

# Fundamentos de programación en R: Conceptos generales



Lo primero que tienes que hacer es instalar R  
<https://cran.r-project.org/bin/windows/base/>



Y después, R Studio, que es donde vamos a programar.  
<https://www.rstudio.com/products/RStudio/#Desktop>

The screenshot shows the R Studio interface with three main panes:

- Script pane (green border):** Contains R code for generating a scatter plot of diamond pricing. It includes library imports, data loading, summary statistics, and plotting commands.
- Console pane (yellow border):** Shows the output of the R code, including summary statistics for the diamonds dataset and the generated plot.
- Plots pane (blue border):** Displays a scatter plot titled "Diamond Pricing" showing Price vs. Carat. The plot uses color to represent diamond clarity levels (I1, SI2, SI1, VS2, VS1, VVS2, VVS1, IF).

Para empezar, sitúate en el Script (recuadro verde) y escribe  
Bienvenida <- "Hola mundo"

## Conjunto de teclas útiles:

Ctrl + intro ejecuta las líneas que tengas seleccionadas. Una sola (en la que estás) si no has seleccionado nada.

Ctrl + L borra los resultados de la consola.

Ctrl + 1 o 2 cambia entre console y el script.

Ctrl + alt + R ejecuta todo el fichero.

Si estás escribiendo una Instrucción, el tabulador te ayudará a completarla, mostrándote las posibilidades existentes.

El cuadro verde es el entorno en el que vamos a programar. Por norma general, este recuadro estará cerrado. Para abrirlo ve a file/ new file/ R Script.

El cuadro amarillo es la Consola. Aquí aparecerán los resultados de la programación. Puedes escribir código directamente en la Consola, pero no se guardará lo que hagas.

En el cuadro rojo podrás ver las variables y funciones que hayas creado, así como su valor actual.

En el cuadro azul podrás indicar dónde estás trabajando (fichero), recorrer los gráficos que hayas creado, buscar e instalar funciones de terceros o usar la ayuda de R Studio.

### **Parar un programa / pedir ayuda**

The screenshot shows the RStudio interface with the following components:

- Script Editor:** Displays the R script "Ejercicio01\_Basic.R" which contains code for generating scenarios, calculating metrics like VAN and TIR, and creating a progress bar.
- Environment Pane:** Shows the global environment with variables like "combinaciones" (500 observations), "aleatorios" (a numeric vector from 1:500), and "combi\_ganado" (a numeric vector from 1:5).
- File Browser:** Shows the file structure under "Home".
- Console:** Displays the output of the R code, including the definition of a function "rdo\_escenario" and its execution.

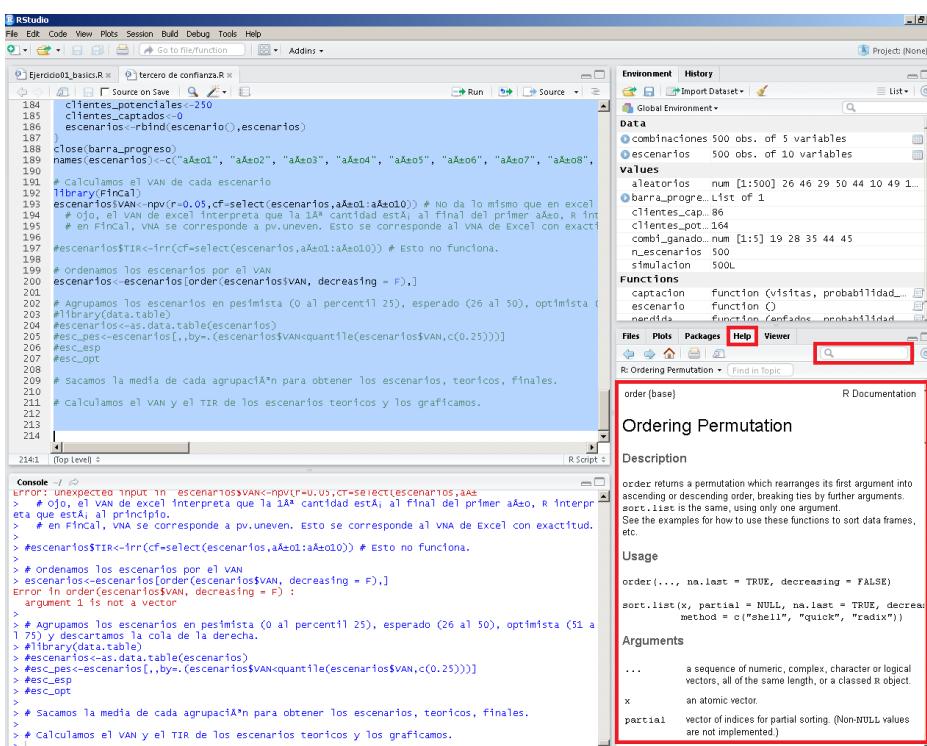
```
Rdoo escenario<-function(ano){  
  rdo_escenario<-c()  
  for (ano in 1:10){  
    rdo_escenario[ano] <- ingresos_ano[ano]-gastos_ano[ano]  
  }  
  
  rdo_escenario  
}  
  
##### Generamos 50.000 escenarios  
escenarios<-data.frame()  
n_escenarios <-500  
barra_progreso <- winProgressBar(title="Barra de progreso", min = 0, max = n_escenarios, width=3  
00)  
for (simulacion in 1:n_escenarios){  
  setwinProgressbar(barra_progreso, simulacion, title=paste(round(simulacion/n_escenarios*100,0),  
  "% realizado"))  
  c1entes_potenciales<-250  
  c1entes_captados<-0  
  escenarios<-rbind(escenario(),escenarios)  
}
```

¿Cómo puedo detener la ejecución de un programa?

En ocasiones, puedes querer detener la ejecución de un programa. Ya sea porque has entrado en un bucle infinito, o porque el programa está tardando demasiado en arrojar un resultado.

Para detener la ejecución basta con pulsar el botón rojo que aparece en la esquina superior derecha de la consola.

Ojo, este botón aparece únicamente cuando se está ejecutando un programa.



**No sé hacer algo, ¿qué puedo hacer?**

Acude al botón de ayuda y escribe allí tu pregunta. Obtendrás documentación muy completa sobre la función, argumentos y ejemplos prácticos.

También puedes escribir en la Console `help(función/duda)`. Se te mostrará la ayuda en la misma ventana de abajo a la derecha.

Si no encuentras lo que buscas, en internet está todo. Para buscar algo de R escribe: [R] duda.

Probablemente, lo que estás intentando programar ya lo ha hecho alguien anteriormente. Busca en Cran algún paquete que haga lo que quieras. Cran es un repositorio de funciones.

Si has localizado una función que hace lo que quieras, pero quieres profundizar en su código, escribe en la Console el nombre de la función, pero sin los paréntesis. Se abrirá el código de la misma para que puedas inspeccionarlo y seleccionar lo que quieras.

## Operadores básicos

+ suma	- resta	* multiplicar	/ dividir	^ o ** potencia	%% resto	%/% división entera
< menor	> mayor	<= menor o igual	>= mayor o igual	!= Distinto	== igual	

Asignar un valor a una variable

```
n <- 15  
22 -> h
```

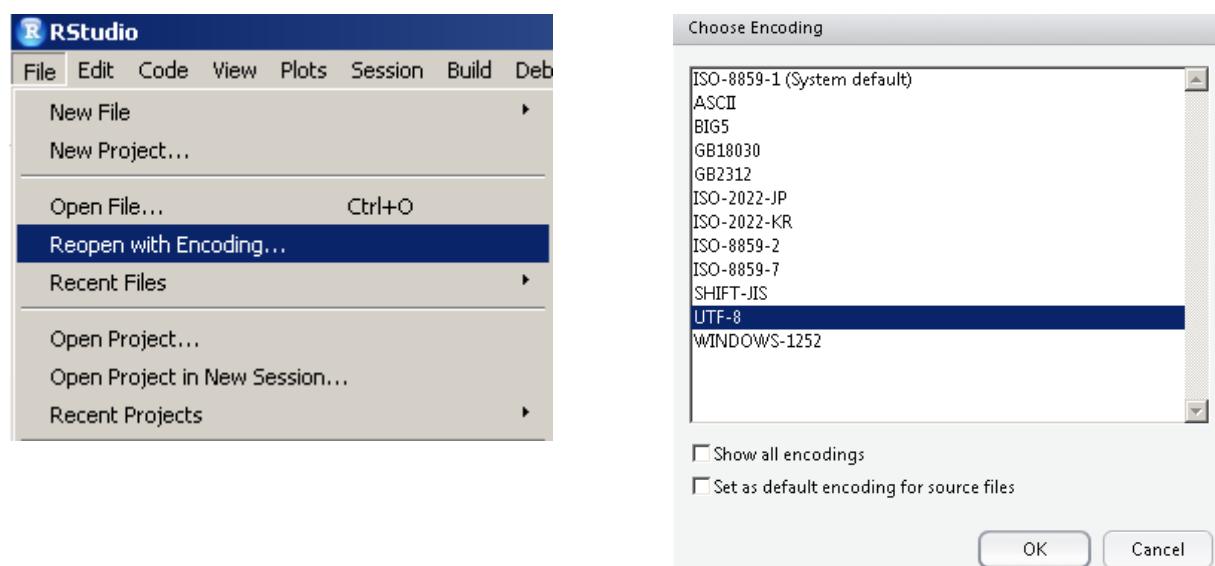
El punto y coma separa comandos en una misma línea. Es decir n<-15; 22->h

#Para realizar comentarios hay que usar la almohadilla.

Frase <- "Hola mundo" # para escribir una frase debemos ponerla entre comillas.

Frase<- "Para poder escribir comillas en mitad de una frase debemos usar \" así podemos no terminará la frase"

Cuidado cuando uses acentos en el código o la letra ñ. Lo más probable es que al volver a abrir el archivo, R no sabrá cómo interpretar esos caracteres. Para evitar tener problemas, guarda siempre todo lo que hagas con la codificación UTF-8. Y cuando abras un Script que no haya sabido interpretar los acentos o las ñ, ve a archivo, reopen with encoding, y selecciona UTF-8.



Limpiar entorno de trabajo. R guarda una imagen del proyecto con el que estás trabajando, así como de las variables y sus valores. Es muy probable que cuando vuelvas a abrir R te encuentres todo tal y como lo dejaste. Lo cual no es siempre deseable (especialmente cuando cambiamos de proyecto y solemos trabajar con el mismo nombre de variables).

```
rm(x,y) # borra los objetos x e y de la memoria.
```

```
rm(list=ls()) # borra todos los objetos que estén definidos. Es muy buena práctica escribir esta instrucción al inicio de los programas.
```

Otra manera de hacerlo es usar la brocha que aparece en el recuadro donde se guardan los objetos declarados (y sus valores).

## Ejemplos

**Creación y asignación de una variable**  
edades <- c(43, 42, 12, 8, 5)

**Para ver el contenido de una variable**  
edades

### Algunas funciones

```
sum(edades)  
mean(edades)  
range(edades)
```

### Operaciones básicas

```
1+2+3  
1+2*3  
(1+2)*5
```

### Operadores relacionales

```
TRUE == TRUE  
TRUE == FALSE  
"hello" == "goodbye"  
3 == 2  
TRUE != TRUE  
TRUE != FALSE  
"hello" != "goodbye"  
3 != 2  
5 >= 3  
3 >= 3
```

### Operadores relacionales sobre vectores

```
a <- c(16, 9, 13, 5, 2, 17, 14)  
a  
a > 10  
b <- c(17, 7, 5, 16, 8, 13, 14)  
a <= b
```

### Operadores lógicos

```
x <- 12  
x > 5 & x < 15  
  
y <- 4  
y < 5 | y > 15  
  
!(x < 5)
```

## Ejercicios

### Operadores aritméticos

# Suma 5 +5 y divide el resultado entre 2  
**(5 + 5) / 2**

# Calcula 2 elevado a 5  
**2^5**

# Calcula el resto de 28 entre 6  
**28%%6**

### Asignación de variables

# Asigna el valor 5 a la variable my\_apples  
**my\_apples <- 5**

# Muestra el contenido de la variable my\_apples  
**my\_apples**

# Asigna el valor 6 a la variable my\_pears  
**my\_pears <- 6**

# Muestra el contenido de la variable my\_pears  
**my\_pears**

# Suma peras con manzanas: crea una nueva variable my\_fruit con la suma  
**my\_fruit <- my\_apples + my\_pears**

# Muestra el número total de piezas de fruta  
**my\_fruit**

### Tipos básicos

# Declara una variable numérica, my\_numeric, con el valor 42  
**my\_numeric <- 42**

# Declara una variable de tipo texto, my\_character, con el valor "forty-two"  
**my\_character <- "forty-two"**

# Declara una variable booleana, my\_logical, con el valor FALSE  
**my\_logical <- FALSE**

# Sentencias y bucles

## Sentencia IF

Ejecuta un código u otro en función de si se cumple o no una condición. Esta estructura es muy útil a la hora de programar, ya que diriges el código en una u otra dirección.

```
x <- 5
if (x < 0) {
  print("x es un número negativo")
} else if (x == 0) {           #Fíjate que para poner igual tiene que poner dos (uno solo asigna valor, dos compara).
  print("x es cero")
} else {
  print("x es un número positivo o cero")
}
```

Se puede escribir la sentencia en una única línea, exactamente igual que en excel (condición, true, false).  
ifelse(x > 0, "x es número positivo", "x es un número negativo")

También es posible poner más de una condición,

```
if (condicion1 && condicion2 && condicion3){
  ...
}
```

&& es un And: todas las condiciones deben cumplirse.

|| es un Or: al menos una de las condiciones debe cumplirse. El símbolo se saca con Alt Gr + nº 1

## Bucle FOR

Utilizamos el bucle for para repetir un código un número determinado de veces.

A diferencia de otros lenguajes, como VBA, el bucle FOR no tiene porqué iterar sobre una secuencia de números (ej de 1 a 50), sino que puede iterar sobre los elementos de una lista, o un vector.

```
for (año in 1:10){ # Este sería un FOR típico, igual a como lo haríamos en VBA.
  print (año)
}
```

```
cities <- c("New York", "Paris", "London", "Tokyo", "Rio de Janeiro", "Cape Town") # Creamos un vector.
```

```
for (city in cities) {      # Recorremos los elementos del vector.
  if (nchar(city) == 6)    # Si el número de letras del vector es igual a 6...
    next                  # Ignoramos esa iteración.
  print(city)             # "Imprimimos el resultado"
}
```

```
for (elemento in 1:length(cities)) { # Recorremos el vector cities desde su posición 1 hasta el último elemento.
  print(paste(cities[elemento], "está en la posición", elemento, "del vector de ciudades."))
} # paste se usa para unir diferentes elementos en una sola frase.
```

Next: Es una instrucción que anula la iteración en la que se encuentra (y solo esa).

Break: Es una instrucción que nos permite terminar el bucle antes de tiempo.

## Bucle While

Utilizamos el bucle While para repetir un número de veces indeterminado un código en concreto. La repetición parará cuando se cumpla la condición que le indiquemos.

```
variable <- 1
while (variable <= 7) { # Repetir hasta que la variable sea mayor o igual a 7.
  if (variable == 5){
    break              # Sal del bucle.
  }
  print(paste("variable vale", variable))
  variable <- variable + 1
}
```

## Ejemplos

### 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 positivo 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 positivo o cero")
}

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

### Bucles

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

variable <- 1
while (variable <= 7) {
  if (variable == 5){
    break
  }
  print(paste("variable vale", variable))
  variable <- variable + 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."))
}
```

## Ejercicios

Recorre los elementos del vector y extrae los que empiecen por C  
numeros<-c("uno","dos","tres","cuatro","cinco","treinta")  
empiezan\_por\_c<-c()  
for (elemento in numeros){  
 if (substr(elemento,1,1)=="c"){  
 # Extrae una letra empezando por la 1<sup>a</sup> y acabando por la 1<sup>a</sup>  
 empiezan\_por\_c<-c(empiezan\_por\_c,elemento)  
 }  
}  
empiezan\_por\_c

Escribe un programa que encuentre todos los enteros entre 2000 y 3200 (ambos incluidos) que sean divisibles por 7 pero no múltiplos de 5  
numeros\_que\_cumplen<-c()  
for (numero in 2000:3200){  
 if (numero %% 7 ==0){  
 if (numero %% 5 !=0){  
 numeros\_que\_cumplen<-c(numeros\_que\_cumplen,numero)  
 }  
 }  
}  
numeros\_que\_cumplen

Escribe un programa que calcule el número de vocales en la frase "En un lugar de la Mancha, de cuyo nombre no quiero acordarme"  
frase<-"En un lugar de la Mancha, de cuyo nombre no quiero acordarme"  
frase<-tolower(frase)  
num\_vocales<-0  
for (letra in 1:nchar(frase)){  
 if (substr(frase,letra,letra) == "a" || substr(frase,letra,letra) == "e" ||  
 substr(frase,letra,letra) == "i" || substr(frase,letra,letra) == "o" ||  
 substr(frase,letra,letra) == "u"){  
 num\_vocales<-num\_vocales+1  
 }  
}  
num\_vocales

Otra manera de hacerlo sería simplificando el IF.  
frase<-"En un lugar de la Mancha, de cuyo nombre no quiero acordarme"  
vocales<-c("a","e","i","o","u")  
frase<-tolower(frase)  
num\_vocales<-0  
for (letra in 1:nchar(frase)){  
 if (substr(frase,letra,letra) %in% vocales){  
 num\_vocales<-num\_vocales+1  
 }  
}  
num\_vocales

## Crear e invocar funciones

Una función es un conjunto de líneas de código que se ejecutará cada vez que la invoquemos.

¿Cuándo debo hacer una función? Cuando el código vaya a ejecutarse más de una vez en el programa, deberías encapsularlo en una función.

```
do_something <- function(a, b = 1) { # La función recibe dos parámetros, pero solo el 1º es obligatorio.  
  return (a * b + a / b) # La función devuelve el parámetro return o la última instrucción ejecutada.  
} # Ojo, solo devuelve un parámetro (aunque puede ser una lista).
```

¿Cómo Invocamos una función?

```
do_something(4) # Llamamos a la función y le pasamos el parámetro obligatorio.  
k <- do_something(4, 3) # Llamamos a la función, le pasamos los parámetros y guardamos el rdo en una variable.  
k <- do_something(a=4, b=3) # Igual que antes, pero le pasamos los parámetros desordenados.
```

```
Funcion2 <- function(x) return(x^2) # Podemos construir una función en una sola línea.
```

Ojo, si ejecutamos una función, esta se declarará en la zona de las variables (estará cargada en memoria). Si cambiamos el código de la función deberemos volver a ejecutarla antes de llamarla. Si no lo hicieras, estarías invocando a la función anterior.

Entorno de función: Cuando creamos una función estamos creando un nuevo entorno. Esto significa que una variable definida dentro de una función no estará disponible fuera de la función (en entornos superiores, aunque sí en inferiores).

R pasa los argumentos por valor. Es decir, una función no puede alterar el valor de la variable que se pasa como argumento en la llamada (genera copias locales).

<<- Si asignamos así el valor dentro de una función estamos modificando el valor de la variable a nivel global (dentro y fuera de la función). Es posible hacerlo, pero se recomienda no usarlo.

## Sentencia Switch

La sentencia Switch se usa para seleccionar elementos dentro de una lista de alternativas.

El primer argumento que le pasamos es el elemento que deberá buscar dentro de la lista, devolviéndonos el valor asociado a dicho elemento.

Un ejemplo práctico sería construir la siguiente función.

```
centre<-function(x,type){ # Creamos una función que recibe dos parámetros.  
  switch(type, mean=mean(x), media=median(x)) # Usamos un switch para indicar qué queremos hacer.  
}  
  
x <- c(10,5,25) # Creamos un vector con varios elementos.  
centre(x, "mean") # Invocamos a la función, pasándole los parámetros.
```

## Ejemplos

### Ejemplo de función sencilla

```
do_something <- function(a, b = 1) {  
  return (a * b + a / b)  
}  
  
k <- do_something (a=4, b=3)
```

### Función que calcula la media o mediana, con los datos que le pases.

```
centre<-function(x,type){  
  switch(type, mean=mean(x), media=median(x))  
}  
  
x <- c(10,5,25)  
centre(x,"mean")
```

## Ejercicios

Escribe una función a la que se le pase una vector de números y calcule su media.

```
media_numeros<-function(numeros){  
  if (length(numeros) == 0){  
    return ("Has pasado una lista vacía")  
  }  
  return(sum(numeros)/length(numeros))  
}  
  
numeros<-c(2,4,6,8,10)  
numeros<-c()  
media_numeros(numeros)
```

Escribe una función que reconozca palíndromos. Un palíndromo es una palabra que se lee igual al derecho que al revés.

```
palindromo<-function(palabra){  
  palabra<-tolower(palabra)  
  capicua<-TRUE  
  for (letra in 1:nchar(palabra)){  
    if (substr(palabra,letra,letra)!=substr(palabra,nchar(palabra)-  
letra+1,nchar(palabra)-letra+1)){  
      capicua<-FALSE  
      break  
    }  
  }  
  if (capicua==TRUE){  
    return("Es un palíndromo")  
  } else {  
    return ("No es un palíndromo")  
  }  
}  
  
palabra<-"Anna"  
palindromo(palabra)
```

# Una manera fácil de hacerlo.

```
library("miscset")  
palindromo<-function(palabra){  
  palabra<-tolower(palabra)  
  if (palabra == str_rev (palabra)){  
    return ("Es un palíndromo")  
  } else {  
    return ("No es un palíndromo")  
  }  
}  
  
palabra<-"Anna"  
palindromo(palabra)
```

Escribe una función que calcule el IVA (21%) de un producto dado su precio de venta sin IVA y devuelva el precio total.

```
precio_total<-function(precio_sin_iva){  
  return(precio_sin_iva*1.21)  
}  
  
precio_sin_iva<-100  
precio_total(precio_sin_iva)
```

# Vectores

Los vectores únicamente tienen una dimensión. Van desde 1 hasta la longitud del vector `length(v)`.

Todos los elementos del vector deben ser del mismo tipo (logical, numeric, carácter...)

Tiene un tamaño fijo que es fijado en su creación (aunque puede modificarse sustituyendo el vector al completo).

## Creación de vectores

`X <- 1:30` genera un vector de nº de 1 a 30. El operador ":" tiene prioridad sobre otros operadores (sumar, dividir...)

`1:10-1` genera un vector de 0 a 9

`1:(10-1)` genera un vector de 1 a 9

`Seq(1, 5, 0.5)` # genera un vector de 1.0 a 5.0 con incrementos de 0.5

`C(1, 2, 3...)` # permite asignar valores a un vector directamente.

`Rep(1, 30)` # crea un vector de 30 valores iguales de valor 1.

`v5 <- seq(from = 0, to = 1, by = 0.1)` # Asignamos el vector a una variable.

`v11 <- c(a = 1, b = 2, c = 3)` # Ponemos nombres a los valores del vector.

## Operaciones con vectores

`total_vector <- v5 + v11`

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

`x<-1:4`

`y<-1:2`

`z<-x+y` # da 2 4 4 6

`sum(x)` # suma de elementos de x

`prod(x)`

`max()`

`min()`

`mean()`

`median()`

`var(x,y)` # Varianza

`cov()` # Covarianza

`cor()` # Correlación

## Indexación de vectores (recuperar valores)

`v=1:100`

`v[1]` # Seleccionamos / recuperamos el primer valor del vector.

`v[c(1, 1, 4, 5)]` # Seleccionamos el primero dos veces, el cuarto y el quinto

`v[-1]` # Seleccionamos todos menos el primer elemento.

`v[-length(v)]` # Seleccionamos todos menos el último elemento.

`v[-c(1, 3, 4, 5)]` # Seleccionamos todos menos el primero, el tercero, el cuarto y el quinto

`v > 30` # Construyendo un índice booleano

`v[v > 30]` # Aplicando un índice booleano

`which(v > 30)` # Localizando posiciones a través de un índice booleano

`v[which(v > 30)]` # Seleccionamos los elementos por posiciones

`v[v > 30 & v <= 50]` # Seleccionamos todos los elementos cuyo valor sea > 30 y <= 50

`v[v == 0]` # Seleccionamos todos los elementos cuyo valor sea 0

`v[v %in% c(10, 20, 30)]` # Seleccionamos los elementos cuyo valor sea 10, 20 y 30

`v[70:100] <- 0` # Asignamos el valor cero a los elementos entre los índices 70 y 100

`v0 = v[1:5]`

`names(v0) <- c("SAN", "TEL", "ACC", "IAG", "MAP")` # ponemos nombres a los valores del vector v0

`v0[c("SAN", "IAG")]` # Podemos seleccionar los elementos por su posición, por su valor o por su nombre...

`v0[!(names(v0) %in% c("TEL", "MAP"))]` #devuelve el nombre y valor de los elementos de v0 que no son alfa o beta

## Operadores relacionales

`&` Compara todo el vector.

`&&` Compara solo el primer elemento del vector.

`|` "Or", compara todo el vector.

`||` Compara solo el primer elemento.

`identical(x,y)` # Da true (compara vectores).

## Ejemplos

### Vectores: Creación

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

```
# Usando el operador de secuencia  
v3 <- 1:5  
v4 <- 1:4:5:4  
v5 <- seq(from = 0, to = 1, by = 0.1)
```

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

```
# Creación de vector nombres  
v10 <- c(a = 1, b = 2, c = 3)
```

### Vectores: Operaciones

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

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

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

### Indexando con números positivos

```
v <- 1:100  
v[1] # Seleccionamos el primer elemento  
v[c(1, 1, 4, 5)] # Seleccionamos el 1º dos veces, el 4º y el 5º  
v[20:30] # Obtenemos los elementos entre el índice 20 y 30  
v[70:100] <- 0 # Asignamos cero a los elementos entre 70 y 100  
v[which(v > 30)] # Seleccionamos las posiciones de los elementos > 30
```

### Indexando con números negativos

```
v[-1] # Seleccionamos todos menos el primer elemento  
v[-c(1, 3, 4, 5)] # Seleccionamos todos menos el 1º, el 3º, el 4º y el 5º  
v[-length(v)] # Todos menos el último
```

### Indexando con vectores lógicos o expresiones booleanas

```
v0 <- v[1:5]  
v0[c(TRUE, FALSE, TRUE, FALSE, FALSE)] # Seleccionamos el 1º y el 3º  
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
```

### 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"))]
```

## Ejercicios

### Crea dos vectores con las ganancias y pérdidas de la semana.

poker_vector	roulette_vector
# On Monday you won 140\$	# On Monday you lost 24\$
# Tuesday you lost 50\$	# Tuesday you lost 50\$
# Wednesday you won 20\$	# Wednesday you won 100\$
# Thursday you lost 120\$	# Thursday you lost 350\$
# Friday you won 240\$	# Friday you won 10\$

### Crea un vector con las ganancias en el póker de lunes a viernes

```
poker_vector <- c(140, -50, 20, -120, 240)
```

### Crea un vector con las ganancias en la ruleta de lunes a viernes

```
roulette_vector <- c(-24, -50, 100, -350, 10)
```

### Crea un vector con los días de la semana y asígnalo como nombre a los elementos de los vectores anteriores.

```
days <- c("Monday", "Tuesday", "Wednesday", "Thursday", "Friday")  
names(poker_vector) <- days  
names(roulette_vector) <- days
```

### Calcula el total de ganancias diarias.

```
total_daily <- poker_vector + roulette_vector
```

### Calcula el total de ganancias en el póker.

```
total_poker <- sum(poker_vector)
```

### Calcula el total de ganancias en la ruleta.

```
total_roulette <- sum(roulette_vector)
```

### Calcula el total de la semana.

```
total_week <- total_poker + total_roulette
```

### Selecciona las ganancias del miércoles en el póker.

```
poker_wednesday <- poker_vector[3]  
poker_wednesday <- poker_vector["Wednesday"]
```

### Selecciona las ganancias del martes, miércoles y jueves en el póker.

```
poker_midweek <- poker_vector[c(2, 3, 4)]  
poker_midweek <- poker_vector[c("Tuesday", "Wednesday", "Thursday")]
```

### Selecciona las ganancias desde el martes al viernes en la ruleta.

```
roulette_selection_vector <- roulette_vector[2:5]
```

### Calcula la media de ganancias en el póker el lunes, martes y miércoles.

```
P_LM= poker_vector[1:3]  
P_LM_G = P_LM[P_LM>0]  
Average = mean(P_LM_G)
```

### # En una línea

```
average <- mean(poker_vector[poker_vector[c("Monday", "Tuesday", "Wednesday")]>0])
```

### ¿Qué días de la semana hubo ganancias al póker?

```
selection_vector <- poker_vector > 0
```

### Haz la selección anterior sobre el vector con los datos del póker.

```
poker_winning_days <- poker_vector[selection_vector]  
poker_winning_days <- poker_vector[poker_vector > 0]
```

### Haz lo mismo sobre el vector con los datos de la ruleta.

```
roulette_winning_days <- roulette_vector[roulette_vector > 0]
```

# Matrices

Las matrices son arrays de dos dimensiones, las cuales contienen elementos del mismo tipo.

La manera más simple de crear una matriz es usando matrix(vector datos, nº filas, nº columnas, rellenar por filas = TRUE)

## Creación de matrices

```
m1 <- matrix(1:9, nrow = 3, byrow = TRUE) #rellena por filas  
m2 <- matrix(c(0, -1, 4)) # Crea una matriz con una columna  
m <- matrix(c(vector1, vector2, vector3), byrow=TRUE, nrow=3) # Ojo, los vectores deben tener la misma longitud.  
m1<- matrix(1, nr = 2, nc = 2) # Creamos una matriz de unos.  
m2<- matrix(2, nr = 2, nc = 2) # Creamos una matriz de doses.
```

## Indexar matrices

Matriz[filas,columnas] Para seleccionar se usan corchetes y hay que indicar dos dimensiones.

```
m[1:2, ] # Seleccionamos las dos primeras filas y todas las columnas.  
m[, c(1, 3)] # Seleccionamos todas las filas, la primera y tercer columna.  
m[-1, ] # Seleccionamos todas las filas menos la primera
```

# Poniendo nombres a las filas y columnas, e indexando por ellos.

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

m[-nrow(m), -ncol(m)] # Quitamos la Última fila y la Última columna (selecciono todos los valores menos estos).

¿Hemos eliminado los valores de esta manera? No, únicamente hemos realizado una selección.

¿Cómo podemos eliminar los valores? m<- m[-nrow(m), -ncol(m)]

m[1, ] <- 0 # Asigna un vector de ceros a la primera fila. Cambiamos los valores.

m[m > 7] # Selecciono todos los valores > 7. Fíjate que estamos usando una máscara booleana.

Si escribiese m>7 me devolvería una matriz de True y False...

m[m == 0] # Selecciono todos los valores iguales a 0.

## Manipular matrices

```
matrix2 <- cbind(a_matrix, b_matrix) # Unión de matrices por columnas.  
matrix2 <- cbind(matrix2, c(1, 5, 6)) # Unión de una matriz con un vector por columnas.  
matrix1 <- rbind(a_matrix, b_matrix) # Unión de matrices por filas.  
matrix1 <- rbind(matrix1, c(1, 5, 6)) # Unión de una matriz y un vector por filas.
```

## Operaciones con matrices

```
d <- det(m1) # Determinante de la matriz  
Matriz_resultante <- matriz + 2; o bien matriz - 5; matriz *3; matriz / 7  
rowSums(total_matrix)  
colMeans(total_matrix)  
max(matrix)  
min(matrix)  
Media <- mean(matriz[, 1]) # Calcula la media de la primera columna.
```

Suma y resta de matrices: Deben de tener la misma dimensión

Matriz\_resultante = matriz1 + matriz2

Producto de matrices:

Multiplicación elemento a elemento: Matriz\_resultante <- matriz1 \* matriz2

Ambas matrices deben de ser de igual dimensión.

Multiplicación matricial: Matriz\_resultante <- matriz1 %\*% matriz2

t(matriz) transpone una matriz.

diag(matriz) extrae la diagonal.

solve(matriz) invierte una matriz.

## Generar una matriz de números aleatorios

```
Matriz_aleatoria <- matrix(sample.int(50, replace = FALSE), 5, 5) # Genera una matriz de 25 elementos distribuidos en 5 filas y 5 columnas. Los números aleatorios son entre 1 y 50, sin reemplazamiento y con distribución uniforme.
```

# Ejemplos

## Matrices: Creación

```
m1 <- matrix(1:9, byrow = TRUE, nrow = 3)  
m2 <- matrix(c(0, -1, 4)) # Crea una matriz con una columna (xq si no indicas nada, se rellena por filas).  
d1 <- diag(3) # Crea una matriz diagonal 3x3  
d2 <- diag(c(1, 2, 3)) # Crea matriz diagonal, asigna vector a la diagonal.  
t_m1 <- t(m1) # Traspuesta de m1  
d <- det(m1) # Determinante de la matriz
```

## Matrices: Operaciones

```
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 <- a_matrix + 2
```

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

## Unión de matrices por columnas

```
big_matrix_2 <- cbind(a_matrix, b_matrix)
```

## Unión de matriz y vector por columnas

```
big_matrix_2 <- cbind(big_matrix_2, c(1, 5, 6))
```

## Unión de matrices por filas

```
big_matrix_1 <- rbind(a_matrix, b_matrix)
```

## Unión de matriz y vector por filas

```
big_matrix_1 <- rbind(big_matrix_1, c(1, 5, 6))
```

## Indexando con números positivos

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

## 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
```

## 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] # Indexamos usando la selección anterior.
```

```
m[m > 7] # Todos los > 7
```

```
m[m == 0] # Todos los 0
```

## 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 F y C por nombre.
```

# Ejercicios

Los siguientes vectores contienen la recaudación de las tres primeras películas de Star Wars en US y fuera de US (non-US)  
nueva\_esperanza <- c(461, 314.4)  
imperio\_contrataca <- c(290.5, 247.9)  
retorno\_jedi <- c(309.3, 165.8)

## Crea una matriz que contenga toda la información (con tres filas)

```
star_wars <- matrix(c(nueva_esperanza, imperio_contrataca, retorno_jedi),  
byrow=TRUE, nrow=3)
```

## Ponles nombres a las columnas: "US" y "non-US"

```
colnames(star_wars) <- c("US", "non-US")
```

## Ponles nombres a las filas: "Una nueva esperanza", "El imperio contraataca" y "El retorno del jedi"

```
rownames(star_wars) <- c("Una nueva esperanza ", " El imperio contraataca ", " El retorno del jedi ")
```

Si el precio de la entrada es de 5\$, estima el número de espectadores de cada película.

```
visitors <- star_wars / 5
```

## Como el precio de las entradas no es el mismo todos los años, creamos una matriz de precios

```
ticket_prices_matrix <- matrix(c(5, 5, 6, 6, 7, 7), nrow = 3, byrow = TRUE,  
dimnames = list(rownames(star_wars), colnames(star_wars)))
```

## Repite el cálculo del número de espectadores con la matriz anterior.

```
visitors <- star_wars / ticket_prices_matrix
```

## Calcula el número de espectadores medio en US

```
average_us_visitors <- mean(visitors[, 1])
```

## Calcula el número de espectadores medio fuera de US

```
average_non_us_visitors <- mean(visitors[, 2])
```

## Calcula los totales de recaudación por película

```
worldwide_vector <- rowSums(star_wars)
```

## Añade el vector anterior como una nueva columna de la matriz star\_wars.

```
all_wars_matrix <- cbind(star_wars, worldwide_vector)
```

## Crea una nueva matriz con las recaudaciones de las siguientes tres películas.

```
amenaza_fantasma <- c(474.5, 552.5)
```

```
ataque_clon <- c(310.7, 338.7)
```

```
venganza_sith <- c(380.3, 468.5)
```

```
star_wars2 <- matrix(c(amenaza_fantasma, ataque_clon, venganza_sith),  
byrow=TRUE, nrow=3)
```

## Ponles nombres a las columnas: "US" y "non-US"

```
colnames(star_wars2) <- c("US", "non-US")
```

## Ponles nombres a las filas: "Amenaza fantasma", "Ataque de los clones" y "Venganza Sith"

```
rownames(star_wars2) <- c("Amenaza fantasma ", " Ataque de los clones ", " Venganza Sith ")
```

Une en una nueva matriz la recaudación de todas las películas, las tres primeras filas corresponderán a las tres primeras películas y las tres siguientes a las últimas películas.

```
all_wars_matrix <- rbind(star_wars, star_wars2)
```

## Calcula los totales de recaudación de todas las películas en US y fuera de US

```
total_revenue_vector <- colSums(all_wars_matrix)
```

## Calcula la media recaudada de las tres primeras películas fuera de US

```
non_us_all <- mean(star_wars[, 2])
```

## Calcula la media recaudada de las 2 primeras películas fuera de US

```
non_us_some <- mean(star_wars[1:2, 2])
```



# Listas

Una lista es una colección de diferentes tipos de objetos.

Estos objetos pueden ser vectores, matrices, dataframes y otras listas.

Al igual que en los vectores, el índice de la lista va desde 1 hasta length().

No es posible realizar operaciones aritméticas sobre los elementos de una lista.

En ocasiones se puede simplificar una lista, convirtiéndola en un vector mediante la función unlist()

## Creación de listas

```
lista <- list(my_vector, my_matrix, my_df) # Creamos una lista.
```

```
lista2 <- list(vec = my_vector, mat = my_matrix, df = my_df) # Otra manera de hacer una lista.
```

```
lista3 <- c(lista1, lista2) # Podemos crear una lista uniendo otras dos.
```

## Manipulación de listas

```
lista <- c(lista, year = 1980) # Añade un nuevo elemento a lista (year = 1980).
```

```
lista[[length(lista)+1]]<-nuevo_elemento # Otra manera de añadir un elemento.
```

```
lista<- append(lista, nuevo_valor) # Añade un elemento. El parámetro after especifica la posición (por defecto length(x)).
```

```
l1<- append(l1, list(my_df), 1) # Añade un elemento en la posición 1.
```

```
Lista[[vector1]]<-c(vector1,nuevo_valor) # Modifica el contenido de un vector dentro de la lista.
```

Otra opción sería modificar el elemento y volver a asignárselo a la lista.

```
lista[[4]] <- NULL # Quita un elemento de la lista.
```

Otra opción sería seleccionar los elementos que queremos conservar y volver a asignarlo a la lista.

```
lista.recursiva <- list(list(my_vector, my_matrix), my_df) # Genera una lista recursiva, donde el primer elemento es una lista.
```

```
lista.recursiva[[1]][[1]] <- my_df # Modificar una lista dentro de una lista.
```

```
lista.recursiva[[1]][[1]] <- NULL # Elimina la lista dentro de la lista.
```

## Indexación de listas

```
lista[[3]]# Si indexamos de este modo, el resultado es un dataframe. Le estamos pidiendo todo el contenido del 3er elemento.
```

```
lista[3]# Si indexamos de este otro modo, lo que nos devuelve es una lista
```

```
lista[[1]][1, ] # Selecciona la primera fila del primer elemento de la lista. Ojo, dicho elemento debe de tener 2 dimensiones.
```

```
lista[[2]][3] # Selecciona el tercer elemento del segundo elemento de la lista. Dicho elemento debe tener solo 1 dimensión.
```

## Ejemplos

### Creación de listas sin nombre

```
my_vector <- 1:10 # Creamos un vector.
```

```
my_matrix <- matrix(1:9, ncol = 3) # Creamos una matriz.
```

```
my_df <- data.frame(my_vector) # Creamos un data frame.
```

```
l1 <- list(my_vector, my_matrix, my_df)
```

### Creación de listas con nombre

```
l2 <- list(vec = my_vector, mat = my_matrix, df = my_df)
```

### Utilizando la función de composición

```
l3 <- c(l1, l2)
```

### Operaciones con listas

```
str(l2) # Resumen del contenido.
```

```
head(l2) # Muestra los primeros elementos de la lista.
```

```
tail(l2) # Muestra los últimos elementos de la lista.
```

### Indexación, selección por índice

```
l2[2] # Devuelve una lista.
```

```
l2[[2]] # Devuelve una matriz.
```

### Indexación, selección por nombre

```
l2[["mat"]] # Devuelve una matriz.
```

```
l2$"mat" # Devuelve una matriz.
```

```
l2["mat"] # Devuelve una lista.
```

```
l2[["mat"]][1, ] # Selecciona la primera fila de la matriz.
```

```
l2[["vec"]][3] # Selecciona el tercer elemento del vector.
```

## Ejercicios

```
actors <- c("Jack Nicholson", "Shelley Duvall", "Danny Lloyd", "Scatman Crothers", "Barry Nelson")
```

```
scores <- c(4.5, 4.0, 5.0)
```

```
comments <- c("Best Horror Film I Have Ever Seen", "A truly brilliant and scary film from Stanley Kubrick", "A masterpiece of psychological horror")
```

```
reviews <- data.frame(scores, comments)
```

### Crea una lista que contenga los siguientes componentes:

1. moviename: "The Shining"

2. actors: el vector de actores

3. reviews: el data frame de reviews

```
shining_list <- list(moviename = "The Shining", actors = actors, reviews = reviews)
```

### Selecciona el último actor del vector de actores de la lista:

```
last_actor <- shining_list[["actors"]][5]
```

### Selecciona la segunda de las críticas del data frame de reviews de la lista:

```
second_review <- shining_list[["reviews"]][2, ]
```

### Añade un nuevo elemento a lista:

```
year: 1980
```

```
shining_list_full <- c(shining_list, year = 1980)
```

### Comprueba el contenido de la lista empleando la función str:

```
str(shining_list_full)
```

```
vector <- 1:10 | matriz <- matrix(1:9, ncol = 3)  
frase <- "En un lugar de la Mancha, de cuyo nombre no quiero acordarme"  
lista <- list(vector, matriz, frase)
```

Añade el nº 11 al vector. | Cambia el valor 9 de la matriz por un 22.  
lista[[1]]<-c(vector,11) | lista[[2]][3,3]<-22

Eliminar la frase de la lista.

## Ejercicio completo: Búsqueda local del primer mejor

El ejercicio consiste en la búsqueda del valor óptimo por minimización.

Se debe recorrer una matriz de valores, empezando por la esquina inferior derecha, y encontrar el valor mínimo “mirando” alrededor del valor actual.

8	7	6	7	5
5	3	4	5	9
6	2	7	6	7
5	1	8	7	9
3	0	8	8	9

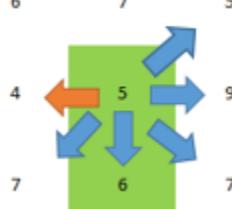
Matriz con la que trabajar y resultado deseado.

Se ha de mirar el entorno del valor en el que estamos. Dado que empezamos en la esquina inferior derecha, el entorno que tenemos es el siguiente:



Miramos en el sentido de las agujas del reloj, quedándonos con el primer mejor valor.

En este caso, iteramos analizando el entorno, hasta que, de nuevo, encontramos un valor que es mejor que el actual.



En el momento que no encontramos un valor mejor, paramos, y devolvemos un vector con los valores recorridos.

```
# Creamos el entorno
entorno<-matrix(c(8,7,6,7,5,5,3,4,5,9,6,2,7,6,7,5,1,8,7,9,3,0,8,8,9),nrow=5,ncol=5,byrow=TRUE)
```

```
#Inicializamos las variables de posición
fila_actual<-5
columna_actual <-5
```

```
# Creamos la función que recorrerá el entorno, buscando el primer mejor.
```

```
busqueda_primer_mejor<-function(entorno,fila_actual,columna_actual){

  valor_actual<-entorno[fila_actual,columna_actual]
  camino<-c(valor_actual)
```

```
# Creamos la matriz para mirar alrededor en el sentido de las agujas del reloj (la 1ª columna son las filas, la 2ª las columnas)
mirar<-matrix(c(-1,1,0,1,1,1,0,1,-1,0,-1,-1,-1,0),nrow=8,ncol=2,byrow=TRUE)
colnames(mirar)<-c("fila","columna")
```

```
#Buscamos en el entorno
```

```
hay_movimiento <-TRUE
while(hay_movimiento==TRUE){
  hay_movimiento<-FALSE
```

```
  for(alrededor in 1:8){ # Como máximo tendremos alrededor 8 posibles movimientos. Los recorremos.
    posible_fila <- fila_actual + mirar[alrededor,1]
    posible_columna <- columna_actual + mirar[alrededor,2]
```

```
  #Comprobamos que no nos hemos salido del entorno
```

```
  if (posible_fila <=5 && posible_fila >=1 && posible_columna <=5 && posible_columna >=1){
    valor_posible_movimiento <- entorno[posible_fila,posible_columna]
```

```
    # Si el valor de la casilla es mejor que el actual, nos movemos a esa casilla y actualizamos valores.
```

```
    if (valor_posible_movimiento < valor_actual){
      fila_actual <-posible_fila
      columna_actual <- posible_columna
      valor_actual <- valor_posible_movimiento
      camino<-c(camino,valor_actual)
      hay_movimiento<-TRUE
      break() # Salimos del bucle for, dado que ya hemos encontrado el primer mejor
    }
  }
```

```
# Mostramos el camino seguido
return(camino)
}
```

```
# Invocamos a la función
busqueda_primer_mejor(entorno, fila_actual, columna_actual)
```

## Data Frames

Una matriz solo puede contener elementos del mismo tipo (lógico, numérico, string, fecha). Debido a esta limitación surgen los Data Frame. Un Data Frame es una tabla donde las columnas son las variables y las filas observaciones. Cada columna puede ser de un tipo distinto, pero dentro de cada columna solo puede haber un tipo de dato.

### Crear un DF, añadir filas y columnas.

```
df <- data.frame(col1 = vector1, col2 = vector2) # Creación de un DF poniendo nombre a las columnas. Las filas no suelen tener nombre.  
df <- read.csv("filename.csv", header = T) # Creamos un DF leyendo los datos desde un fichero. Header=T el DF tiene nombre en las col.
```

```
df <- rbind(df, data.frame(col1 = 22, col2 = 5, col3 = 202, col4 = 4, row.names=c("seat"))) # Añadir una fila al final del DF.
```

```
df$newcolumn <- rep(1, nrow(df)) # Añadimos una columna creando un vector de unos, por cada fila del DF.
```

```
df[, 'copyofhp'] <- df$hp # Añadimos una columna copiando otra que ya existía (hp).
```

```
df$hp.gear <- df$hp / df$gear. # Añadimos una columna que es el rdo de una operación entre otras dos columnas.
```

```
df <- cbind(df, vector) # Añadimos una columna que es un vector (importante que el nº de filas se igual en el vecto y en el DF).
```

```
nombres <- c("Columna A", "Columna B", "Columna C");
```

```
colnames(dataSet)[5:7] <- nombres # Renombrar columnas usando el vector nombres que acabamos de crear.
```

```
Rownames(dataset) <- dataset[,1] # Poner nombre a las filas
```

### Indexar (ojo, si indexas una columna obtienes un vector, si indexas filas o combinaciones de celdas obtienes un DF)

```
df[5, 2] # Cómo indexar una celda.
```

```
df[1:5, 1:2] # Cómo indexar varias celdas.
```

```
df[1:2, c("col3", "col6")] <- 0 # Cómo asignar valores a celdas.
```

```
df[-nrow(df), ] # Selecciono todas las filas menos la última
```

```
df[(df$col5 > 150 & df$col5 < 200), ] # Obtengo as filas que, en la columna 5, son mayores de 150 y menores de 200.
```

```
df$hp # Extraemos una columna como un vector
```

```
df[, "hp"] o df[, 4] # Extraemos una columna como un vector.
```

```
df["hp"] o df[4] # Extraemos una columna como un DF.
```

```
indice_booleano <- df$col > mean(df$col) # Obtenemos un vector de T y F.
```

```
indice_numerico <- which(indice_booleano) # Al usar el índice booleano obtenemos un vector con los valores de True.
```

### Ordenar el Data Frame

```
positions <- order(df$col5, decreasing = TRUE) # Devuelve el índice con las posiciones. Ojo, no ordena el Df.
```

```
largest <- df[positions, ] # Al usar el índice, obtenemos el Df ordenado.
```

### Merge (Unión de DFs mediante una key)

```
join <- merge(df.x, df.y, by = c("col1")) # Crea un DF con los elementos comunes de X e Y en la col1.
```

```
left.join <- merge(df.x, df.y, by = c("col1"), all.x = T) # Crea un DF con todos los elementos de X y solo los coincidentes de Y.
```

```
right.join <- merge(df.x, df.y, by = c("col1"), all.y = T) # Crea un DF con todos los elementos de Y y solo los coincidentes de X.
```

```
full.join <- merge(df.x, df.y, by = c("col1"), all = T) #Fusiona todos los elementos de X e Y. En los no coincidentes pone NA.
```

# Ejemplos

## Creación de un Data Frame

```
empty <- data.frame() # Creamos uno vacío.  
c1 <- 1:10 # vector de enteros  
c2 <- letters[1:10] # vector de strings  
df <- data.frame(col1 = c1, col2 = c2) # Creamos uno a partir de 2 vec.  
df <- read.csv("filename.csv", header = T) # Lectura desde fichero
```

## Análisis exploratorio de un Data Frame

```
Análisis exploratorio de la base de datos mtcars.  
head(mtcars) # Devuelve las primeras observaciones.  
head(mtcars, 10)  
head(mtcars, -10) # Todas menos las 10 primeras.  
  
tail(mtcars) # Devuelve las últimas observaciones.  
tail(mtcars, 10)  
tail(mtcars, -10) # Todas menos las 10 últimas.  
  
str(mtcars) # Resumen de variables, tipo y ejemplo de datos.  
summary(mtcars) # Análisis estadístico de cada variable.
```

## Manipulación de Data Frames

```
Añade una fila al final del DF  
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")) # La nueva fila debe tener las mismas variables.
```

```
Añadir columnas al final del DF  
df$newcolumn <- rep(1, nrow(df)) # Creamos un vector de "unos"  
df[, 'copyofhp'] <- df$hp # Creamos una copia de la columna hp  
df$hp.gear <- df$hp / df$gear # Nueva columna de una operación.  
  
v <- 1:nrow(df) # Vector que va desde 1 hasta el nº de filas del DF  
df <- cbind(df, v)
```

## Indexación de Data Frames

```
Indexando celdas  
df <- data.frame(mtcars)  
df[5, 2] # Obtenemos una única celda  
df[1:5, 1:2] # Varias celdas, filas 1 a 5, columnas 1 a 2.  
df[1:2, c("gear", "am")] # filas 1 a 2, columnas por nombre.  
df[1:2, c("gear", "am")] <- 0 # Asignamos un valor a las celdas
```

```
Indexando filas (devuelve data frames)  
df[1, ] # Primera fila.  
df[-nrow(df), ] # Todas menos la última.  
df[1:5, ] # filas 1 a 5.  
df[(df$hp > 150 & df$hp < 200), ] # Devuelve las filas que cumplen.  
subset(df, hp > 150 & hp < 200) # Otra manera de hacerlo.
```

```
Podemos convertir el resultado de la indexación en vector.  
vrow <- as.numeric(as.vector(df[1, ]))
```

```
Indexando columnas (distintas maneras de hacer lo mismo).  
df$hp # Devuelve una columna como 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
```

## Unión de Data frames mediante una key: merge

```
c1 <- 1:10; c2 <- letters[1:10]; c3 <- 5:20; c4 <- letters[5:20]  
df.x <- data.frame(col1 = c1, col2 = c2)  
df.y <- data.frame(col1 = c3, col2 = c4)  
  
join <- merge(df.x, df.y, by = c("col1")) #join  
left.join <- merge(df.x, df.y, by = c("col1"), all.x = T) #left join  
right.join <- merge(df.x, df.y, by = c("col1"), all.y = T) #right join  
full.join <- merge(df.x, df.y, by = c("col1"), all = T) #full join
```

# Ejercicios

## Crea a partir de los siguientes vectores un Data Frame

```
planets <- c("Mercury", "Venus", "Earth", "Mars", "Jupiter", "Saturn",  
"Uranus", "Neptune")  
  
type <- c("Terrestrial planet", "Terrestrial planet", "Terrestrial planet",  
"Terrestrial planet", "Gas giant", "Gas giant", "Gas giant", "Gas giant")  
  
diameter <- c(0.382, 0.949, 1, 0.532, 11.209, 9.449, 4.007, 3.883)  
  
rotation <- c(58.64, -243.02, 1, 1.03, 0.41, 0.43, -0.72, 0.67)  
  
rings <- c(FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, TRUE, TRUE)  
  
planets_df <- data.frame(planets, type, diameter, rotation, rings)
```

Comprueba el contenido del Data Frame (nº variables y tipo).  
str(planets\_df)

Selecciona la información de los tres primeros planetas.  
closest\_planets\_df <- planets\_df[1:3, ]

Selecciona la información de los últimos tres planetas.  
furthest\_planets\_df <- planets\_df[6:8, ]

Selecciona la columna diameter de los últimos seis planetas.  
furthest\_planets\_diameter <- planets\_df[3:8, "diameter"]

Selecciona sólo los planetas que tienen anillos.  
planets\_with\_rings\_df <- planets\_df[planets\_df\$rings==TRUE, ]

Selecciona los planetas que tienen un diámetro inferior al de la tierra  
(aquellos que tienen diámetro < 1, la variable es relativa a la tierra)

```
indice <- planets_df$diameter < 1  
small_planets_df <- planets_df[indice, ]
```

Ordena el Data Frame según el diámetro de los planetas, ascendente, usando la función order.

```
positions <- order(planets_df$diameter, decreasing = FALSE)  
largest_first_df <- planets_df[positions, ]
```

# Manipulación de datos

Editor de datos (similar a una hoja de cálculo): `data.entry(x)` abre un editor donde se puede modificar directamente.

Tipos de datos básicos en R: Logical, numeric, character y date.

Tipos de objetos en R: Vector, matrix, factor, data.frame y list.

¿Cómo se puede analizar la naturaleza de un objeto?

```
Class(x)  
Attributes(x)  
Str(x) # imprime un resumen.  
Dim(x) o length(x) # Indica las dimensiones del data frame o longitud del vector.
```

Es posible convertir datos u objetos a otros tipos:

```
as.numeric(), as.logical(), as.character(),  
as.factor(), as.vector(), as.matrix(), as.list(), as.data.frame(),
```

## Funciones muy útiles

```
sort(x) # ordena el vector.
```

```
order(x) # devuelve un índice, pero no devuelve el vector ordenado.
```

```
str_rev(x) # Necesita library("miscset")
```

```
gsub("in", "adios", vector) # busca los in en el vector y los reemplaza por adiós.
```

```
paste("a", "b", "c", sep = ";") # concatena
```

```
strsplit(vector, ' ') # separa
```

```
grepl('in', vector) # Devuelve si la subcadena existe o no dentro de la cadena
```

```
any(x %in% c(1, 3, 5)) # ¿Alguno de estos valores están en x?
```

```
all(x %in% c(1, 3, 5)) # ¿Están todos estos valores en el vector?
```

```
which(c(48, 50, 1, 20, -9) %in% x) # busca valores en X y devuelve sus índices.
```

```
match(26, x) # Si existe el 26 en el vector te devuelve la posición, sino, NA
```

## Trabajar con fechas



```
today <- Sys.Date()  
now <- Sys.time()  
d <- as.Date("2016-03-17")  
d <- as.Date("17-03-2016", format = "%d-%m-%Y")  
d + 1 # Añade un día  
  
d2 <- as.Date("2015-03-17")  
d - d2 # Diferencia en días
```

# Ejemplos

## Funciones muy útiles

```
rep(NA, 10)  
append(1:20, c(1, 2, 3))  
seq(from = 5, to = 100, by = 5)  
x <- c(15, 26, 5, 9, 1, -9)  
sort(x)  
order(x)  
str_rev(x) # Necesita library("miscset")  
  
any(x %in% c(1, 3, 5))  
all(x %in% c(1, 3, 5))  
all(c(15, 9, 1) %in% x)  
which(c(48, 50, 1, 20, -9) %in% x)  
match(26, x)
```

## Funciones matemáticas

```
x <- seq(from = 5, to = 100, by = 5)  
is.na(x) # vector de T o F  
is.finite(x) # Vector de T  
is.infinite(x) # Vector de F  
abs(x)  
sqrt(x)  
log(x)  
log10(x)  
exp(x)  
ceiling(log(x)) # Redondeo al más cercano.  
floor(log(x)) # Redondeo hacia abajo.  
round(log(x), digits = 2) # Redondeo con dos dígitos.  
sin(x)  
cos(x)  
tan(x)  
sum(x)  
prod(x)  
cumsum(x)  
cumprod(x)
```

# Ejemplos

## Funciones sobre cadenas

```
s <- "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod  
tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim  
veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea  
commodo consequat. Duis aute irure dolor in reprehenderit in voluptate  
velit esse cillum dolore eu fugiat nulla pariatur."  
nchar(s) # Número de caracteres.  
toupper(s) # Todo mayúsculas.  
tolower(s) # Todo minúsculas.  
gsub("in", "adios", s) # Busca los in y los cambia por adios.  
substr(s, 7, 11) # Devuelve una subcadena, inicio – fin.  
substr(s, 7, 11) <- "IPSUM" # Modifica una subcadena.  
paste("a", "b", "c", sep = ";")  
strsplit(s, ' ') # Separa la cadena por espacios en blanco.  
grepl('dolor', s) # Devuelve si la subcadena existe o no dentro de la cadena
```

## library(stringr)

```
str_locate_all(pattern = 'dolor', s) # Devuelve posiciones de la subcadena
```

## Funciones sobre fechas

```
today <- Sys.Date()  
class(today)  
  
now <- Sys.time()  
class(now)  
  
d <- as.Date("2016-03-17")  
d <- as.Date("17-03-2016") # No da error pero lo hace mal  
d <- as.Date("17-03-2016", format = "%d-%m-%Y")  
  
t <- as.POSIXct("2016-03-17 22:32:00")  
t <- as.POSIXct("17-03-2016 22:32:00", format = "%d-%m-%Y %H:%M:%S")  
  
Cálculos con fechas
```



```
d + 1 # Añade un día  
d2 <- as.Date("2015-03-17")  
d - d2 #Diferencia en días.  
  
t + 1 #Añade un segundo  
t2 <- as.POSIXct("2015-03-17 22:32:00")  
t - t2 #Diferencia en días.  
  
unclass(d) # Da un entero: número de días desde el 1 Enero 1970.  
unclass(t) # Da un entero: número de segundos desde el 1 Enero 1970.
```

# Cómo importar y guardar datos

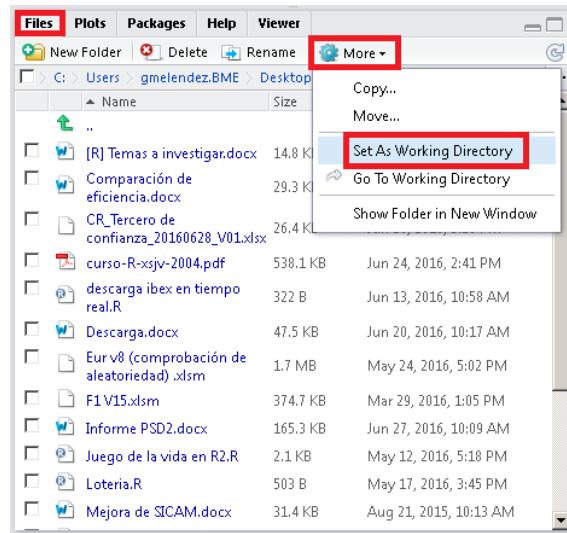
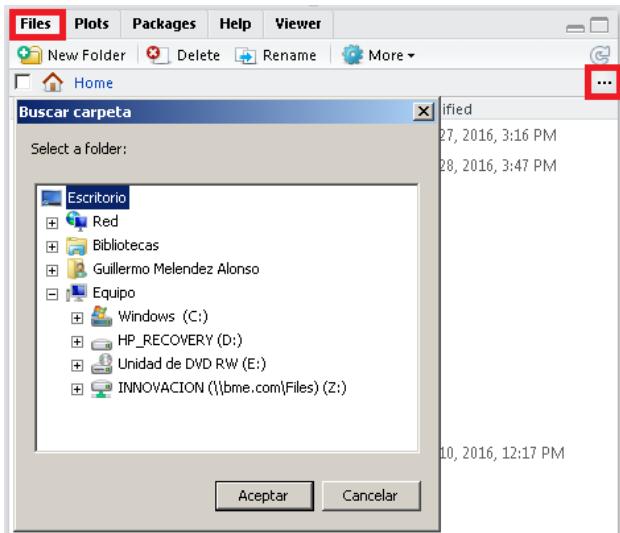
## Importar datos

Parámetro encoding: utf-8 (para que lea bien los acentos y la “ñ”).

Utils es la librería estándar de R, pero existen otras librerías que nos permiten importar ficheros.

	Utils	xlsx
txt	read.table()	
csv	read.csv()	
tsv	read.delim()	
Excel		read.xlsx

Para poder importar, lo primero que tenemos que hacer es establecer el directorio de trabajo. Establece, por ejemplo, el escritorio.



```
mun_csv_1 <- read.table("dat/municipios1.csv",
  header = TRUE,
  sep = ",",
  stringsAsFactors = FALSE, # No deseamos convertir el resultado en un factor (poner siempre esto).
  dec = ",",
  quote = "") #Especificamos las cadenas de caracteres
```

```
mun_csv_2 <- read.csv("dat/municipios1.csv", stringsAsFactors = FALSE)
mun_tsv_2 <- read.delim("dat/municipios2.tsv", stringsAsFactors = FALSE) # Lee TSV separados por tab.
```

```
library(xlsx)
read.xlsx(paste(getwd(),"/warrants.xls", sep = ""), sheetName="Warrants", startRow=6, endRow=NULL, , as.data.frame=TRUE,
header=TRUE, keepFormulas=FALSE, encoding="UTF-8")
```

## Guardar datos

write.table guarda el obj en un archivo (suele ser un data.frame, pero puede ser de cualquier tipo).

write.table(x, file=""). x es el nombre del objeto a guardar (vector, matriz...). Append True anexa los datos sin borrar los existentes.

```
write.table(MyData, file = "MyData.csv", row.names=FALSE, na="", col.names=FALSE, sep=",")
```

```
write.csv(MyData, file = "MyData.csv", row.names=FALSE, na="")
```

```
write.xlsx(DF_a_guardar, "nombre_fichero.xlsx") # Guardamos un excel.
```

```
Save(x, y, z, file ="xyz.RData") # Guardamos las variables X, Y y Z. Se pueden cargar con load("xyz.RData").
```



# Limpieza de datos

## Exploración de los datos

```
vinos <- read.csv("wine.csv", header = T)
class(vinos)      head(vinos)      hist(vinos$alcohol)
dim(vinos)        tail(vinos)      plot(vinos$alcohol, vinos$proline)
names(vinos)       print(vinos)
str(vinos)
summary(vinos)
```



## Preparación

```
library(stringr)
str_trim(" this is a test ") # Elimina los espacios en blanco al principio y al final de la frase.
str_pad("244493", width = 7, side = "left", pad = "0") # Lo transforma en un nº de 7 dígitos añadiendo un 0 a la izq.

is.na(df) # Busca los na del dataset y devuelve matriz de T y F.
any(is.na(df)) # si hay alguno devuelve T
sum(is.na(df)) # Cuenta los na

df[complete.cases(df), ] # Elimina las filas que tienen Na. Otra manera de hacerlo es na.omit(df)
```

## Ejemplos

vinos <- read.csv("dat/wine.csv", header = T) # Cargamos el dataset

### Exploración de la estructura

```
class(vinos)
dim(vinos)
names(vinos)
str(vinos)
summary(vinos)
```

### Exploración de los datos

```
head(vinos)
tail(vinos)
print(vinos)
```

### Visualizando los datos

```
hist(vinos$alcohol)
plot(vinos$alcohol, vinos$proline)
```

### Limpieza: preparación

```
library(lubridate) # Trabajando con formatos de fecha.
ymd("2015-08-25")
ymd("2015 August 25")
mdy("August 25, 2015")
hms("14:17:07")
ymd_hms("2015/08/25 13.33.09")
```

```
library(stringr) # Trabajando con string
str_trim(" this is a test ")
str_pad("244493", width = 7, side = "left", pad = "0")
names <- c("Sarah", "Tom", "Alice")
str_detect(names, "Alice")
str_replace(names, "Alice", "David")
```

### NAs

```
df <- data.frame(A = c(1, NA, 8), B = c(3, NA, 88), C = c(2, 45, 3))
```

### Detección

```
is.na(df) #Da una matriz de T o F si tiene na.
any(is.na(df))
sum(is.na(df))
summary(df)
```

### Eliminación

```
df[complete.cases(df), ] # elimina las filas con na.
na.omit(df) # elimina las filas con na.
```

### Outliers

```
set.seed(10) # Inicializamos la semilla para generar valores aleatorios
```

```
x <- c(rnorm(30, mean = 15, sd = 5), -5, 28, 35)
# Generamos 30 aleatorios y añade -5, 28...
```

```
boxplot(x, horizontal = T) # Lo graficamos para ver qué pinta tiene.
# Muestra una "caja" con el 1er y 3er cuartil, y la mediana
# Las mechas son 1,5 veces Q3 – Q1
# Los atípicos están fuera de las mechas
```

```
df2 <- data.frame(A = rnorm(100, 50, 100), B = c(rnorm(99, 50, 10), 500),
= c(rnorm(99, 50, 10), -1))
```

```
hist(df2$B, 20)
boxplot(df2)
```

## Dplyr

Es un paquete de manipulación de datos, creado por un usuario al que no le convencían las funciones de Dataframe.

Crea un nuevo objeto al que llama tabla (similar a un Data Frame): Permite una mejor visualización del contenido.

El paquete permite trabajar con los objetos Data Frame y tabla.

Para poder trabajar con Dplyr, primero debes instalarlo (packages/install) y luego invocarlo: library(dplyr)

tabla <- tbl\_df(datos)# Convierte los datos a una tabla.

glimpse(tabla) # El equivalente a str.

### Selección

hflights.select <- select(tabla, col1, col2, col7, col32)#1er parametro es tabla y el resto son las columnas seleccionadas.

select(hflights.tbl, colum6:colum12) # Selecciona desde la columna 6 a la 12.

select(hflights.tbl, -colum3:colum5) # Selecciono todas las variables menos aquellas entre las columnas 3 y 5.

select(hflights.tbl, starts\_with ("Delay")) # Selecciono las variables que empiezan con "Delay".

select(hflights.tbl, ends\_with("Time"), ends\_with("Delay")) # Selecciono las variables que terminan en "Time" o "Delay"

### Creación de columnas

seleccion <- mutate(hflights.select, loss = ArrDelay - DepDelay)# creamos una columna usando los datos de otras dos.

m1 <- mutate(hflights.tbl, loss = ArrDelay - DepDelay, loss\_percent = loss / DepDelay \* 100) # Creamos dos columnas.

hflights.tbl\$UniqueCarrier <- recode.carrier[hflights.tbl\$UniqueCarrier] #Estamos sustituyendo los datos de la columna UniqueCarrier por los datos un de vector llamado recode.carrier.

### Filtramos

hflights.filter <- filter(hflights.select, Cancelled == 1)#Seleccionamos las filas donde la columna Cancelled sean = 1

filter(hflights.tbl, Distance >= 1000) # Filtra los vuelos cuya distancia recorrida es mayor o igual a 1000.

filter(hflights.tbl, UniqueCarrier %in% c("JetBlue", "Southwest", "Delta")) # Filtra la columna UniqueCarrier, buscando los valores que sean JetBlue, Southwest o Delta.

filter(hflights.tbl, DepTime < 500 | ArrTime > 2200) # Filtra los vuelos que salieron antes de las 5am o llegaron después de las 10pm.

filter(hflights.tbl, Cancelled == 1 & DayOfWeek %in% c(6, 7)) # Filtra los vuelos cancelados en fin de semana.

### Ordenar

hflights.arrange <- arrange(hflights.select, DepDelay) # ordenamos por la columna DepDelay.

hflights.arrange <- arrange(hflights.select, DepDelay, ArrDelay)# ordenamos por dos columnas (de izq a der).

dtc <- filter(hflights.tbl, Cancelled == 1, !is.na(DepDelay)) ; arrange(dtc, DepDelay)

# Ordenamos por las columnas dtc y depdelay. Filtramos las filas que son = 1 de la columna Cancelled y quitamos las filas que contienen Na de la columna DepDelay.

### Summarise (permite usar funciones de agregación de R)

summarise(hflights.select, min = min(DepDelay), max = max(DepDelay), mean = mean(DepDelay), median = median(DepDelay)) # Obtengo los estadísticos usando la selección anterior.

summarise(filter(hflights.tbl, Diverted == 1), max\_div = max(Distance)) # En una sola instrucción, filtramos los vuelos desviados y calculamos la distancia máxima recorrida.

Summarise permite usar las siguientes funciones: min(x), max(x), mean(x), median(x), quantile(x), sd(x) desviación estándar, var(x), first(x), last(x), nth(x,n) es el enésimo elemento del vector x, n() nº de filas del DF o tabla, n\_distinct(x) nº de valores únicos de x.

### Agrupar filas Group by

hflights.group <- group\_by(hflights.tbl, UniqueCarrier) # Agrupamos valores (filas) por la columna UniqueCarrier.

## Pipes

Encadena llamadas a funciones, enviando la salida a la siguiente línea.

Ahorra espacio al no ser necesario declarar variables intermedias y facilita la lectura del código.

Con programación normal

```
Summarise(  
  Mutate(  
    Filter(  
      Select(tabla, x, y, z),  
      x > y),  
      Q = x + y + z),  
  All = sum(Q))
```

Con pipes

```
Tabla %>% # Indico con qué tabla trabajamos.  
Select(x, y, z) %>% # Selecciono las columnas.  
Filter (x > y) %>% # Filtro los datos que me interesan.  
Mutate(Q = x + y + z) %>% # Creo una nueva variable.  
Summarise (all = sum(Q)) # Extraigo la suma.
```

hflights.tbl %>% # Selecciono la tabla.

```
filter(!is.na(DepDelay)) %>% # Elimino los datos con NA (valor no disponible).  
summarise(min = min(DepDelay), max = max(DepDelay), mean = mean(DepDelay), median = median(DepDelay))
```

hflights.tbl %>% # Selecciono la tabla.

```
group_by(UniqueCarrier) %>% # Agrupo la información por compañías.  
summarise(n_flights = n(), # Obtendo el nº de vuelos.  
  n_canc = sum(Cancelled), # Calculo el nº de vuelos cancelados.  
  p_canc = mean(Cancelled) * 100, # Calculo el % de vuelos cancelados.  
  avg_delay = mean(ArrDelay, na.rm = TRUE)) %>% # Calculo el retraso medio de llegada quitando los NA.  
arrange(avg_delay, p_canc) # Ordena los resultados por el retraso medio y el porcentaje de vuelos cancelados
```

hflights.tbl %>%

```
filter(!(is.na(ArrDelay))) %>% # Elimino los NA  
group_by(UniqueCarrier) %>% # Agrupo por compañía  
summarise(p_delay = mean(ArrDelay > 0)) %>% # Calculo el retraso medio (pero solo de los vuelos con retraso).  
mutate(rank = rank(p_delay)) %>% # creo una variable nueva que es un ranking de los vuelos con retraso.  
arrange(rank) # ordeno el ranking.
```

# Ejemplos

## dplyr

```
Instala el paquete hflights y dplyr  
library(dplyr) # Invocamos a la librería ya instalada.  
library(hflights) # Información del aeropuerto de Houston.  
hflights.tbl <- tbl_df(hflights) # Convertimos los datos en una table.  
hflights.tbl # Muestra la información de una manera amigable.  
glimpse(hflights.tbl) # Muestra un resumen de la info.
```

## Select (selección de datos)

```
hflights.select <- select(hflights.tbl, ActualElapsedTime, AirTime,  
ArrDelay, DepDelay) # El primer parámetro es la tabla y el resto las  
columnas seleccionadas.
```

## Mutate (creación de variables)

```
hflights.mutate <- mutate(hflights.select, loss = ArrDelay - DepDelay)  
# Usando el resultado del select, creamos una nueva variable "loss",  
restando dos columnas (ArrDelay y DepDelay)
```

## Filter (selección de datos)

```
hflights.select <- select(hflights.tbl, starts_with("Cancel"), DepDelay)  
# Seleccionamos las columnas cuyo nombre empieza por "Cancel" y  
la columna DepDelay.
```

```
hflights.filter <- filter(hflights.select, Cancelled == 1) #  
Seleccionamos las filas que sean = 1 de la columna Cancelled.
```

## Arrange (ordenación de datos)

```
hflights.select <- select(hflights.tbl, TailNum, contains("Delay")) #  
Seleccionamos las filas que contengan "Delay" de la columna  
TailNum.
```

```
hflights.arrange <- arrange(hflights.select, DepDelay) # Ordena la  
selección por la columna DepDelay.
```

```
hflights.arrange <- arrange(hflights.select, DepDelay, ArrDelay) #  
Ordena por dos columnas (izq 1º).
```

## Summarise (estadísticos)

```
hflights.summarise <- summarise(hflights.select, min =  
min(DepDelay), max = max(DepDelay), mean = mean(DepDelay),  
median = median(DepDelay)) # Obtengo los estadísticos de la  
selección.
```

## Group by (agrupar filas)

```
hflights.group <- group_by(hflights.tbl, UniqueCarrier) # Agrupa las  
filas por la columna UniqueCarrier.
```

```
hflights.summarise.group <- summarise(hflights.group, avgDep =  
mean(DepDelay, na.rm = T), avgArr = mean(ArrDelay, na.rm = T)) #  
Sacamos los estadísticos de la agrupación, quitando los "NA".
```

## Pipes (encadena llamadas a funciones)

Pasa el resultado a la siguiente fila / función.

```
hflights.tbl %>%  
filter(!is.na(DepDelay)) %>%  
summarise(min = min(DepDelay), max = max(DepDelay), mean =  
mean(DepDelay), median = median(DepDelay))
```

Indicamos la tabla con la que trabajamos.

Realizamos un filtro eliminando los Na.

Sacamos los estadísticos.

# Ejercicios

**Carga la librería de vuelos del aeropuerto de Houston, convíerte la información en una tabla y guárdala en una variable.**

```
library(hflights)  
library(dplyr)  
hflights.tbl <-tbl_df(hflights)
```

**Recodifica la variable UniqueCarrier a partir del vector recode.carrier**  
recode.carrier <- c("AA" = "American", "AS" = "Alaska", "B6" = "JetBlue",  
"CO" = "Continental", "DL" = "Delta", "OO" = "SkyWest", "UA" = "United",  
"US" = "US\_Airways", "WN" = "Southwest", "EV" = "Atlantic\_Southeast",  
"F9" = "Frontier", "FL" = "AirTran", "MQ" = "American\_Eagle", "XE" =  
"ExpressJet", "YV" = "Mesa")

```
hflights.tbl$UniqueCarrier <- recode.carrier[hflights.tbl$UniqueCarrier]
```

**Recodifica la variable CancellationCode a partir del vector recode.cancellation**

```
hflights.tbl[hflights.tbl$CancellationCode == "", "CancellationCode"] <- "E"  
recode.cancellation <- c("A" = "carrier", "B" = "weather", "C" = "FFA", "D" =  
"security", "E" = "not cancelled")
```

```
hflights.tbl$CancellationCode <-  
recode.cancellation[hflights.tbl$CancellationCode]
```

**Selecciona las variables: ActualElapsedTime, AirTime, ArrDelay, DepDelay**  
select(hflights.tbl, ActualElapsedTime, AirTime, ArrDelay, DepDelay)

**Selecciona las variables desde Origin a Cancelled**  
select(hflights.tbl, Origin:Cancelled)

**Selecciona todas las variables menos aquellas entre DepTime y AirTime**  
select(hflights.tbl, -DepTime:-AirTime)

**Selecciona todas las variables que terminan en "Delay"**  
select(hflights.tbl, ends\_with("Delay"))

**Selecciona las variables que terminan en "Num" o empiezan por "Cancel"**  
select(hflights.tbl, ends\_with("Num"), starts\_with("Cancel"))

**Selecciona las que terminan en "Num" y la variable UniqueCarrier**  
select(hflights.tbl, UniqueCarrier, ends\_with("Num"))

**Crea la variable ActualGroundTime como la resta de ActualElapsedTime y AirTime. Guarda el resultado en g1.**

```
g1 <- mutate(hflights.tbl, ActualGroundTime = ActualElapsedTime - AirTime)
```

**Añade a g1 una nueva variable GroundTime como la suma Taxiln y TaxiOut**  
g2 <- mutate(g1, GroundTime = Taxiln + TaxiOut)

**Crea dos variables a la vez y guardalas en m1:**

```
loss como la resta de ArrDelay y DepDelay  
loss_percent como el ratio de ArrDelay - DepDelay entre DepDelay  
m1 <- mutate(hflights.tbl, loss = ArrDelay - DepDelay, loss_percent =  
(ArrDelay - DepDelay) / DepDelay * 100)
```

**Filtrar los vuelos con distancia recorrida mayor o igual que 3000**  
filter(hflights.tbl, Distance >= 3000)

**Filtrar los vuelos de JetBlue, Southwest, o Delta**  
filter(hflights.tbl, UniqueCarrier %in% c("JetBlue", "Southwest", "Delta"))

**Filtrar los vuelos cuyo tiempo en Taxi fue mayor que el tiempo de vuelo.**  
filter(hflights.tbl, TaxiOut + Taxiln > AirTime)

**Filtrar los vuelos que salieron con retraso y llegaron a tiempo**  
filter(hflights.tbl, DepDelay > 0 & ArrDelay < 0)

**Filtrar los vuelos cancelados después de haber sido retrasados**  
filter(hflights.tbl, Cancelled == 1, DepDelay > 0)

**Ordena la tabla dtc según el retraso de salida**

```
dtc <- filter(hflights.tbl, Cancelled == 1, !is.na(DepDelay))  
arrange(dtc, DepDelay)
```

**Ordena los vuelos según compañía y retraso de salida descendente**  
arrange(hflights.tbl, UniqueCarrier, desc(DepDelay))

**Ordena los vuelos según el retraso total**

```
arrange(hflights.tbl, ArrDelay + DepDelay)
```

Filtra los vuelos con destino DFW y hora de salida anterior a las 8am, ordenalos según el tiempo en vuelo descendenteamente.

```
arrange(filter(hflights.tbl, (Dest == "DFW") & (DepTime < 800)),  
desc(AirTime))
```

Calcula la distancia máxima de los vuelos desviados

```
summarise(filter(hflights.tbl, Diverted == 1), max_div = max(Distance))
```

Crea en una variable con los vuelos que no tienen NA en TaxiIn y TaxiOut  
Calcula un estadístico (max\_taxi\_diff) que contenga la mayor diferencia en valor absoluto entre TaxiIn y TaxiOut

```
temp2 <- filter(hflights.tbl, (!is.na(TaxiIn)) & (!is.na(TaxiOut)))  
summarise(temp2, max_taxi_diff = max(abs(TaxiIn - TaxiOut)))
```

**Usando pipes a partir de este punto, realiza:**

1. Crea una variable diff como la diferencia de TaxiIn y TaxiOut
2. Filtra aquellas filas para las que diff no es NA
3. Calcula la media

```
hflights.tbl %>%  
mutate(diff = TaxiOut - TaxiIn) %>%  
filter(!is.na(diff)) %>%  
summarise(avg = mean(diff))
```

Usando los pipes calcula cuantos vuelos nocturnos hay. Los vuelos nocturnos son aquellos para los que la hora de llegada es menor que la de salida.

```
hflights.tbl %>%  
filter(!is.na(DepTime) & !is.na(ArrTime) & ArrTime < DepTime) %>%  
summarise(n = n())
```

Usando los pipes calcula, para cada compañía los siguientes estadísticos:

n\_flights: número de vuelos  
n\_canc: número de vuelos cancelados  
p\_canc: porcentaje de vuelos cancelados  
avg\_delay: retraso medio de llegada (cuidado con los NAs)  
Ordena los resultados por el retraso medio y % de vuelos cancelados

```
hflights.tbl %>%  
group_by(UniqueCarrier) %>%  
summarise(n_flights = n(),  
n_canc = sum(Cancelled),  
p_canc = mean(Cancelled) * 100,  
avg_delay = mean(ArrDelay, na.rm = TRUE)) %>%  
arrange(avg_delay, p_canc)
```

Calcula qué avión (TailNum) voló más veces. ¿Cuántas?

```
hflights.tbl %>%  
group_by(TailNum) %>%  
summarise(n = n()) %>%  
filter(n == max(n))
```

¿Cuántos aviones volaron a un único destino?

```
hflights.tbl %>%  
group_by(TailNum) %>%  
summarise(ndest = n_distinct(Dest)) %>%  
filter(ndest == 1) %>%  
summarise(nplanes = n())
```

¿Cuál es el destino más visitado de cada compañía?

```
hflights.tbl %>%  
group_by(UniqueCarrier, Dest) %>%  
summarise(n = n()) %>%  
mutate(rank = rank(desc(n))) %>%  
filter(rank == 1)
```

¿Qué compañías viaja más a cada destino?

```
hflights.tbl %>%  
group_by(Dest, UniqueCarrier) %>%  
summarise(n = n()) %>%  
mutate(rank = rank(desc(n))) %>%  
filter(rank == 1)
```

## Apply

La familia apply realiza una operación a todos los elementos de un vector o lista. Es como un for que aplica una función. Recorre los elementos de un vector o lista, les aplica una función y devuelve el resultado en un vector.

apply(matriz, 1, mean) # Hace la media por filas.

apply(matriz, 2, mean) # Hace la media por columnas.

apply(matriz, 1:2, function(x) x/2) # Divide entre 2 cada valor (usando las filas y columnas).

## Lapply

Recorre los elementos de un vector o lista, les aplica una función y devuelve el resultado en una lista.

### Usando bucle

```
for (elemento in vector) {  
  print(class(elemento))  
}
```

num\_chars <- c() #vector en blanco.

```
for (i in 1:length(cities)) { #recorre los elementos  
  num_chars[i] <- nchar(cities[i]) #nº caract del elemento  
}
```

### Usando lapply

lapply(vector, class) #hace lo mismo que el bucle de la izq, pero más eficiente.

lapply(cities, nchar) # Hace lo mismo que el de la izq. Siempre devuelve una lista.

unlist(lapply(cities, nchar)) #Con unlist, obtenemos un vector.

## Ejemplos

```
nyc <- list(pop = 8405837, boroughs = c("Manhattan", "Bronx",  
"Brooklyn", "Queens", "Staten Island"), capital = FALSE)
```

Usando bucle

```
for (info in nyc) {  
  print(class(info))  
}
```

```
cities <- c("New York", "Paris", "London", "Tokyo", "Rio de Janeiro",  
"Cape Town")
```

```
num_chars <- c()  
for (i in 1:length(cities)) {  
  num_chars[i] <- nchar(cities[i])  
}
```

lapply(cities, nchar) #Ojo, siempre devuelve una lista  
unlist(lapply(cities, nchar)) #Con unlist, pasamos a vector

### Con funciones propias

```
oil_prices <- list(2.37, 2.49, 2.18, 2.22, 2.47, 2.32)  
triple <- function(x) {  
  x * 3  
}  
unlist(lapply(oil_prices, triple))
```

### Con funciones propias pasando argumentos

```
oil_prices <- list(2.37, 2.49, 2.18, 2.22, 2.47, 2.32)  
multiply <- function(x, factor) {  
  x * factor  
}  
unlist(lapply(oil_prices, multiply, factor = 3))
```

## Ejercicios

Vector con listado de nombres de matemáticos y sus años de nacimiento.  
pioneers <- c("GAUSS:1777", "BAYES:1702", "PASCAL:1623",  
"PEARSON:1857")

**Separa los nombres y años utilizando la función strsplit.**  
split\_math <- strsplit(pioneers, ":")

**Aplica la función tolower a todos los elementos de split\_math.**  
split\_low <- lapply(split\_math, tolower)

**Escribe una función que devuelva el primer elemento de un vector.**  
select\_first <- function(x) {  
 x[1]  
}

**Aplica la función select\_first a split\_low**  
names <- lapply(split\_low, select\_first)

**Escribe una función que devuelva el segundo elemento de un vector.**  
select\_second <- function(x) {  
 x[2]  
}

**Aplica la función select\_second a split\_low**  
years <- lapply(split\_low, select\_second)

## Sapply

Devuelve un vector o matriz con los rdos (muy parecido al Apply original).

### Ejemplos

```
cities <- c("New York", "Paris", "London", "Tokyo", "Rio de Janeiro",
"Cape Town")
sapply(cities, nchar) # Obtenemos un vector
first_and_last <- function(name) {
  name <- gsub(" ", "", name) # Elimina los espacios en blanco
  letters <- strsplit(name, split = "")[[1]] #Separa ciudades en vector.
  c(first = min(letters), last = max(letters))
} #Devuelve la letra “menor” y la “mayor” de la ciudad.
sapply(cities, first_and_last) # Obtenemos una matriz
```

### Ejercicios

Muestras de temperatura de cada día de la semana.  
temp <- list(monday = c(3, 7, 9, 6, -1), tuesday = c(6, 9, 12, 13, 5),
wednesday = c(4, 8, 3, -1, -3), thursday = c(1, 4, 7, 2, -2), friday = c(5, 7, 9,
4, 2), saturday = c(-3, 5, 8, 9, 4), sunday = c(3, 6, 9, 4, 1))

Utiliza lapply para encontrar la temperatura mínima de cada día.  
`lapply(temp, min)`

Utiliza sapply para encontrar la temperatura mínima de cada día.  
`sapply(temp, min)`

Crea una función que calcule la media entre el día mínimo y el máximo.  
extremes\_avg <- `function(x) {`  
  `mean(c(max(x), min(x)))`  
`}`

Aplica la nueva función utilizando sapply.  
`sapply(temp, extremes_avg)`

Crea una función extremes que devuelva una matriz con:  
1. La temperatura mínima.  
2. La temperatura máxima.

```
extremes <- function(x) {
  c(min = min(x), max = max(x))
}
```

Aplica la nueva función utilizando sapply.  
`sapply(temp, extremes)`

Crea una función que devuelva las muestras menores que cero.  
below\_zero <- `function(x) {`  
  `x[x < 0]`  
`}`

Aplica la nueva función utilizando sapply y guardarla en freezing\_s  
`freezing_s <- sapply(temp, below_zero)`

# Vapply

Controla el tipo de dato de retorno (el resto es igual que Sapply).

## Ejemplos

```
cities <- c("New York", "Paris", "London", "Tokyo", "Rio de Janeiro",
"Cape Town")
vapply(cities, nchar, numeric(1)) # El dato de retorno es numérico.
first_and_last <- function(name) {
  name <- gsub(" ", "", name)
  letters <- strsplit(name, split = "")[[1]]
  c(first = min(letters), last = max(letters))
}
vapply(cities, first_and_last, character(2)) # # El 2 es xq devuelve una
matriz de 2 filas.
vapply(cities, first_and_last, character(1)) # Esto daría un error, ya
que estamos devolviendo un vector de longitud 2.
```

## Ejercicios

Muestras de temperatura de cada día de la semana.  
temp <- list(monday = c(3, 7, 9, 6, -1), tuesday = c(6, 9, 12, 13, 5),
wednesday = c(4, 8, 3, -1, -3), thursday = c(1, 4, 7, 2, -2), friday = c(5, 7, 9,
4, 2), saturday = c(-3, 5, 8, 9, 4), sunday = c(3, 6, 9, 4, 1))

Crea la función basics que devuelve un vector con nombres con:

1. **min**: temperatura mínima
2. **mean**: temperatura media
3. **max**: temperatura máxima

```
basics <- function(x) {
  c(min = min(x), mean = mean(x), max = max(x))
}
```

Usando vapply aplica la función basics.

```
vapply(temp, basics, numeric(3))
```

Con esta nueva definición de la función basics.

```
basics <- function(x) {
  c(min = min(x), mean = mean(x), median = median(x), max = max(x))
}
```

Arregla el error:

```
vapply(temp, basics, numeric(3))
```

```
vapply(temp, basics, numeric(4))
```

Convierte este sapply en un vapply.

```
sapply(temp, max)
vapply(temp, max, numeric(1))
```

Convierte este sapply en un vapply.

```
sapply(temp, function(x, y) { mean(x) > y }, y = 5)
vapply(temp, function(x, y) { mean(x) > y }, y = 5, logical(1))
```

Con esta función get\_info, convierte el sapply en un vapply.

```
get_info <- function(x, y) {
  if (mean(x) > y) {
    return("Not too cold!")
  } else {
    return("Pretty cold!")
  }
}

sapply(temp, get_info, y = 5)
vapply(temp, get_info, y = 5, character(1))
```

## Data table

Paquete de manipulación de datos. Es una versión mejorada de los data frames.

Su objetivo es reducir el tiempo de programación y procesamiento.

Para poder trabajar con DT, primero debes instalarlo (packages/install) y luego invocarlo: library(data.table)

iris.dt <- as.data.table(iris) # Convierzo un data frame en un data table. Es a la vez un DF y DT (acepta ambas instrucciones).

Funcionamiento DT[selección de filas, selección de columnas, by agrupación de filas] en SQL [where, select, group by].

### Conversión a Data Table

```
DT <- data.table(x = c("a", "b", "c", "d", "e"), y = c(1, 2, 3, 4, 5)) # Creación de un data.table
```

```
DT <- as.data.table(DF) # Convertimos un Data Frame en un Data Table.
```

### Selección de filas

```
DT[3] # Seleccionamos la tercera fila, devuelve un data.table.
```

```
DT[[3]] # Seleccionamos la tercera fila, devuelve un vector.
```

```
DT[2:3] o bien DT[2:3,] # Seleccionamos la segunda y tercera fila.
```

```
DT[.N - 1] # Selecciona la penúltima fila
```

```
DT[c(2, 2, 3)] # Selecciona la segunda fila dos veces y la tercera
```

```
iris.dt[base * altura > 10] # Filtra las filas para las que el área es mayor que 10
```

```
iris.dt[Species %in% c("virginica", "versicolor")] # Filtra las filas de la especie virginica y versicolor
```

### Selección de columnas

```
DT[, .(B)] # Devuelve un data.table con la columna B.
```

```
DT[, B] # Devuelve un vector
```

```
DT[c(1, 3), .(B, C)] # Selección de las filas 1 y 3 y columnas B y C.
```

```
ans <- DT[, .(B, val = A * C)] # Selección de la columna B y creación de una columna val como la multiplicación de A por C.
```

### Agrupando con By

```
DT[, .(MySum = sum(A), MyMean = mean(A)), by = .(B)] # Agrupa por la col B, hace la suma y media de los valores de la col A.
```

```
DT[, .(MySum = sum(C)), by = .(Grp = C%%2)] # Hace la suma de la columna C, agrupándola por par e impar.
```

### Encadenando llamadas

```
DT[, sum(B), by = A][order(A)] # Agrupando por la columna A, suma la columna B y ordena los rdos por la columna A.
```

```
DT[, .(C = sum(C)), by = .(A, B)][, .(C = tail(C, 2)), by = A]
```

```
# Agrupando por las columnas A y B, suma los valores de C; y muestra los dos últimos valores de C ordenados por la col A.
```

### .SD Extrae subconjunto de datos

```
DT[, lapply(.SD, mean), by = A] # Hace la media de todas las columnas agrupando por los valores de la columna A.
```

```
DT[, lapply(.SD, sum), .SDcols = 2:4] # Hace la suma de las columnas 2 a 4.
```

```
DT[, lapply(.SD, sum), by = x, .SDcols = c("x", "y", "z")] # Agrupando por x, suma las columnas (x, y, z).
```

```
DT[, lapply(.SD, cumsum), by = .(by1 = x, by2 = z > 8), .SDcols = c("x", "y")]
```

```
# Calcula la suma acumulada de las columnas X e Y, agrupando por los valores de X y los valores de Z > 8.
```

### := crea, modifica o elimina el contenido de una columna

```
DT[, Total := sum(B), by = A] # Añade una nueva columna "Total" agregando por los valores de A.
```

```
DT[c(2, 4), B := B + 1] # Añade 1 a la columna B en las filas 2 y 4.
```

```
DT[, Total := NULL] # Elimina columna Total.
```

```
DT[, 2 := NULL] # Borra la segunda columna.
```

```
DT[, `:=`(B = B + 1, C = A + B, D = 2)] # Añade 1 a la columna B y crea las columnas C y D.
```

### Key (crea índices para optimizar filtrados)

```
setkey(DT, A, B) # Creación del índice por las columnas A y B.
```

## Ejemplos

```
library(data.table)
Agrupando con by
DT <- data.table(A = c(1, 2, 3, 4, 5), B = c("a", "b", "c", "a", "b"), C = c(6, 7, 8, 9, 10))
DT[, .(MySum = sum(A), MyMean = mean(A)), by = .(B)]
DT[, .(MySum = sum(C)), by = .(Grp = C%>%2)]

Encadenando llamadas
DT <- data.table(A = c("c", "b", "a", "c", "b", "a"), B = c(1, 2, 3, 4, 5, 6))
DT[, sum(B), by = A][order(A)] # Suma B agrupando y ordenando A
DT <- data.table(A = c("b", "b", "b", "a", "a", "a", "a"), B = c(1, 1, 2, 2, 3, 3, 4, 4), C = c(3, 8, 4, 5, 1, 7, 2, 6))
DT[, .(C = sum(C)), by = .(A, B)]# Agrupan por columnas A y B y suma C.

Usando .SD
DT[, lapply(.SD, mean), by = A] # Media de las columnas.
DT[, lapply(.SD, median), by = A] # Mediana.

DT <- data.table(grp = c(6, 6, 8, 8, 8), Q1 = c(2, 2, 3, 5, 2), Q2 = c(5, 5, 4, 4, 1), Q3 = c(2, 1, 4, 2, 3), H1 = c(3, 4, 5, 2, 4), H2 = c(5, 2, 4, 1, 2))
DT[, lapply(.SD, sum), .SDcols = 2:4] # suma de las columnas Q
DT[, lapply(.SD,sum), .SDcols = paste0("H", 1:2)] # suma de H1 y H2

DT <- data.table(x = c(2, 1, 2, 1, 2, 2, 1), y = c(1, 3, 5, 7, 9, 11, 13), z = c(2, 4, 6, 8, 10, 12, 14))
DT[, lapply(.SD, sum), by = x, .SDcols = c("x", "y", "z")] # Suma las columnas (x, y, z) agrupándolas por x.

DT[, lapply(.SD, cumsum), by = .(by1 = x, by2 = z > 8), .SDcols = c("x", "y")] # Calcula la suma acumulada de x e y agrupando por x y z > 8

Usando :=
DT <- data.table(A = c("a", "a", "a", "b", "b"), B = c(1, 2, 3, 4, 5))
DT[, Total := sum(B) , by = A] # nueva columna agregando por A.
DT[c(2, 4), B := B + 1] # Añade 1 a la columna B en las filas 2 y 4
DT[2:4, Total2 := sum(B), by = A] # Nueva columna agregando por A pero solo en las filas 2, 3 y 4.
DT[, Total := NULL] # Elimina la columna
DT <- data.table(A = c(1, 1, 1, 2, 2), B = 1:5)
DT[, `:=`(B = B + 1, C = A + B, D = 2)] # Actualiza B y añade C y D
DT[, 2 := NULL] # Borra la segunda columna

Usando key
DT <- data.table(A = letters[c(2, 1, 2, 3, 1, 2, 3)], B = c(5, 4, 1, 9, 8, 6), C = 6:12)
setkey(DT, A, B) # Creación del índice
DT["b"] # Filtra por A="b"
DT[c("b", "c")] # Filtra por A="b" | A="c"
DT[c("b", "c"), mult = "first"] # Selecciona primera fila de cada grupo
DT[c("b", "c"), .SD[c(1, .N)], by = .EACHI] #Selecciona la primera y ultima fila de cada grupo. EACHI (por columnas).
```

## Ejercicios

```
library(data.table)

Convierte iris en un data.table
iris.dt <- as.data.table(iris)

Para cada especie calcula la longitud media del sépalo
iris.dt[, mean(Sepal.Length), by = Species]

Agrupando por la primera letra del nombre de cada especie, calcula la longitud media del sépalo.
iris.dt[, mean(Sepal.Length), by = substr(Species, 1, 1)]

Filtrar las filas de la especie virginica.
iris.dt[Species == "virginica"]

Filtrar las filas de la especie virginica y versicolor.
iris.dt[Species %in% c("virginica", "versicolor")]

Para cada especie calcula la mediana de todas las columnas. Ordena el resultado por el nombre de la especie descendenteamente.
iris.dt[, .(Sepal.Length = median(Sepal.Length),
           Sepal.Width = median(Sepal.Width),
           Petal.Length = median(Petal.Length),
           Petal.Width = median(Petal.Width)), by = Species][order(-Species)]

Repite el ejercicio anterior usando .SD
iris.dt[, lapply(.SD, median), by = Species][order(-Species)]

Elimina el prefijo sepal del nombre de las columnas.
setnames(iris.dt, gsub("Sepal.", "", names(iris)))

Elimina las dos columnas que comienzan por "Petal".
iris.dt[, grep("Petal", names(iris)) := NULL]

Filtrar las filas para las que el área es mayor que 20.
iris.dt[Width * Length > 20]

Añade una columna que indique si la fila tiene un área mayor de 25.
iris.dt[, is_large := Width * Length > 25]

Filtrar aquellas filas que cumplen la condición anterior.
iris.dt[is_large == TRUE]
```

## Caso a resolver: Análisis de una base de datos



Simulacro de la práctica

En esta hoja encontrarás un dataset (bikes.csv) que contiene información agregada por horas de un sistema de alquiler de bicicletas (parecido al servicio BiciMad).

Las variables del dataset son las siguientes:

- date: fecha (en formato yyyy-mm-dd).
- season: estación del año. Los valores son:
  - 1 (winter)
  - 2 (spring)
  - 3 (summer)
  - 4 (fall)
- hour: la hora del día (0 a 23).
- is.holiday: 1 si es día festivo, 0 en caso contrario.
- weekday: día de la semana. Los valores son:
  - 0 (Sunday)
  - 1 (Monday)
  - 2 (Tuesday)
  - 3 (Wednesday)
  - 4 (Thursday)
  - 5 (Friday)
  - 6 (Saturday)
- is.workingday: si el día no es ni fin de semana ni vacaciones 1, en caso contrario 0.
- weathersit: variable categórica con los siguientes valores:
  - 1: Clear, Few clouds, Partly cloudy, cloudy
  - 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist
  - 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds
  - 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
- temp: temperatura en grados Celsius.
- atemp: sensación térmica en grados Celsius.
- hum: humedad.
- windspeed: velocidad del viento.
- casual: número de alquileres de usuarios no registrados en el servicio.
- registered: número de alquileres de usuarios registrados en el servicio.

## Parte 1 Análisis básico

Lee el fichero y asígnalo a una variable

```
setwd("C:/Users/Guillermo/Desktop")
bikes_csv<-read.table("bikes.csv",header=T,sep=",",stringsAsFactors = F,dec=".")
```

¿De qué clase es el objeto?

```
class(bikes_csv)
```

¿Cómo se puede ver el tipo de cada columna y una muestra de ejemplos?

```
str(bikes_csv)
```

Muestra los primeros 15 registros del dataset.

```
head(bikes_csv,15)
```

Muestra los últimos 20 registros del dataset.

```
tail(bikes_csv,20)
```

¿Cuáles son las dimensiones del dataset?

```
dim(bikes_csv)
```

¿Cuáles son los nombres de las variables del dataset?

```
names(bikes_csv)
```

Las variables season y weekday deberían ser categóricas. Crea un factor con etiquetas para dichas columnas (no hace falta que sea ordenable) y asígnalo a las columnas de nuevo.

```
season<-factor(bikes_csv[,2],levels=c(1,2,3,4)) #lo convierto en un factor
```

```
levels(season)[1:4]<-c("winter","spring","summer","fall") #renombramos los niveles
```

```
bikes_csv[,2]<-season #Lo reasigno al data frame.
```

```
weekday<-factor(bikes_csv[,5],levels=c(0,1,2,3,4,5,6)) #lo convierto en un factor
```

```
levels(weekday)[1:7]<-c("Sunday","Monday","Tuesday","Wednesday","Thursday","Friday","Saturday") #renombramos
```

```
bikes_csv[,5]<-weekday #Lo reasigno al data frame.
```

Las variables i s.holiday y i s.workingday son variables booleanas. Conviértelas a variable booleana y asígnalas a las columnas de nuevo.

```
bikes_csv[,4]<-as.logical(bikes_csv[,4])
```

```
bikes_csv$is.holiday<- as.logical(bikes_csv$is.holiday)#otra manera de hacerlo
```

```
bikes_csv[,6]<-as.logical(bikes_csv[,6])
```

Calcula la suma de la columna casual.

```
sum(bikes_csv[,12])
```

```
sum(bikes_csv$casual)
```

Guarda en un vector la suma de las columnas casual y registered.

```
total_por_fila<-as.vector(bikes_csv$casual+bikes_csv$registered)
```

```
total_por_columna <-c(sum(bikes_csv$casual), sum(bikes_csv$registered))
```

¿Qué variables son numéricas? PISTA: utiliza sapply junto con la función is.numeric.

```
sapply(bikes_csv,is.numeric)
```

Utilizando el resultado anterior, selecciona aquellas columnas numéricas y calcula la media de todos los registros.

```
seleccion<- bikes_csv[sapply(bikes_csv,is.numeric)]
```

```
sapply(seleccion,mean)
```

```
colMeans(seleccion) # Otra manera de hacerlo.
```

Selecciona aquellas observaciones del dataset para las que el día de la semana es Monday, están entre las 15 y 20 horas y la temperatura es superior a 25 grados.

```
bikes_csv[(bikes_csv$weekday=="Monday" & bikes_csv$hour>15 & bikes_csv$hour<20 & bikes_csv$temp>25),]
```

Selecciona las 100 primeras filas y todas las columnas menos las dos últimas (con índices positivos).

```
bikes_csv[1:100,1:11]
```

Selecciona las 100 primeras filas y todas las columnas menos las dos últimas (con índices negativos).

```
bikes_csv[1:100,1:(length(bikes_csv)-2)]
```

Obtén los cuantiles de la variable hum.

```
quantile(bikes_csv$hum)
```

**Obtén los deciles de la variable hum.**

```
quantile(bikes_csv$hum,c(0, .1, .2, .3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1))  
quantile(bikes_csv$hum,probs=seq(0,1,0.1)) #una manera más eficiente de hacerlo.
```

**Obtén los estadísticos básicos de todas las variables en un solo comando.**

```
summary(bikes_csv)
```

**¿Cuantos valores únicos existen para la variable atemp?**

```
length(unique(bikes_csv$atemp))
```

**¿Cuantos registros tienen un valor superior a 100 para la variable casual?**

```
nrow(bikes_csv[bikes_csv$casual>100,])  
sum(bikes_csv$casual>100) #otra manera de hacerlo.
```

**Ordena de menor a mayor los 100 primeros elementos de la variable registered.**

```
sort(bikes_csv[1:100,13])  
sort(bikes_csv$registered[1:100]) #otra manera de hacerlo
```

**Ordena el dataset por la variable registered de manera ascendente. Inspecciona los primeros resultados.**

```
indice<-order(bikes_csv$registered, decreasing = F) #Obtengo el índice por el que realizar la ordenación.  
bikes_csv<-bikes_csv[indice,]
```

```
bikes_csv<-bikes_csv[order(bikes_csv$registered,decreasing = F),] #otra manera de hacerlo
```

**Obtén los índices de los registros para los que el valor de la variable temp es superior a la media.**

```
indice_booleano<- bikes_csv$temp>mean(bikes_csv$temp)  
indice_numerico <- which(indice_booleano)
```

**Obtén los registros para los que el valor de la variable windspeed es máximo.**

```
maximo<-max(bikes_csv[,11])  
registros<-bikes_csv[bikes_csv$windspeed==maximo,]
```

**Comprueba si alguna de las variables contiene NAs.**

```
any(is.na(bikes_csv))
```

**Comprueba utilizando el boxplot si la variable windspeed tiene outliers.**

```
boxplot(bikes_csv$windspeed) #Sí que los tiene.
```

**Pinta un histograma de la variable atemp.**

```
hist(bikes_csv$atemp)
```

**Crea una función (cold.heat) que reciba dos parámetros: temperatura y límite; y devuelva cold o heat.**

```
cold.heat<-function(temp,limit=20){
```

```
  if (temp>limit){  
    return ("Heat")  
  } else if (temp<limit){  
    return ("Cold")  
  } else {  
    return ("ideal")  
  }  
}
```

```
cold.heat(5)  
cold.heat(20)  
cold.heat(30)
```

**Aplica la función a la columna temp y almacena el resultado en una nueva columna cold.heat del data.frame.**

```
cold.heat<-sapply(bikes_csv$temp,cold.heat)
```

```
bikes_csv<-cbind(bikes_csv,cold.heat)
```

```
bikes_csv$cold.heat<- sapply(bikes_csv$temp,cold.heat)#en una sola línea.
```

## Parte 2: Utiliza Data frame, dplyr o data.table (según prefieras)

```
library(data.table)
```

### Calcula el número medio de usuarios registrados por día de la semana

```
bikes<-data.table(bikes) #Convierto el data frame en un data table.
```

```
bikes[.,(media_usuarios=mean(registered)),by=.(weekday)]
```

```
bikes.group.semana <- group_by(bikes,weekday) # Otra manera de hacerlo.
```

```
summarise(bikes.group,media_usuario = mean(registered))
```

### Ordena de menor a mayor las temperaturas máximas de cada estación

```
bikes[.,(temperaturas_maximas=max(temp)),by=.(season)][order(temperaturas_maximas)]
```

```
bikes.group.estacion <- group_by(bikes,season) # Otra manera de hacerlo.
```

```
arrange(summarise(bikes.group.estacion,temp_max = max(temp)),desc(temp_max))
```

### Calcula el número total de usuarios (registered + casual) en días festivos y no festivos.

```
bikes[.,(num_total_usuarios=sum(casual+registered)),by=.(is.holiday)]
```

```
bikes.group.festivos<-group_by(bikes,is.holiday) # Otra manera de hacerlo.
```

```
bikes.group.festivos$usuarios = bikes.group.festivos$casual + bikes.group.festivos$registered
```

```
summarise(bikes.group.festivos,suma =sum(usuarios))
```

```
bikes %>%
```

```
  group_by(is.holiday) %>%
```

```
  mutate(usuarios = casual + registered) %>%
```

```
  summarise(suma = sum(usuarios))
```

### Calcula el día que se produjeron el mayor número de alquileres casuales.

```
bikes[, .(max=sum(casual)), by = date][order(-max)] # El primer registro tienen el día con el máximo de alquileres casuales.
```

```
library(dplyr) # Otra manera de hacerlo.
```

```
bikes %>%
```

```
  group_by(date) %>%
```

```
  select(date,casual) %>%
```

```
  summarise(suma_dia = sum(casual)) %>%
```

```
  filter(suma_dia==max(suma_dia))
```

```
bikes[casual==bikes[,max(casual)]] # MAL, no obtenemos el día en que más se produjeron, sino el día con máximo por hora.
```

### Calcula el ratio de alquileres registrados/casuales por hora en verano.

```
alquileres_en_verano<-bikes[season==3]
```

```
registrados<-alquileres_en_verano[.,(suma_registrados=sum(registered)),by=.(hour)]
```

```
casuales<-alquileres_en_verano[.,(suma_casuales=sum(casual)),by=.(hour)]
```

```
rdo<-data.table(registrados,casuales[,suma_casuales])
```

```
rdo[,ratio:=suma_registrados/V2]
```

```
library(dplyr) #otra manera de hacerlo.
```

```
bikes %>%
```

```
  filter(season==3) %>%
```

```
  group_by(hour) %>%
```

```
  summarise(suma_reg = sum(registered),suma_cas=sum(casual))%>%
```

```
  mutate(proporcion = suma_reg/suma_cas)
```

```
# La manera más sencilla de hacerlo.
```

```
data.table: bikes[season == 3, .(registered.casual.ratio = sum(registered) / sum(casual)), by = .(hour)]
```

### Para todos los lunes festivos calcular el número total de alquileres registrados, temperatura media, humedad máxima y velocidad del viento mínima.

```
lunes_festivos<-bikes[weekday==1 & is.holiday==1]
```

```
lunes_festivos[.,(total_alquileres_registrados=sum(registered),temperatura_media=mean(temp),humedad_maxima=max(hum),velocidad_viento_minimo=min(windspeed))]
```

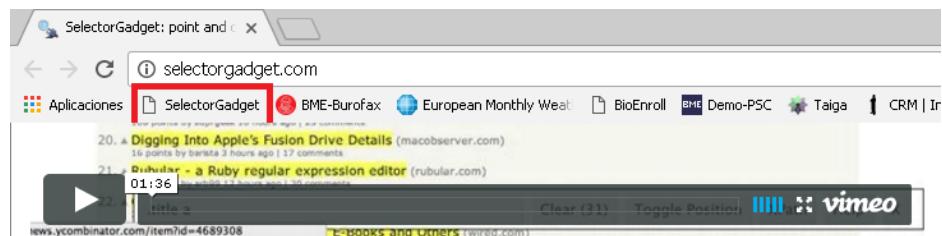
```
bikes %>% # Otra manera de hacerlo.
```

```
  filter(weekday==1,is.holiday==1) %>%
```

```
  summarise(total_alquileres_registrados=sum(registered),temperatura_media=mean(temp),hum_max=max(hum),min_vel=min(windspeed))
```

## Web Scraping (descarga de datos web)

A la hora de bajarse datos web, lo primero que tenemos que hacer es averiguar en qué nodo del HTML se encuentra dicha información. SelectorGadget es una forma fácil de determinar qué selector extrae los datos que nos interesan sin necesidad de conocer HTML: Ve a esta página <http://selectorgadget.com/> y pon el programa en tu barra de marcadores.



[SelectorGadget Screencast](#) from [Andrew Cantino](#) on [Vimeo](#).

SelectorGadget is an open source tool that makes [CSS selector](#) generation and discovery on complicated sites a breeze. Just install the [Chrome Extension](#) or drag the bookmarklet to your bookmark bar, then go to any page and launch it. A box will open in the bottom right of the website. Click on a page element that you would like your selector to match (it will turn green). SelectorGadget will then generate a minimal CSS selector for that element, and will highlight (yellow) everything that is matched by the selector. Now click on a highlighted element to remove it from the selector (red), or click on an unhighlighted element to add it to the selector. Through this process of selection and rejection, SelectorGadget helps you come up with the perfect CSS selector for your needs.

How could I use this?

- for webpage scraping with tools such as [Nokogiri](#) and [Beautiful Soup](#)
- to generate [jQuery](#) selectors for dynamic sites
- as a tool to examine JavaScript-generated DOM structures
- as a tool to help you style only particular elements on the page with your stylesheets
- for selenium or phantomjs testing

Created by [Andrew Cantino](#) and [Kyle Maxwell](#). You can find the [current version on GitHub](#), and please feel free to leave comments below.

Try our new [Chrome Extension](#)!

Or drag this link to your bookmark bar: [SelectorGadget \(updated August 7, 2013\)](#)

Or use the development version: [SelectorGadget unstable \(updated August 7, 2013\)](#)

Ya lo tienes. A partir de ahora, en cualquier página en la que estés puedes pinchar en selectorgadget para seleccionar información. Cuando lo hagas, te saldrá el siguiente recuadro en la parte inferior derecha de tu pantalla:



Necesitarás las siguientes librerías en R, acuérdate de instalarlas e invocarlas antes de hacer web scraping.

**library(rvest)**

## Ejemplo: Extraemos datos del Lobo de Wall Street de wikipedia



```
library(rvest)

datos_pelicula <- read_html ("https://es.wikipedia.org/wiki/El_lobo_de_Wall_Street")

# Obtenemos la ficha técnica (la columna de la derecha con los datos de dirección, producción, etc)
ficha_tecnica <- datos_pelicula %>%
  html_nodes("table") %>%
  .[1] %>% # Sabemos lo que queremos extraer gracias a selectorgadget
  html_table()
ficha_tecnica<-as.data.frame(ficha_tecnica)
ficha_tecnica<-ficha_tecnica[2:dim(ficha_tecnica)[1],1:2]

# Obtenemos las referencias
referencias<- datos_pelicula %>%
  html_nodes("span.reference-text") %>% # Sabemos lo que queremos extraer gracias a selectorgadget
  html_text()
referencias<-as.data.frame(referencias)

# Obtenemos el contenido de la página con el argumento
argumento<- datos_pelicula %>%
  html_nodes("p") %>% # Sabemos lo que queremos extraer gracias a selectorgadget
  html_text()
argumento<-as.data.frame(argumento)
argumento<-as.data.frame(argumento[4:(dim(argumento)[1]-1),1])
```

## ¿Necesito usar selector Gadget para encontrar el nodo?

No. Otra manera de encontrarlo es pulsar el botón derecho del ratón en el apartado de la web que queremos capturar y seleccionar la opción “inspeccionar”.

A la derecha se nos abrirá un menú donde veremos los distintos niveles de nodos, de mayor (izquierda) a menor (derecha). Pasando el ratón por encima de ellos se nos iluminará en la web la información que nos descargaríamos.

The screenshot shows a news article from the 'ECONOMÍA' section of the Expansión website. The article is titled 'Rajoy debatirá sobre las pensiones el día 14 pero sin votar las propuestas de la oposición'. Below the article, there is a sidebar with 'BLOGS ECONOMÍA' featuring three posts by Pedro Biurrun, RIFDE, and Javier Morillas. To the right, the Chrome DevTools developer console is open, specifically the 'Elements' tab. The DOM tree shows the structure of the page, with nodes like 'article.noticia.economia' being inspected. The right-hand panel of the DevTools shows the element's style properties, including 'position: absolute', 'margin: 11.500px', and 'border: 1px solid black'. The 'Styles' tab is also visible.

# Ejercicio: Captura los hechos relevantes de la CNMV

finanzas para todos.es  
portal de la transparencia

Inicio Enlaces Mapa Contacto Català Euskera Galego English Canal RS:

Sede electrónica @CNMV SEDE ELECTRÓNICA/ REGISTRO ELECTRÓNICO ADVERTENCIAS AL PÚBLICO ANCV AGENCIA NACIONAL DE CODIFICACIÓN DE VALORES XBRL

28/10/2016 INSTRUCCIONES DE REPORTE Y ESQUEMAS XML DEL FIRDS ASÍ COMO ESPECIFICACIONE

Sección del inversor	Emisores y cotizadas	Inversión colectiva y capital riesgo	Empresas de servicios de inversión	Infraestructuras de mercados
----------------------	----------------------	--------------------------------------	------------------------------------	------------------------------

Sobre la CNMV  
Comunicaciones de la CNMV  
Consultas a los registros oficiales  
Folletos  
Sala de prensa  
Publicaciones, estadísticas e investigación  
Legislación  
Actividades internacionales  
Ofertas de empleo

Consultas a los registros oficiales

Búsqueda por entidades  
Búsqueda en 5 últimos días  
Hechos relevantes  
Hechos relevantes del día  
Entidades Emisoras: Información regulada y otra  
Emisiones, admisiones y OPAS  
Instituciones de Inversión Colectiva

Entidades de Capital-Riesgo  
Empresas Servicios Inversión  
Infraestructuras de mercados  
Comunicaciones empresas que prestan servicios inversión  
Agencias de calificación crediticia  
Registro de sanciones  
Plataformas de Financiación Participativa

Inicio > Consultas a registros oficiales

» Hechos relevantes del día

Suscripciones | | | |

10/11/2016

16:14 **SACYR, S.A. Convocatorias de reuniones o actos informativos**  
La sociedad envía presentación de resultados del tercer trimestre de 2016  
 1325 KB

14:36 **FERROVIAL, S.A. Programas de recompra de acciones, estabilización y autocartera**  
Operaciones efectuadas por Ferrovial, S.A. al amparo de su programa de recompra de acciones entre el 3 y el 9 de noviembre de 2016  
 85 KB

14:31 **TECNICAS REUNIDAS, S.A. Información sobre resultados**  
La Sociedad remite presentación sobre los resultados del Tercer Trimestre de 2016. The Company sends presentation regarding the 3Q 2016 Results.  
 494 KB

14:16 **DESARROLLOS ESPECIALES DE SISTEMAS DE ANCLAJES, S.A. Información financiera intermedia**  
La sociedad remite información sobre los resultados del tercer trimestre de 2016  
 IFI

13:48 **COCA-COLA EUROPEAN PARTNERS PLC Información sobre resultados**  
Información sobre los resultados del tercer trimestre de 2016  
 1174 KB

13:45 **COCA-COLA EUROPEAN PARTNERS PLC Composición del consejo de administración**  
Nombramiento de Damian Gammell como nuevo consejero delegado de la compañía en sustitución de John Brock con motivo de su jubilación  
 196 KB

```
CNMV <- read_html("http://www.cnmv.es/portal/HR/HRAldia.aspx")
```

```
hechos_relevantes<- CNMV %>%
```

```
html_nodes("#elementoPrimerNivel") %>% # Sabemos lo que queremos extraer gracias a selectorgadget
```

```
html_text()
```

```
hechos_relevantes<-as.data.frame(hechos_relevantes)
```

## Ejercicio: Captura las noticias sobre economía de Expansion

# ECONOMÍA

## ECONOMÍA

### La deuda media por fraude fiscal liquidada por Hacienda creció un 44% en 2015

En el Impuesto sobre Sociedades el fraude descubierto asciende a 1.386 millones, el triple que un año antes.

## ECONOMÍA

### El Pacto de Toledo cita a Báñez y a los agentes sociales la próxima semana

EXPANSIÓN.COM/EFE

La Comisión del Pacto de Toledo ha aprobado la comparecencia de la ministra de Empleo y Seguridad Social, Fátima Báñez, y de los dirigentes de CCOO, UGT, CEOE y CEPYME, para que acudan al Congreso la próxima semana con el fin de hablar del futuro de las pensiones.

## ECONOMÍA

### Hacienda pone en marcha un sistema electrónico para devolver el IVA a viajeros

## BLOGS ECONOMÍA



IVAN REDONDO,  
CONSULTOR POLÍTICO  
*The War Room*  
*El EE.UU de Trump*



PEDRO BIURRUN  
*La estadística del pollo*  
*Por qué ha ganado Trump*



MANUEL CONTHE  
*El sueño de Jardiel*  
*¿Nos hace estúpidos la Política?*

## ENCUENTROS



14.11.2016  
**ALBERT ENGUIX**  
Gestor de fondos de GVC Gaesco Gestión

**¿Puede el Ibex acabar el año en positivo?**



10.11.2016  
**TOMÁS GARCÍA-PURRIÑOS**  
gestor de carteras  
**¿Qué sectores tienen mejores expectativas para invertir?**

```
expansion<-read_html("http://www.expansion.com/economia.html?intcmp=MENUHOM24101&s_kw=economia")
noticias <- expansion %>%
  html_nodes("h2") %>%
  html_text()
noticias<-as.data.frame(noticias)
```

# Ejercicio: Captura la agenda macroeconómica de investing.com

Investing.com EUR/USD o BBVA

Ayer Hoy Mañana Esta semana Próxima semana  Filtros

○ Hora Actual: 16:27 (GMT +1:00) Todos los datos son transmitidos y actualizados automáticamente.

Hora	Div.	Importancia	Evento	Actual	Previsión	Anterior
Jueves, 10 de noviembre de 2016						
00:50	● JPY	▼ □ ▲	Pedidos de maquinaria (subyacente) (Mensual) (Sep)	-3,3%	-0,8%	-2,2%
00:50	● JPY	▼ □ ▲	Pedidos de maquinaria (subyacente) (Anual) (Sep)	4,3%	3,5%	11,6%
00:50	● JPY	▼ □ ▲	Compras de bonos extranjeros	604,9B	892,3B	
00:50	● JPY	▼ □ ▲	Inversión extranjera en acciones japonesas	-106,4B	108,9B	
00:50	● JPY	▼ □ ▲	Masa monetaria M2 (Anual)	3,7%	3,6%	3,5%
01:00	■ AUD	▼ □ ▲	Expectativas de inflación del MI	3,2%		3,7%
01:01	■ GBP	▼ □ ▲	Índice RICS de precios de la vivienda (Oct)	23%	19%	18%
Declaraciones de Wheeler, gobernador del Banco de la Reserva de Nueva Zelanda						
01:10	■ NZD	▼ □ ▲				
01:30	■ AUD	▼ □ ▲	Hipotecas sobre viviendas (Mensual) (Sep)	1,6%	-2,0%	-2,7%
01:30	■ AUD	▼ □ ▲	Financiación para inversiones en vivienda (Mensual) (Sep)	4,6%		0,1%
03:00	■ USD	▼ □ ▲	Declaraciones de Williams, miembro del FOMC			
04:45	● JPY	▼ □ ▲	Subasta de deuda a 30 años (JGB)	0,511%		0,514%
07:00	● JPY	▼ □ ▲	Pedidos de herramientas de maquinaria (Anual) P	-8,9%		-6,3%
08:00	■ IDR	▼ □ ▲	Ventas de motocicletas (Anual) (Oct)			-7,80%
08:45	■ EUR	▼ □ ▲	Producción industrial de Francia (Mensual) (Sep)	-1,1%	-0,3%	2,3%
08:45	■ EUR	▼ □ ▲	Nóminas no agrícolas de Francia (Trimestral) (3T) P	0,3%	0,2%	0,2%
10:00	■ USD	▼ □ ▲	Informe mensual de la AIE			
10:00	■ EUR	▼ □ ▲	Producción industrial de Italia (Anual) (Sep)	1,8%	2,2%	4,4%

```
investing<-read_html("http://es.investing.com/economic-calendar/")
```

```
eventos<- investing %>%
```

```
html_nodes(".event") %>% # Capturamos los eventos. No podemos capturar la tabla entera xq tiene elementos complejos.
```

```
html_text
```

```
eventos <- as.data.frame(eventos)
```

```
hora<- investing %>%
```

```
html_nodes(".js-time") %>% # Capturamos las horas asociadas a los eventos.
```

```
html_text
```

```
hora <- as.data.frame(hora)
```

```
actual<- investing %>%
```

```
html_nodes(".act") %>%
```

```
html_text
```

```
actual <- as.data.frame(actual)
```

```
prevision<- investing %>%
```

```
html_nodes(".fore") %>%
```

```
html_text
```

```
prevision <- as.data.frame(prevision)
```

```
anterior<- investing %>%
```

```
html_nodes(".prev") %>%
```

```
html_text
```

```
anterior <- as.data.frame(anterior)
```

```
eventos <- as.data.frame(eventos[3:dim(eventos)[1],1]) # No todos los DF tienen el mismo tamaño. Lo corregimos.
```

```
actual <- as.data.frame(actual[3:dim(actual)[1],1])
```

```
anterior <- as.data.frame(anterior[3:dim(anterior)[1],1])
```

```
agenda_macro<-cbind(eventos, hora, actual, prevision, anterior) # Reconstruimos la agenda
```

```
colnames(agenda_macro)<-c("eventos", "hora", "actual", "prevision", "anterior")
```

```
# Otra manera, más fácil, de hacerlo
```

```
investing<-read_html("http://es.investing.com/economic-calendar/")
```

```
calendario_economico <- investing %>%
```

```
html_nodes("#economicCalendarData") %>%
```

```
html_table(fill = TRUE)
```

# Ejercicio: Captura las cotizaciones de los commodities de investing.com

## Materias primas



### Tabla de Comportamiento

Materias primas	15 minutos	1 hora	Diario	Semanal	Mensual	Anual	3 años
Oro	0,15%	0,41%	0,93%	1,49%	0,64%	1,55%	5,24%
Plata	0,00%	0,75%	1,17%	1,08%	-0,57%	-2,33%	-0,78%
Cobre	-0,03%	0,27%	1,16%	-0,07%	1,17%	11,29%	40,30%
Platino	0,03%	0,32%	1,61%	7,51%	12,35%	14,80%	-5,03%
Petróleo Brent	-0,40%	-0,25%	0,16%	2,09%	7,16%	30,95%	67,98%
Petróleo crudo WTI	-0,33%	-0,27%	0,14%	2,57%	12,66%	39,11%	59,04%
Gas natural	0,00%	1,16%	1,16%	-0,48%	-5,93%	-8,33%	35,43%
Aceite de Calefacción	-0,37%	-0,44%	0,07%	2,80%	2,20%	21,60%	70,26%
Café C EE.UU.	-0,50%	-0,70%	-0,45%	0,74%	-2,64%	-8,91%	-22,97%
Maíz EE.UU.	0,00%	0,07%	0,17%	0,24%	2,22%	-3,30%	0,10%
Trigo EE.UU.	0,00%	0,06%	-0,78%	0,14%	5,70%	-7,88%	0,41%
Azúcar N°5 Londres	-37,95%	0,09%		0,06%	-2,42%	-0,66%	-22,95%
Algodón N°2 EE.UU.	-0,13%	-0,09%	-0,35%	0,92%	6,34%	7,90%	29,86%

```
investing <- read_html("https://es.investing.com/commodities/")
```

```
cotizaciones <- investing %>%
```

```
html_nodes("table") %>% # Así capturamos la información en forma de tabla.
```

```
.[1] %>% # De lo que tenemos que darnos cuenta es que la tabla que queremos es la primera.
```

```
html_table()
```

```
cotizaciones<-as.data.frame(cotizaciones)
```

```
cotizaciones<-cotizaciones[,2:dim(cotizaciones)[2]]
```

# Ejercicio: Captura las cotizaciones del Ibex 35 de Bolsa de Madrid

## Precios de la sesión

Mercado	Seleccione mercado ▾	Índice	IBEX 35®	Sector	Seleccione sector ▾	Consultar
---------	----------------------	--------	----------	--------	---------------------	-----------

**IBEX 35®**

Nombre	Anterior	Último	% Dif.	Máximo	Mínimo	Fecha	Hora	% Dif. Año 2016
▼ IBEX 35®	8.901,50	8.803,30	-1,10	9.053,90	8.794,40	10/11/2016	16:20	-7,76

Nombre	Últ.	% Dif.	Máx.	Mín.	Volumen	Efectivo (miles €)	Fecha	Hora
▼ ABERTIS	12,6950	-2,79	13,1900	12,6700	2.339.932	30.363,53	10/11/2016	16:15
▼ ACCIONA	65,3100	-2,88	67,9600	65,1800	227.517	15.098,77	10/11/2016	16:15
▲ ACERINOX	11,8350	3,23	12,0250	11,6500	3.354.976	40.005,12	10/11/2016	16:16
▲ ACS	28,9250	2,12	29,7250	28,7500	1.318.000	38.662,99	10/11/2016	16:15
▼ AENA	124,8000	-4,51	132,2600	124,5500	163.584	21.028,41	10/11/2016	16:15
▼ AMADEUS	40,3400	-4,15	42,4050	40,2250	552.332	22.736,47	10/11/2016	16:15
▲ ARCELOM. IT.	6,6340	2,49	6,7400	6,4700	5.906.347	39.231,97	10/11/2016	16:15
▲ BA.POPULAR	0,9960	3,97	1,0300	0,9660	72.102.011	72.218,24	10/11/2016	16:16
▲ BA.SABADELL	1,3110	5,05	1,3300	1,2560	36.644.643	47.873,18	10/11/2016	16:15
▲ BA.SANTANDER	4,5160	2,13	4,6300	4,4840	69.772.626	318.783,11	10/11/2016	16:16
▲ BANKIA	0,8730	7,25	0,8790	0,8210	54.917.020	47.235,40	10/11/2016	16:16
▲ BANKINTER	7,2030	1,48	7,2850	7,1300	4.005.261	28.817,77	10/11/2016	16:15
▼ BBVA	6,0880	-0,52	6,2830	6,0750	45.352.838	280.742,88	10/11/2016	16:15
▲ CAIXABANK	2,9570	3,72	2,9900	2,8730	20.032.408	58.990,36	10/11/2016	16:16
▼ CELLNEX	12,9950	-6,48	14,0100	12,9800	1.754.254	23.707,48	10/11/2016	16:15
▼ DIA	4,6480	-1,46	4,7700	4,6300	4.331.657	20.305,32	10/11/2016	16:15
▼ ENAGAS	23,5250	-5,99	25,1450	23,5050	1.079.273	28.294,58	10/11/2016	16:15
▼ ENDESA	17,9150	-4,30	18,9000	17,9150	1.154.444	21.205,26	10/11/2016	16:12
▼ FERROVIAL	16,6100	-2,41	17,2600	16,5700	1.323.117	22.568,45	10/11/2016	16:16
▼ GAMESA	17,5900	-4,35	18,5750	17,4200	4.518.168	81.632,70	10/11/2016	16:15
▼ GAS NATURAL	16,5000	-3,25	17,1700	16,4600	1.420.284	23.966,58	10/11/2016	16:15
▼ GRIFOLS CL.A	18,3350	-1,45	18,7900	18,2350	1.240.038	22.998,05	10/11/2016	16:15
▲ IAG	5,1000	1,27	5,1700	5,0440	5.372.297	27.473,15	10/11/2016	16:15
▼ IBERDROLA	5,7240	-3,33	5,9580	5,7070	30.810.466	179.841,37	10/11/2016	16:15
▼ INDITEX	30,5400	-2,04	31,5000	30,4700	1.716.643	53.323,13	10/11/2016	16:15
▼ INDRA A	10,6800	-2,20	11,0100	10,6650	527.267	5.683,03	10/11/2016	16:15
▼ MAPFRE	2,6920	-1,32	2,7250	2,6670	6.613.021	17.807,52	10/11/2016	16:15
▲ MEDIASET	9,8950	2,77	10,0100	9,6940	1.001.700	9.930,54	10/11/2016	16:15
▼ MELIA HOTELS	10,8750	-0,14	11,1250	10,8600	470.083	5.181,58	10/11/2016	16:15
▼ MERLIN	9,2930	-4,00	9,7220	9,2630	2.324.548	22.074,94	10/11/2016	16:15
▼ R.E.C.	16,8500	-5,07	17,8400	16,8000	1.715.257	29.583,27	10/11/2016	16:15
▲ REPSOL	12,6750	0,32	12,9900	12,6400	4.570.209	58.666,31	10/11/2016	16:15
▲ TEC.REUNIDAS	33,3450	2,52	34,6650	33,3000	381.778	12.930,66	10/11/2016	16:15
▼ TELEFONICA	8,6390	-2,38	8,9510	8,6200	18.080.189	159.099,37	10/11/2016	16:15

```
bolsa <- read_html("http://www.bolsamadrid.es/esp/aspx/Mercados/Precios.aspx?indice=ESI1000000000")
cotizaciones <- bolsa %>%
  html_nodes("table") %>%
  .[5] %>% #
  html_table()
cotizaciones<-as.data.frame(cotizaciones)
```

Ejercicio: Obtén las últimas noticias (expansión), precios objetivo (expansión) y posiciones cortas (CNMV) de: Telefónica, Santander y Siemens Gamesa.

```
# Noticias: https://www.expansion.com/mercados/cotizaciones/valores/telefonica\_M.TEF.html
# Precios objetivo: https://www.expansion.com/mercados/bolsa/recomendaciones/consenso-mercados/t/telefonica\_M.TEF.html
# Posiciones cortas: https://www.cnmv.es/portal/Consultas/EE/PosicionesCortas.aspx?nif=A-28015865

library(rvest)

lista_empresas = c("santander_M.SAN", "siemensgamesa_M.SGRE", "telefonica_M.TEF")
lista_cifs = c("A-39000013", "A-01011253", "A-28015865")

for (empresa in 1:length(lista_empresas)){

  # Ultimas noticias
  url = paste("https://www.expansion.com/mercados/cotizaciones/valores/", lista_empresas[empresa], ".html", sep="")
  web <- read_html(url)
  noticias<- web %>%
    html_nodes("div.fila") %>%
    html_text()
  noticias<-as.data.frame(noticias)

  # Precios objetivo
  url = paste("https://www.expansion.com/mercados/bolsa/recomendaciones/consenso-mercados/",
  substring(lista_empresas[empresa],1,1), "/", lista_empresas[empresa], ".html" , sep="")
  web <- read_html(url)
  precio_objetivo<- web %>%
    html_nodes("table") %>%
    .[1] %>%
    html_table()
  precio_objetivo<-as.data.frame(precio_objetivo)

  # Posiciones cortas
  url = paste("https://www.cnmv.es/portal/Consultas/EE/PosicionesCortas.aspx?nif=", lista_cifs[empresa], sep="")
  web <- read_html(url)
  pos_cortas<- web %>%
    html_nodes("table") %>%
    .[1] %>%
    html_table()
  pos_cortas<-as.data.frame(pos_cortas)

  Sys.sleep(3)
}

# Fíjate que solo estás guardando la última empresa. No todas las que te has descargado.
# ¿Qué tendrías que hacer para guardar todas las empresas descargadas?
```

Ejercicio: Crea una función que descargue noticias de Google, pasándole como parámetros el nombre de la empresa que quieras descargar.

Todo Maps **Noticias** Imágenes Vídeos Más Configuración Herramientas

Aproximadamente 3.720.000 resultados (0,21 segundos)



Noticias Bancarias

[Los accionistas de Banco Santander cobrarán un dividendo de 0,23 ...](#)

Estrategias de Inversión - Hace 3 horas

[Banco Santander abona a primeros de noviembre el segundo dividendo a cuenta del actual ejercicio a través del programa Santander ...](#)

[Banco Santander pagará un dividendo de 23 céntimos por acción, un ...](#)

Noticias Bancarias - Hace 28 minutos

[Ver todos](#)



[La Guardia Civil avisa del engaño que afecta a clientes de Banco ...](#)

Ideal Digital - 24 oct. 2018

Si la semana pasada alertaban del engaño que afectaba de lleno al BBV ahora es el

[Banco Santander](#) el protagonista involuntario de un timo ...

[Ojo con una campaña que suplanta al Banco Santander](#)

Opinión - El Periódico - 24 oct. 2018



[La gestora de Banco Santander y la firma de la Sicav de Alierta aumentan su ...](#)

OKDIARIO - Hace 4 horas

La gestora de [Banco Santander](#) y la firma de inversión que gestiona la Sicav de César Alierta han redoblado en los últimos días su apuesta ...



[Ruptura millonaria de Banco Santander y Allianz por el Popular](#)

Economía Digital - 23 oct. 2018

El [Banco Santander](#) y Allianz están preparados para romper la relación que les une por la integración del Popular. Las empresas aún ...

[Santander y Allianz ultiman la ruptura millonaria del acuerdo en ...](#)

El Confidencial - 21 oct. 2018

# Obtén las noticias del Banco Santander

```
library(rvest)
library(tidyverse)
```

```
news <- function(term) {
  html_dat <- read_html(paste0("https://news.google.com/search?q=",term,"&hl=en-US&gl=US&ceid=US%3Aen"))
  dat <- data.frame(Link = html_dat %>%
    html_nodes('.VDXfz') %>%
    html_attr('href') %>%
    mutate(Link = gsub("./articles/","https://news.google.com/articles/",Link)))
  news_dat <- data.frame(
    Title = html_dat %>%
    html_nodes('.DY5T1d') %>%
    html_text(),
    Link = dat$Link,
    Description = html_dat %>%
    html_nodes('.Rai5ob') %>%
    html_text()
  )
  return(news_dat)
}
noticias = news("banco+santander")
```

# Captura todas las noticias que tengan relación con Telefónica, Santander y BBVA entre enero y noviembre de 2016 de la hemeroteca de ABC.

TEMAS ▶ Telefonica , Telefónica Moviles  
ABC

TODO (4257) NOTICIAS (4257)

Ordenar: Fecha Resultados por página 20 1 2 3 4 5 6 7 8 9 10 ...

 GENTE Kim Kardashian reaparece en las redes sociales un mes después del atraco que sufrió en París 01/11/2016 21:32:22 ABC.ES ...martes por fin ha actualizado su cuenta personal de Facebook. Kim Kardashian ha publicado una foto suya en la que aparece sentada con su teléfono móvil. Sin embargo, la foto podría ser antigua, ya que en su regazo se puede ver el teléfono que tenía

 GALICIA Móviles pinchados y tres sospechosos, los relevantes avances en el caso Diana Quer 01/11/2016 19:48:25 PATRICIA ABET ...tráfico de drogas. Tanto es así que estas mismas fuentes confirman que hay varios teléfonos móviles pinchados desde hace días. Pese al hermetismo con el que se maneja la información, se sabe que los agentes están volcados en esta línea de trabajo, aunque

 Trum reabasa a Clinton por un punto en un sondeo a siete días de las elecciones 01/11/2016 18:29:12 EFE , divulgado por el diario The Washington Post y la cadena ABC News, Trump logra un apoyo del 46 por ciento, frente al 45 por ciento para Clinton. La encuesta, elaborada del 27 al 30 de octubre entre 1.128 votantes probables por teléfono, indica también que

VALENCIA XarxaLlibres, Servef y asistencia tributaria: las consultas ciudadanas más frecuentes 01/11/2016 18:07:15 ABC ...de los distintos canales habilitados: servicio de atención del teléfono 012, Internet y oficinas presenciales PROP. La mayoría se realizaron en la Guía PROP a través de Internet, con un total de 12,4 millones de consultas, seguidas del canal telefónico de la ... de Transparencia continúa con el proceso de mejora de la información que ofrece al ciudadano a través de los diferentes puntos PROP (presenciales y on line), en coordinación con el teléfono 012 y el nuevo Portal de Transparencia GvaObera de la Generalitat. En

 CASTILLA LA MANCHA Desmantelan una trama que se hacía pasar por la Guardia Civil 01/11/2016 18:03:19 ABC Civil conocido que varios agricultores de Jumilla (Murcia) estaban recibiendo llamadas telefónicas de personas que, identificándose como guardias civiles, solicitaban colaboraciones económicas mediante la inserción de publicidad en revistas oficiales ... en varias publicaciones, aunque solo habían autorizado su colaboración con una. La Guardia Civil ha averiguado que alguno de los perjudicados habría recibido llamadas telefónicas en las que su interlocutor, simulando ser miembro de estos cuerpos

**El problema que tenemos aquí es que los resultados se encuentran en varias páginas.**

```
# http://www.abc.es/hemeroteca/resultados-busqueda-avanzada/noticia?exa=telefonica&rfec=20160101;20161031&or=1&nres=20
# http://www.abc.es/hemeroteca/resultados-busqueda-avanzada/noticia/pagina-2?exa=telefonica&rfec=20160101;20161031&or=1&nres=20
# http://www.abc.es/hemeroteca/resultados-busqueda-avanzada/noticia/pagina-3?exa=telefonica&rfec=20160101;20161031&or=1&nres=20
# http://www.abc.es/hemeroteca/resultados-busqueda-avanzada/noticia/pagina-4?exa=telefonica&rfec=20160101;20161031&or=1&nres=20
```

**# Creamos dataframes vacíos donde guardaremos todas las noticias que bajemos.**

```
noticias_acumuladas<-data.frame()
cuerpo_noticias<-data.frame()
fecha_noticias<-data.frame()
```

**# Creamos los parámetros de la búsqueda**

```
empresa<-"bbva" # Busca noticias que contengan alguno de estos términos. Si buscas más de una empresa escribe "telefonica+santander+bbva"
fecha_inicio<-"20150101" # año mes día
fecha_fin<-"20161115"
```

**# Extraemos los títulos de las noticias de la primera página.**

```
url<-gsub("","\"",paste("http://www.abc.es/hemeroteca/resultados-busqueda-avanzada/todo?exa=",empresa,"&rfec=",fecha_inicio,";",fecha_fin,"&or=1&nres=20"),fixed = TRUE)
abc<-read_html(url, encoding = "UTF-8") # La url de la primera página es siempre distinta.
noticias<- abc %>%
  html_nodes(".titulo") %>%
  html_text
noticias <- as.data.frame(noticias) # Con esto obtenemos los títulos de las noticias de la página en la que estemos.
noticias_acumuladas<-rbind(noticias_acumuladas,noticias)
```

cuerpo<- abc %>%
 html\_nodes("p") %>%
 html\_text
cuerpo <- as.data.frame(cuerpo) # Con esto obtenemos el cuerpo de las noticias de la página en la que estemos.
cuerpo\_noticias<-rbind(cuerpo\_noticias,cuerpo)

fecha<- abc %>%
 html\_nodes(".date") %>%
 html\_text
fecha <- as.data.frame(fecha)

# Obtenemos el número de páginas que tiene la búsqueda que hemos realizado.

num\_paginas<- abc %>%

html\_nodes("strong") %>%

html\_text

num\_paginas <- as.numeric(num\_paginas[12]) # El número de páginas está siempre en la posición 12 del nodo strong.

# Recorremos las páginas extrayendo los títulos de las noticias.

for (pagina in 2:num\_paginas){

url<-gsub(" ", "", paste("http://www.abc.es/hemeroteca/resultados-busqueda-avanzada/noticia/pagina-", "?exa=", empresa, "&rfec=", fecha\_inicio, ";", fecha\_fin, "&or=1&nres=20"), fixed = TRUE)

abc<-read\_html(url, encoding = "UTF-8")

noticias<- abc %>%

html\_nodes(".titulo") %>%

html\_text

noticias <- as.data.frame(noticias)

noticias\_acumuladas<-rbind(noticias\_acumuladas,noticias)

cuerpo<- abc %>%

html\_nodes("p") %>%

html\_text

cuerpo <- as.data.frame(cuerpo) # Con esto obtenemos el cuerpo de las noticias de la página en la que estemos.

cuerpo\_noticias<-rbind(cuerpo\_noticias,cuerpo)

fecha<- abc %>%

html\_nodes(".date") %>%

html\_text

fecha <- as.data.frame(fecha)

fecha\_noticias<-rbind(fecha\_noticias,fecha)

}

# Quitamos las cadenas de texto que no tienen que ver con los títulos de las noticias.

quitar<-c("Especiales", "Clasificados", "Institucional", "ABC", "Mapa web", "Enlaces Vocento", "Siga ABC en")

for (palabra in quitar){

indice<-!(palabra == noticias\_acumuladas)

noticias\_acumuladas<-as.data.frame(noticias\_acumuladas[indice,])

}

# Quitamos las cadenas de texto que no tienen que ver con el cuerpo de las noticias.

quitar<-c("", "Las noticias se actualizan cada 5 minutos.")

for (palabra in quitar){

indice<-!(palabra == cuerpo\_noticias)

cuerpo\_noticias<-as.data.frame(cuerpo\_noticias[indice,])

}

# Unimos los data frame

resultado<-data.frame(noticias\_acumuladas,cuerpo\_noticias, fecha\_noticias)

colnames(resultado)<-c("Título", "Contenido", "Fecha")

## Resultado deseado

Título	Contenido	Fecha
1 La gran banca cobró casi 14.000 millones en comisió...	Servimedia.La gran banca española (Santander, BBVA,...	30/10/2016 17:07:46
2 Merlin cierra la fusión con Metrovacesa y lanza la ma...	S. E. ....notario del traspaso de los edificios y centros ...	20/10/2016 22:03:48
3 ¿Cómo enviar dinero a un amigo con el móvil en 5 se...	Moncho Veloso ...corriente en uno de los bancos ad...	07/10/2016 9:20:06
4 La banca española lanza una aplicación común de pa...	Moncho Veloso ...podrá realizar previsiblemente ace...	06/10/2016 12:57:49
5 La banca española lidera en Europa el blindaje frente...	Maria Cuesta ...entidades españolas de mayor tamañ...	30/08/2016 10:51:01
6 Lassalle: «España no permitirá que se comercialicen ...	JESÚS GARCÍA CALERO ...proyección exterior de nue...	10/08/2016 17:29:00
7 Las tecnológicas son las empresas con mayor valor ...	Daniel Caballero ...de estas cinco tecnológicas no es...	08/08/2016 18:27:15
8 El interés del bono español a diez años baja del 1%...	EFE ...apertura en el -0,112% frente al -0,119 anterior, ...	01/08/2016 17:07:55
9 Metrovacesa y Merlin: historia de una fusión	Luis M. Ontoso ...omnipresente estigma en el sector ...	22/06/2016 13:40:16
10 Merlin y Metrovacesa se fusionan y crean la mayor in...	EFE ...otra serie de activos, fundamentalmente suelo, ...	21/06/2016 20:19:35
11 El Santander será el primer accionista de la nueva Me...	EUROPA PRESS ..En la práctica, la transacción supone...	21/06/2016 20:18:34
12 Todo lo que puede convertir un aula en un universo ...	ALEJANDRO CARRA ...empresas como Santander, BBV...	02/06/2016 13:40:41
13 La gran banca tiene refinanciados 80.070 millones d...	Moncho Veloso ...además 205.000 millones en prést...	26/05/2016 11:18:56
14 ¿Qué salud tiene la cartera crediticia de cada banco?	Moncho Veloso ...cuantía, un 48,7% tiene ya problem...	26/05/2016 11:08:16
15 Abierta oficialmente la temporada de bodas y banque...	MARÍA JESÚS PÉREZ , claro (que tiene que seguir su fir...	24/05/2016 8:26:34
16 Las 15 entidades españolas aprobaron los test de es...	...anteriores ocasiones. Solo habrá seis bancos espa...	23/05/2016 11:33:50
17 La atonía del negocio por los bajos tipos lastra las cu...	MONCHO VELOSO, MARÍA CUESTA ...la baja. Las siete ...	03/05/2016 10:19:52
18 La banca extranjera inicia su salida de las autopistas ...	LUIS M.ONTOSO 70% de la deuda, cerca de 2.380 mill...	26/04/2016 17:09:32
19 Telepizza pide un crédito de 215 millones para refin...	EUROPA PRESS ...sobre la base de un apalancamiento...	18/04/2016 23:11:10
20 La atonía del negocio y la digitalización fuerzan a la ...	...notable empeoramiento. Cuando esta tasa es del 5...	17/04/2016 14:09:54
21 La nulidad de las cláusulas suelo afectará principalm...	EUROPA PRESS ...de préstamos hipotecarios está difi...	16/04/2016 10:19:15
22 Los bancos españoles cumplen desde septiembre la...	EP Los bancos españoles, incluyendo al Santander, B...	11/04/2016 13:27:31

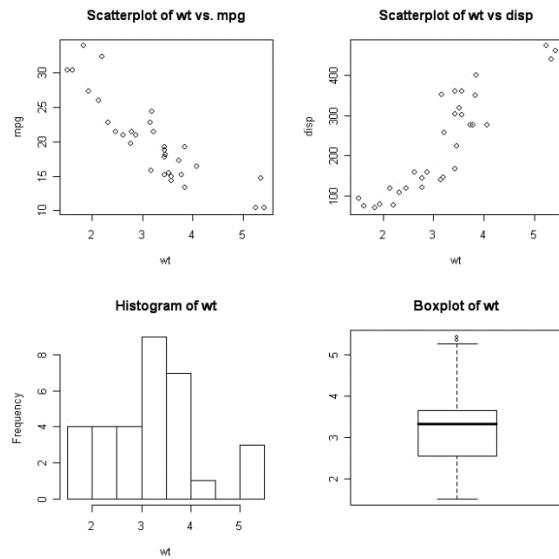
# Gráficos

## Código

## Gráfico resultante

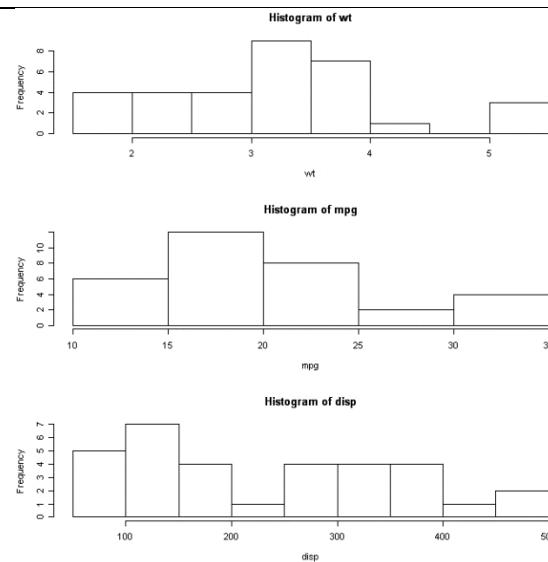
### 4 gráficos en 2 filas y dos columnas.

```
attach(mtcars) # Cargamos la base de datos.  
par(mfrow=c(2,2)) # Dividimos en 2 filas y 2 columnas.  
plot(wt,mpg, main="Scatterplot of wt vs. mpg")  
plot(wt,disp, main="Scatterplot of wt vs disp")  
hist(wt, main="Histogram of wt")  
boxplot(wt, main="Boxplot of wt")
```



### 3 filas 1 columna

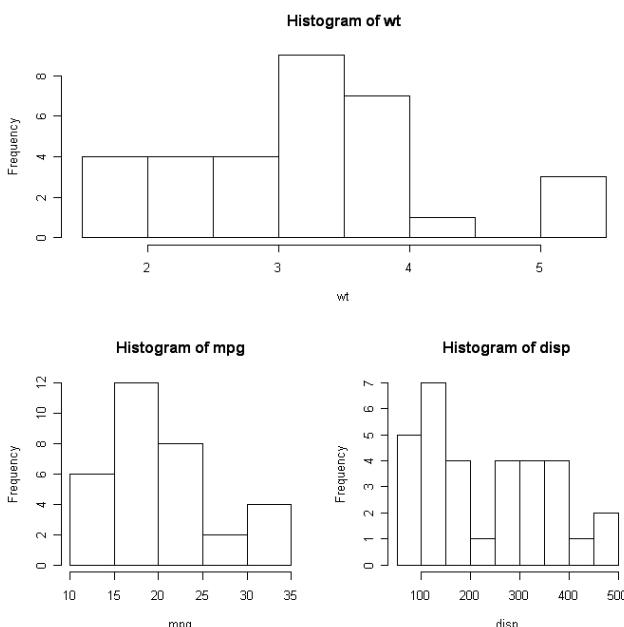
```
attach(mtcars)  
par(mfrow=c(3,1))  
hist(wt)  
hist(mpg)  
hist(disp)
```



### Combinaciones

```
attach(mtcars)  
layout(matrix(c(1,1,2,3), 2, 2, byrow = TRUE))  
hist(wt)  
hist(mpg)  
hist(disp)
```

# layout(matrix(c(1,1,2,3) le estamos indicando que el primer gráfico ocupa las dos primeras posiciones de la matriz (la fila de arriba), el segundo gráfico está en la 3<sup>a</sup> posición de la matriz (2<sup>a</sup> fila 1<sup>a</sup> columna) y el tercer gráfico en el hueco restante (2<sup>a</sup> fila 2<sup>a</sup> columna).



## Añadir diagramas de caja a un gráfico de dispersión

Enhanced Scatterplot

```
# Creamos el gráfico de dispersión.
```

```
par(fig=c(0,0.8,0,0.8), new=TRUE)
```

```
plot(mtcars$wt, mtcars$mpg, xlab="Car Weight",
```

```
ylab="Miles Per Gallon")
```

```
# Añadimos el boxplot de arriba.
```

```
par(fig=c(0,0.8,0.55,1), new=TRUE)
```

```
boxplot(mtcars$wt, horizontal=TRUE, axes=FALSE)
```

```
# Añadimos el boxplot de la derecha.
```

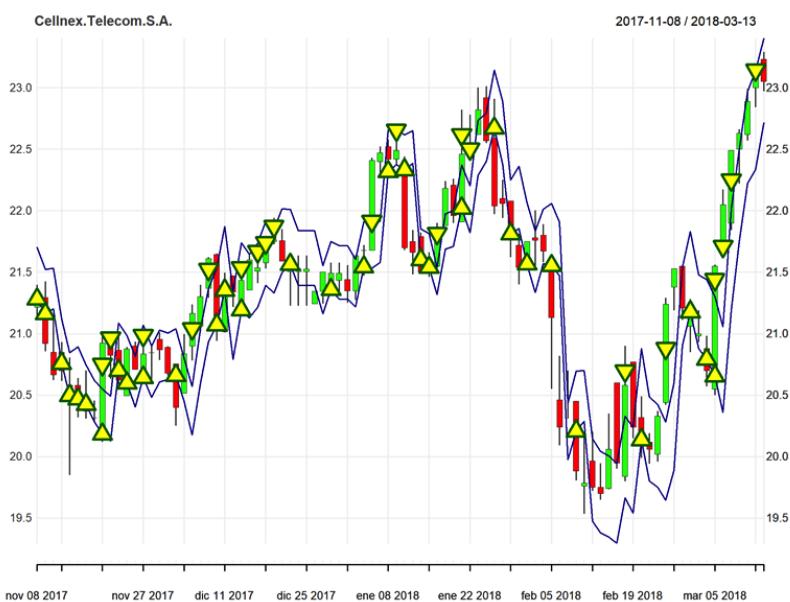
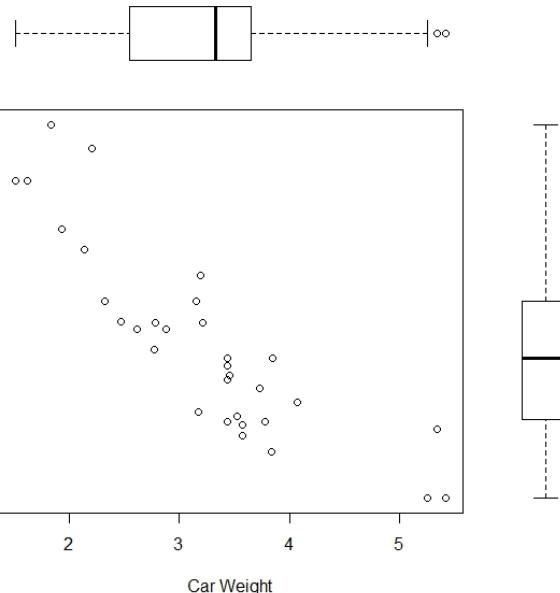
```
par(fig=c(0.65,1,0,0.8),new=TRUE)
```

```
boxplot(mtcars$mpg, axes=FALSE)
```

```
# Añadimos el título.
```

```
mtext("Enhanced Scatterplot", side=3, outer=TRUE,
```

```
line=-3)
```



## Gráficos personalizados

Ejemplo de gráfico personalizado con la librería base de R.

Las velas están generadas a partir de los precios descargados (máximo, mínimo, apertura y cierre).

Las bandas suponen los precios objetivo de compra y venta.

Los triángulos hacia arriba son las compras efectivas y los triángulos hacia abajo las ventas realizadas.

## Ejercicios

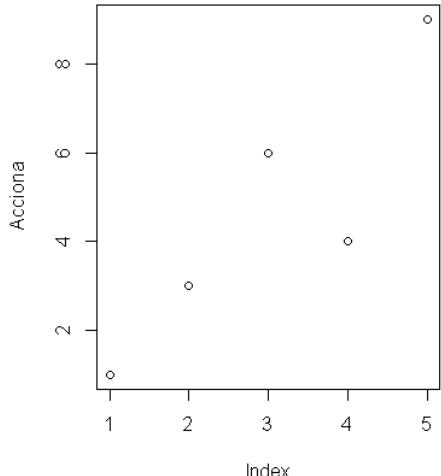
Programa los siguientes gráficos:

# Definimos un vector con 5 valores.

```
Acciona <- c(1, 3, 6, 4, 9)
```

# Graficamos el vector con todas las opciones en default.

```
plot(Acciona)
```



# Definimos 2 vectores

```
Acciona <- c(1, 3, 6, 4, 9)
```

```
Telefonica <- c(2, 5, 4, 5, 12)
```

# Graficamos Acciona con una línea azul, delimitando el eje Y entre 0 y 12

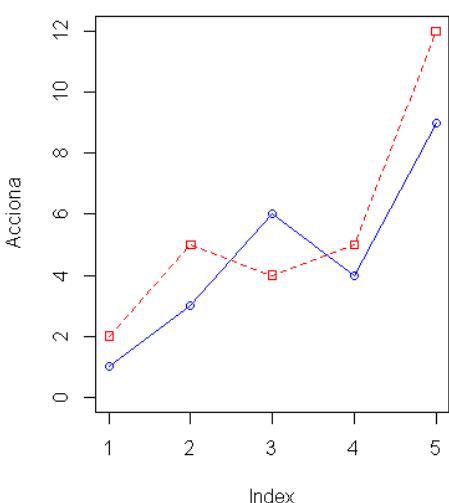
```
plot(Acciona, type="o", col="blue", ylim=c(0,12))
```

# Graficamos Telefonica con una línea de puntos roja y la añadimos al gráfico anterior.

```
lines(Telefonica, type="o", pch=22, lty=2, col="red")
```

# Ponemos el título en rojo y tamaño 4.

```
title(main="Comparativa de acciones", col.main="red", font.main=4)
```



# Definimos 3 vectores

```
Acciona<-c(1,3,6,4,9)
```

```
Santander<-c(2,5,4,5,12)
```

```
BBVA<-c(4,4,6,6,16)
```

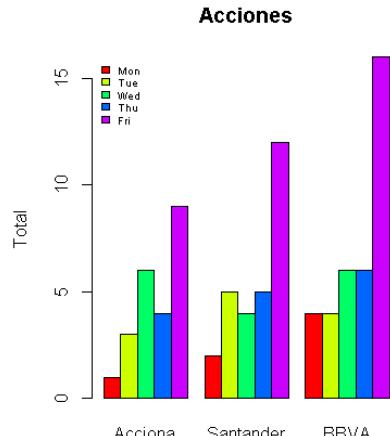
```
acciones<-data.frame(Acciona, Santander, BBVA)
```

# Graficamos los vectores usando rainbow colors

```
barplot(as.matrix(acciones), main="Acciones", ylab= "Total",  
beside=TRUE, col=rainbow(5))
```

# Ponemos la leyenda en la parte superior izquierda.

```
legend("topleft", c("Mon", "Tue", "Wed", "Thu", "Fri"), cex=0.6,  
bty="n", fill=rainbow(5))
```



```
# Definimos 3 vectores
```

```
Acciona<-c(1,3,6,4,9)
```

```
Santander<-c(2,5,4,5,12)
```

```
BBVA<-c(4,4,6,6,16)
```

```
acciones<-data.frame(Acciona, Santander, BBVA)
```

```
# Modificamos el margen derecho para que quepa la leyenda.
```

```
par(xpd=T, mar=par()$mar+c(0,0,0,4))
```

```
# Graficamos las acciones usando heat colors,
```

```
# Ponemos un 10% del espacio entre cada barra y hacemos las etiquetas más pequeñas
```

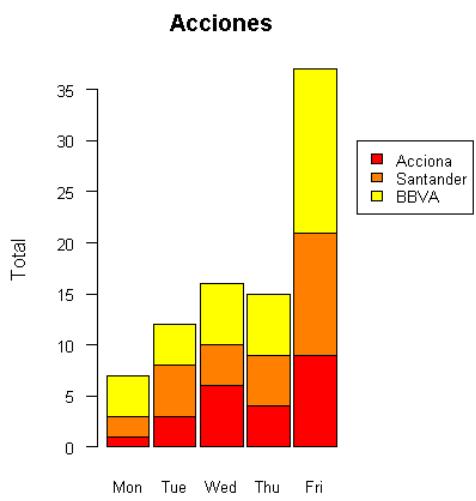
```
barplot(t(acciones), main="Acciones", ylab="Total",
```

```
col=heat.colors(3), space=0.1, cex.axis=0.8, las=1,
```

```
names.arg=c("Mon","Tue","Wed","Thu","Fri"), cex=0.8)
```

```
# Ponemos la leyenda en (6,30) usando heat colors
```

```
legend(6, 30, names(acciones), cex=0.8, fill=heat.colors(3))
```



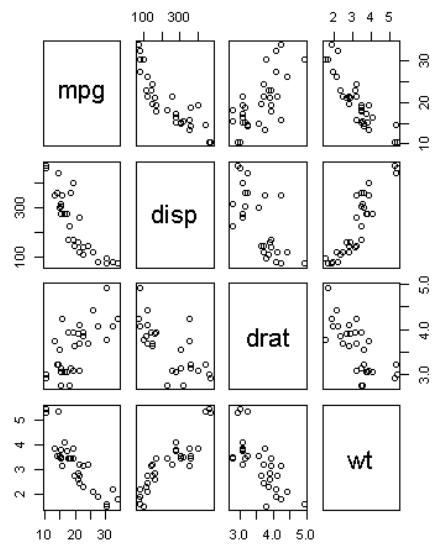
---

```
# Cargamos los datos de mtcars
```

```
attach(mtcars)
```

```
# Graficamos las variables mpg, disp, drat y wt, unas contra otras.
```

```
pairs(~mpg+disp+drat+wt,data=mtcars)
```



# Gráficos avanzados (ggplot2)

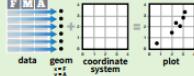
<https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf>

## Data Visualization with ggplot2 Cheat Sheet

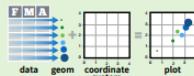


### Basics

ggplot2 is based on the **grammar of graphics**, the idea that you can build every graph from the same few components: a **data set**, a set of **geoms**—visual marks that represent data points, and a **coordinate system**.



To display data values, map variables in the data set to aesthetic properties of the geom like **size**, **color**, **x** and **y** locations.



Build a graph with **qplot()** or **ggplot()**

**qplot(x = city, y = hwy, color = cyl, data = mpg, geom = "point")**  
Creates a complete plot with given data, geom, and mappings. Supplies many useful defaults.

**ggplot(data = mpg, aes(x = cyl, y = hwy))**

Begins a plot that you finish by adding layers to. No defaults, but provides more control than qplot().

**ggplot(mpg, aes(hwy, cyl)) + geom\_point(aes(color = cyl)) + geom\_smooth(method = "lm") + coord\_cartesian() + scale\_color\_gradient() + theme\_bw()**

Add a new layer to a plot with a **geom\_\*** or **stat\_\*** function. Each provides a geom, a set of aesthetic mappings, and a default stat and position adjustment.

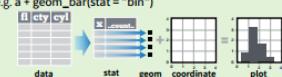
**last\_plot()**

Returns the last plot

**ggsave("plot.png", width = 5, height = 5)**  
Saves last plot as 5'x5' file named "plot.png" in working directory. Matches file type to file extension.

### Stats - An alternative way to build a layer

Some plots visualize a **transformation** of the original data set. Use a **stat** to choose a common transformation to visualize, e.g. **a + geom\_bar(stat = "bin")**



Each stat creates additional variables to map aesthetics to. These variables use a common **..name..** syntax.

stat functions and geom functions both combine a stat with a geom to make a layer, i.e. **stat\_bin(geom="bar")** does the same as **geom\_bar(stat="bin")**

**i + stat\_density2d(aes(fill = ..level..), geom = "polygon", n = 100)**  
geom for layer parameters for stat

**a + stat\_bin(binwidth = 1, origin = 10)** 1D distributions  
**x, y, ..count.., ..density.., ..density..**  
**a + stat\_bindot(binwidth = 1, radius = "x")**  
**x, y, ..count.., ..density..**  
**a + stat\_density(sjfun = 1, kernel = "gaussian")**  
**x, y, ..count.., ..density.., ..scaled..**

**f + stat\_bin2d(bins = 30, drop = TRUE)** 2D distributions  
**x, y, ..fill.., ..density.., ..density..**  
**f + stat\_binnedx(bins = 30)**  
**x, y, ..fill.., ..density..**  
**f + stat\_density2d(contour = TRUE, n = 100)**  
**x, y, size | ..level..**

**m + stat\_contour(aes(z = z))** 3 Variables  
**x, y, ..order.. | ..level..**

**m + stat\_spoke(aes(radius = z, angle = z))**  
angle, radius, x, xend, y, yend | ..x.., ..y.., ..xend.., ..y.., ..yend..

**m + stat\_summary\_hexes(aes(z = z), bins = 30, fun = mean)**  
**x, y, ..fill.. | ..value..**

**m + stat\_summary2d(aes(z = z), bins = 30, fun = mean)**  
**x, y, ..fill.. | ..value..**

**g + stat\_boxplot(coef = 1.5)** Comparisons  
**x, y | ..lower.., ..middle.., ..upper.., ..outliers..**

**g + stat\_ydensity(aes(adjust = 1, kernel = "gaussian", scale = "area"))**  
**x, y | ..scaled.., ..count.., ..n.., ..lowinwidth.., ..width..**

**f + stat\_ecdf(n = 40)** Functions  
**x, y | ..x.., ..y..**

**f + stat\_quantile(aes = c(0.25, 0.5, 0.75), formula = y ~ log(x), method = "rq")**  
**x, y | ..quantile.., ..x.., ..y..**

**f + stat\_smooth(method = "auto", formula = y ~ x, se = TRUE, n = 80, fullrange = FALSE, level = 0.95)**  
**x, y | ..se.., ..x.., ..y.., ..ymin.., ..ymax..**

**ggplot() + stat\_function(aes(x = -3:3))** General Purpose  
**x, y | ..n.., ..args.. = list(sd = 0.5)**

**f + stat\_identity()**  
**ggplot() + stat\_qq(aes(sample = 1:100), distribution = qt, dparams = list(dfr = 5))**  
**sample, x, y | ..x.., ..y..**

**f + stat\_sum()**  
**x, y, size | ..size..**

**f + stat\_summary(fun.data = "mean\_cl\_boot")**

**Geoms** - Use a geom to represent data points, use the geom's aesthetic properties to represent variables. Each function returns a layer.

#### One Variable

##### Continuous

**a + geom\_area(stat = "bin")**

**a + geom\_density(kernel = "gaussian")**

**a + geom\_dotplot()**

**a + geom\_freqpoly()**

**a + geom\_histogram(binwidth = 5)**

**b + geom\_rug(sides = "l")**

**b + geom\_smooth(model = lm)**

**C f + geom\_text(aes(label = cyl))**

**AB**

##### Discrete

**b < ggplot(mpg, aes(fl))**

**b + geom\_bar()**

**x, alpha, color, fill, linetype, size, weight**

**Graphical Primitives**

**c < ggplot(map, aes(long, lat))**

**c + geom\_polygon(aes(group = group))**

**d < ggplot(economics, aes(date, unemploy))**

**d + geom\_path(lineend = "butt", linejoin = "round", linemetre = 1)**

**d + geom\_ribbon(aes(ymin = unemploy - 900, ymax = unemploy + 900))**

**e < ggplot(seals, aes(x = long, y = lat))**

**e + geom\_segment(aes(xend = long + delta\_long, yend = lat + delta\_lat))**

**e + geom\_rect(aes(xmin = long, ymin = lat, xmax = long + delta\_long, ymax = lat + delta\_lat))**

**h + geom\_jitter()**

**x, y, alpha, color, fill, shape, size**

**Discrete X, Continuous Y**

**g + geom\_bar(stat = "identity")**

**g + geom\_boxplot()**

**g + geom\_dotplot(binaxis = "y", stackdir = "center")**

**g + geom\_violin(scale = "area")**

**Discrete X, Discrete Y**

**h < ggplot(diamonds, aes(cut, color))**

**h + geom\_jitter()**

**x, y, alpha, color, fill, shape, size**

**Continuous Function**

**j < ggplot(economics, aes(date, unemploy))**

**j + geom\_area()**

**j + geom\_line()**

**j + geom\_step(direction = "hv")**

**Continuous Bivariate Distribution**

**i < ggplot(movies, aes(year, rating))**

**i + geom\_bin2d(binwidth = c(0.5, 0.5))**

**xmax, xmin, ymax, ymin, alpha, color, fill, linetype, size, weight**

**i + geom\_hex()**

**x, y, alpha, colour, fill, size**

**Visualizing error**

**df < data.frame(grp = c("A", "B"), fit = 4:5, se = 1:2)**

**k < ggplot(df, aes(grp, fit, ymin = fit - se, ymax = fit + se))**

**k + geom\_crossbar(fatten = 2)**

**x, y, ymax, ymin, alpha, color, fill, linetype, size**

**k + geom\_errorbar()**

**x, ymax, ymin, alpha, color, linetype, size, width (also geom\_errorbar())**

**k + geom\_linerange()**

**x, ymin, ymax, alpha, color, linetype, size**

**k + geom\_pointrange()**

**x, y, ymin, ymax, alpha, color, fill, linetype, size**

**Maps**

**data <- data.frame(murder = USArrests\$Murder,**

**state = tolower(rownames(USArrests)))**

**map <- map\_data("state")**

**l < ggplot(data, aes(fill = murder))**

**l + geom\_map(aes(map\_id = state, map = map) +**

**expand\_limits(x = map\$xlong, y = map\$lat)**

**map\_id, alpha, color, fill, linetype, size**

**Three Variables**

**seals\$z <- with(seals, sqrt(delta\_long^2 + delta\_lat^2))**

**m < ggplot(seals, aes(long, lat))**

**m + geom\_contour(aes(z = z))**

**x, y, z, alpha, colour, linetype, size, weight**

**Coordinate Systems**

**r < b + geom\_bar()**

**r + coord\_cartesian(xlim = c(0, 5))**

**xlim, ylim**

**The default cartesian coordinate system**

**r + coord\_fixed(ratio = 1/2)**

**ratio, xlim, ylim**

**Cartesian coordinates with fixed aspect ratio between x and y units**

**r + coord\_flip()**

**xlim, ylim**

**Flipped Cartesian coordinates**

**r + coord\_polar(theta = "x", direction = 1)**

**theta, start, direction**

**Polar coordinates**

**r + coord\_trans(transt = "sqrt")**

**xtrans, ytrans, xlim, ylim**

**Transformed cartesian coordinates. Set extras and strans to the name of a window function.**

**z + coord\_map(projection = "ortho", orientation = c(41, -74, 0))**

**projection, orientation, xlim, ylim**

**Map projections from the mapproj package (mercator (default), azequarea, lagrange, etc.)**

**Position Adjustments**

**Position adjustments determine how to arrange geoms that would otherwise occupy the same space.**

**s < ggplot(mpg, aes(ttl, fill = drv))**

**s + geom\_bar(position = "dodge")**

**Arrange elements side by side**

**s + geom\_bar(position = "fill")**

**Stack elements on top of one another, normalize height**

**s + geom\_bar(position = "stack")**

**Stack elements on top of one another**

**f + geom\_point(position = "jitter")**

**Add random noise to X and Y position of each element to avoid overplotting**

**s + geom\_bar(position = position\_dodge(width = 1))**

**Each position adjustment can be recast as a function with manual width and height arguments**

**Labels**

**t + ggtitle("New Plot Title")**

**Add a main title above the plot**

**t + xlab("New X label")**

**Change the label on the X axis**

**t + ylab("New Y label")**

**Change the label on the Y axis**

**t + labs(title = "New title", x = "New x", y = "New y")**

**All of the above**

**Legends**

**t + theme(legend.position = "bottom")**

**Place legend at "bottom", "top", "left", or "right"**

**t + guides(color = "none")**

**Set legend type for each aesthetic: colorbar, legend, or none (no legend)**

**t + scale\_fill\_discrete(name = "Title", labels = c("A", "B", "C"))**

**Set legend title and labels with a scale function.**

**Themes**

**t + theme\_bw()**

**White background with grid lines**

**t + theme\_classic()**

**White background no gridlines**

**t + theme\_grey()**

**Grey background (default theme)**

**t + theme\_minimal()**

**Minimal theme**

**ggthemes - Package with additional ggplot2 themes**

**Zooming**

**Without clipping (preferred)**

**t + coord\_cartesian(xlim = c(0, 100), ylim = c(10, 20))**

**With clipping (removes unseen data points)**

**t + xlim(0, 100) + ylim(10, 20)**

**t + scale\_x\_continuous(limits = c(0, 100)) +**

**scale\_y\_continuous(limits = c(0, 100))**

# Análisis de rendimiento

Incluso los programadores experimentados tienen dificultades para la identificación de cuellos de botella en su código.

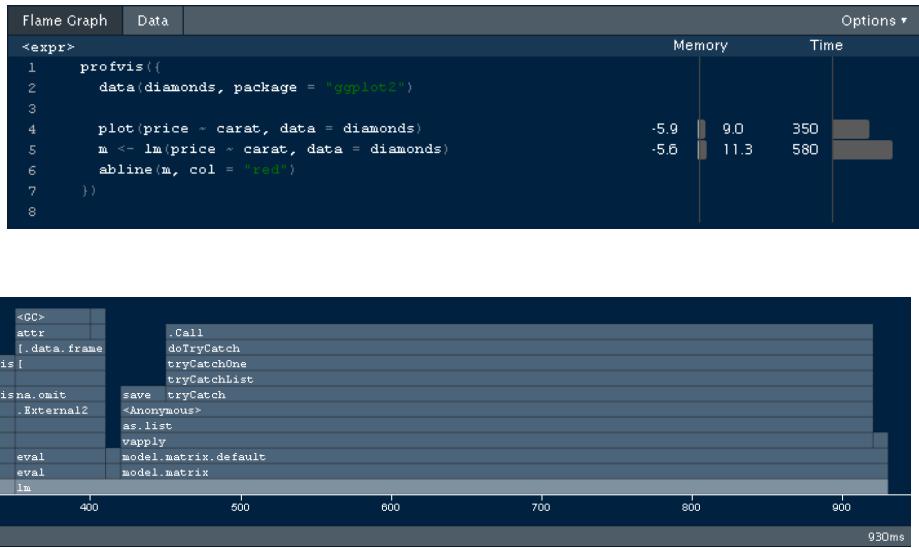
Para poder realizar el análisis instálate profvis en R. `install.packages("profvis")`

Para realizar el análisis de rendimiento de un programa, lo único que tenemos que hacer es “meter” el código que queremos analizar dentro de la instrucción profvis:

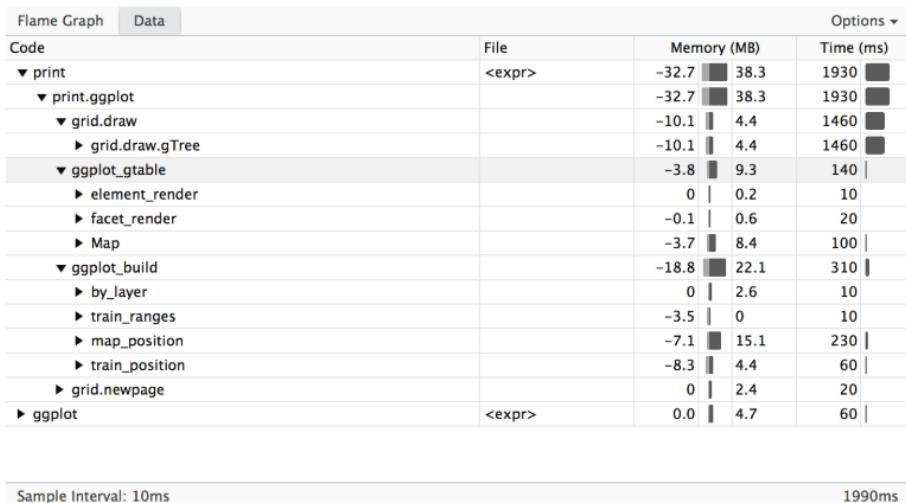
```
library(profvis)
library(ggplot2)

profvis({
  data(diamonds, package = "ggplot2")

  plot(price ~ carat, data = diamonds)
  m <- lm(price ~ carat, data = diamonds)
  abline(m, col = "red")
})
```



Al ejecutarse se abrirá una nueva pestaña de Data en donde podremos ver el tiempo, en milisegundos, que ha consumido cada línea de código. Justo debajo podremos ver un gráfico “de llamas” interactivo que nos muestra la pila de llamadas que ha realizado cada instrucción y su tiempo consumido.



Puedes consultar toda la información de la librería y ver más ejemplos en el siguiente enlace. <https://rstudio.github.io/profvis/>

## Medición de la duración de un programa

Una manera rápida de analizar la duración de un código es poner al inicio y final del mismo las siguientes instrucciones.

```
tiempo <- proc.time() # Inicia el cronómetro
...
proc.time()-tiempo # Detiene el cronómetro (elapsed es el tiempo que ha tardado el programa)
```

Como podemos ver, en el ejemplo de la izquierda, es muy fácil detectar cuellos de botella dentro del código.

La línea `grid.draw.gTree` ha consumido 1460 milisegundos del total de 1930 del programa.

Esa línea sola consume el 75% del tiempo de todo el programa. Deberemos pensar, por lo tanto, si hay otra manera de realizar la misma función de un modo más eficiente, o si es un tiempo razonable y, por lo tanto, asumible para nuestro código.

Ejemplo de análisis de eficiencia: 50.000 combinaciones de lotería.

Objetivo: Análisis de rendimiento del programa. ¿Dónde están los cuellos de botella?

```
library(profvis)
```

```
profvis({
```

```
  tiempo <- proc.time() # Inicia el cronómetro
```

**# Establecemos una semilla de generación aleatoria determinada para que los rdos sean siempre iguales.**  
set.seed(1000)

**# Hacemos una función que saque una bola y compruebe que no ha salido ya en la combinación actual.**

```
sacar_bola <- function(combi,nbola){  
  bola <- sort(sample(seq(1,50,1), 1, replace = TRUE))
```

**# Comprobamos que este número no ha salido ya en la combinación actual**

```
for (comprobar_bola in 1:nbola){  
  if (bola == combi[comprobar_bola]){  
    bola <- sort(sample(seq(1,50,1), 1, replace = TRUE)) # Si la bola ya ha salido sacamos otra.  
  }  
}  
return(bola)
```

**# Sacamos la combinación ganadora**

```
combi_ganadora<- matrix(0,nrow=1,ncol=5,byrow=T) # Creamos una matriz para la combi ganadora.  
for (nbola in 1:5){  
  combi_ganadora[nbola]<- sacar_bola(combi_ganadora,nbola)  
}
```

**# Sacamos las combinaciones apostadas y comprobamos cuantos aciertos tenemos en cada una de ellas.**

```
combinaciones <- 50000  
apuestas<- matrix(0,nrow=combinaciones,ncol=5,byrow=T) # Creamos una matriz para las apuestas.  
aciertos<-matrix(0,nrow=combinaciones,ncol=1,byrow=T) # Creamos un vector para los aciertos.  
barra_progreso <- winProgressBar(title= "Barra de progreso", min = 0, max = combinaciones, width=300)  
# width es el nº de pixeles de la barra.
```

```
for (combinacion in 1:combinaciones){
```

**# Obtenemos las apuestas realizadas (sacamos una combinación)**

```
combi<- matrix(0,nrow=1,ncol=5,byrow=T) # Creamos una matriz donde guardaremos la combi apostada  
for (nbola in 1:5){  
  combi[nbola]<- sacar_bola(combi,nbola)  
}  
apuestas[combinacion,>]<-combi
```

**# Comprobamos los aciertos que tenemos entre nuestras apuestas y la combinación ganadora.**

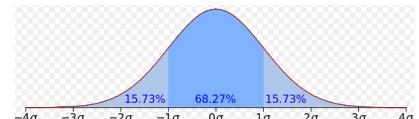
```
for (bola_apostada in 1:5){  
  for (bola_premiada in 1:5){ # comparamos cada bola_apostada con cada bola_premiada  
    if(apuestas[combinacion,bola_apostada]==combi_ganadora[bola_premiada]){  
      aciertos[combinacion]<- aciertos[combinacion]+1  
    }  
  }  
}  
setWinProgressBar(barra_progreso, combinacion, title=paste(round(combinacion/combinaciones*100,0), "% realizado"))  
}  
close(barra_progreso)
```

**# Calculamos la frecuencia de los aciertos (cuantas veces hemos acertado 1 nº, cuantas veces 2 etc)**

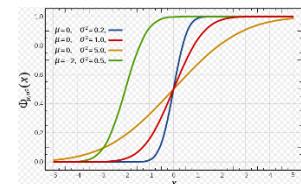
```
library(plyr)  
aciertos<-count(aciertos)  
aciertos  
print(proc.time()-tiempo) # Detiene el cronómetro (elapsed es el tiempo que ha tardado el programa)  
})
```

# Trabajar con distribuciones en R

**Función de densidad:** Probabilidad de que una variable tome un determinado valor.



**Función de distribución:** Indica la probabilidad de que la variable tome un valor igual o menor que  $x$ . Se calcula acumulando el valor de la función de densidad. Es la probabilidad acumulada. Es decir, da la probabilidad de un suceso.



**Función cuantil:** Es la inversa de una función de distribución. Dada una probabilidad indica el valor que tomará la variable.

R denomina de la siguiente manera a las funciones en todas las distribuciones:

dxxx(función de densidad)  
pxxx(función de distribución)  
qxxx (función cuantil)  
rxxx (generación de nº aleatorios)

## Librerías estadísticas útiles

```
library(stats) # paquete de funciones estadísticas de R  
library(pastecs) # Análisis de series temporales.
```

## Simulación de variables aleatorias discretas

```
sample(x, tamaño, replace = FALSE, prob = NULL)
```

x: vector de más de un elemento del que elegir las ocurrencias.

Tamaño: nº de ocurrencias o extracciones a realizar.

Replace: Indica si la extracción se hace o no con remplazamiento.

Prob: vector de pesos a asignar a cada uno de los posibles valores. Por defecto, todos los valores son equiprobables.

Ejemplo

```
dadoTrucoNum=sample(c(1:6), 1000, replace = TRUE, prob = c(2,3,1,9,8,5))
```

Obtenemos un dado trucado, con mayor probabilidad de que salga el 4 y 5, que el resto.

Si ponemos peso 0, ese valor no saldrá nunca.

table(dadoTrucoNum) → Te pone los resultados en bonito, en una tabla.

## Simulación de variables aleatorias continuas

### Distribución de Bernoulli

Arroja 1 o 0, cara o cruz. Se suele utilizar para realizar simulaciones de default (con distintas probabilidades al 50%, 5%, 1%...)

```
dbern(x, prob, log=FALSE)  
pbern(q, prob, lower.tail=TRUE, log.p=FALSE)  
qbern(p, prob, lower.tail=TRUE, log.p=FALSE)  
rbern(n, prob)
```

X y Q son vectores de cuantiles (ver ayuda para estos parámetros).  
p es el vector de probabilidades.  
n cuantos números aleatorios queremos.  
Prob: probabilidad del suceso a medir. Ej: simulamos default con prob 2%  
log o log.p da T o F en lugar de 1 o 0.  
Lower.tail: Si es true, las probabilidades son  $X \leq x$ , sino  $X > x$ .

```
library(Rlab)  
operaciones <- rbern(1000, 0.02)  
table(operaciones)
```

Resultado

0	1
979	21

### Distribución uniforme

dunif(x, min = 0, max = 1, log = FALSE) función de densidad (da la probabilidad de un suceso. Ej un dado 1/6)  
punif(q, min = 0, max = 1, lower.tail = TRUE, log.p = FALSE) Función de distribución (da la probabilidad acumulada en un pto)  
qunif(p, min = 0, max = 1, lower.tail = TRUE, log.p = FALSE) Función cuantil (da un punto a una probabilidad)  
runif(n, min = 0, max = 1) # Sacamos números aleatorios con distribución uniforme.

operaciones <- runif(1000, min = 100, max = 200) # Sacamos 1000 nº aleatorios.  
table(operaciones) # Esto no vale de nada porque es una tabla enorme, que no es útil de ver.  
hist(operaciones) # Para ver la uniformidad de los números, hacemos el histograma.

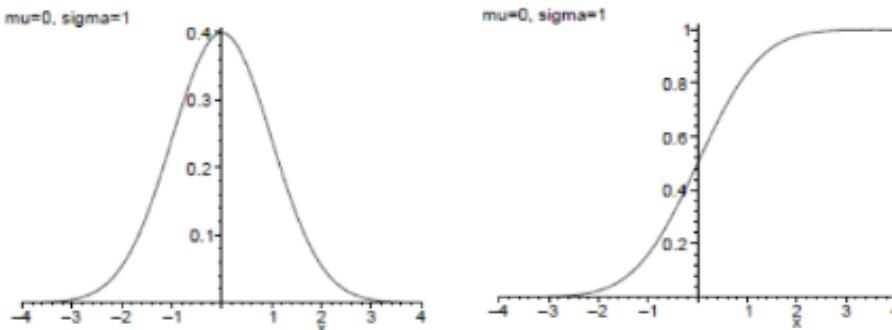
## Distribución Normal

`dnorm(x, mean = 0, sd = 1, log = FALSE)` función de densidad (da la probabilidad de un suceso).

`pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)` Probabilidad acumulada en un pto.

`qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)` Punto en función de una probabilidad.

`rnorm(n, mean = 0, sd = 1)` Obtención de números aleatorios.



Ojo, al sacar números aleatorios no suelen surgir por encima o debajo de 5 desviaciones típicas, pero los límites son  $-\infty$  y  $+\infty$ , por lo que podría pasar.

Percentiles y desviaciones.

90% 1,6448	En 1,6 desviaciones típicas tenemos el 90% de los datos.
95% 1,9599	En 1,9 desviaciones tenemos el 95% de los datos.
99% 2,5758	En 2,5 desviaciones tenemos el 99% de los datos.
99,99% 4	En 4 desviaciones tenemos el 99,99% de los datos.

El parámetro  $\mu$  (media) desplaza el centro de la distribución, el parámetro  $\sigma$  (desv) la ensancha o estrecha.  $(X - \mu) / \sigma \sim N(0,1)$

operaciones <- rnorm(10000, mean = 0, sd = 1)

hist(operaciones, breaks=seq(from=-5,to=5,by=0.2))

plot(operaciones) # Graficarlo así no tiene sentido.

**Ejercicio:** ¿Cómo consigo 1000 nº aleatorios que sigan una distribución normal entre 100 y 200 con una probabilidad del 99%?

1º determinar el % de números que deseó estén dentro de este rango. Para un 99% le corresponde una desviación del 2,57.

2º Obtenemos el nº de desviaciones necesarias  $\rightarrow$  media - 2,57  $\sigma$  = límite inferior.

En este caso  $150 - 2,57 \sigma = 100 \rightarrow \sigma = 19,41$

3º rnorm(1000,150,19.41) # Utilizando los parámetros anteriores, sacamos los 1.000 números aleatorios.

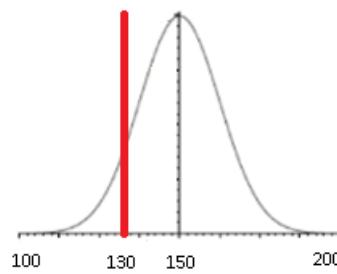
hist(operaciones) # Hacemos el histograma.

length(operaciones) # Comprobamos que hemos obtenido 1.000 números aleatorios.

**Ejercicio:** Quiero acotar la distribución anterior y que no me de los nº menores de 130, pero me siga dando 1000 nº

```
numeros_sacados <- 0
operaciones <- 1:1000
while (numeros_sacados<1000){
  numero_nuevo<- rnorm(1,150,19.41)
  if (numero_nuevo>=130) {
    numeros_sacados <- numeros_sacados +1
    operaciones[numeros_sacados] <- numero_nuevo
  }
}
hist(operaciones, breaks=seq(from=120,to=220,by=5))
length(operaciones)
```

Es decir, quiero obtener esta distribución



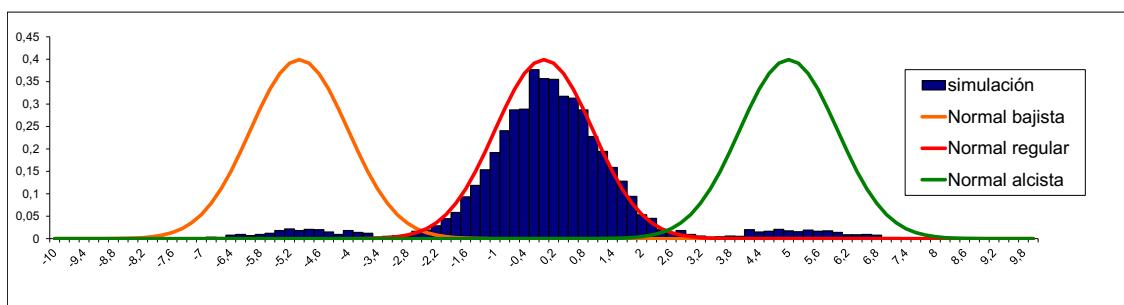
**Mixturas:** ¿Cómo se puede realizar una mezcla de distribuciones?

**Ejercicio:** Queremos mezclar tres distribuciones normales distintas (como en el gráfico). ¿Cómo lo hacemos?

Lo único que hay que saber es qué % de cada una queremos (ej: 5% de la 1ª, 90% de la segunda y 5% de la tercera).

Elegimos qué distribución usamos mediante una aleatoria uniforme.

Para comprobar la muestra resultante hacemos el histograma, donde comprobamos el resultado.

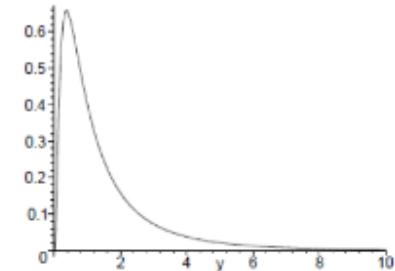


## Distribución Log-Normal

Los rendimientos siguen una distribución normal, los precios log normales (xq no pueden ser negativos)

```
dlnorm(x, meanlog = 0, sdlog = 1, log = FALSE)
plnorm(q, meanlog = 0, sdlog = 1, lower.tail = TRUE, log.p = FALSE)
qlnorm(p, meanlog = 0, sdlog = 1, lower.tail = TRUE, log.p = FALSE)
rlnorm(n, meanlog = 0, sdlog = 1)

operaciones <- rlnorm(1000, meanlog = 0, sdlog = 1)
hist(operaciones, breaks=seq(from=0,to=30,by=1))
# Ojo, si da error, es porque ha salido un dato por encima del "to 30"
# Habría que aumentar el límite.
```



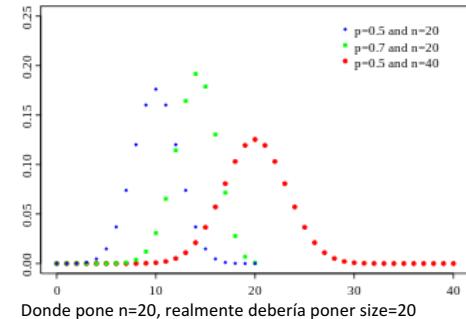
## Distribución Binomial

Es una distribución de Bernoulli N veces (sucesos independientes y con la misma probabilidad)

Se usa en distribuciones de pérdidas.

```
dbinom(x, size, prob, log = FALSE)
pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)
rbinom(n, size, prob)

operaciones <- rbinom(1000, 20, 0.5)
hist(operaciones, breaks=seq(from=0,to=30,by=1))
```



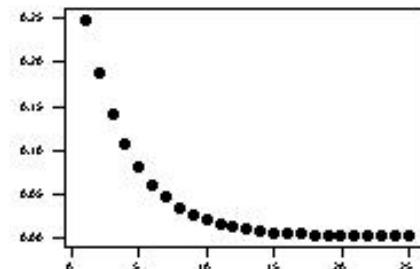
Donde pone n=20, realmente debería poner size=20

## Distribución geométrica

La distribución geométrica es un modelo adecuado para aquellos procesos en los que se repiten pruebas hasta la consecución del éxito. Por ejemplo, hasta que una empresa con una probabilidad de default p entra en quiebra.

```
dgeom(x, prob, log = FALSE)
pgeom(q, prob, lower.tail = TRUE, log.p = FALSE)
qgeom(p, prob, lower.tail = TRUE, log.p = FALSE)
rgeom(n, prob)

operaciones<- rgeom(1000, 0.05)
hist(operaciones, breaks=seq(from=0,to=120,by=5))
```



## Distribución de Poisson

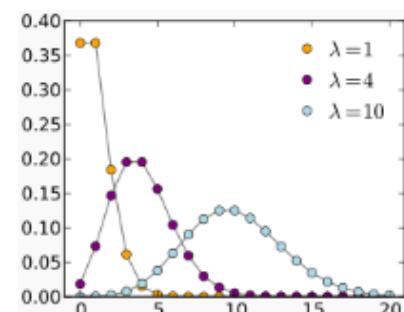
Se utiliza para simular nº de incumplimientos (o nº de errores en una página).

```
dpois(x, lambda, log = FALSE)
ppois(q, lambda, lower.tail = TRUE, log.p = FALSE)
qpois(p, lambda, lower.tail = TRUE, log.p = FALSE)
rpois(n, lambda)
```

Permite aproximar binomiales  $\lambda=n*p$

```
operaciones <- rpois(1000, 1)
hist(operaciones)
```

```
operaciones <- rpois(1000, 50)
hist(operaciones)
```



## Distribución $\chi^2$

Una distribución  $\chi^2$  centrada es igual a una distribución normal.

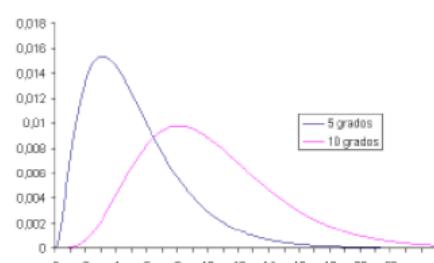
Se usa para contrate de varianzas o hipótesis.

La "chepa" se desplaza a la derecha cuantos más grados pongamos.

```
dchisq(x, df, ncp = 0, log = FALSE)
pchisq(q, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
qchisq(p, df, ncp = 0, lower.tail = TRUE, log.p = FALSE)
rchisq(n, df, ncp = 0) # Df son grados de libertad.

operaciones <- rchisq(1000, 5, ncp = 0)
hist(operaciones)

operaciones <- rchisq(1000, 10, ncp = 0)
hist(operaciones)
```



## Distribución t de Student

Simétrica respecto del origen, con colas más gruesas que la normal.

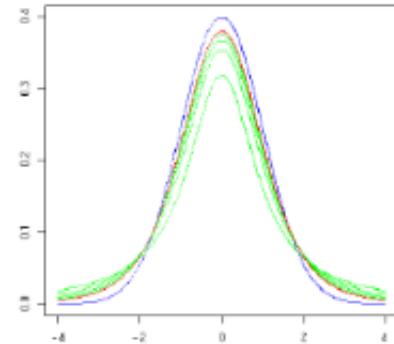
Se usa en finanzas para simular pérdidas (por la cola izq), usando grados de libertad bajos.

Cuando los grados de libertad son grandes se asemeja a una  $N(0,1)$ .

```
dt(x, df, ncp, log = FALSE)  
pt(q, df, ncp, lower.tail = TRUE, log.p = FALSE)  
qt(p, df, ncp, lower.tail = TRUE, log.p = FALSE)  
rt(n, df, ncp) # ncp es centralidad de la gráfica.
```

```
operaciones <- rt(1000, df=5, ncp=0)  
hist(operaciones)
```

```
operaciones <- rt(1000, df=1000, ncp=0) # Similar a una normal(0,1)  
hist(operaciones)
```



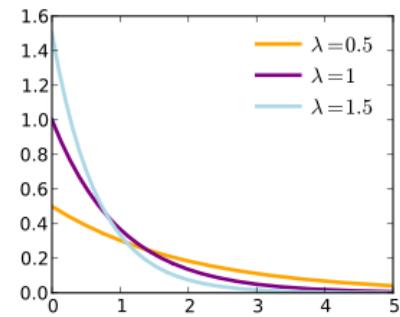
## Distribución exponencial

Se usa para calcular la probabilidad de default, siendo  $\lambda$  la intensidad de default.

```
dexp(x, rate = rate, log = FALSE)  
pexp(q, rate = rate, lower.tail = TRUE, log.p = FALSE) # Rate es lambda  
qexp(p, rate = rate, lower.tail = TRUE, log.p = FALSE)
```

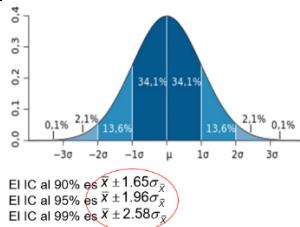
```
operaciones <- rexp(1000, rate=1)  
hist(operaciones)
```

```
operaciones <- rexp(1000, rate=5)  
hist(operaciones)
```



## Intervalos de confianza

Se usa para estimar si un resultado está dentro de un rango de valores en el que el valor del parámetro estará con una probabilidad de  $1-\alpha$  (nivel de confianza).



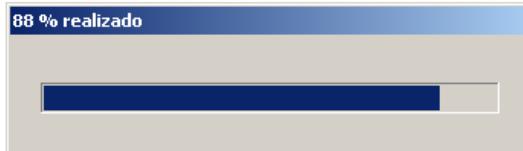
```
norm.interval = function(datos, varianza = var(datos), nivel.conf = 0.95)
{
  z = qnorm((1 - nivel.conf)/2, lower.tail = FALSE)
  m = mean(datos)
  dt = sqrt(varianza/length(datos))
  c(m - z * dt, m + z * dt)
}

X = rnorm(50,0,1) # Genero 50 nº aleatorios de media 0 y desviación típica 1.
norm.interval(X) # Quiero comprobar si el 0 está dentro del intervalo con un 95% de probabilidad.
# Da: -0.2566292 0.4148183 # Aceptamos la hipótesis nula porque el valor 0 está dentro del intervalo.
```

## Barra de progreso de un programa

En ocasiones el programa que realices tardará mucho en ejecutarse y te entrará la duda de si está "pensando" o si se ha bloqueado. Para evitar estas situaciones, lo mejor es utilizar la típica barra de progreso.

```
n_escenarios <- 10000
barra_progreso <- winProgressBar(title= "Barra de progreso", min = 0, max = n_escenarios, width=300) # width es el nº de pixeles de la barra.
for (escenario in 1:n_escenarios){
  setWinProgressBar(barra_progreso, escenario, title=paste(round(escenario/n_escenarios*100,0), "% realizado"))
}
close(barra_progreso)
```



# Depurar un programa

La forma más común para parar en una línea de código es **establecer un punto de interrupción** en esa línea. Puedes hacer esto en RStudio haciendo clic a la izquierda del número de línea en el editor (ojo, no funciona dentro de una función o de un bucle).

```
18 best <- 0
19 for (x in 100:999) {
20   for (y in x:999) {
21     candidate <- x * y
22     if (candidate > best && palindrome(candidate)) {
23       best <- candidate
24     }
25   }
26 }
```

**browser()** detiene la ejecución e invoca un navegador. Se puede poner en cualquier parte del código. Aquí, por ejemplo, se utiliza para detener cuando una función está a punto de devolver TRUE:

```
9   digit2 <- (num %% (10 ^ (place2+1))) %/ (10 ^ place2)
10  if (digit1 != digit2)
11    return(FALSE)
12  }
13 browser()
14 return(TRUE)
15 }
```

A diferencia del punto de interrupción, el navegador es parte del código, por lo que necesita ser ejecutado. El navegador es la herramienta de depuración de nivel más bajo y puede ser usado incluso en las situaciones en las que el punto de interrupción no funciona.

El navegador es útil para crear puntos de interrupción condicionales. Por ejemplo, si quieras iniciar la depuración después de cientos de iteraciones del ciclo puedes hacer:

```
for (i in 1:1024) {
  start_work()
  if (i == 512)
    browser()
  finish_work()
}
```

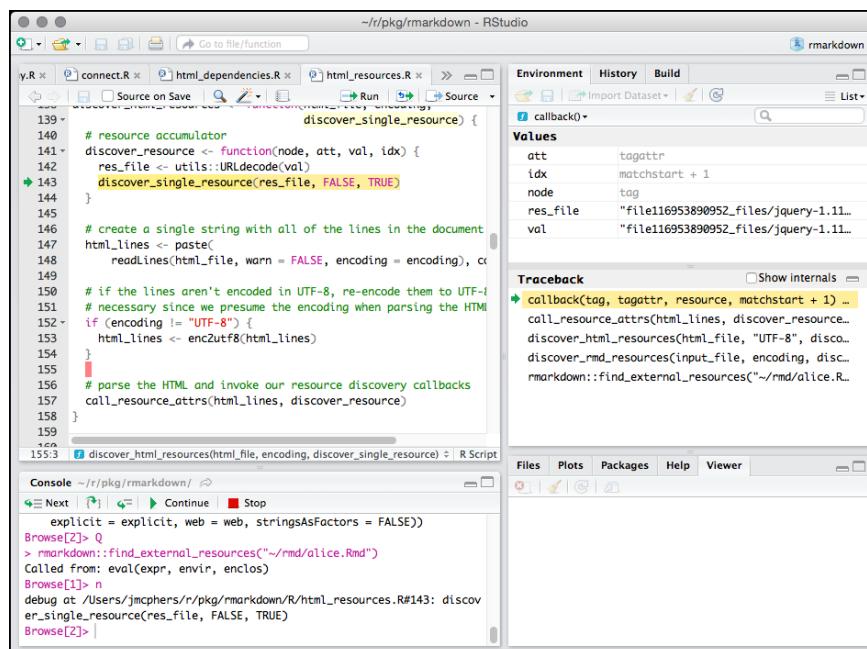
## Detenerse cuando se está ejecutando una función

Hay que establecer un punto de interrupción antes de la declaración de una función. No cambia la función en sí, sino que hace que el depurador se active inmediatamente cuando se ejecuta la función.

Usa la función `debugonce(nombre_funcion)` para establecer un punto de depuración dentro de una función (librería `devtools`).

`debugonce()` establece un punto de ruptura, es decir, la función entra en el depurador la próxima vez que se ejecute, pero no todas las veces que entre. Si lo que quieras es depurar una función cada vez que se ejecute, invoca `debug(nombre_funcion)` en la función y `undebug(nombre_funcion)` cuando ya no quieras seguir depurando.

## Usando debugger



Una vez se detiene el código, el IDE entra automáticamente en modo de depuración, en donde disponemos de varias herramientas para inspeccionar y alterar el estado del programa.

**Environment** (la mayoría de las veces no usarás esto)

R interactúa con el "entorno global" (objetos, valores, funciones y datos).

Los valores en gris son argumentos de la función que no han sido evaluados todavía.

**Traceback** (Callstack / pila de llamadas)

Muestra cómo se ha alcanzado el punto actual, desde la primera línea ejecutada hasta el punto actual.

```

shiny.R x html_dependencies.R x html_resources.R x
Source Save Run Source
139
140 # resource accumulator
141 discover_resource <- function(node, att, val, idx) {
142   res_file <- utils::URLdecode(val)
143   discover_single_resource(res_file, FALSE, TRUE)
144 }
145
146 # create a single string with all of the lines in the doc
147 html_lines <- paste0(
148   readLines(html_file, warn = FALSE, encoding = encoding)
149
150 # if the lines aren't encoded in UTF-8, re-encode them to
151 # necessary since we presume the encoding when parsing them
152 if (encoding != "UTF-8") {
153   html_lines <- enc2utf8(html_lines)
154 }
155
156 # parse the HTML and invoke our resource discovery callback
157 call_resource_attrs(html_lines, discover_resource)
158 }
159
discover_html_resources(html_file, encoding, discover_single_resource)
  
```

Console ~ /r/pkg/rmarkdown/

Browse[2]> Q  
> rmarkdown:::find\_external\_resources("~/rmd/alice.Rmd")  
Called from: eval(expr, envir, enclos)  
Browse[1]> n  
debug at /Users/jmcphers/r/pkg/rmarkdown/R/html\_resources.R#143: discover\_single\_resource(res\_file, FALSE, TRUE)  
Browse[2]>

Next | ⌂ | ⌄ | Continue | Stop

## Environment (la mayoría de las veces no usarás esto)

R interactúa con el "entorno global" (objetos, valores, funciones y datos).

Los valores en gris son argumentos de la función que no han sido evaluados todavía.

Encima de la lista de objetos locales, hay una lista desplegable que muestra la cadena de herencia para el entorno. Es una lista de lugares en donde se buscará el valor de las variables que utilice la función.

## Traceback (Callstack / pila de llamadas)

Muestra cómo se ha alcanzado el punto actual, desde la primera línea ejecutada hasta el punto actual.

Puedes hacer clic en cualquier función de la pila de llamadas para ver el contenido actual de su environment y el punto de ejecución. Ten en cuenta que la selección en la pila de llamadas no cambia el entorno activo en la consola.

## Ventana de código

La ventana de código muestra la función que se está ejecutando. La línea que se está a punto de ejecutar se resalta en amarillo.

## Console

Mientras se depura, te darás cuenta de dos cambios en la consola.

El primero es que el símbolo es diferente Browse[1]>

Pulsando Enter en la consola, esta ejecutará la instrucción actual y pasará a la siguiente.

La segunda es que hay una nueva barra de herramientas en la parte superior de la consola con los siguientes comandos:

n or Enter	F10	Ejecutar siguiente línea
s	Shift+F4	Entrar dentro de la función
f	Shift+F6	Terminar función / bucle
c	Shift+F5	Seguir ejecutando
Q	Shift+F8	Salir del depurador

Adjunto un link donde se explican estas y otras técnicas de depuración de errores con mayor profundidad.

<http://adv-r.had.co.nz/Exceptions-Debugging.html>

# Kaggle

<https://www.kaggle.com/>

Fundada en 2010, Kaggle es una plataforma de competición en análisis de modelos predictivos donde las empresas publican sus datos y los programadores compiten para generar el mejor modelo predictivo.

Este enfoque se basa en que existen infinidad de maneras de resolver un problema y es “imposible” conocer de antemano qué técnica será la más eficaz.

Por otro lado, Kaggle es un “centro de reclutamiento” de Data Scientist para empresas del calibre de Google, Facebook, Amazon...

## ¿Cómo funcionan las competiciones?

- La empresa prepara y publica un set de datos, indicando el objetivo deseado, el plazo y el premio en metálico.
- Los programadores compiten entre sí para producir los mejores modelos predictivos. El trabajo es compartido públicamente al resto de programadores. Las soluciones se califican inmediatamente en función de su precisión predictiva en relación con un archivo de soluciones oculto y se presenta un ranking de programadores.
- Transcurrido el plazo, la empresa paga el premio en metálico a cambio de una “licencia mundial, perpetua, irrevocable y libre de royalties para usar la solución ganadora.

## Active Competitions

		<b>Ultrasound Nerve Segmentation</b> Identify nerve structures in ultrasound images of the neck	<b>29 days</b> 573 teams 590 kernels <b>\$100,000</b>
		<b>State Farm Distracted Driver Detection</b> Can computer vision spot distracted drivers?	<b>12 days</b> 1355 teams 873 kernels <b>\$65,000</b>
		<b>Grupo Bimbo Inventory Demand</b> Maximize sales and minimize returns of bakery goods	<b>41 days</b> 1360 teams 1843 kernels <b>\$25,000</b>
		<b>TalkingData Mobile User Demographics</b> Get to know millions of mobile device users	<b>47 days</b> 581 teams 839 kernels <b>\$25,000</b>

Si no te interesa competir, esta página es muy interesante para descargarse Set de Datos con los que poner a prueba tus conocimientos y habilidades de programación.

Uno de los principales problemas a la hora de programar es la ausencia de Set de Datos gratuitos.

Estos son algunos de los Datasets que te puedes descargar (échales un vistazo).

# Datasets

[Publish a Dataset](#)



101

## Health Insurance Marketplace

Explore health and dental plans data in the US Health Insurance Marketplace

[U.S. Department of Health and Human Services](#) · updated 6 months ago

3,801 downloads  
74 kernels  
9 comments



98

## 2016 US Election

Explore data related to the 2016 US Election

[Ben Hamner](#) · updated 3 weeks ago

5,682 downloads  
293 kernels  
26 comments



93

## World Food Facts

Explore nutrition facts from foods around the world

[Open Food Facts](#) · updated 6 months ago

5,009 downloads  
143 kernels  
12 comments



87

## Twitter US Airline Sentiment

Analyze how travelers in February 2015 expressed their feelings on Twitter

[Crowdflower](#) · updated 7 months ago

4,254 downloads  
124 kernels  
14 comments



84

## World Development Indicators

Explore country development indicators from around the world

[World Bank](#) · updated 6 months ago

3,901 downloads  
161 kernels  
13 comments

Por último, y quizás lo más importante cuando estás empezando a programar, es que puedes consultar las soluciones de otros programadores a un problema determinado. El ranking de soluciones te facilita qué es lo que tienes que buscar.

La posibilidad de ver el código de otra persona, que programa mejor que tú, te permitirá avanzar mucho más rápido...

## 2016 US Election

Explore data related to the 2016 US Election

Ben Hamner · last updated 19 days ago

Overview   Kernels   Discussion   Download (17 MB)   New Notebook   New Script

69 kernels   Sort By: Hotness

All   Mine   R   All Output Types

Rank	User	Title	Last Run	Actions
59		Predictions in the Republican Primary	run 2 weeks ago by <a href="#">Alexandru Papiu</a>	   17 
14		'Clinton: Champion of the Primaries	run 1 month ago by <a href="#">Joshua Ellis</a>	  
5		Bernie Sanders County Results Map	run 2 weeks ago by <a href="#">Ben Hamner</a> (+16 / -1 / -8)	  2 
3		2016 Democratic Primary Predictions	run 2 months ago by <a href="#">MassimoMannino</a>	  
5		Cluster Analysis of Democrat Primary Vot	run 4 months ago by <a href="#">Saritha Ramkumar</a>	  

### Plot a bunch of different maps showing the locations of the events

```
# Set up plot
df_events_sample = df_events.sample(n=100000)
plt.figure(1, figsize=(12,6))

# Mercator of World
m1 = Basemap(projection='merc',
              llcrnrlat=-60,
              urcrnrlat=65,
              llcrnrlon=-180,
              urcrnrlon=180,
              lat_ts=0,
              resolution='c')

m1.fillcontinents(color="#191919",lake_color='#000000') # dark grey land, black lakes
m1.drawmapboundary(fill_color='#000000') # black background
m1.drawcountries(linewidth=0.1, color="w") # thin white line for country borders

# Plot the data
mxy = m1(df_events_sample["longitude"].tolist(), df_events_sample["latitude"].tolist())
m1.scatter(mxy[0], mxy[1], s=3, c="#1292db", lw=0, alpha=1, zorder=5)

plt.title("Global view of events")
plt.show()
```

