Características Básicas de R

Santiago Lozano

27 de enero de 2020

Calculadora

Como otros lenguajes similares, R puede facilmente reemplzar todas las funcionalidades de una calculadora y de manera más eficiente, una de sus mayores fortalezas es que R permite las operaciones entre arreglos (arrays)

```
5 + 7
```

```
## [1] 12
```

Calculadora

```
log(42/7.3)
```

```
## [1] 1.749795
```

Cada línea puede tener a lo sumo 8192 carateres, pero si se quiere tener una instrucción o una expresión en la pantalla visualmente más legible, se puede continuar en las líneas de abajo simplemente cambiando la línea

```
5 + 6 + 3 + 6 + 4 + 2 + 4 + 8 +
  3 + 2 + 7
```

```
## [1] 50
```

Calculadora

también se puede usar punto y coma (;) para separar varias instrucciones

```
2 + 3; 5 * 7; 3 - 7
```

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

```
## [1] 35
```

```
## [1] -4
```

Calculadora

y los grandes números pueden ser expuestos en notación científica

```
1.2e3 #1200
```

```
## [1] 1200
```

```
1.2e-2 #0.012
```

```
## [1] 0.012
```

a su vez, es posible relizar operaciones entre arreglos

```
c(1, 2, 3, 4, 5) * 2
```

```
## [1]  2  4  6  8 10
```

Calculadora

Los símbolos aritméticos corresponden a + (suma), - (resta), * (multiplicación), / (división), %% (módulo: Residuo de la división), %% (cociente de la división)

```
119 %/% 13
```

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

```
119 %% 13
```

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

Calculadora

```
9 %% 2
```

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

```
8 %% 2
```

```
## [1] 0
```

Calculadora

PEMDAS

- Paréntesis
- Exponenciales
- Multiplicación
- División
- Adición
- Sustracción

```
6/3+4*2
```

Calculadora

PEMDAS

- Paréntesis
- Exponenciales
- Multiplicación
- División
- Adición
- Sustracción

```
6/3+4*2
```

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

```
6*4-12/3-8
```

Calculadora

PEMDAS

- Paréntesis
- Exponenciales
- Multiplicación
- División
- Adición
- Sustracción

```
6/3+4*2
```

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

```
6*4-12/3-8
```

```
## [1] 12
```

Calculadora

```
4*4-3*3-16/4
```

Calculadora

```
4*4-3*3-16/4
```

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

```
20-(3*2^3-5)
```

Calculadora

```
4*4-3*3-16/4
```

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

```
20-(3*2^3-5)
```

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

```
(5+2)^2-9*3+2^3
```

Calculadora

```
4*4-3*3-16/4
```

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

```
20-(3*2^3-5)
```

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

```
(5+2)^2-9*3+2^3
```

```
## [1] 30
```

Calculadora

```
(2^4+(16-3*4))/((6+3^2)/(7-4))
```

Calculadora

```
(2^4+(16-3*4))/((6+3^2)/(7-4))
```

```
## [1] 4
```

Redondeo

Tenemos varios tipos de Redondeo

- **El entero más grande menor que**

```
floor(5.7)
```

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

- **El entero siguiente**

```
ceiling(5.7)
```

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

Redondeo

- **Lugar decimal**: Especificando bajo que lugar decimal se hace la aproximación

```
round(5.7,0)
```

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

Redondeo

```
round(5.4,0)
```

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

```
x2 <- pi * 100^(-1:3)
x2
```

```
## [1] 3.141593e-02 3.141593e+00 3.141593e+02
## [4] 3.141593e+04 3.141593e+06
```

```
round(x2, 3)
```

```
## [1]     0.031     3.142   314.159 31415.927 3141592.654
```

Redondeo

- **Dígitos de Significancia**

```
signif(12345678,4)
```

```
## [1] 12350000
```

```
signif(12345678,5)
```

```
## [1] 12346000
```

```
signif(12345678,3)
```

```
## [1] 12300000
```

Funciones Matemáticas

`log(x)`	log to base e of x
`exp(x)`	antilog of x (e^x)
`log(x,n)`	log to base n of x
`log10(x)`	log to base 10 of x
`sqrt(x)`	square root of x
`factorial(x)`	$x! = x \times (x-1) \times (x-2) \times \cdots \times 3 \times 2$
`choose(n,x)`	binomial coefficients $n!/(x!\,(n-x)!)$
`gamma(x)`	$\Gamma(x)$, for real x $(x–1)!$, for integer x
`lgamma(x)`	natural log of $\Gamma(x)$
`floor(x)`	greatest integer less than x
`ceiling(x)`	smallest integer greater than x
`trunc(x)`	closest integer to x between x and 0, e.g. trunc(1.5) = 1, trunc(–1.5) = –1; trunc is like floor for positive values and like ceiling for negative values
`round(x, digits=0)`	round the value of x to an integer
`signif(x, digits=6)`	give x to 6 digits in scientific notation
`runif(n)`	generates n random numbers between 0 and 1 from a uniform distribution
`cos(x)`	cosine of x in radians
`sin(x)`	sine of x in radians
`tan(x)`	tangent of x in radians
`acos(x), asin(x), atan(x)`	inverse trigonometric transformations of real or complex numbers
`acosh(x), asinh(x), atanh(x)`	inverse hyperbolic trigonometric transformations of real or complex numbers
`abs(x)`	the absolute value of x, ignoring the minus sign if there is one

Asignación de Variables

En los campos anteriores observábamos que los resultados se obtenían directamente en la ventana de comandos, sin embargo, R es un lenguaje de progrmación y a menudo la razón de usar un lenguaje de programación como opuesto a una calculadora es automatizar algún proceso o establecer repeticiones.

Por ejemplo, quisierámos usar el resultado de $5 + 7$ en un segundo cálculo y en vez de repetir aquella sentencia podemos crear una variable que la almacene mediante <- una flecha hacía la izquierda

```
x <- 5 + 7
```

para ver el contenido de la variable x, simplemente la escribimos

```
x
```

```
## [1] 12
```

Asignación de Variables

Es erróneo tipear

```
x < - 5 + 7
```

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

Ahora

```
y <- x - 3
```

```
y
```

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

Asignación de Variables

Algunas cuestiones importante a la hora de establecer variables

- Cuando a algún valor no le asignamos alguna variable este no se almacena, pero si le asignamos una variable, esta variable se almacena en la memoria RAM, así que es de vital importancia tener un manejo debido de esta y evitar tener variables que contengan el mismo valor
- no es lo mismo y que Y
- no se puede poder un nombre de variable con número primero "1x" está mal, "x1" está bien.
- los nombre de las variable no pueden contener espacios
- Evitar poner nombres engorrosos a las variables

Tipos de Datos en R: Números reales (double)

```
a <- 1
b <- 3.4
c <- pi
```

```
is.double(a)
```

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

Tipos de datos en R: Números reales (double)

```
is.double(b)
```

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

```
is.double(c)
```

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

```
typeof(b)
```

```
## [1] "double"
```

Tipos de datos en R: Número entero (integer)

Inicialmente los número son enteros y reales a la misma vez son establecidos como double, pero estos pueden ser transformados a integer, otra nota importante que puede ser tenida en cuenta es que los integer consumen menos memoria que los numeric

```
d <- as.integer(a)
```

```
is.integer(d)
```

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

```
typeof(d)
```

```
## [1] "integer"
```

Tipos de datos en R: Números complejos (complex)

Los números complejos pueden ser creados gracias a la letra i

```
1i
```

```
## [1] 0+1i
```

```
z <- 1 + 2i
```

```
typeof(z)
```

```
## [1] "complex"
```

Tipos de datos en R: Números complejos (complex)

```
is.complex(z)
```

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

```
Re(z)
```

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

```
Im(z)
```

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

Tipos de datos en R: Números complejos (complex)

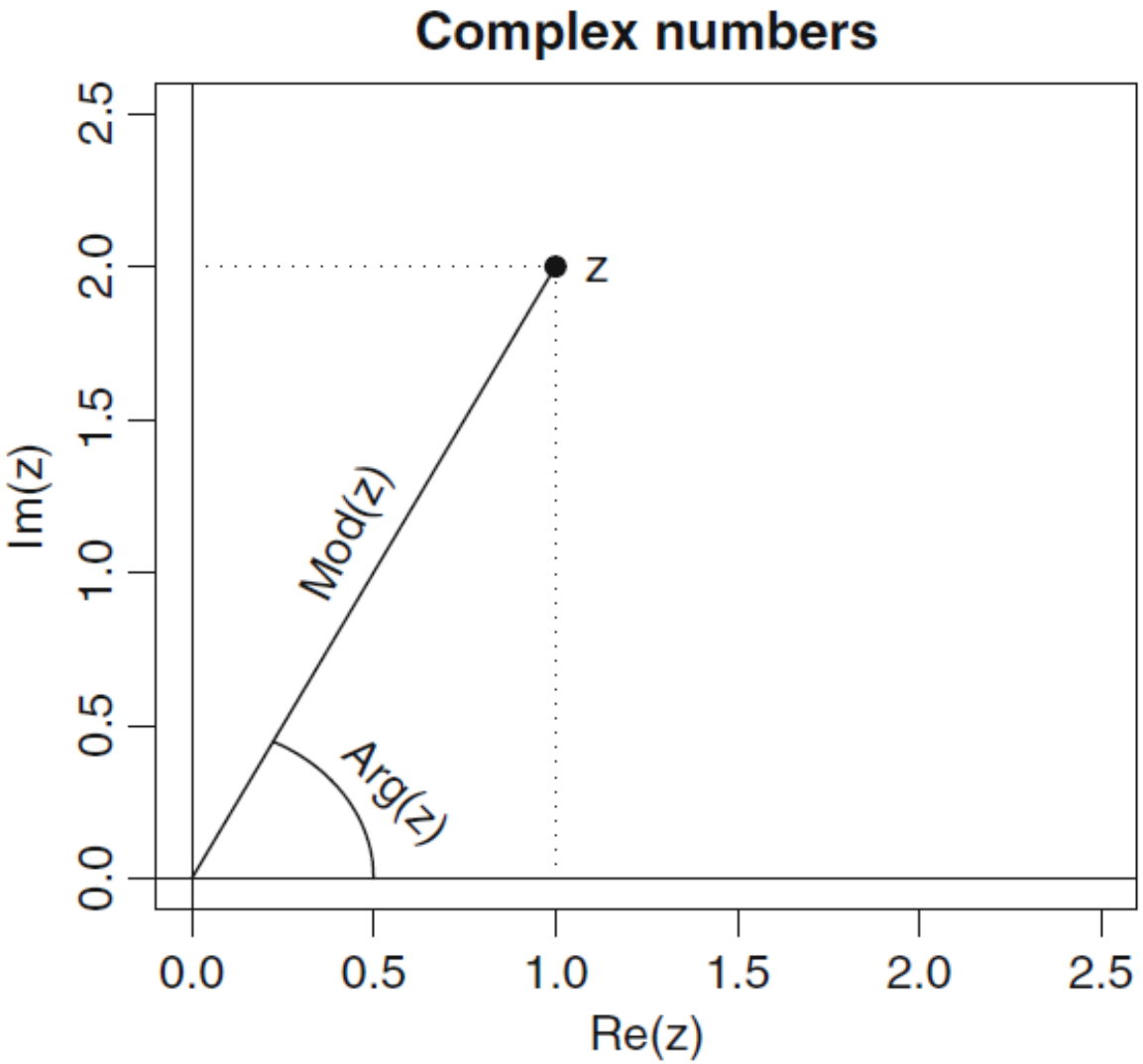
```
Mod(z)
```

```
## [1] 2.236068
```

```
Arg(z)
```

```
## [1] 1.107149
```

Tipos de datos en R: Números complejos (complex)



Booleano o de tipo Lógico (logical)

El tipo logical() es el resultado de una operación lógica. Este puede tomar los valores TRUE o FALSE, el cálculo de operaciones lógica corresponde a preguntas sobre cosas

```
TRUE == FALSE
```

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

```
T == F
```

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

Booleano o de tipo Lógico (logical)

```
T <- 0
T == FALSE
```

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

```
F <- 1
TRUE == F
```

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

```
T != F
```

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

Booleano o de tipo Lógico (logical)

Se debe tener cuidado con los números que contienen dígitos bastante grandes, pues pueden conducir a diferentes errores pues la mayoría de números se redondean sobre 53 dígitos binarios

```
x <- sqrt(2)
x * x == 2
```

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

```
x * x - 2
```

```
## [1] 4.440892e-16
```

Booleano o de tipo Lógico (logical)

El consejo es no usar ==, lo mejor es usar all.equal

```
x <- 0.3 - 0-2
y <- 0.1
x == y
```

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

la función identical tampoco funciona

```
identical(x,y)
```

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

Booleano o de tipo Lógico (logical)

Pero all.equal si funciona, pues permite diferencias insignificantes

```
all.equal(x,y)
```

```
## [1] "Mean relative difference: 1.058824"
```

otro ejemplos

```
b > a
```

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

Booleano o de tipo Lógico (logical)

Symbol	Meaning
`!`	logical NOT
`&`	logical AND
`|`	logical OR
`<`	less than
`<=`	less than or equal to
`>`	greater than
`=`	greater than or equal to
`==`	logical equals (double =)
`!=`	not equal
`&&`	AND with IF
`||`	OR with IF
`xor(x,y)`	exclusive OR
`isTRUE(x)`	an abbreviation of `identical(TRUE,x)`

Booleano o de tipo Lógico (logical)

```
TRUE & FALSE
```

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

```
FALSE | FALSE
```

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

```
!TRUE
```

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

Missing Data (NA)

Un valor faltante o valor indefinido es indicado por la instrucción NA (por no-disponible), varias funciones existen para manejar este tipo de dtos. De hecho, R lo considera como un valor lógico constante

```
x <- c(3,NA,6)
is.na(x)
```

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

```
mean(x)
```

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

Missing Data (NA)

```
mean(x,na.rm = TRUE)
```

```
## [1] 4.5
```

Los Missing Values son una fuente de irritación, porque ellos afectan las funciones que ajustan modelos y pueden reducir el poder del modelamiento

Infinitos y cosas que no son números

Los cálculos pueden conducir a respuestas como más infinito ó menos infinito

```
3/0
```

```
## [1] Inf
```

```
-12/0
```

```
## [1] -Inf
```

Infinitos y cosas que no son números

Podemos operar infinitos también

```
exp(-Inf)
```

```
## [1] 0
```

```
0/Inf
```

```
## [1] 0
```

Infinitos y cosas que no son números

Otros cálculos conducen a cantidades que no son números

```
0/0
```

```
## [1] NaN
```

```
Inf-Inf
```

```
## [1] NaN
```

```
Inf/Inf
```

```
## [1] NaN
```

Infinitos y cosas que no son números

Es necesario entender la diferencia entre NaN y NA. NaN significa "no es un número" y significa que es un resultado, pero no puede ser representado en el computador. NA explica que el data es faltante por razones desconocidas

NaN implica un resultado que no puede ser calculado por cualquier razón

Caracteres tipo String (character)

Este tipo de dato corresponde a las citas que queramos realizar

```
a <- "R es mi amigo"
a
```

```
## [1] "R es mi amigo"
```

```
typeof(a)
```

```
## [1] "character"
```

```
is.character(a)
```

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

Caracteres tipo String (character)

Podemos convertir números (numeric) a carater (character) y viceversa

```
as.character(2.3)
```

```
## [1] "2.3"
```

```
as.integer("3")
```

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

```
as.integer("3.cuatro")
```

```
## Warning: NAs introducidos por coerción
```

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

Caracteres tipo String (character)

También podemos generar frases de tipo string

```
cat("Hola","mundo","!")
```

```
## Hola mundo !
```

```
cat("Hello",2.7,"!")
```

```
## Hello 2.7 !
```

```
paste(1:12)
```

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

Caracteres tipo String (character)

```
paste("Hola","mundo","!",sep = " ")
```

```
## [1] "Hola mundo !"
```

```
paste("Hola",2,"!",collapse = " ")
```

```
## [1] "Hola 2 !"
```

Estructuras de Datos

En R se pueden organizar varios tipod de datos, mediante distintas estructuras que permiten realizar diferentes acciones sobre ellas

Estructuras de Datos: Vectores (vector)

Es la estructura de datos más simple. Representa un secuencia de datos del mismo tipo

```
0:10
```

```
##  [1]  0  1  2  3  4  5  6  7  8  9 10
```

```
15:5
```

```
##  [1] 15 14 13 12 11 10  9  8  7  6  5
```

```
c(55, 76, 92, 103, 84, 88, 121, 91, 65, 77, 99)
```

```
##  [1]  55  76  92 103  84  88 121  91  65  77  99
```

Estructuras de Datos: Vectores (vector)

```
seq(from=0.04,by=0.01,length=11)
```

```
##  [1] 0.04 0.05 0.06 0.07 0.08 0.09 0.10 0.11 0.12
##  [9] 0.13 0.14
```

```
seq(from=0.2, to=4, by=0.2)
```

```
##  [1] 0.2 0.4 0.6 0.8 1.0 1.2 1.4 1.6 1.8 2.0 2.2
## [12] 2.4 2.6 2.8 3.0 3.2 3.4 3.6 3.8 4.0
```

```
seq(0, 1.5, 0.1)
```

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0 1.1
## [12]1.2 1.3 1.4 1.5
```

Estructuras de Datos: Vectores (vector)

```
seq(6, 4, -0.2)
```

```
##  [1] 6.0 5.8 5.6 5.4 5.2 5.0 4.8 4.6 4.4 4.2 4.0
```

```
N <- c(55, 76, 92, 103, 84, 88, 121, 91, 65, 77, 99)
```

Si queremos usar el largo de N en otro vector usamos

```
seq(from=0.04, by=0.01, along=N)
```

```
##  [1] 0.04 0.05 0.06 0.07 0.08 0.09 0.10 0.11 0.12
## [10] 0.13 0.14
```

Estructuras de Datos: Vectores (vector)

```
seq(from=0.04, to=0.14, along=N)
```

```
##  [1] 0.04 0.05 0.06 0.07 0.08 0.09 0.10 0.11 0.12
## [10] 0.13 0.14
```

Observe que cuando el incremento no coincide con el valor final, entonces la secuencia generada para antes del último valor

```
seq(1.4, 2.1, 0.3)
```

```
## [1] 1.4 1.7 2.0
```

Estructuras de Datos: Vectores (vector)

Si queremos generar varias sequencias de vectores con diferente tamaños usamos

```
sequence(c(4,3,4,4,4,5))
```

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

para generar reprticiones de números o caracteres usamos

```
rep(9,5)
```

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

Estructuras de Datos: Vectores (vector)

```
rep(1:4, 2)
```

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

```
rep(1:4,each = 2)
```

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

```
rep(1:4, each = 2, times = 3)
```

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

Estructuras de Datos: Vectores (vector)

Para repetir números pero diferente número de veces usamos un vector que nos especifique el números de veces con respecto al ubicación del número

```
rep(1:4,1:4)
```

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

```
rep(1:4,c(4,1,4,2))
```

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

Estructuras de Datos: Vectores (vector)

usándolo para caracteres

```
rep(c("gato","perro","pez","ratón","conejo")
    ,c(2,3,2,1,3))
```

```
##  [1] "gato"   "gato"   "perro"  "perro"  "perro"  "pez"
##  [7] "pez"    "ratón"  "conejo" "conejo" "conejo"
```

Cuando usamos la función c() establecemos un vector de datos tipo double, sin embargo, si los datos son de tipo entero los podemos convertir a enteros (integer), con el objetivo de consumir menor memoria

```
x <- c(5,3,7,8)
```

estos vectores de datos tipo double se denominan numeric

```
class(x)
```

Estructuras de Datos: Vectores (vector)

```
is.numeric(x)
```

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

```
is.integer(x)
```

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

realizando coerción

```
x <- as.integer(x)
```

Estructuras de Datos: Vectores (vector)

```
is.integer(x)
```

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

Para ponerle nombre a los elementos de un vector procedemos

```
vec<-c(1,3,6,2,7,4,8,1,0)
names(vec)<- c("aa","bb","cc","dd","ee","ff","gg"
               ,"hh","ii")
vec
```

```
## aa bb cc dd ee ff gg hh ii
##  1  3  6  2  7  4  8  1  0
```

Matrices (matrix) y Arrays (array)

son representaciones de estructuras de datos con dos índices para matrices y múltiple para arrays, como los vectores, deben ser del mismo tipo.

```
X <- matrix(c(1,2,0,0,1,0,0,0,1),nrow = 3)
X
```

```
##      [,1] [,2] [,3]
## [1,]    1    0    0
## [2,]    2    1    0
## [3,]    0    0    1
```

```
class(X)
```

```
## [1] "matrix"
```

Matrices (matrix) y Arrays (array)

```
attributes(X)
```

```
## $dim
## [1] 3 3
```

```
vector <- c(1,2,3,4,4,3,2,1)
V <- matrix(vector,byrow = T,nrow = 2)
V
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    3    4    2
## [2,]    2    4    3    1
```

otra forma de convertir un vector a matrix es con

```
dim(vector) <- c(4,2)
```

Matrices (matrix) y Arrays (array)

```
is.matrix(vector)
```

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

```
vector
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    3
## [3,]    3    2
## [4,]    4    1
```

Matrices (matrix) y Arrays (array)

```
vector <- t(vector)
vector
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    4    3    2    1
```

```
Y <- matrix(1:12,nrow = 4,ncol = 3,byrow = TRUE)
Y
```

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

Matrices (matrix) y Arrays (array)

Para colocar nombres a nuestras filas y columnas en una matriz usamos

```
X <- matrix(rpois(20,1.5),nrow=4)
X
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    4    2    2    1    3
## [2,]    2    0    0    4    4
## [3,]    2    0    0    1    1
## [4,]    2    3    2    2    1
```

Matrices (matrix) y Arrays (array)

donde do.NULL = FALSE nos da los números y el prefix el prefijo

```
rownames(X) <- rownames(X,do.NULL=FALSE,prefix="Trial.")
X
```

```
##          [,1] [,2] [,3] [,4] [,5]
## Trial.1    4    2    2    1    3
## Trial.2    2    0    0    4    4
## Trial.3    2    0    0    1    1
## Trial.4    2    3    2    2    1
```

Matrices (matrix) y Arrays (array)

```
drug.names <- c("aspirin", "paracetamol", "nurofen",
                "hedex", "placebo")
colnames(X) <- drug.names
X
```

```
##         aspirin paracetamol nurofen hedex placebo
## Trial.1       4           2       2     1       3
## Trial.2       2           0       0     4       4
## Trial.3       2           0       0     1       1
## Trial.4       2           3       2     2       1
```

Matrices (matrix) y Arrays (array)

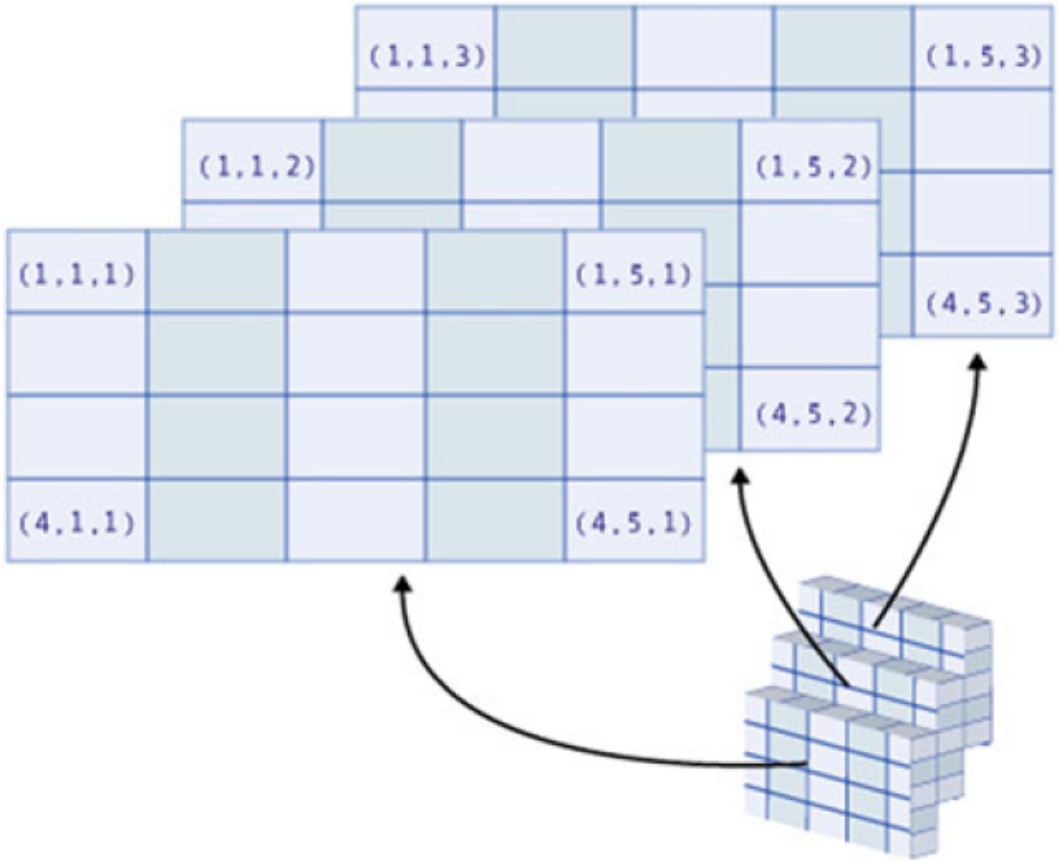
de forma alternativa

```
dimnames(X) <- list(NULL,paste("drug.",1:5,sep=""))
X
```

```
##      drug.1 drug.2 drug.3 drug.4 drug.5
## [1,]      4      2      2      1      3
## [2,]      2      0      0      4      4
## [3,]      2      0      0      1      1
## [4,]      2      3      2      2      1
```

Matrices (matrix) y Arrays (array)

Los arrays corresponden a matrices multidiensionales, veamos un ejemplo de 3 dimensiones



Matrices (matrix) y Arrays (array)

```
P <- array(as.integer(rnorm(60)),dim = c(4,5,3))
P
```

```
## , , 1
##
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    0    1    0    0
## [2,]    0   -2    0   -1    0
## [3,]    0    0    0    0    0
## [4,]    0    0    0    0    0
##
```

Matrices (matrix) y Arrays (array)

```
## , , 2
##
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    1    0
## [2,]    0    0    0    0    0
## [3,]    0    0    0    1    0
## [4,]    0   -1    1    0    0
##
```

Matrices (matrix) y Arrays (array)

```
## , , 3
##
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0   -1   -1    0
## [2,]   -1    0    0    1    0
## [3,]    0    0    0   -1    0
## [4,]   -1    0    0    1    0
```

Matrices (matrix) y Arrays (array)

```
class(P)
```

```
## [1] "array"
```

Listas (list)

Son estrucutras de datos extremadamente importantes en R debido a su riqueza y a la capacidad de agrupar en una estructura datos de diferente tipos sin alterarlos, es decir, cada elemento de una lista puede ser un dato, vector, matrix, array o incluso otra lista

Listas (list)

```
A <- list(TRUE,-1:3,matrix(1:4,nrow = 2), c(1+2i,3),
          "A es un character")
A
```

```
## [[1]]
## [1] TRUE
##
## [[2]]
## [1] -1  0  1  2  3
##
## [[3]]
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
```

Listas (list)

```
## [[4]]
## [1] 1+2i 3+0i
##
## [[5]]
## [1] "A es un character"
```

Listas (list)

```
class(A)
```

```
## [1] "list"
```

```
apples <- c(4,4.5,4.2,5.1,3.9)
orange <- c(TRUE, TRUE, FALSE)
chalk <- c("limestone", "marl", "oolite", "CaCO3")
cheese <- c(3.2-4.5i,12.8+2.2i)
```

Listas (list)

```
items <- list(apples,orange,chalk,cheese)
items
```

```
## [[1]]
## [1] 4.0 4.5 4.2 5.1 3.9
##
## [[2]]
## [1]  TRUE  TRUE FALSE
##
## [[3]]
## [1] "limestone" "marl"      "oolite"    "CaC03"
##
## [[4]]
## [1]  3.2-4.5i 12.8+2.2i
```

Listas (list)

Si quiero ponerle nombre a cada elemento de la lista

```
items <- list(first=apples,second=orange,third=chalk,
              fourth=cheese)
items
```

```
## $first
## [1] 4.0 4.5 4.2 5.1 3.9
##
## $second
## [1]  TRUE  TRUE FALSE
##
## $third
## [1] "limestone" "marl"      "oolite"    "CaCO3"
##
## $fourth
## [1]  3.2-4.5i 12.8+2.2i
```

Listas (list)

```
names(items)
```

```
## [1] "first"  "second" "third"  "fourth"
```

```
is.list(items)
```

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

Factores (factor) y variables ordinales (ordered)

Los factores son una estructura de variables categóricas que tienen un número fijo de niveles, un simple ejemplo de variables categóricas puede ser hobre o mujer

```
genero <- factor(c("mujer", "hombre", "mujer"
                   , "hombre", "mujer"))
genero
```

```
## [1] mujer  hombre mujer  hombre mujer
## Levels: hombre mujer
```

```
levels(genero)
```

```
## [1] "hombre" "mujer"
```

Factores (factor) y variables ordinales (ordered)

```
class(genero)
```

```
## [1] "factor"
```

podemos establecer factores con número que ellos establezcan una caraterística

```
Temp <- factor(c(0,1,0,3,1,0,1,0,3,3),
               labels = c("Low","medium", "High"))
Temp
```

```
##  [1] Low    medium Low    High   medium Low
##  [7] medium Low    High   High
## Levels: Low medium High
```

Factores (factor) y variables ordinales (ordered)

La función gl permite generar niveles y es útil para cuando se quieren codificar grandes vectores de niveles de factor, los argumentos dice n: hasta donde, k: cuantas veces se repite, labels: que niveles quiero

```
Imp <- gl(n = 2,k = 8,labels = c("Control", "Treat"))
Imp
```

```
##  [1] Control Control Control Control Control Control
##  [7] Control Control Treat   Treat   Treat   Treat
## [13] Treat   Treat   Treat   Treat
## Levels: Control Treat
```

Factores (factor) y variables ordinales (ordered)

```
gl(4,3,24)
```

```
##  [1] 1 1 1 2 2 2 3 3 3 4 4 4 1 1 1 2 2 2 3 3 3 4 4 4
## Levels: 1 2 3 4
```

Los elementos de tipo ordered tratan cuanto tenemos jerarquía en nuestras variables cualitativas

Factores (factor) y variables ordinales (ordered)

```
z <- ordered(c("Small","Tall","Average","Tall",
               "Average","Small","Small"),
             levels=c("Small","Average","Tall"))
z
```

```
## [1] Small   Tall   Average   Tal   Average
## [6] Small    Small
## Levels: Small < Average < Tall
```

```
class(z)
```

```
## [1] "ordered" "factor"
```

Unidades de tiempo (date)

Son las unidades que conciernen a horarios, para desplegar la hora actual

```
Sys.time()
```

```
## [1] "2020-02-14 01:06:14 -05"
```

```
date()
```

```
## [1] "Fri Feb 14 01:06:14 2020"
```

Unidades de tiempo (date)

Realizando un vector de fechas, tenemos que

```
fechas <- c("92/27/02", "92/02/27", "92/01/14",
            "92/02/28", "92/02/01")
fechas <- as.Date(fechas, "%y/%m/%d")
fechas
```

```
## [1] NA "1992-02-27" "1992-01-14" "1992-02-28"
## [5] "1992-02-01"
```

```
class(fechas)
```

```
## [1] "Date"
```

Series de tiempo (ts)

Los datos correspondientes a Series de tiempo se establecen con la función ts()

```
ti <- ts(1:10, frequency = 12, start = c(1959, 2))
ti
```

```
##      Feb Mar Apr May Jun Jul Aug Sep Oct Nov
## 1959   1   2   3   4   5   6   7   8   9  10
```

```
ts(1:10, frequency = 4, start = c(1959, 2))
```

```
##      Qtr1 Qtr2 Qtr3 Qtr4
## 1959         1    2    3
## 1960    4    5    6    7
## 1961    8    9   10
```

Series de tiempo (ts)

```
ti2 <- ts(1:10, frequency =1/12 , start = c(1959))
ti2
```

```
## Time Series:
## Start = 1959
## End = 2067
## Frequency = 0.0833333333333333
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
attributes(ti)
```

```
## $tsp
## [1] 1959.083 1959.833   12.000
##
## $class
## [1] "ts"
```

data frame (data.frame)

Es una tbla de individuo X variable que es esencial en el análisis estadísticos

```
datos <- data.frame(Genero=c("M","F","M","F"," ","F"),
                    Altura=c(1.83,1.76,1.82,1.60,1.90,
                             1.66),
                    Peso=c(67,58,66,48,75,55),
                    row.names=c("Jorge","Julia",
                                "Henry","Emilia",
                                "William","Elisa"))
datos
```

data frame (data.frame)

```
##         Genero Altura Peso
## Jorge        M   1.83   67
## Julia        F   1.76   58
## Henry        M   1.82   66
## Emilia       F   1.60   48
## William          1.90   75
## Elisa        F   1.66   55
```

data frame (data.frame)

```
is.data.frame(datos)
```

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

```
class(datos)
```

```
## [1] "data.frame"
```

```
str(datos)
```

```
## 'data.frame':    6 obs. of  3 variables:
##  $ Genero: Factor w/ 3 levels " ","F","M": 3 2 3 2 1 2
##  $ Altura: num  1.83 1.76 1.82 1.6 1.9 1.66
##  $ Peso  : num  67 58 66 48 75 55
```

data frame (data.frame)

```
attributes(datos)
```

```
## $names
## [1] "Genero" "Altura" "Peso"
##
## $class
## [1] "data.frame"
##
## $row.names
## [1] "Jorge"   "Julia"   "Henry"   "Emilia"  "William"
## [6] "Elisa"
```

así un data frame puede ser visto como una lista cuyos elemntos tienen igual longitud, realmente, esta es la forma como R los almacena

Tabla adicional

Type	Testing	Coercing
Array	`is.array`	`as.array`
Character	`is.character`	`as.character`
Complex	`is.complex`	`as.complex`
Dataframe	`is.data.frame`	`as.data.frame`
Double	`is.double`	`as.double`
Factor	`is.factor`	`as.factor`
List	`is.list`	`as.list`
Logical	`is.logical`	`as.logical`
Matrix	`is.matrix`	`as.matrix`
Numeric	`is.numeric`	`as.numeric`
Raw	`is.raw`	`as.raw`
Time series (ts)	`is.ts`	`as.ts`
Vector	`is.vector`	`as.vector`