Tipos de datos en R

M. A. Ibáñez
Dpto. Economía Agraria, Estadística y Gestión de Empresas
ETSIAAB*

14 de septiembre de 2023

Índice

1. Introducción

En R se trabaja con objetos en los que se almacena información.

Para crear un objeto utilizamos el símbolo de asignación `<-` que conseguimos pulsando las teclas `<` y `-` [a].

Por ejemplo, la siguiente instrucción:

```
> ## Crea el objeto a al que se le asigna el valor numérico de tres
> a <- 3
```

ha creado el objeto `a` al que se le ha asignado el valor numérico de 3.

[a] Una alternativa es utilizar el símbolo `=`. Durante el curso siempre utilizaremos el símbolo `<-` para crear objetos

Dicho objeto aparece el área de trabajo, panel `Environmet` en RStudio.

en este momento sólo hemos creado el objeto `a`.

Si queremos que nos muestre por pantalla lo que tiene almacenado el objeto `a`, tecleamos la siguiente instrucción.

```
> ## Muestra por pantalla la información almacenada en el objeto a
> a

[1] 3
```

o también podemos utilizar

```
> ## Muestra por pantalla la información almacenada en el objeto a
> print(a)

[1] 3
```

Esta opción no la utilizaremos en este curso.

En R se distingue mayúsculas de minúsculas, es decir, el objeto `a` es distinto del objeto `A`.

Si tecleamos la instrucción:

```
> A

Error in eval(expr, envir, enclos): objeto 'A' no encontrado
```

nos indica que el objeto `A` no lo puede encontrar en el área de trabajo.

R tiene cinco **clases** básicas o 'atómicas' de objetos:

- character (carácter)
- numeric (numérico). Representa los números reales o decimales
- integer (entero)
- complex (número complejo)
- logical (lógico). Verdadero (TRUE) o Falso (FALSE)

Normalmente este tipo de objetos tan básicos no se encuentran de forma aislada sino que forman parte de una clase objeto básico que es el **vector**. Un vector es una colección de objetos básicos que puede contener cero o más objetos pero todos ellos tienen que ser de la misma clase. **Un vector solo puede contener objetos de la misma clase**.

Sin embargo, existe la clase denominada **list** que puede contener objetos de diferentes clases.

2. Números

En R generalmente los números son tratados como datos de tipo `numeric` (reales o decimales). Esto implica que si almacenas el número 1 o 2 en un objeto pensando que es un entero, se almacena como 1.00 o 2.00.

```
> x <- 1 ## Es un número real
> class(x) ## Muestra cual es la clase de x

[1] "numeric"
```

La función `class()` permite saber cual es la clase del objeto `x`

```
> x <- 6/2
> x

[1] 3

> class(x)

[1] "numeric"
```

Si queremos que lo almacene como `integer` hay que decírselo de forma explicita poniendo la letra `L` al final del número

```
> x <- 1L # es un entero
> x

[1] 1

> class(x)

[1] "integer"
```

En R existe un número especial llamado `Inf` que representa infinito. Es como un número real y puede ser utilizado en cálculos y obtener los resultados esperados.

Dividir por infinito:

```
> x <- 1/Inf # Dividir por infinito
> x

[1] 0
```

Dividir por 0:

```
> y <- 1/0   # Dividir por 0
> y

[1] Inf
```

Menos infinito:

```
> (-y)      # Menos infinito

[1] -Inf
```

Existe otro valor especial denominado `NaN` (Not a Number) y que representa un valor no definido, tal como dividir 0 por 0.

```
> x <- 0/0 # Valor no definido
> x

[1] NaN
```

3. Vectores

Como ya hemos dicho un vector es un objeto que contiene múltiples objetos de la misma clase.

Cuando asignamos un número a un objeto se crea un vector de la clase `numeric` de longitud 1.

```
> x <- 3
> x
[1] 3
```

El [1] nos indica que está mostrando el primer y único elemento del vector `x`.

Existen diferentes formas de crear vectores de longitud mayor que 1. En las siguientes secciones veremos algunas formas utilizadas habitualmente.

3.1. La función `c()` para crear vectores

Con la función `c()` se crea el vector concatenando los valores que lo componen, separando cada valor con una coma (,).

Vector numérico:

```
> x <- c(0.1,0.2,0.3) ## vector numérico de longitud 3
> x
[1] 0.1 0.2 0.3
> class(x)
[1] "numeric"
```

Vector lógico:

```
> x <- c(TRUE,FALSE,FALSE) ## vector lógico de longitud 3
> x
[1]  TRUE FALSE FALSE
> class(x)
[1] "logical"
> x <- c(T,F,F) ## Igual pero utilizando T (TRUE) y F (FALSE)
> x
[1]  TRUE FALSE FALSE
```

Vector carácter.

```
> x <- c("a","b","d") # vector carácter de longitud 3
> x
[1] "a" "b" "d"
> class(x)
[1] "character"
```

Los caracteres que queremos almacenar deben ir entre comillas.

Pero la función `c()` también puede concatenar varios vectores para formar un nuevo vector.

Concatena dos vectores:

```
> x <- c(2,3,7)
> y <- c(8,9,10)
>
> z <- c(x,y) # concatena dos vectores
> z
[1]  2  3  7  8  9 10
```

Concatena 3 vectores:

```
> w <- c(x,y,c(5,6)) ## concatena 3 vectores
> w
[1]  2  3  7  8  9 10  5  6
```

3.2. Creando un vector utilizando secuencias

El operador `:`, permite generara un vector a partir de una secuencia creciente o decreciente de enteros. El número a la izquierda de `:` indica donde empieza la secuencia y el número a la derecha donde termina.

```
> x <- 1:7  # vector de longitud 7 formado por la secuencia de
>         # enteros del 1 al 7
> x
[1] 1 2 3 4 5 6 7
```

El vector creado es de la clase integer.

```
> class(x) # integer
[1] "integer"
```

También se pueden crear secuencias decrecientes.

```
> x <- 40:12 # vector de enteros formado por la secuencia
>         #   decreciente de enteros del 40 al 12
> x
 [1] 40 39 38 37 36 35 34 33 32 31 30 29 28 27 26
[16] 25 24 23 22 21 20 19 18 17 16 15 14 13 12
```

Como la longitud del vector es grande, se necesita dos líneas de la consola para mostrar los valores. Cada línea comienza, indicando entre corchetes (`[]`), la posición del elemento en el vector.

La función `seq()` permite generar una mayor variedad de secuencias numéricas. Tiene tres argumentos: `from=`, `to=` y `by=` que podemos utilizar de la siguiente forma:

```
> x <- seq(from=2,to=10, by=2) ## secuencia numérica del 2
>                              ## hasta el 10 con incrementos de 2
> class(x) # con seq la clase es numeric y no integer
[1] "numeric"
```

Con la función `seq()` el vector es de la clase `numeric` y no `integer`.

Pero la función `seq()` tiene otros argumentos que podemos utilizar para crear secuencias. Por ejemplo,

el argumento `length.out`:

Indicando el inicio, el final y la longitud de la secuencia:

```
> x <- seq(from=2, to=10,length.out=4) ## secuencia de 4 números
>                              ## iniciada en 2 y acabada en 10
> x
[1]  2.000000  4.666667  7.333333 10.000000
```

Indicando el inicio, el incremento y la longitud de la secuencia:

```
> y <- seq(from=2, by=2,length.out=4) ## secuencia de 4 números
>                                   ## iniciada en 2 y de 2 en 2
> y
[1] 2 4 6 8
```

Aveces es necesario repetir un conjunto de números varias veces para generar el vector deseado. La función `rep()` nos permite hacer esto de varias formas.

```
> x <- c(3,4,7)
> y <- rep(x,times=5) # concatenamos el vector x 5 veces
>                   # para crear el vector y
> y
 [1] 3 4 7 3 4 7 3 4 7 3 4 7 3 4 7
```

Con el argumento `times` pegamos el vector `x` el número de veces deseado.

o, alternativamente, podemos utilizar el argumento `each` para pegar cada elemento del vector el número de veces que queramos.

```
> y <- rep(x,each=5) # pega cada número del vector x
>                   # 5 veces para crear el vector y
> y
 [1] 3 3 3 3 3 4 4 4 4 4 7 7 7 7 7
```

3.3. Índices de un vector

La función `length()` nos proporciona el número de elementos o longitud de un vector.

```
> length(x)
[1] 3
```

Los elementos de un vector (posiciones) pueden identificarse mediante la utilización de un índice que es un entero. Así, por ejemplo:

```
> x <- c(3,7,11,15,19)
> x[2]  # el segundo elemento de x
[1] 7
```

Utilizamos los corchetes para indicar a que posición del vector queremos acceder, en este ejemplo a la posición 2.

Además de acceder a la posición, también la podemos modificar el valor en dicha posición. El siguiente ejemplo almacena en la posición 2 del vector `x` el resultado de restar el valor de `x` en la posición 1 menos el valor de `x` en la posición 2.

```
> x

[1]  3  7 11 15 19

> x[2] <- x[2]-x[1]  # almacena en la posición 2 del vector x,
>                    # el resultado de restar el valor de x
>                    # en la posición 1 menos el valor de x
>                    # en la posición 2
```

```
> x # nuevo valor de x en la posición 2

[1]  3  4 11 15 19
```

Podemos crear un vector de posiciones para acceder a distintas posiciones de un vector.

El siguiente ejemplo muestra las posiciones 2, 1 y 4 del vector `x`, en el orden indicado.

```
> x[c(2,1,4)] # muestra las posiciones 2, 1 y 4 del vector x

[1]  4  3 15
```

De forma similar, utilizando el operador `:` para crear un vector con una secuencia de enteros.

```
> x[1:3] # muestra las tres primeras posiciones del vector x

[1]  3  4 11
```

La instrucción anterior explicada paso a paso:

```
> y <- 1:3 # vector con la secuencia de enteros del 1 al 3
> y

[1] 1 2 3

> x[y] # valores del vector x en las posiciones indicadas en y

[1]  3  4 11
```

3.4. Operaciones con vectores numéricos

Cuando trabajamos con vectores numéricos podemos realizar las operaciones aritméticas más comunes: la suma (+), la resta (-), la división (/) y la potenciación (^ o **). Todas estas operaciones se realizan elemento a elemento entre dos vectores.

La suma:

```
> x <- c(2,3,4)
> y <- c(1,2,3)
> z <- x+y ## suma de los elemento de x con los elementos de y.
>          # Posición por posición
> z

[1] 3 5 7
```

La resta:

```
> w <- x-y ## resta los elemento de x de los elementos de y.
>        ## Posición por posición
> w
[1] 1 1 1
```

La división:

```
> r <- x/y ## divide cada elemento de x
>        # por cada elemento de y
> r
[1] 2.000000 1.500000 1.333333
```

La potenciación:

```
> t <- x**y
> t
[1]  2  9 64
```

A todos los elementos de un vector numérico le podemos sumar un número

```
> x+2 ## sumar 2 a cada valor del vector x
[1] 4 5 6
```

Esto sería también válido para la resta, multiplicación, división y la potencia.

```
> x*2
[1] 4 6 8
> x/2
[1] 1.0 1.5 2.0
> x**2
[1]  4  9 16
```

Algunas funciones aritméticas utilizan como argumento el vector, y producen como salida un vector con la misma longitud que el original. Por ejemplo la función `sqrt()` para la raíz cuadrada.

```
> x <- c(9,81,6561)
> sqrt(x) ## raíz cuadrada de cada elemento del vector x
[1]  3  9 81
```

Además están también disponibles las funciones `log(), exp(), sin(), cos(), tan(), sqrt(),..`

4. Factores

Los factores son vectores de caracteres pero que se utilizan en R para representar categorías de datos no numéricos.

Para entender como funcionan los factores en R, supongamos que tenemos el vector `sexo` que recoge el sexo de 10 personas encuestadas.

```
> ## Sexo de 10 personas encuestadas
> sexo <- c("H", "M", "H", "H", "M", "M", "M", "M", "H", "H")
> sexo
 [1] "H" "M" "H" "H" "M" "M" "M" "M" "H" "H"
> class(sexo)
[1] "character"
```

De tal forma que el primer individuo es un hombre (`H`), la segunda persona es una mujer (`M`), etc.

Con la función `factor()` o con la función `as.factor()` podemos crear o convertir el vector `sexo` en un factor.

```
> ## Convierte el vector de caracteres sexo en un factor
> sexo <- factor(sexo)
```

```
> ## Muestra sexo por pantalla
> sexo
 [1] H M H H M M M M H H
Levels: H M
```

El objeto `sexo` ya no es de la clase `character` sino que ahora es de la clase `factor`.

```
> ## Indica de que clase es el vector sexo
> class(sexo)
[1] "factor"
```

Los niveles representan las distintas valores (categorías) que tiene el factor `sexo`, en este ejemplo dos: `H`, y `M`.

Cuando le pedimos a R que muestre los valores del factor `sexo` ademas de mostrar los valores en cada posición, nos indica los niveles que tiene dicho factor.

```
> ## Muestra el factor sexo
> sexo
 [1] H M H H M M M M H H
Levels: H M
```

Hay dos funciones que utilizamos habitualmente con un factor. La función `levels()` que nos muestra los niveles que tiene un factor.

```
> ## Muestra los niveles del factor sexo
> levels(sexo)
[1] "H" "M"
```

y la función `nlevels()` para obtener el número de niveles que tiene el factor.

```
> ## muestra el número de niveles del factor sexo
> nlevels(sexo)
[1] 2
```

Los factores tienen especial relevancia en el análisis estadístico de datos.

Como muestra de su utilidad, podemos utilizar la función `table()` para calcular el número de posiciones del vector `sexo` que son "H"(número de personas entrevistadas que son hombres) y el número de posiciones que son "M".

```
> ## Numero de observaciones en cada nivel del factor sexo
> Tab <- table(sexo)
> Tab

sexo
H M
5 5
```

Si además, queremos la proporción de cada nivel con respecto al total de personas entrevistadas, podemos utilizar la función `prop.table()`.

```
> # Proporción de observaciones en cada nivel del factor sexo
> prop.table(Tab)

sexo
  H   M
0.5 0.5
```

5. Listas

Una lista, de la clase `list`, es una clase de datos que puede contener 0 o más elementos, cada uno de los cuales puede ser de una clase distinta.

Las listas se pueden crear con la función `list()` y en este ejemplo vamos a crear una agenda con el nombre, apellido y teléfono de 3 personas.

```
> ## Lista con tres componentes
> Agenda <- list(Nombre=c("Maria","Juan","Pedro"),
+              Apellido=c("Garcia","Perez","Hernandez"),
+              Telf=c(6237863,6253478))
```

La lista creada tiene tres componentes, el primer componente lo hemos llamado `Nombre` y es un vector de la clase carácter con los nombres de las tres personas, el segundo componente, al que hemos llamado `Apellido` es otro vector de la clase carácter con el apellido y el tercer componente, con el nombre `Telf`, es un vector de la clase numérico con los teléfonos. En este ejemplo, solo tenemos el teléfono de las dos primeras personas. Cada componente se separa por una coma.

```
> Agenda

$Nombre
[1] "Maria" "Juan"  "Pedro"

$Apellido
[1] "Garcia"    "Perez"     "Hernandez"

$Telf
[1] 6237863 6253478
```

```
> class(Agenda)

[1] "list"
```

Podemos utilizar la función `names()` para obtener los nombres de los componentes.

```
> names(Agenda)  # Muestra el nombre de los componentes

[1] "Nombre"   "Apellido" "Telf"
```

La lista es una clase de objeto muy importante en R y muchas salidas de funciones estadísticas son objetos de la clase `list`.

5.1. Índices de una lista

Al igual que en el caso de los vectores podemos acceder a distintos componentes de una lista.

Cuando los componentes de una lista tienen nombre podemos acceder utilizando el nombre de las dos maneras siguientes:

Mostrar el componente `Apellido`.

```
> ## Muestra los elementos del componente Apellido
> Agenda$Apellido

[1] "Garcia"    "Perez"     "Hernandez"
```

o bien:

```
> Agenda[["Apellido"]]

[1] "Garcia"    "Perez"     "Hernandez"
```

Pero también podemos utilizar el doble corchete y la posición del componente en la lista.

```
> ## Muestra el primer componente
> Agenda[[1]]

[1] "Maria" "Juan"  "Pedro"
```

Es un vector carácter de longitud 3.

Si solo utilizamos un corchete.

```
> ## Muestra el nombre del componente y su contenido
> Agenda[1]

$Nombre
[1] "Maria" "Juan"  "Pedro"
```

El resultado muestra el nombre del componente y su contenido.

6. Marco de datos

Un marco de datos es un objeto de la clase `data.frame` y al igual que las listas puede contener componentes de diferentes clases, pero todos ellos tienen que tener la misma longitud. Es como una matriz pero en cada columna puedo tener vectores de diferentes clases.

La apariencia de un marco de datos es la de una tabla de datos y es la forma habitual de almacenar los datos en R.

Para crear un marco de datos podemos utilizar la función `data.frame()`, aunque no es la forma habitual de hacerlo. Normalmente se crean importando ficheros de datos con la función `read.table()` o similares (lo veremos más adelante).

En el siguiente ejemplo, se crea un marco de datos con la función `data.frame()`. Tiene tres componentes que representan el peso (kg), la altura (m) y sexo de tres individuos. Cada componente se separa con una coma.

```
> ## Creación de un marco de datos
> ## El componente Sexo es un factor
> datos.Ind <- data.frame(Peso=c(57,65,78),
+                         Altura=c(1.60,1.75,1.67),
+                         Sexo=factor(c("M","M","H")))
```

```
> datos.Ind

  Peso Altura Sexo
1   57   1.60    M
2   65   1.75    M
3   78   1.67    H

> class(datos.Ind)

[1] "data.frame"
```

Los componentes del marco de datos son las columnas, cada una de ellas con un nombre que podemos obtener utilizando la función `names()` o también la función `colnames()`.

```
> ## Nombres de los componentes del marco de datos datos.Ind
> colnames(datos.Ind)

[1] "Peso"   "Altura" "Sexo"
```

Cada fila representa a un individuo, en total 3 individuos.

6.1. Índices de los marcos de datos

Para poder acceder a determinada información contenida en un marco de datos podemos operar de diferentes formas.

Así, por ejemplo, si quiero obtener el valor almacenado en la primera fila y segunda columna:

```
> ## Muestra el valor de la fila 1 columna 1
> ## Utilizando la posición
> datos.Ind[1,2]

[1] 1.6
```

```
> ## Utilizando el nombre de la columna
> datos.Ind[1,"Altura"]

[1] 1.6
```

Para obtener los valores de la primera fila:

```
> ## Muestra la primera fila, todas las columnas
> datos.Ind[1,]

  Peso Altura Sexo
1   57    1.6    M
```

Los valores de la columna `Peso`

```
> ## Muestra la columna Peso. Tres formas distintas
> datos.Ind[,"Peso"]

[1] 57 65 78

> datos.Ind$Peso

[1] 57 65 78

> datos.Ind[,1]

[1] 57 65 78
```

Pero también podemos crear nuevas columnas o modificar los valores de las ya existentes.

Crear la columna `IMC` (Índice de masa corporal) utilizando el peso y la altura de cada individuo.

```
> ## Indice de masa corporal de cada individuo en el marco de datos
> datos.Ind$IMC <- datos.Ind$Peso/(datos.Ind$Altura**2)
> datos.Ind

  Peso Altura Sexo      IMC
1   57   1.60    M 22.26562
2   65   1.75    M 21.22449
3   78   1.67    H 27.96802
```

La función `str()` es muy útil para darnos información de que es lo que hay almacenado en un marco de datos.

```
> str(datos.Ind)

'data.frame': 3 obs. of  4 variables:
 $ Peso  : num  57 65 78
 $ Altura: num  1.6 1.75 1.67
 $ Sexo  : Factor w/ 2 levels "H","M": 2 2 1
 $ IMC   : num  22.3 21.2 28
```

En este ejemplo nos indica que tenemos 3 observaciones y 4 variables o columnas.

Que las columnas `Peso`, `Altura` y `IMC` son vectores `numeric` y que la columna `Sexo` es un factor. También nos muestra los primeros valores almacenados en cada columna.