Práctica 2. Introducción y manipulación de datos en R (1)

Jesús Martín Fernández

Contenidos

1.Tipos de objetos en R.

R es capaz de gestionar diversos tipos de **datos**, que se organizan en distintas estructuras de datos. A continuación se presenta una tabla que describe los tipos de datos básicos en R:

Tipo de dato	**Descripción**	**Ejemplo**
Numeric	Números decimales	`numero <- 1.0`
Integer	Números enteros	`int <- 1`
Character	Cadenas de texto	`str <- "texto"`
Complex	Números complejos	`comp <- 3+2i`
Logical	Valores de verdad: `TRUE` o `FALSE`, a menudo generados como resultado de operaciones lógicas.	`a <- 1; b <- 2; a < b`
Factor	Los factores son un tipo de variable, no de dato, las variables categóricas. Los vectores de caracteres suelen convertirse en factores para aprovechar las funciones que manejan datos categóricos, como en el análisis de regresión.	Usando `as.factor()`.

Las **estructuras de datos** son objetos que contienen datos y organizados de diferentes maneras. Las estructuras vienen definidas por su número de dimensiones y por la homogeneidad o no de sus datos.

La siguiente tabla muestra las principales estructuras de datos que maneja R.

Objeto	Modos	Descripción	¿Múltiples Modos ?	Dimensiones
Vector	Numérico, Carácter, Complejo, Lógico	Secuencia de elementos del mismo tipo (numérico, carácter, complejo, lógico).	No	1D
Array	Numérico, Carácter, Complejo, Lógico	Datos en varias dimensiones con elementos del mismo tipo (numérico, carácter, complejo, lógico).	No	Multi-dimensional (2D, 3D, …)
Matriz	Numérico, Carácter, Complejo, Lógico	Colección bidimensional de elementos del mismo tipo (numérico, carácter, complejo, lógico).	No	2D
Data Frame	Numérico, Carácter, Complejo, Lógico	Similar a una matriz pero puede contener diferentes tipos de datos (numéricos, caracteres, etc.) en sus columnas.	Sí	2D
TS	Numérico, Carácter, Complejo, Lógico	Serie temporal (time series), un conjunto de datos ordenados por tiempo, del mismo tipo (numérico, carácter, complejo, lógico).	No	1D (univariada) o 2D (multi-variada)
Lista	Numérico, Carácter, Complejo, Lógico, Función, Expresión, etc.	Contenedor flexible que puede almacenar elementos de diferentes tipos (numérico, carácter, complejo, lógico, función, expresión, etc.).	Sí	1D (pero puede contener estructuras de cualquier dimensión)

2. Operadores básicos en R

Vamos a hacer un resumen de los principales operadores y las funciones matemáticas más utilizadas en R

Operador	Descripción	Ejemplo
==	Igual a	x == 5
!=	Distinto de	x != 3
>	Mayor que	x > 3
<	Menor que	x < 5
>=	Mayor o igual que	x >= 5
<=	Menor o igual que	x <= 5
&	Y lógico (AND)	x>2 & x<3
|	O lógico (OR)	x<2 | x>3
!	Negación lógica (NOT)	!TRUE
all()	Devuelve TRUE si todos los elementos son TRUE	all(c(TRUE, TRUE))
any()	Devuelve TRUE si al menos un elemento es TRUE	any(c(FALSE, TRUE))

Ahora vamos a recordar las funciones matemáticas básicas más utilizadas

Función	Descripción	Ejemplo	Resultado
+	Suma dos números	5 + 3	8
-	Resta un número a otro	5 - 3	2
*	Multiplica dos números	5 * 3	15
/	Divide un número por otro	6 / 3	2
^	Exponenciación	2^3	8
%%	Resto de la división (módulo)	5 %% 2	1
%/%	División entera	5 %/% 2	2
round()	Redondea a un número especificado de decimales	round(3.14159, 2)	3.14
ceiling()	Redondea al entero superior más cercano	ceiling(2.3)	3
floor()	Redondea al entero inferior más cercano	floor(2.7)	2
trunc()	Trunca la parte decimal, dejando solo la parte entera	trunc(2.7)	2
sqrt()	Calcula la raíz cuadrada	sqrt(16)	4
exp()	Calcula el exponencial de un número (e^x)	exp(1)	2.718.282
log()	Calcula el logaritmo natural (base e)	log(10)	2.302.585
log10()	Calcula el logaritmo en base 10	log10(100)	2
log2()	Calcula el logaritmo en base 2	log2(8)	3
abs()	Calcula el valor absoluto	abs(-5)	5
sum()	Suma todos los elementos de un vector	sum(c(1, 2, 3))	6

Función	Descripción	Ejemplo	Resultado
prod()	Calcula el producto de todos los elementos de un vector	prod(c(1, 2, 3))	6
mean()	Calcula el promedio de un conjunto de números	mean(c(1, 2, 3))	2
min()	Encuentra el mínimo valor en un conjunto de números	min(c(1, 2, 3))	1
max()	Encuentra el máximo valor en un conjunto de números	max(c(1, 2, 3))	3

3. Vectores

Un **vector** es una secuencia ordenada de datos del mismo tipo (siendo éstos númericos, cadenas, fechas, etc). Es la estructura de datos más sencilla. Un simple valor (por ejemplo el número 5) constituye un vector numérico. Los escalares son vectores de longitud 1

```
x<- 5
x
```

```
[1] 5
```

Pero un vector puede contener otro tipo de datos como cadenas de texto o datos lógicos

```
x<- as.vector (TRUE)
x
```

```
[1] TRUE
```

```
y<-as.vector(c("Andrés", "Luis"))
y
```

```
[1] "Andrés" "Luis"
```

3.1 Creación de vectores

El vector es el elemento más básico en R, y contiene elementos de la misma clase.

Se crea con la función **c()**, que significa '**concatenar**' o '**combinar**' y el paréntesis debe incluir r los datos que deseamos integrar en ese vector, separados por comas (y en caso de datos de tipo `charater` flanqueados por comillas).

Vector formado por números enteros:

Con la función concatenar

```
x <- c(1,2,3,4,5,6,7,8,9,10)
x
```

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

Hay otras formas de generar una secuencia de números

Con expresiones específicas

```
x <- 1:10
# Crea un vector con elementos del 1 al 10
x
```

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

El operador : tiene prioridad

```
x <- 1:10-2 #Primero se genera el vector 1:10 (de 1 a 10) y luego se resta -2 a cada element
x
```

```
 [1] -1  0  1  2  3  4  5  6  7  8
```

Se pueden cambiar las prioridades usando paréntesis

```
x <- 1:(10-2) # Genera una serie de 1 a 8
x
```

```
[1] 1 2 3 4 5 6 7 8
```

Se puede crear secuencias de números enteros con `sequence`

```
x<-sequence(4:5)
#Genera una secuencia hasta 4 y otra hasta 5
x
```

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

La función `seq` es una función más general y flexible para generar secuencias numéricas. seq(from, to, by, length.out, along.with), Sus parámetros son from: El valor inicial de la secuencia. to: El valor final de la secuencia. by: El incremento (o decremento si es negativo) entre elementos de la secuencia. length.out: El número total de elementos en la secuencia. Si se especifica, el valor de by se ajusta automáticamente.

```
x<- seq(1, 2, by= 0.1)
#Genera una secuencia de números entre 1 y 2 con valores separados por 0,1
x
```

```
 [1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0
```

```
x<- seq(length=9, from=1, to=5)
#La función seq crea una secuencia de 9 elementos, con valor inicial 1 y final 5.
x
```

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

Otra forma de crear un vector con elementos repetidos es con `rep`

```
x <-rep(1, 10) #Crea un vector en el que aparece 10 veces el numero 1
x
```

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

Se pueden generar secuencias con distribuciones fijas, por ejemplo la normal con la función `rnorm`

```
x<-rnorm(10, 50, 5) # 10 valores de media 50 y desviación típica 5
x
```

```
 [1] 53.12499 46.94577 53.12924 59.06649 45.74317 48.38027 54.97049 49.12769
 [9] 50.43847 57.17353
```

Se puede crear un vector con elementos repetidos en forma de series con la función `gl` (generate levels)

```
x<-gl(3, 4) #gl (generate levels) genera un factor con 3 niveles (1, 2, 3),\
#repitiendo cada nivel 4 veces.
x
```

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

Vector formado por caracteres:

Cada uno de los elementos de un vector de caracteres debe escribirse entre comillas. En caso contrario, la máquina devuelve un mensaje de error.

```
ciudades<- c("Jaen", "Córdoba", "Sevilla")
ciudades
```

```
[1] "Jaen"    "Córdoba" "Sevilla"
```

Vector lógico:

Solo requiere definir los valores verdaderos (`TRUE`) y falsos (`FALSE`):

```
vector_logico <- c(TRUE, FALSE, TRUE, FALSE)
print(vector_logico)
```

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

Los vectores lógicos se pueden crear al poner una condición a otro vector:

```
# Crear un vector numérico
numeros <- c(1, -5, 7, -2, 8, 9, 2)

# Crear un vector lógico que indica si cada elemento es positivo o no
positivo <- numeros>0
print(positivo)
```

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

3.2 Operaciones con vectores

La primera operación que describiremos será comprobar si un objeto es un vector

```
v0 <- 10
v0
```

```
[1] 10
```

```r
is.vector(v0) # Verificar si el objeto o variable 'v0' es un vector.
```

```
[1] TRUE
```

Una vez creados los vectores, pueden ser modificados posteriormente. Por ejemplo, si deseamos agregar un elemento a un vector ya existente, lo añadimos asignándolo a nuestro vector original.

Vamos a operar con el vector `v1(2,5,10)`

```r
v1<-c(2,5,10)
v1
```

```
[1]  2  5 10
```

Se pueden añadir elementos a un vector

```r
v2<-c (v1,8)
v2
```

```
[1]  2  5 10  8
```

Ahora tenemos dos vectores `v1(2,5,10) , v2(2,5,10,8)`

Se pueden realizar operaciones aritméticas con vectores. Se pueden sumar `v1` y `v2`

```r
v1+v2
```

```
Warning in v1 + v2: longitud de objeto mayor no es múltiplo de la longitud de
uno menor
```

```
[1]  4 10 20 10
```

Aviso en v1 + v2 : longitud de objeto mayor no es múltiplo de la longitud de uno menor. Al último valor de v2, le suma el primero de V1 (“reciclaje”)
No ocurriría si v1 y v2 tuviesen igual tamaño.

También pueden multiplicarse `v1` y `v2`

```
v1*v2
```

```
Warning in v1 * v2: longitud de objeto mayor no es múltiplo de la longitud de
uno menor
```

```
[1]   4  25 100  16
```

Aviso en v1*v2 : longitud de objeto mayor no es múltiplo de la longitud de uno menor. Multiplica último valor de v2, por el primero de V1 ("reciclaje"). #No ocurriría si v1 y v2 tuviesen igual tamaño.

Se pueden elevar al cuadrado los términos de un vector

```
v1^2
```

```
[1]   4  25 100
```

Le podemos pedir a R que nos calcule la raiz cuadrada de los elementos de un vector con la función `sqrt`,

```
sqrt (v2) #v2(2,5,10,8)
```

```
[1] 1.414214 2.236068 3.162278 2.828427
```

Se pueden sumar los términos de un vector, con la función ´sum´

```
sum (v2) #v2(2,5,10,8)
```

```
[1] 25
```

Se puede calcular la media de los términos de un vector con la función ´mean´

```
mean (v2) #v2(2,5,10,8)
```

```
[1] 6.25
```

Atención, si un vector tiene valores ausentes `NA` hay que hacérselo notar a la función. Imaginemos `v3(0,10,8,NA,6)`

```r
v3<-c(0,10,8,NA,6)
mean (v3)
```

```
[1] NA
```

la forma correcta de calcular la media sería

```r
mean(v3, na.rm = TRUE)
```

```
[1] 6
```

Existen otras funciones que operan con los valores de un vector

```r
v3<-c(0,10,8,NA,6)
length(v3) #número de valores de v3
```

```
[1] 5
```

```r
max(v3,na.rm = TRUE) #Valor máximo de v3, sabiendo que hay valores NA
```

```
[1] 10
```

```r
min(v3, na.rm = TRUE) #Valor mínimo de v3, sabiendo que hay valores NA
```

```
[1] 0
```

Se puede operar con un solo vector

```r
v2*2+10
```

```
[1] 14 20 30 26
```

Cuando le indicamos a R que calcule `v2* 2 + 10`, lo que realmente está calculando es:`v2 * c(2, 2, 2, 2) + c(10, 10, 10, 10)`

Algunas operaciones se pueden escribir de forma simplificada

```
v2 + c (2, 10)
```

```
[1]  4 15 12 18
```

Esta operación suma 2 a los valores impares y 10 a los pares. Recuerde que `v2(2,5,10,8)`

‘

Los elementos de un vector se pueden recuperar, por medio de corchetes. Siendo `v1(2,5,10)` `v2(2,5,10,8)`

```
v1[2]
```

```
[1] 5
```

```
v2[3]
```

```
[1] 10
```

Se puede operar con valores de determinada posición de los vectores, vamos a sumar el valor de la segunda posición de `v1` con el de la tercera posición de `v2`

```
v1[2] +v2[3]
```

```
[1] 15
```

También se pueden añadir los corchetes para añadir un valor en una posición o para cambiarlo. Veamos `v1(2,5,10)`

```
v1[4] <- 12
v1
```

```
[1]  2  5 10 12
```

```
v1[2] <- 12
v1
```

```
[1]  2 12 10 12
```

Los elementos de un vector se pueden recuperar en función de una orden lógica. Siendo `v1(2,12,10,12)`

```r
v1 > 7
```

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

```r
v1 < 7
```

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

```r
v1 == 12
```

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

También puede usarse la función `which`. Tenemos `v3(0,10,8,NA,6)`

```r
which(v3 > 2) # Posiciones de los elementos mayores que 2
```

```
[1] 2 3 5
```

```r
which(v3 < 2 | v3 >= 8) # Posiciones de los elementos <2 o >= 8.
```

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

Atención, hay un elemento NA, aunque R en esta versión lo identifica como tal, es más recomendable

```r
which(!is.na(v3) & v3 < 2 | v3 >= 8)
```

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

Posiciones de los elementos >2 y <= 8 sabiendo que existen valores ausentes.

4. Matrices

En R, una matriz es una colección bidimensional de elementos que deben ser del mismo tipo (numérico, carácter, lógico, etc.).

4.1 Creación de matrices

La función `matrix()` es la forma más común de crear matrices en R.

```r
matrix(1:12, nrow = 4, ncol = 3)
```

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

Cuando no se especifica nada la matriz se cumplimenta por columnas.

Pero se puede pedir que se cumplimente por filas

```r
matrix(1:12, nrow = 4, ncol = 3, byrow = TRUE)
```

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

La función `matrix()` se puede sustituir con la instrucción `dim`, que transforma un vector en una matriz

```r
x<-1:15
dim(x) <- c(5, 3)
x
```

```
     [,1] [,2] [,3]
[1,]    1    6   11
[2,]    2    7   12
[3,]    3    8   13
[4,]    4    9   14
[5,]    5   10   15
```

4.2 Operaciones con matrices

Las funciones básicas explicadas se pueden emplear en una matriz

```r
range(x) #Rango de la matriz x
```

```
[1]  1 15
```

```r
length(x) #Número de elementos en la matriz x
```

```
[1] 15
```

```r
mean(x) #Media de los valores de la matriz x
```

```
[1] 8
```

```r
median(x) #Mediana de los valores de la matriz x
```

```
[1] 8
```

```r
sum(x) #Suma de los valores de la matriz x
```

```
[1] 120
```

```r
prod(x) #Producto de los valores de la matriz x
```

```
[1] 1.307674e+12
```

```r
max(x) #Valor máximo de la matriz x
```

```
[1] 15
```

```r
min(x) #Valor mínimo de la matriz x
```

```
[1] 1
```

```r
which.max(x) # Devuelve el valor máximo de la matriz x
```

```
[1] 15
```

```
which.min(x) #Devuelve valor mínimo de la matriz x
```

```
[1] 1
```

Las matrices pueden unirse entre sí

```
m1<-matrix(1:4, nrow=2, ncol=2)
m2<-matrix(5:8, nrow=2, ncol=2)
m1
```

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

```
m2
```

```
     [,1] [,2]
[1,]    5    7
[2,]    6    8
```

Se pueden unir por filas con la orden `rbind`

```
rbind (m1,m2)
```

```
     [,1] [,2]
[1,]    1    3
[2,]    2    4
[3,]    5    7
[4,]    6    8
```

Y se puden unir por columnas con la orden `cbind`

```
cbind (m1,m2)
```

```
     [,1] [,2] [,3] [,4]
[1,]    1    3    5    7
[2,]    2    4    6    8
```

´Las matrices también se pueden sumar o multiplicar

```
m1+m2
```

```
     [,1] [,2]
[1,]    6   10
[2,]    8   12
```

```
m1*m2
```

```
     [,1] [,2]
[1,]    5   21
[2,]   12   32
```

Las matrices se pueden transponer con la orden `t`

```
t (m1)
```

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

```
t (m2)
```

```
     [,1] [,2]
[1,]    5    6
[2,]    7    8
```

Las filas y las columnas de las matrices pueden etiquetarse

```
colnames(m1) <- c("Columna 1", "Columna 2")
rownames (m1) <- c("Fila 1", "Fila 2")
m1
```

```
       Columna 1 Columna 2
Fila 1         1         3
Fila 2         2         4
```

Las posiciones de los valores de una matriz pueden recuperarse, supongamos una nueva matriz `m3`

```r
m3<- rbind(m1,m2)
rownames (m3) <- c("Fila 1", "Fila 2", "Fila 3", "Fila 4")
m3
```

```
       Columna 1 Columna 2
Fila 1         1         3
Fila 2         2         4
Fila 3         5         7
Fila 4         6         8
```

Vamos a intentar localizar algunas posiciones en la matriz

```r
m3[1,] # fila 1 todas las columnas.
```

```
Columna 1 Columna 2
        1         3
```

```r
m3[,2] # todas filas, columna 2
```

```
Fila 1 Fila 2 Fila 3 Fila 4
     3      4      7      8
```

```r
m3[1:2,1] #filas 1 y 2,  columna 1
```

```
Fila 1 Fila 2
     1      2
```

```r
m3[-(1:2),2] #excluir filas 1 a 2, incluir columna 2.
```

```
Fila 3 Fila 4
     7      8
```

```r
m3[,"Columna 2"] #La columna 2, usando el nombre de la columna
```

```
Fila 1 Fila 2 Fila 3 Fila 4
     3      4      7      8
```

```r
m3[3,c("Columna 1","Columna 2")] # La fila 3 y todas las columnas, llamadas por su nombre
```

```
Columna 1 Columna 2
        5         7
```