Indexación y operaciones

Carlos Iván Espinosa

06 de mayo de 2019

Contents

Pueden descargar este documento en pdf haciendo clic aquí

Indexación

Una vez creada una estructura de datos la flexibilidad de `R` para manipular los objetos es enorme, podemos seleccionar cualquier elemento o grupo de elementos y operar sobre ellos. La manipulación de los objetos depende de la dimensionalidad de los objetos, usaremos los corchetes `[]` para acceder a las diferentes dimensiones, dentro de cada objeto podemos acceder a la dimensión filas, columnas y hojas. cada uno de estos elementos separados por una coma. Así, si tengo un vector (una dimensión), solo necesito especificar la dimensión filas, si tengo una matriz o una data.frame necesito especificar dos dimensiones, filas y columnas. Finalmente, si tengo un arreglo, necesito especificar tres dimensiones, filas, columnas y hojas.

objeto[filas,columnas,hojas]

Veamos un ejemplo;

```
x <- 1:30
x[8:12]
```

```
## [1]  8  9 10 11 12
```

```
y <- matrix(1:9,3,3)
y[1:2,3]
```

```
## [1] 7 8
```

```r
z <- array(1:27,c(3,3,3))
z[2:3,2,3]
```

```
## [1] 23 24
```

Como vemos para el vector especificamos solamente las filas que queremos obtener (elementos del 8 al 12), para la matriz le dimos dos dimensiones los elementos de la fila uno y dos de la columna 3. Finalmente, en el arreglo le pedimos los elementos de la fila dos y tres, de la columna dos y de la hoja tres. Si queremos todos los elementos de alguna de las dimensiones dejamos el espacio en blanco. Así si me interesa tener todas las filas de la segunda columna de la matriz usaríamos:

```r
y[,2] #todas las filas de la columna dos
```

```
## [1] 4 5 6
```

```r
y[2,] #todas las columnas de la fila dos
```

```
## [1] 2 5 8
```

Extracción de elementos

La extracción de elementos del objeto puede realizarse de al menos 4 formas distintas.

1. Un vector de números naturales, usamos un vector numérico que extrae los elementos que se encuentran en la posición determinada por el vector numérico, si el vector de extracción es 2:3 extrae los elementos de la posición dos y tres.

```r
x <- seq(1,40, by=2)
x #todos los elementos del vector
```

```
##  [1]  1  3  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
```

```r
x[1:3] #los elementos de la posición uno a la tres
```

```
## [1] 1 3 5
```

```r
x[c(1,3,4)] #los elementos en la posición uno, tres y cuatro
```

```
## [1] 1 5 7
```

2. Un vector lógico de la misma longitud que el vector original. Podemos usar un vector lógico que permite extraer los elementos que cumplen la condición lógica. Podemos utilizar el mismo vector u otro vector.

```r
cat <- sample(c("a","b"), 20, replace =TRUE)
vl <- x>10 #Generamos el vector lógico

x[vl]#Extraemos los datos en función del vector lógico creado
```

```
##  [1] 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
```

```r
x[cat=="a"] #Usamos directamente la operación lógica
```

```
##  [1]  3  5  9 15 17 19 21 25 27 39
```

3. Un vector de números naturales negativos. Por medio de esta indexación se seleccionan todos los elementos del vector excepto los indicados con valores negativos.

```r
x[-(1:3)] #elementos que no están en posición uno a tres
```

```
##  [1]  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
```

```r
x[-c(1,3,4)] #elementos que no están en posición uno, tres y cuatro
```

```
##  [1]  3  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39
```

4. Extracción de elementos que cumplen determinada condición Muchas veces resulta conveniente extraer de un vector las **posiciones** que cumplen determinada condición

```r
a <- seq(1,30, by=3)
which(a>20) #Las posiciones 8,9 y 10 sus elementos son mayores a 20
```

```
## [1]  8  9 10
```

```r
a <- c (1,3,5,6,7,9,2)

which (a == 2) #Las posiciones 7 el elemento es igual a 2
```

```
## [1] 7
```

La indexación de las matrices, marco de datos y los arreglos se pueden acceder bajo los mismos principios, únicamente debemos recordar que debemos manejar más de una dimensión.

El caso de los marco de datos podemos acceder a las variables (columnas), usando el nombre de la variable usando el signo $. Veamos un ejemplo:

```r
y <- data.frame(peso=rnorm(10,20,5), tipo=rep(c("a","b"),5))
y$peso #selecciona toda la variable peso
```

```
##  [1] 24.096276 19.031501 19.737454  9.154021 22.053057 32.116259 20.368588
##  [8] 29.885017 29.793268 20.808359
```

```r
y$peso[y$tipo=="a"] #selecciona la variable peso, las filas que
```

```
## [1] 24.09628 19.73745 22.05306 20.36859 29.79327
```

```r
                    #corresponden a la variable tipo con elementos igual a "a"
```

En el segundo ejemplo vemos que, aunque y es una matriz, solo especifico las filas, esto es debido a que la primera parte (y$peso) es un vector por tanto tiene una sola dimensión.

Para acceder a las listas podemos utilizar los nombres de cada elemento de la lista o el doble corchete. Veamos un ejemplo.

```r
z <- list("a", matrix(letters[1:9],3,3), 1:20)#Creamos la matriz
names(z) <- c("letra","mat","num") #le damos nombres

z$letra #Extraemos por nombre
```

```
## [1] "a"
```

```r
z[[2]][,3] #Extraemos por posición
```

```
## [1] "g" "h" "i"
```

Operaciones con los objetos

Una de las enormes ventajas de R es que yo puedo utilizar operadores para modificar los objetos. Estos operadores pueden ser utilizados para realizar operaciones con los mismos vectores.

```r
x <- 1:20
x*2; sqrt(x); x^3
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20 22 24 26 28 30 32 34 36 38 40
```

```
##  [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751
##  [8] 2.828427 3.000000 3.162278 3.316625 3.464102 3.605551 3.741657
## [15] 3.872983 4.000000 4.123106 4.242641 4.358899 4.472136
```

```
##  [1]    1    8   27   64  125  216  343  512  729 1000 1331 1728 2197 2744
## [15] 3375 4096 4913 5832 6859 8000
```

Existen algunas funciones que nos permiten tener información sobre el vector.

```r
sum(x) #nos devuelve la suma del vector
```

```
## [1] 210
```

```r
length(x) #nos devuelve el largo del vector
```

```
## [1] 20
```

```r
max(x) #nos devuelve el valor máximo del vector
```

```
## [1] 20
```

```r
min(x) #nos devuelve el valor mínimo del vector
```

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

```r
median (x) #nos devuelve la mediana del vector
```

```
## [1] 10.5
```

```r
mean(x) #nos devuelve la media del vector
```

```
## [1] 10.5
```

```r
quantile(x) #nos devuelve los cuartiles del vector
```

```
##    0%   25%   50%   75%  100%
##  1.00  5.75 10.50 15.25 20.00
```

```r
range (x)  #nos devuelve el rango del vector
```

```
## [1]  1 20
```

```r
sd (x) #nos devuelve la desviación estándar del error
```

```
## [1] 5.91608
```

```r
y <- rep(c(25, 15, 5), 10)

unique(y) #nos devuelve un vector con valores únicos
```

```
## [1] 25 15  5
```

```r
duplicated(y) #nos devuelve un vector lógico indicándonos   si el elemento esta duplicado.
```

```
##  [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [12]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
## [23]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
```

Ordenando vectores

Muchas veces nos interesa ordenar un vector para esto podemos utilizar dos funciones; `order` y `sort`

```r
z <- sample(1:30, 20)
z
```

```
##  [1] 30 16 28  2 27 29  9 17  8 22  5 20 13 23  3 11 15  1 19 10
```

```r
sort(z)
```

```
##  [1]  1  2  3  5  8  9 10 11 13 15 16 17 19 20 22 23 27 28 29 30
```

```
order(z, decreasing=FALSE)
```

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

La diferencia entre estas dos funciones es que sort lo que hace es ordenar la función en orden ascendente. Mientras que order genera un vector con la posición de cada elemento, si cambiamos el argumento de decreasing a TRUE entonces el orden será decreciente.

Tabulando datos

Las variables categóricas nos pueden servir para tabular datos. Vamos a utilizar el objeto que generamos antes *cat3* y *cat2* para generar una tabla con la frecuencia por cada nivel.

```
cat <- rep(c("alto", "medio", "bajo"), c(10, 20,25))
cat <- factor(cat)
levels(cat)
```

```
## [1] "alto"  "bajo"  "medio"
```

```
cat1 <- relevel(cat, ref = "bajo")
cat2 <- factor(cat, levels = c("bajo", "medio", "alto"))

cat3 <- cat
levels(cat3) <- list(no.contaminado = "bajo", contaminado= c("medio", "alto"))
cat3
```

```
##  [1] contaminado    contaminado    contaminado    contaminado
##  [5] contaminado    contaminado    contaminado    contaminado
##  [9] contaminado    contaminado    contaminado    contaminado
## [13] contaminado    contaminado    contaminado    contaminado
## [17] contaminado    contaminado    contaminado    contaminado
## [21] contaminado    contaminado    contaminado    contaminado
## [25] contaminado    contaminado    contaminado    contaminado
## [29] contaminado    contaminado    no.contaminado no.contaminado
## [33] no.contaminado no.contaminado no.contaminado no.contaminado
## [37] no.contaminado no.contaminado no.contaminado no.contaminado
## [41] no.contaminado no.contaminado no.contaminado no.contaminado
## [45] no.contaminado no.contaminado no.contaminado no.contaminado
## [49] no.contaminado no.contaminado no.contaminado no.contaminado
## [53] no.contaminado no.contaminado no.contaminado
## Levels: no.contaminado contaminado
```

```
table(cat3)
```

```
## cat3
## no.contaminado    contaminado
##             25             30
```

```r
table(cat2)
```

```
## cat2
##  bajo medio  alto
##    25    20    10
```

Como vemos la función `table` nos permite ver las frecuencias de cada nivel. Pero podríamos utilizar esta función para hacer una tabla de contingencia entre ambas variables.

```r
table(cat2, cat3)
```

```
##        cat3
## cat2    no.contaminado contaminado
##   bajo              25           0
##   medio              0          20
##   alto               0          10
```

Pero nos podría interesar no tener un dato de frecuencias sino de proporciones, para esto podemos utilizar la función `prop.table`.

```r
x <- table(cat2, cat3)
prop.table(x,2)
```

```
##        cat3
## cat2    no.contaminado contaminado
##   bajo       1.0000000   0.0000000
##   medio      0.0000000   0.6666667
##   alto       0.0000000   0.3333333
```

La función prop.table funciona sobre un arreglo o tabla, el segundo argumento indica si la proporción la queremos por filas(1), columnas (2)

Funciones apply

Las funciones de la familia apply son de las funciones más utilizadas para manipular los datos. Vamos a ver algunas de estas funciones.

```r
a <- matrix(1:12, 3,4)

apply (a, 2, sum)
```

```
## [1]  6 15 24 33
```

La función `apply` nos permite obtener valores de una matriz usando funciones resumen. En este caso lo que hacemos es obtener la suma de las columnas. La función apply requiere los siguientes argumentos; una matriz sobre la que se ejecutará una función (a), indicar si esta función queremos que la ejecute por filas (1), o por columnas (2), la función de resumen que queremos obtener, podemos utilizar todas las funciones que vimos para los vectores.

La función apply() se puede ejecutar sobre una matriz, pero si tenemos una lista y queremos aplicar funciones resumen debemos ocupar funciones como `lapply()` o `sapply()`. La diferencia de estas dos funciones es que cuando usamos lapply() el resultado es una lista, y si usamos sapply() el resultado es un vector.

```r
x <- list (a = rnorm(10, 4), b = rnorm(10, 12), c = rnorm(10, 25))
x
```

```
## $a
##  [1] 4.168187 2.453147 3.168634 5.517918 2.869359 3.369860 3.342093
##  [8] 4.113620 2.526343 4.422402
##
## $b
##  [1] 11.26745 13.13708 10.83377 11.13119 13.39572 11.50568 12.98762
##  [8] 12.56497 11.49238 11.03458
##
## $c
##  [1] 25.34959 23.86407 25.52820 24.94501 25.43195 25.19227 23.40552
##  [8] 25.93170 25.50517 24.64594
```

```r
sapply(x, mean); lapply(x, mean)
```

```
##         a         b         c
##  3.595156 11.935044 24.979942
```

```
## $a
## [1] 3.595156
##
## $b
## [1] 11.93504
##
## $c
## [1] 24.97994
```

Estas funciones nos permiten tener datos resumen de matrices y arreglos, pero puede ser que me interese realizar cálculos condicionados sobre una variable en función de los niveles de un factor, en este caso utilizaremos la función `tapply()`.

```r
mercurio <- rnorm(55, 0.020, 0.003)

tapply(mercurio, cat2, mean)
```

```
##       bajo      medio       alto
## 0.02021007 0.02121791 0.01880898
```

Lo que hacemos es usar el vector factor que generamos antes cat2 y calcular la media de valores de mercurio por cada nivel de contaminación.

Fusionando datos

Cuando trabajamos con datos una de las cosas que necesitamos muy a menudo es unir datos o matrices de datos. Imaginemos que tenemos unos datos de contaminación con datos bióticos y por otra, datos abióticos, es posible que nos interese unir estas matrices, pero asegurarnos que los datos correspondan a cada estación de muestreo.

```r
bio<- data.frame(paste(rep(letters[1:5],c(5,5,5,5,5)),rep(1:5, 5), sep=""),
                 round(rnorm(25, 60, 15), 0),
                 round(rnorm(25, 30, 10), 0))

colnames(bio) <- c("sitio", "sp1", "sp2")


abio<- data.frame(paste(rep(letters[1:5],c(5,5,5,5,5)), rep(sample(1:5, 5), 5), sep=""),
                  rnorm(25, 8, 1),
                  rnorm(25, 1, 0.5))
colnames(abio) <- c("sitio", "N", "P")
```

Bueno hemos generado dos matrices con datos bióticos y abióticos. Ahora vamos a unirlos usando dos funciones distintas para ver las diferencias.

```r
datos <- cbind(bio, abio)

datos1 <- merge(bio, abio, by ="sitio")

head(datos1, 5); head(datos, 5)
```

```
##   sitio sp1 sp2         N         P
## 1    a1  71  33  6.886316 0.7166076
## 2    a2  45  14  7.821234 0.8641204
## 3    a3  97  37  7.931301 1.1527174
## 4    a4  66  27  7.404261 1.1561314
## 5    a5  64  34 10.076891 0.8965074
```

```
##   sitio sp1 sp2 sitio         N         P
## 1    a1  71  33    a1  6.886316 0.7166076
## 2    a2  45  14    a5 10.076891 0.8965074
## 3    a3  97  37    a2  7.821234 0.8641204
## 4    a4  66  27    a3  7.931301 1.1527174
## 5    a5  64  34    a4  7.404261 1.1561314
```

Como podemos ver los dos casos son completamente distintos, el que realmente nos interesa es la opción dos que la obtenemos con la función merge.

Ejercicios

---

1. Construir 3 vectores que contengan la siguiente información de 22 individuos:

- edad: 4 personas de 18 años, 3 personas de 24 años, 6 personas de 30 años y 9 personas de 35 años (ej. 18, 18, 18 . . . . . .)
- genero: deben estar ordenados masculino, femenino las 22 personas
- peso: un vector que tenga un peso medio de 150 y una desviación de 25

2. Realizar algunas operaciones con los vectores

- Obtener la cantidad de libras/año de cada persona
- La suma, valor máximo y mínimo, el promedio de los pesos de las mujeres y de los hombres
- Obtener un vector con los pesos de las mujeres y uno con el de hombres
- Obtener un vector con los pesos mayores de 115 y menores de 123

3. Con los datos proporcionados calcule lo siguiente:

- Convertir el vector provincias en niveles
- Obtenga datos de suma (sum), desviación estándar (sd),valores máximos (max) o mínimos (min) y conteo (length), por cada una de las provincias.

```
provincia <- c("loj", "azu", "zam", "mor", "mor", "oro", "gua", "gua",
               "zam", "can", "mor", "can", "zam", "zam", "azu", "loj", "azu", "oro",
               "gua", "can", "zam", "mor", "mor", "gua", "azu", "rios", "mor", "can", "can", "rios")

ingresos <- c(60, 49, 40, 61, 64, 60, 59, 54, 62, 69, 70, 42,
              56, 61, 61, 61, 58, 51, 48, 65, 49, 49, 41, 48, 52, 46, 59, 46, 58, 43)
```