Estructuras de datos

David García Callejas

Hemos visto los tipos de datos básicos en R: numérico (real y entero), carácter, lógico, complejo. También hemos visto cómo crear y manejar vectores de estos tipos. Pero existen estructuras de datos más complejas que permiten almacenar más información. Las más importantes son las listas, los dataframes, y las matrices/arrays.

Listas

Las listas pueden ser intuitivamente pensadas como una estructura semejante a un vector, cuyos elementos pueden contener *cualquier estructura*. Los elementos de una lista pueden ser numéricos, caracteres, vectores, o incluso otras listas. Además, cada elemento puede ser de tipo diferente.

```
# una lista con dos elementos numéricos, un carácter,
# y un vector de caracteres
mi.info <- list(18,04, "David", c("extremoduro","radio tarifa","tool"))
mi.info
```

Los elementos de una lista se pueden acceder de dos maneras

```
# un solo corchete, como los vectores
l1 <- mi.info[1]
l1
# ¿qué tipo de dato devuelve?
str(l1)

# doble corchete
l2 <- mi.info[[1]]
l2
# ¿qué tipo de dato devuelve?
str(l2)
```

En resumen, acceder a un elemento de una lista con un solo corchete nos devuelve *otra lista* con un solo elemento (¡sí, confuso!). Otro ejemplo:

```
# un vector de caracteres
musica1 <- mi.info[4]
# nos devuelve una lista con un único elemento, que es nuestro vector
musica1
str(musica1)
```

Acceder mediante doble corchete nos devuelve el elemento en sí:

```
musica2 <- mi.info[[4]]
# nos devuelve el vector de caracteres
str(musica2)
```

En general, para trabajar con listas, lo más útil es usar *doble corchete*. Más detalles, como siempre, en stack overflow! https://stackoverflow.com/questions/1169456/the-difference-between-bracket-and-double-bracket-for-accessing-the-el

Los elementos de las listas pueden nombrarse

```
mi.info <- list(dia = 18, mes = 4, nombre = "David",
                musica = c("extremoduro", "radio tarifa", "tool"))
mi.info
```

Tener listas "nombradas" (*named lists*, en inglés) nos da aun más flexibilidad para acceder a los elementos:

```
mi.musica <- mi.info[["musica"]]
mi.musica
```

Como en un vector, podemos añadir elementos:

```
# añado un quinto elemento
mi.info[[5]] <- c("breaking bad", "treme", "the expanse")
mi.info

# puedo nombrar a este quinto elemento
names(mi.info)[5] <- "series"
mi.info
```

Como decía, una lista puede contener otras listas como elementos.

```
# si quiero un número como nombre de un elemento, lo pongo entre comillas
mis.viajes <- list("2020" = c("Grazalema", "Parc del Cadí", "Cerdeña"),
                   "2019" = c("Portugal", "Doñana", "Picos de Europa"))
mis.viajes

# añado la lista "mis.viajes" a la lista general
mi.info[[6]] <- mis.viajes
names(mi.info)[6] <- "viajes"
mi.info

# ¿y para acceder a los elementos de una lista dentro de otra lista?
viajes.20 <- mi.info[[6]][["2020"]]
viajes.20
```

Las listas son estructuras enormemente flexibles, que R utiliza por defecto en muchas funciones. Por ejemplo, en una gran mayoría de modelos estadísticos, lo que R devuelve al ejecutar las funciones son listas con toda la información de los resultados en diferentes elementos.

Dataframes

Las listas, con toda su flexibilidad, son complejas de manejar, y no se ajustan bien a tipos de datos en los que tenemos varias columnas (variables) y filas (observaciones). Ese tipo de datos, similares a los que podríamos importar de una hoja de Excel, se adecuan mejor a la estructura llamada 'dataframe'. Un dataframe no es más que una serie de vectores distribuidos en columnas.

```
# seguramente el dataframe más usado en todos los tutoriales de R...
# características de diferentes especies de "Iris"
head(iris)

# ¿cómo crear un dataframe? fácil!
info.personal <- data.frame(dia = c(18,17), mes = c(4,5), nombre = c("David","Adam"))
info.personal

# para explorar un dataframe, tenemos varias funciones a nuestra disposición
```

```r
# las primeras 6 filas
head(info.personal)
# tipos de datos en cada columna
str(info.personal)
# resumen de los datos
summary(info.personal)
# número de filas
nrow(info.personal)
# número de columnas
length(info.personal)
# nombres de las columnas
names(info.personal)
```

Las columnas de un dataframe siempre vienen nombradas. Esta restricción es muy útil, porque nos permite acceder a los datos de cualquier columna por su nombre, tal y como en las listas.

```r
# ¿qué significa la coma?
info.personal[,"dia"]
```

También es muy común acceder a las columnas de un dataframe con otra sintaxis:

```r
info.personal$dia

# esto nos permite acceder cualquier combinación de fila y columna
info.personal$dia[1] # el día asociado a la primera observación
info.personal$nombre[2] # el nombre de la segunda observación
```

Hay varias maneras de añadir información a un dataframe ya existente

```r
# no se puede añadir elemento a elemento,
# porque no existe la fila 3 del dataframe
# info.personal$dia[3] <- 5

# lo ideal es crear un nuevo dataframe y añadirlo completo
# con la función "rbind" (row-wise bind, literalmente enganche de filas).
# Es como apilar piezas de Lego
nueva.fila <- data.frame(dia = 1, mes = 5, nombre = "Sofia")
info.personal <- rbind(info.personal,nueva.fila)

# La manera óptima de trabajar con dataframes es crear
# desde cero la estructura, con todas las filas que necesitemos.
# Esto permite a R asignar la memoria necesaria para esos datos,
# aunque estén temporalmente "vacíos".

personas <- 1000

info.long <- data.frame(dia = numeric(personas),
                        mes = numeric(personas),
                        nombre = character(personas),
                        stringsAsFactors = FALSE)

# este dataframe contiene 1000 entradas, todas vacías
head(info.long)

# podemos "rellenarlo" columna a columna,
info.long$mes <- 9
```

```r
# o elemento a elemento
info.long$dia[2] <- 18
info.long$nombre[1] <- "Anna"

# pero hacerlo fila a fila es problemático
# info.long[3,] <- c(1,2,"Mario")

# fijáos que lo que estamos asignando es un vector de caracteres.
# R asume que si mezclas números y caracteres en un mismo vector,
# todo será convertido a carácter.
mi.vector <- c(1,2,"Mario")
str(mi.vector)

# Si asignamos esto a una fila de nuestro dataframe,
# convertirá las columnas numéricas en columnas de caracteres.
# Y encima, lo hace sin avisar, así que tenemos que tener mucho cuidado!
info.long.copia <- info.long

info.long.copia[3,] <- c(1,2,"Mario")
str(info.long)
str(info.long.copia)
```

Ya que las columnas de un dataframe no son más que vectores, podemos operar con ellas de manera habitual. Por ejemplo, podemos generar nuevas columnas que sean el resultado de operaciones entre columnas existentes

```r
info.long$nueva.columna <- info.long$dia * info.long$mes
```

También es posible eliminar columnas (pero no filas) por completo

```r
info.long$nueva.columna <- NULL
# info.long[2,] <- NULL
```

NULL es un término reservado, que indica un valor no definido. En este caso, elimina la definición de una columna.

Matrices

Una matriz es una estructura de datos bidimensional, es decir, tiene filas y columnas igual que un dataframe. Sin embargo, las matrices por definición sólo pueden almacenar valores de un solo tipo. Mientras que un dataframe puede tener columnas numéricas y de caracteres, una matriz sólo puede tener columnas o numéricas, o lógicas, o de caracteres. Esta restricción las hace más eficientes en memoria.

```r
mi.matriz <- matrix(data = c(1,2,3,4,5,6,7,8,9),nrow = 3)
mi.matriz

# para acceder y modificar los elementos de una matriz, usamos los corchetes

# la primera fila
mi.matriz[1,]
# la primera columna
mi.matriz[,1]
# elemento de la segunda fila, tercera columna
mi.matriz[2,3]
```

```r
# las filas y columnas se pueden nombrar
rownames(mi.matriz) <- c("fila1","fila2","fila3")
colnames(mi.matriz) <- c("col1","col2","col3")
mi.matriz

# al igual que en listas y dataframes, si tenemos una matriz nombrada,
# podemos acceder a sus elementos por sus nombres
mi.matriz["fila1","col3"]
```

Por supuesto, R integra operaciones matemáticas matriciales

```r
# suma
mi.matriz + mi.matriz

# multiplicación de matrices
mi.matriz %*% mi.matriz

# multiplicación elemento a elemento
mi.matriz * mi.matriz

# o por un escalar
mi.matriz * 2

# podemos extraer la diagonal
diag(mi.matriz)

# o la transpuesta
t(mi.matriz)
```

A veces nos interesa transformar una matriz en dataframe, o viceversa. R nos permite estas transformaciones a través de una serie de funciones estándar

```r
# de matriz a dataframe
mi.df <- as.data.frame(mi.matriz)
mi.df

# de dataframe a matriz
mi.matriz.2 <- as.matrix(info.long)
# hemos intentado transformar a matriz un dataframe que
# contenía números y caracteres. R lo convierte en una matriz de caracteres
head(mi.matriz.2)
```

En R también podemos crear matrices de varias dimensiones, los llamados "arrays". No son tan comunes, por lo que no los veremos aquí. Pero si os interesa, se crean con la función "array". De manera general, un array de una dimensión es un vector, un array de dos dimensiones es una matriz, etc.

```r
?array
```