

Resumen de comandos de R

Contents

Constantes y variables	2
Ayuda	2
Funciones	2
Paquetes	3
Operadores	3
Aritméticos	3
Relacionales	4
Lógicos	4
Tipos de datos	5
NA y NULL	6
Condicionales	6
if	6
ifelse	6
for	7
while	7
Break y next	7
Repeat	7
Estructuras de datos	7
Vector	8
Propiedades	8
Operaciones	9
Extracción de valores	9
Sucesiones	9
Repetición	10

Función all y any	10
Matriz	10
Propiedades	11
Listas	13
Dataframe	13

Bibliografía 13

Este notebook ha sido creado en RStudio, un entorno que incluye una consola para insertar código, historial, gráficos, paquetes y librerías.

Constantes y variables

En R, se usa `<-` para asignar valores a una variable. Los nombres de las variables pueden incluir letras, números, puntos y guiones bajos, sin embargo, siempre deben empezar con una letra.

```
# Asignándole 1 a la variable a
a<-1
a
```

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

Ayuda

Es posible obtener la documentación de una función digitando un signo de interrogación (?) al inicio de esta o escribiendo `help(" ")` con el nombre de la función dentro de las comillas. Al hacerlo, en la pestaña de Help se mostrará cómo utilizar dicha función, sus parámetros y ejemplos.

```
?sum()
# o help("sum")
```

La documentación de un paquete se puede obtener con:

```
help(package = "datasets")
```

Funciones

En R, una función posee la siguiente sintaxis: **nombre_funcion()**. Dentro de los paréntesis, van los argumentos de la función. Algunas funciones básicas definidas de R son:

- `sum()`: suma.
- `mean()`: promedio.
- `max()`: máximo.
- `min()`: mínimo.
- `sqrt()`: raíz cuadrada.
- `sort()`: ordena los valores de menor a mayor.

- `unique()`: muestra valores únicos.

Las funciones también pueden ser definidas por el usuario usando la siguiente sintaxis:

```
nombre_función <- function(argumentos) {
  # código
}
```

Paquetes

En R, un paquete es una colección con funciones que no están en R base. CRAN es el repositorio de paquetes oficial de R, los cuales se pueden instalar mediante `install.packages()`. Por ejemplo:

```
install.packages("stats")
```

```
## Warning: package 'stats' is in use and will not be installed
```

Luego de instalar el paquete, las funciones de este se podrán utilizar después de ejecutar `library()` con el nombre del paquete dentro de la función.

```
library(stats)
```

Cada vez que se inicia una nueva sesión y se requiera usar una función que pertenezca a un paquete, se debe ejecutar `library()`.

Para saber qué paquetes están instalados, se debe ejecutar `installed.packages()`, sin argumento.

Operadores

Aritméticos

Los operadores aritméticos de R base que se pueden utilizar con datos enteros y numéricos son:

- `+`: suma.
- `-`: resta.
- `*`: multiplicación.
- `/`: división.
- `^`: exponencial.
- `%%`: módulo, devuelve el residuo.
- `%*%`: multiplicación entre matrices.

Ejemplo:

```
3*5+2
```

```
## [1] 17
```

El orden de las operaciones es el siguiente:

1. ^.
2. *.
3. /.
4. +.
5. -.
6. <, >, <=, >=, ==, !=.
7. !.
8. &.
9. |.
10. <-.

Relacionales

Los operadores relacionales se utilizan para comparar un valor con otro. Siempre devuelven TRUE o FALSE.

- > : mayor que.
- >= : mayor o igual que.
- < : menor que.
- <= : menor o igual que.
- == : igual.
- != : distinto.

Ejemplo:

```
234 > 243
```

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

Lógicos

Los operadores lógicos se utilizan para crear condiciones. Devuelven TRUE o FALSE.

- & : y.
- | : o.
- ! : not, negación lógica.

Ejemplo:

```
# Al usar &, si uno de los valores es FALSE, devuelve FALSE
# Como 234 es menor que 243, devuelve FALSE
234 & 2000 > 243
```

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

```
# Al usar |, si uno de los valores es TRUE, devuelve TRUE
# Como 2000 es mayor que 243, devuelve TRUE
234 | 2000 > 243
```

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

Tipos de datos

Los tipos de datos más comunes en R son:

1. integer: entero. Ejemplo: 1.
2. double: decimales. Ejemplo: 1.7
3. numeric: real. Ejemplo 4.5.
4. character: cadena de texto. Ejemplo: "Hola mundo"
5. factor: se utiliza para representar variables categóricas. Ejemplo: Categoría de productos como A, B, C.
6. logical: lógico, booleano. Ejemplo: FALSE.

Para que devuelva el tipo de dato de una variable, se usa `typeof`:

```
typeof(1.5)
```

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

Se pueden convertir un tipo de dato en otro por medio de funciones que comienzan con `as..`

1. `as.integer()`: convertir a entero.
2. `as.double()`: convertir a decimal.
3. `as.character()`: convertir a cadena de texto.
4. `as.logical()`: convertir a booleano.

Ejemplo:

```
# Convirtiendo la variable b en una cadena de texto  
b<-456 # integer  
b<-as.character(b)  
typeof(b)
```

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

```
# Como b es un texto, devuelve el resultado entre comillas  
b
```

```
## [1] "456"
```

Si al aplicar el `as.` no se puede convertir el dato al tipo de dato deseado, retornará un NA.

```
c <- "abuela"  
c <- as.integer(c)
```

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

Por otro lado, se puede determinar el tipo de dato de un dato con `is..`

1. `is.integer()`: verifica si es entero.

2. `is.double()`: verifica si es decimal.
3. `is.character()`: verifica si es cadena de texto.
4. `is.logical()`: verifica si es booleano.

Ejemplo:

```
# Verificando si 432 es una cadena de texto.  
is.character(432)
```

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

NA y NULL

La diferencia entre NA y NULL es que NULL representa datos inexistentes y NA datos faltantes o no disponibles.

Es posible eliminar estos valores usando `drop_na()` de la librería tidy. Se puede usar en una sola columna o en todas las columnas del marco de datos:

```
drop_na(df, column = "nombre columna")
```

Si se especifica `column = NULL`, se eliminarán todas las filas que contengan un valor NA en el marco de datos.

Condicionales

if

```
if (condicion){  
  # operacion si la condicion es TRUE  
} else {  
  # operacion si la condicion es FALSE  
}
```

ifelse

```
ifelse(vector, valor_si_es_TRUE, valor_si_es_FALSE)
```

Ejemplo:

```
# El siguiente codigo imprimira en pantalla Par si el modulo entre el numero y 2 es cero, o impar si no  
n <- 1:6  
ifelse(n%%2==0,"Par", "Impar")
```

```
## [1] "Impar" "Par" "Impar" "Par" "Impar" "Par"
```

for

```
for (elemento in objeto){  
  # operacion con el elemento  
}
```

Ejemplo:

```
dado<-1:6  
for (cara in dado){  
  d<-dado^2  
}  
d
```

```
## [1] 1 4 9 16 25 36
```

while

```
while (condicion){  
  # operaciones  
}
```

Break y next

El comando **break** interrumpe un bucle, mientras que **next** avanza a la siguiente iteración del bucle saltándose la actual. Estos comandos se ocupan para **for** y **while**.

Repeat

repeat es un bucle que se repetirá un número específico de veces. Se usa **break** para detenerlo. Ejemplo:

```
valor<-0  
repeat{  
  valor<-valor+1  
  if (valor ==5){  
    break  
  }  
}  
# imprime el resultado  
valor
```

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

Estructuras de datos

Una estructura de datos es una especie de contenedor que guarda datos en una posición específica. En R, las estructuras de datos más comunes son:

1. Vector

2. Matriz
3. Factores
4. Listas
5. DataFrame

Vector

Es la estructura de datos más básica en R. Almacena datos del mismo tipo y su única dimensión es el largo. Para crear un vector, se usa la función `c()` (concatenación). Ejemplo:

```
# Vector numerico
v1<-c(1,2,3,4,5,6,7)
v1
```

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

```
# Vector numerico del 2 al 20
v2<-c(2:20)
v2
```

```
## [1] 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
```

Propiedades

```
# Vector
x<-c(3,4,5,4,3,2)
x
```

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

```
# Verifica si un vector es vector. Si es vector, devuelve TRUE
is.vector(x)
```

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

```
# Imprime el largo del vector
length(x)
```

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

```
# Imprime el tipo de vector
typeof(x)
```

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

```
# Nombra los elementos de un vector
names(x)<-c("a","b","c","d","e","f")
x
```

```
## a b c d e f
## 3 4 5 4 3 2
```


Operaciones

Se pueden realizar operaciones aritméticas con los vectores e incluso operaciones relacionales, devolviendo un TRUE o un FALSE en este caso.

```
v3<-c(2,4,6,8)
v4<-c(1,3,5,7)
```

```
# 1.
v3*2
```

```
## [1] 4 8 12 16
```

```
v3+2
```

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

```
# 2.
v4==3
```

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

Extracción de valores

```
v3
```

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

```
# Imprime el valor de la posición 4 del vector v3
v3[c(4)]
```

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

```
# Entrega un rango de valores del vector v3
v3[1:3]
```

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

```
# Elimina el valor de la posición 1 (elimina el 2)
v3[-1]
```

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

```
# Elimina un rango de valores (elimina el 2,3 y 6)
v3[-1:-3]
```

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

Sucesiones

Para crear una sucesión en R se utiliza `seq(inicio, fin, by=numero)`:

```
# Sucesion que inicia en 1 y va de 2 en 2 hasta 20  
seq(1,20,by=2)
```

```
## [1] 1 3 5 7 9 11 13 15 17 19
```

```
# Sucesion que inicia en 3 y termina en 16  
seq(3:16)
```

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

Repetición

En R, se puede repetir un número *n* veces usando `rep(numero, num_de_repeticiones)`:

```
# Repite el 12 cinco veces  
rep(12, 5)
```

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

Función all y any

Las funciones `all` y `any` se utilizan para hacer comprobaciones lógicas en vectores.

- `all()`: si todos los elementos de un vector son TRUE, devolverá TRUE. De lo contrario, si hay uno que sea FALSE, devolverá FALSE. Ejemplo:

```
# Comprueba si todos los elementos son mayores a 5  
y<-1:20  
all(y>5)
```

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

- `any()`: si al menos uno de los elementos es TRUE, devolverá TRUE. Si todos son FALSE, devolverá FALSE. Ejemplo:

```
# Comprueba si hay al menos un elemento menor a 5  
x<-1:10  
any(x<5)
```

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

Matriz

Una matriz es una estructura de datos que posee filas y columnas. Contiene elementos del mismo tipo.

Para crear una matriz, se utiliza la función `matrix()`:

```
matrix(rango, nrow= n_filas, ncol=n_col)
```

Por ejemplo:

```
# Creando una matriz de datos NA
z<-matrix(nrow=4,ncol=3)
z
```

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

```
z1<-matrix(1:6, nrow=2, ncol=3)
z1
```

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

Se pueden crear matrices con `rbind` y `cbind`:

```
# Vectores
vec1 <- 1:4
vec2 <- 5:8
# cbind usa cada vector como una columna
vec3 <- cbind(vec1, vec2)
vec3
```

```
##      vec1 vec2
## [1,]    1    5
## [2,]    2    6
## [3,]    3    7
## [4,]    4    8
```

```
# rbind usa cada vector como una fila
vec3 <- rbind(vec1, vec2)
vec3
```

```
##      [,1] [,2] [,3] [,4]
## vec1    1    2    3    4
## vec2    5    6    7    8
```

Propiedades

Usando las matrices anteriores, es posible cambiar el nombre de las filas y las columnas mediante los siguientes comandos.

Para ponerle nombre a las filas, se usa `rownames()`:

```
rownames(z1) <- c("Fila_1", "Fila_2")
z1
```

```
##      [,1] [,2] [,3]
## Fila_1    1    3    5
## Fila_2    2    4    6
```

Para nombrar las columnas se usa `colnames()`:

```
colnames(z1) <- c("Col_1", "Col_2", "Col_3")
z1
```

```
##      Col_1 Col_2 Col_3
## Fila_1    1    3    5
## Fila_2    2    4    6
```

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

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

```
m2 <- matrix(12:23, nrow=4, ncol=3)
m2
```

```
##      [,1] [,2] [,3]
## [1,]   12   16   20
## [2,]   13   17   21
## [3,]   14   18   22
## [4,]   15   19   23
```

```
# Devuelve la dimension de una matriz
dim(m)
```

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

```
# Transpone una matriz
t(m)
```

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

```
# Multiplicacion entre matrices
m %*% m2
```

```
##      [,1] [,2] [,3]
## [1,]  312  400  488
## [2,]  366  470  574
## [3,]  420  540  660
```

Siendo A una matriz y B otra matriz:

```
# Producto cruzado
crossprod(a,b) # t(A) %*% B
tcrossprod(A,B) # A %*% t(B)

#Producto de Kronecker
A %x% B

# Potencia
A %^% 2

# Potencia elemento a elemento
A^2

# Determinante
det(A)

# Diagonal
diag(A)

# Diagonal secundaria
diag(apply(A,2,rev))

# Matriz diagonal
diag(c(1,2,3,4,5))

# Matriz identidad
diag(3)

# Autovalores
# Se almacenan en el elemento values
eigen(A)$values

# Autovectores
# Se almacenan en el elemento vectors
eigen(A)$vectors
```

Listas

Dataframe

Bibliografía

1. <https://r-coder.com/inicio/>
2. <https://bookdown.org/jboscomendoza/r-principiantes4/>