escuela de Finanzas, S.A.

Dónde encontrar esto y más

- DataCamp: https://www.datacamp.com/
- R Programming: https://es.coursera.org/learn/r-programming

- R in a nutshell
- R cookbook





© Afi Escuela de Finanzas, S.A.