Lo que debemos saber de R

Explorando las capacidades de R y su uso en el entorno Linux

Edison Achalma

2023-06-11

Lo que debemos saber

Tipos de datos

En R, es fundamental comprender los diferentes tipos de datos disponibles. A continuación, exploraremos los tres tipos básicos de datos en R y cómo se utilizan en la programación.

1. Tipos de datos numéricos

Los datos numéricos en R se dividen en dos tipos principales:

a. Números reales, se conoce como `double`. Estos son los números más comunes y se utilizan para representar valores decimales. Por ejemplo, 3.14 y 2.71828 son números reales en R. La precisión de los números reales en R depende de la máquina en la que se ejecuta el programa.

b. Números enteros, se conoce como `integer`. Estos son números que no contienen decimales y se utilizan para representar valores enteros. Por ejemplo, 1, 2, -5 son ejemplos de números enteros en R. Los números enteros se utilizan cuando no se requiere precisión decimal.

2. Tipo de datos lógico

El tipo de dato lógico en R se conoce como `booleano`. Este tipo de dato puede tener uno de dos valores: TRUE o FALSE. Los valores booleanos se utilizan principalmente para realizar operaciones de comparación y evaluación lógica en los programas. Por ejemplo, se puede usar una expresión lógica para verificar si una condición es verdadera o falsa.

3. Tipo de datos carácter

El tipo de dato carácter en R se utiliza para almacenar letras `text` y símbolos `strings`. Los datos de tipo carácter se definen utilizando comillas simples (' ') o comillas dobles (" "). Por ejemplo,"Hola" y 'RStudio' son ejemplos de datos de tipo carácter en R. Los datos de tipo carácter se utilizan con frecuencia para almacenar texto legible por humanos, como nombres, descripciones o mensajes.

> Es importante comprender estos tipos de datos en R, ya que nos permiten manipular y realizar operaciones en los datos de manera adecuada. Cada tipo de dato tiene sus propias características y funciones asociadas que nos permiten realizar tareas específicas en la programación.

Estructura de datos

Las estructuras de datos nos permiten organizar y manipular la información de manera eficiente. A continuación, exploraremos las principales estructuras de datos disponibles en R y cómo se utilizan en la programación.

1. Escalar

Un escalar es un dato individual, como un número o una palabra, que no está agrupado con otros elementos. En R, los escalares pueden ser de diferentes tipos de datos, como numéricos, lógicos o caracteres. Estos datos se utilizan cuando solo necesitamos almacenar una única observación.

2. Vector

Un vector es una colección ordenada de elementos del mismo tipo de dato. Puede contener números, valores lógicos o caracteres. En R, los vectores son utilizados para almacenar conjuntos de datos relacionados. Por ejemplo, podemos tener un vector de edades o un vector de nombres. Los vectores son una de las estructuras de datos más utilizadas en R y nos permiten realizar operaciones y cálculos de manera eficiente.

Vectores

Concatenación de elementos con `c()`: Se utiliza la función `c()` para concatenar elementos y crear vectores en R.

```
c(0.5, 0.6, 0.25) # números decimales (double)
```

```
[1] 0.50 0.60 0.25
```

```
c(9L, 10L, 11L, 12L, 13L) # números enteros (integer)
```

```
[1]  9 10 11 12 13
```

```
c(9:13) # secuencia de números enteros (integer sequence)
```

```
[1]  9 10 11 12 13
```

```
c(TRUE, FALSE, FALSE) # valores lógicos (logical)
```

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

```
c(1 + 0i, 2 + 4i) # números complejos (complex)
```

```
[1] 1+0i 2+4i
```

```
c("a", "b", "c") # caracteres (character)
```

```
[1] "a" "b" "c"
```

Acciones con vectores

1. Asignar los vectores a nombres:

   Creamos un vector llamado “dbl” que contiene los números decimales 0.5, 0.6 y 0.25.

   ```
   dbl <- c(0.5, 0.6, 0.25)
   ```

   Creamos un vector llamado “chr” que contiene los caracteres “a”, “b” y “c”.

   ```
   chr <- c("a", "b", "c")
   ```

2. Imprimir los vectores “dbl” y “chr” en la consola:

   Visualizamos en la consola el contenido del vector “dbl”, que son los números decimales 0.5, 0.6 y 0.25.

   ```
   dbl
   ```

   ```
   [1] 0.50 0.60 0.25
   ```

   Visualizamos en la consola el contenido del vector “chr”, que son los caracteres “a”, “b” y “c”.

   ```
   chr
   ```

   ```
   [1] "a" "b" "c"
   ```

3. Verificar el número de elementos en “dbl” y “chr”:

   Calculamos y mostramos en la consola la longitud del vector “dbl”, que es 3.

   ```
   length(dbl)
   ```

   ```
   [1] 3
   ```

   Calculamos y mostramos en la consola la longitud del vector “chr”, que es 3.

   ```
   length(chr)
   ```

   ```
   [1] 3
   ```

4. Verificar el tipo de dato de “dbl” y “chr”:

   Visualizamos en la consola el tipo de dato del vector “dbl”, que es “double” (números decimales).

   ```
   typeof(dbl)
   ```

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

   Visualizamos en la consola el tipo de dato del vector “chr”, que es “character” (caracteres).

   ```
   typeof(chr)
   ```

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

5. Combinar dos vectores:

   Se puede combinar el vector “dbl” consigo mismo utilizando la función “c()”, creando un nuevo vector que contiene los elementos duplicados del vector original.

   ```
   c(dbl, dbl)
   ```

   ```
   [1] 0.50 0.60 0.25 0.50 0.60 0.25
   ```

   Tambien se puede combina el vector “dbl” con el vector “chr” utilizando la función “c()”, creando un nuevo vector que contiene los elementos de ambos vectores concatenados.

   ```
   c(dbl, chr)
   ```

   ```
   [1] "0.5"  "0.6"  "0.25" "a"    "b"    "c"
   ```

> **Nota**
>
> El cambio automático del tipo de datos del vector resultante se denomina coerción. La coerción garantiza que se mantiene el mismo tipo de datos para cada elemento del vector.

Operaciones aritméticas con vectores

1. Definamos dos nuevos vectores numéricos llamados `a` y `b` con 4 elementos cada uno:

   ```
   a <- c(1, 2, 3, 4)
   b <- c(10, 20, 30, 40)
   ```

2. Realizamos una multiplicación escalar de `a` por 5, lo que significa que cada elemento en `a` se multiplica por 5:

   ```
   a * 5
   ```

   ```
   [1]  5 10 15 20
   ```

3. Realizamos una multiplicación de vectores entre `a` y `b`, lo que implica multiplicar cada elemento en `a` por el elemento correspondiente en `b`:

   ```
   a * b
   ```

   ```
   [1]  10  40  90 160
   ```

4. Creamos un nuevo vector numérico llamado `v` con longitud 5.

   ```
   v <- c(1.1, 1.2, 1.3, 1.4, 1.5)
   a * v
   ```

   ```
   [1] 1.1 2.4 3.9 5.6 1.5
   ```

> **Nota**
>
> Las operaciones aritméticas de los vectores se realizan por elementos. si dos vectores no tienen la misma longitud, el vector más corto se reciclará para que coincida con el más largo (en este caso, se vuelve a utilizar el primer elemento de a).

3. Matriz

Una matriz es una estructura bidimensional que contiene elementos organizados en filas y columnas. Todos los elementos de una matriz deben ser del mismo tipo de dato. Las matrices son útiles para almacenar datos tabulares, como una tabla de datos con variables en filas y observaciones en columnas. En R, podemos realizar operaciones matriciales y manipular los datos de manera eficiente utilizando esta estructura.

Matrices

1. Combinamos los vectores `a` y `b`, definidas anteriormente, por columnas utilizando la función `cbind()`:

```
A <- cbind(a, b)
A
```

```
     a  b
[1,] 1 10
[2,] 2 20
[3,] 3 30
[4,] 4 40
```

   Esta opción combina los vectores `a` y `b` por columnas, creando una matriz `A` donde los elementos de `a` forman la primera columna y los elementos de `b` forman la segunda columna.

2. Combinamos los vectores `a` y `b` por filas utilizando la función `rbind()`:

```
B <- rbind(a, b)
B
```

```
  [,1] [,2] [,3] [,4]
a    1    2    3    4
b   10   20   30   40
```

   En esta opción, los vectores `a` y `b` se combinan por filas para crear una matriz `B`. Los elementos de `a` forman la primera fila y los elementos de `b` forman la segunda fila.

3. Creamos una matriz a partir de los elementos de vector `a` utilizando la función `matrix()`:

```
A <- matrix(a, ncol = 2, nrow = 2)
A
```

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

   Aquí se utiliza la función `matrix()` para crear una matriz `A` a partir de los elementos del vector `a`. Se especifica que la matriz tendrá 2 columnas y 2 filas. Los argumentos nrow y ncol indican el número de filas y el número de columnas de que consta la matriz resultante.

4. Para 4 elementos y ncol =2 la matriz sólo puede tener 2 filas. Por lo tanto no es necesario especificar ambos argumentos

```
A <- matrix(a, ncol = 2)
A
```

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

   En esta variante, se crea una matriz `A` con 2 columnas y se ajusta automáticamente el número de filas según la longitud del vector `a`.

5. Por defecto la matriz se rellena columna a columna (R trata internamente un objeto matriz como vector columna). si la matriz debe rellenarse fila a fila se requiere el argumento `byrow = TRUE`

```
B <- matrix(a, ncol = 2, byrow = TRUE)
B
```

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

En esta opción, se crea una matriz `B` con 2 columnas y se especifica que los elementos del vector `a` se distribuirán por filas `byrow = TRUE`, es decir, los primeros elementos de `a` formarán la primera fila, los siguientes elementos formarán la segunda fila, y así sucesivamente.

Acciones con matrices

1. Verificamos el número de filas de la matriz `A` utilizando la función `nrow()`:

   ```
   nrow(A)
   ```

   ```
   [1] 2
   ```

   Esta línea de código devuelve el número de filas de la matriz `A`.

2. Verificamos el número de columnas de la matriz `A` utilizando la función `ncol()`:

   ```
   ncol(A)
   ```

   ```
   [1] 2
   ```

   Aquí se obtiene el número de columnas de la matriz `A`.

3. Verificamos la dimensión (número de filas y columnas) de la matriz `A` utilizando la función `dim()`:

   ```
   dim(A)
   ```

   ```
   [1] 2 2
   ```

   Esta línea de código devuelve la dimensión de la matriz `A` en formato `[nrow, ncol]`.

4. Combinamos dos matrices `A` por columnas utilizando la función `cbind()` y almacenamos el resultado en `D.wide`:

   ```
   D.wide <- cbind(A, A)
   D.wide
   ```

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

   En esta línea se crea una nueva matriz `D.wide` que combina las matrices `A` y `A` por columnas.

5. Combinamos dos matrices `A` por filas utilizando la función `rbind()` y almacenamos el resultado en `D.long`:

   ```
   D.long <- rbind(A, A)
   D.long
   ```

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

   Aquí se crea una nueva matriz `D.long` que combina las matrices `A` y `A` por filas.

6. Combinamos las matrices `D.wide` y `D.long` por columnas utilizando la función `cbind()` y almacenamos el resultado en `D`:

   ```
   # D <- cbind(D.wide, D.long)
   ```

   En esta línea se crea una nueva matriz `D` que combina las matrices `D.wide` y `D.long` por columnas.

Operaciones aritméticas con matrices

1. Suma de la matriz `B` consigo misma utilizando el operador `+`:

   ```
   B + B
   ```

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

Esta línea de código realiza la suma de la matriz `B` con ella misma.

2. Multiplicación escalar de la matriz `B` por 2 utilizando el operador `*`:

```
B * 2
```

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

Aquí se realiza la multiplicación de cada elemento de la matriz `B` por 2.

3. Multiplicación elemento a elemento de la matriz `B` consigo misma y almacenar el resultado en `a`:

```
a <- B * B
a
```

```
     [,1] [,2]
[1,]    1    4
[2,]    9   16
```

En esta línea se realiza la multiplicación elemento a elemento de la matriz `B` con ella misma, y el resultado se almacena en la matriz `a`.

4. Multiplicación de matrices utilizando el operador `%*%`:

```
C <- B %*% B
C
```

```
     [,1] [,2]
[1,]    7   10
[2,]   15   22
```

Aquí se realiza la multiplicación de matrices entre la matriz `B` y ella misma, y el resultado se almacena en la matriz `C`.

Otras operaciones con matrices:

1. Transposición de la matriz `D.wide` utilizando la función `t()`:

```
t(D.wide)
```

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

Esta línea de código transpone la matriz `D.wide`, es decir, intercambia las filas por columnas y viceversa.

2. Cálculo del determinante de la matriz `B` utilizando la función `det()`:

```
det(B)
```

```
[1] -2
```

Aquí se calcula el determinante de la matriz `B`.

3. Cálculo de la inversa de la matriz `B` utilizando la función `solve()` (solo si el determinante es diferente de 0):

```
solve(B)
```

```
     [,1] [,2]
[1,] -2.0  1.0
[2,]  1.5 -0.5
```

En esta línea se calcula la inversa de la matriz `B`, siempre y cuando el determinante sea diferente de 0.

4. Cálculo de los valores propios (eigenvalues) de una matriz cuadrada y simétrica utilizando la función `eigen()`:

```
eigen(B)
```

```
eigen() decomposition
$values
[1]  5.3722813 -0.3722813

$vectors
           [,1]       [,2]
[1,] -0.4159736 -0.8245648
[2,] -0.9093767  0.5657675
```

Aquí se calculan los valores propios de la matriz `B`. Esta operación solo es aplicable a matrices cuadradas y simétricas.

4. Data frame

Un data frame es una estructura similar a una matriz, pero más flexible. Puede contener columnas con diferentes tipos de datos, lo que lo hace ideal para almacenar conjuntos de datos heterogéneos. Los data frames son muy utilizados en el análisis de datos, ya que nos permiten manipular y explorar datos de manera eficiente. Podemos realizar operaciones de filtrado, selección y transformación en los data frames para obtener información significativa.

Creación del data frame:

1. Creamos vectores con diferentes tipos de datos, como números decimales (`dbl`), números enteros (`int`), valores lógicos (`lgl`) y caracteres (`chr`):

```
dbl <- c(0.5, 0.6, 0.25, 1.2, 0.333) # números decimales (double)
int <- c(9L, 10L, 11L, 12L, 13L) # números enteros (integer)
lgl <- c(TRUE, FALSE, FALSE, TRUE, TRUE) # valores lógicos (logical)
chr <- c("a", "b", "c", "d", "e") # caracteres (character)
```

Cada vector tiene elementos que representan valores de su respectivo tipo de dato.

2. Utilizamos la función `data.frame()` para combinar los vectores en un data frame llamado `df`:

```
df <- data.frame(dbl, int, lgl, chr)
```

El data frame `df` se crea utilizando los vectores `dbl`, `int`, `lgl` y `chr` como columnas.

3. Mostamos el contenido del data frame en la consola:

```
df
```

```
    dbl int   lgl chr
1 0.500   9  TRUE   a
2 0.600  10 FALSE   b
3 0.250  11 FALSE   c
4 1.200  12  TRUE   d
5 0.333  13  TRUE   e
```

Esto imprime el contenido del data frame `df`.

Acciones con data frames:

1. Verificamos el número de filas del data frame utilizando la función `nrow()`:

```
nrow(df)
```

```
[1] 5
```

Esta línea de código devuelve el número de filas en el data frame `df`.

2. Verificamos el número de columnas del data frame utilizando la función `ncol()`:

```
ncol(df)
```

```
[1] 4
```

Aquí se obtiene el número de columnas en el data frame `df`.

3. Verificamos la dimensión (número de filas y columnas) del data frame utilizando la función `dim()`:

```
dim(df)
```

```
[1] 5 4
```

Esta línea de código devuelve la dimensión del data frame `df` en formato `[nrow, ncol]`, es decir, el número de filas y columnas que tiene el data frame.

5. Lista

Una lista es una estructura de datos genérica que puede contener diferentes objetos, como vectores, matrices, data frames o incluso otras listas. A diferencia de las otras estructuras, las listas no tienen restricciones en cuanto a los tipos de datos o la longitud de los componentes individuales. Las listas son muy flexibles y se utilizan cuando necesitamos almacenar objetos de diferentes tipos o estructuras complejas.

Creación de la lista

1. Creamos una variable `a` que contiene un **escalar** de tipo entero (`1L`):

```
a <- 1L
```

2. Creamos un **vector numérico** `dbl` con 5 elementos:

```
dbl <- c(0.5, 0.6, 0.25, 1.2, 0.333)
```

3. Creamos un **vector de caracteres** `chr` con 3 elementos:

```
chr <- c("a", "b", "c")
```

4. Creamos un vector `v` con 4 elementos de tipo numérico:

```
v <- c(1.1, 1.2, 1.3, 1.4)
```

5. Creamos una matriz `mat` de tamaño 2x2 a partir del vector `v`:

```
mat <- matrix(v, ncol = 2)
```

La matriz `mat` tiene 2 columnas y los elementos del vector `v` se llenan por columnas.

6. Creamos una lista `l` que contiene los elementos `a`, `dbl`, `chr` y `mat`:

```
l <- list(a, dbl, chr, mat)
```

La lista `l` contiene estos elementos en ese orden.

7. Finalmente, visualizamos el contenido de la lista en la consola:

```
l
```

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

[[2]]
[1] 0.500 0.600 0.250 1.200 0.333

[[3]]
[1] "a" "b" "c"

[[4]]
     [,1] [,2]
[1,]  1.1  1.3
[2,]  1.2  1.4
```

Esto imprime el contenido de la lista `l`.

Es importante comprender estas estructuras de datos en R, ya que nos permiten organizar y manipular la información de manera efectiva. Cada estructura tiene sus propias características y funciones asociadas que nos facilitan el trabajo con los datos en la programación.