uc3m | Universidad Carlos III de Madrid

Curso OpenCourseWare

Aprendizaje del Software Estadístico R: un entorno para simulación y computación estadística

Alberto Muñoz García

6. Estructuras de datos en R

Vectores

Se puede construir un vector de tipo numérico, lógico o carácter. Ejemplos de vectores son:

> c(1,5,3,2) **[1] 1 5 3 2**	**Crea un vector numérico de 4 elementos**
> c(T,F,T,T,F) **[1] TRUE FALSE TRUE TRUE FALSE**	**Crea un vector lógico de 5 elementos**
>c("barcelona","tarragona","lerida","gerona") **[1] "barcelona" "tarragona"** **"lerida" "gerona"**	**Crea un vector de 4 cadenas de caracteres**

La letra c significa "concatenar", y de hecho es un acrónimo para dicha palabra. Vamos a crear y a concatenar dos vectores:

> x = c(1,3,5)

> y = c(2,4,6)

> c(x,y)

[1] 1 3 5 2 4 6

>

La primera orden crea un vector formado por los números 1,3,5 y lo asigna a la variable x. El operador de asignación se escribe <- , o también _ (guión de subrayado).

Extracción de elementos de un vector

Hay tres maneras:

1. Especificar los índices de los elementos a extraer:

> x = c(18,11,12,10,7,6,17)

> x[c(1,3,6)]

[1] 18 12 6

La orden anterior extrae los elementos 1, 3 y 6 del vector. Un número negativo precediendo al índice significa exclusión. Con el vector x creado anteriormente:

> x[-3]

[1] 18 11 10 7 6 17

> x[-c(1,2)]

[1] 12 10 7 6 17

2. Especificar una condición lógica. En el caso del vector x creado arriba:

> x>10

[1] TRUE TRUE TRUE FALSE FALSE FALSE TRUE

> x[x>10]

[1] 18 11 12 17

3. En el caso de un vector de variables, podemos utilizar los nombres de las variables para extraer los elementos:

> A = 1

> B = 3

> C = 5

> y = c(A,B,C)

> y

[1] 1 3 5

> y[B]

[1] 5

En el ejemplo precedente, creamos tres variables A, B y C con los valores 1, 3 y 5 respectivamente.

A continuación creamos un vector y formado por dichas variables, y después extraemos el valor referenciado por la variable B.

Creación de patrones

Hay varias órdenes para crear vectores de modo automático:

from:to, seq y rep.

> 1:5

[1] 1 2 3 4 5

Genera números enteros entre 1 y 5.

> seq(1,6)

[1] 1 2 3 4 5 6

> seq(1,6,by=0.5)

[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0

> seq(1,6,length=10)

[1] 1.000000 1.555556 2.111111 2.666667 3.222222 3.777778 4.333333 4.888889

[9] 5.444444 6.000000

Hemos generado números entre 1 y 6 en los tres casos. En el segundo se va sumando al número anterior 0.5

hasta que se llega a 6. En el tercer caso hemos ordenado generar una secuencia de 10 números entre 1 y 6.

> rep(1,5)

[1] 1 1 1 1 1

> rep(c(1,2),5)

[1] 1 2 1 2 1 2 1 2 1 2

> rep(1:4,2)

[1] 1 2 3 4 1 2 3 4

> rep(1:3,c(1,4,5))

[1] 1 2 2 2 2 3 3 3 3 3

En el primer caso se ha repetido el 1 cinco veces. En el segundo, se ha repetido el patrón (1,2) cinco veces. En el tercer caso, la secuencia 1,2,3 ha sido repetida de acuerdo al vector (1,4,5), esto es, 1 vez el 1, 4 veces el 2 y 5 veces el 3.

Valores faltantes

El símbolo de valor faltante es NA (significa Not Available). Cualquier operación aritmética que involucre a un NA da por resultado un NA. Esto se aplica también a los operadores lógicos tales como **<, <=, >, >=, = =, !=** (= = es para comprobar si dos objetos son iguales, y != comprueba si dos objetos son distintos).

Veamos algunos ejemplos:

```
> x = c(1,2,3,NA,4,5)
> x
[1]  1  2  3 NA  4  5
> is.na(x)
[1] FALSE FALSE FALSE  TRUE FALSE FALSE
> x[x>2]
[1]  3 NA  4  5
> x*2
[1]  2  4  6 NA  8 10
```

is.na(x) pregunta qué elementos de x son faltantes. Sólo da valor cierto para el cuarto. La siguiente orden

pregunta qué valores de x superan a 2. La última multiplica cada elemento de x por 2.

```
> x
[1]  1  2  3 NA  4  5
> x<-x[!is.na(x)]
> x
```

[1] 1 2 3 4 5

En el párrafo precedente vemos que x contiene un valor faltante. La siguiente instrucción selecciona los valores de x no faltantes y asigna el resultado al mismo vector x. De esta manera eliminamos los valores faltantes del vector x.

Factores

Un factor es un vector que se usa para especificar una clasificación discreta de los componentes de otros vectores de la misma longitud.

Supongamos que tenemos un vector con la población de origen de 15 estudiantes, y que este vector se ha creado así:

>estudiantes.origen = c("getafe","mostoles","madrid","madrid","mostoles",

"leganes","getafe","leganes","madrid","mostoles","parla","alcorcon","mostoles",

"getafe","leganes")

> estudiantes.origen

[1] "getafe" "mostoles" "madrid" "madrid" "mostoles" "leganes"

[7] "getafe" "leganes" "madrid" "mostoles" "parla" "alcorcon"

[13] "mostoles" "getafe" "leganes"

> length(estudiantes.origen)

[1] 15

En la primera orden se crea el vector que contiene las 15 poblaciones de origen de los estudiantes. En la segunda orden se muestran dichos orígenes. En la tercera se pregunta la longitud de dicho vector.

Ahora creamos una variable de tipo factor, a partir de la existente:

> festudiantes = as.factor(estudiantes.origen)

> festudiantes

[1] getafe mostoles madrid madrid mostoles leganes getafe leganes

[9] madrid mostoles parla alcorcon mostoles getafe leganes

Levels: alcorcon getafe leganes madrid mostoles parla

> levels(festudiantes)

[1] "alcorcon" "getafe" "leganes" "madrid" "mostoles" "parla"

> summary(festudiantes)

alcorcon getafe leganes madrid mostoles parla

1 3 3 3 4 1

Al pedir un sumario de la variable de tipo factor 'festudiantes', el resultado es una tabla que nos muestra los niveles del factor (las poblaciones de origen), junto con el número de estudiantes correspondiente a tales niveles.

Supongamos ahora que disponemos de las estaturas de cada uno de los estudiantes del ejemplo anterior:

> estudiantes.estaturas = c(1.83, 1.71, 1.79, 1.64, 1.74, 1.81, 1.62, 1.84, 1.68, 1.81, 1.82, 1.74, 1.84, 1.61, 1.84)

> estudiantes.estaturas

[1] 1.83 1.71 1.79 1.64 1.74 1.81 1.62 1.84 1.68 1.81 1.82 1.74 1.84 1.61 1.84

Vamos a calcular ahora la estatura promedio de los estudiantes de cada población a partir de la muestra de la que disponemos:

> tapply(estudiantes.estaturas,festudiantes,mean)

alcorcon getafe leganes madrid mostoles parla

1.740000 1.686667 1.830000 1.703333 1.775000 1.820000

La función **tapply()** se utiliza para aplicar una función, en este caso **mean()** para cada grupo de componentes del primer argumento, definidos por los niveles de la segunda componente, en este caso, festudiantes.

Factores ordenados

Son factores cuyos niveles guardan un determinado orden. Para crear un factor ordenado o para transformar un factor en ordenado se usa la función **ordered().**

Supongamos que tenemos un vector con el nivel de inglés de 10 estudiantes:

```
> nivel.ingles = c("medio", "medio", "bajo",  "medio", "bajo",  "medio", "alto",  "alto",
"bajo", "bajo" )
> nivel.ingles
 [1] "medio" "medio" "bajo"  "medio" "bajo"  "medio" "alto"  "alto"  "bajo"
[10] "bajo"
```

Ahora creamos un factor ordenado con el nivel de inglés de los estudiantes:

```
> fnivel.ingles = ordered(nivel.ingles,levels=c("bajo","medio","alto"))
> fnivel.ingles
 [1] medio medio bajo  medio bajo  medio alto  alto  bajo  bajo
Levels:  bajo < medio < alto
```

Si ahora queremos saber qué estudiantes tienen un nivel de inglés por debajo de "medio":

```
> fnivel.ingles<"medio"
 [1] FALSE FALSE  TRUE FALSE  TRUE FALSE  TRUE  TRUE  TRUE  TRUE
```

Matrices y arrays

Una matriz en R es un conjunto de objetos indizados por filas y columnas. Un array en R es lo mismo, salvo que puede tener más de dos dimensiones.

La sintaxis general de la orden para crear una matriz es la siguiente:

matrix(data, nrow, ncol, byrow=F)

donde:

data	datos que forman la matriz
nrow	número de filas de la matriz
ncol	número de columnas de la matriz
byrow	Los datos se colocan por filas o por columnas según se van leyendo. Por defecto se colocan por columnas.

Algunos ejemplos:

> matrix(1:6) **[,1]** **[1,] 1** **[2,] 2** **[3,] 3** **[4,] 4** **[5,] 5** **[6,] 6**	Crea una matriz con 6 elementos. Al no especificarse nada, se entiende que se desea crear un vector columna
> matrix(1:6,nrow=2) **[,1] [,2] [,3]** **[1,] 1 3 5** **[2,] 2 4 6**	Crea una matriz con 6 elementos y dos filas. Los elementos, que son los números 1,2,3,4,5,6 se van leyendo por columnas.
> matrix(1:6,nrow=2,byrow=T) **[,1] [,2] [,3]** **[1,] 1 2 3** **[2,] 4 5 6**	Igual que en el caso anterior, pero se lee por filas, al especificar que la lectura por filas está activada.

Los datos que contiene una matriz deben ser todos del mismo tipo: todos numéricos, o de tipo carácter o lógico, pero no mezclados.

Algunas funciones sobre matrices

dim	devuelve las dimensiones de una matriz
dimnames	devuelve el nombre de las dimensiones de una matriz
colnames	devuelve el nombre de las columnas de una matriz
rownames	devuelve el nombre de las filas de una matriz
mode	devuelve el tipo de datos de los elementos de una matriz
length	devuelve el número total de elementos de una matriz
is.matrix	devuelve T si el objeto es una matriz, F si no lo es
[,]	accede a elementos dentro de la matriz
apply	Aplica una función sobre las filas o columnas de una matriz
cbind	Añade una columna a una matriz dada
rbind	Añade una fila a una matriz dada

Veamos algunos ejemplos:

```
> x = matrix(1:6,nrow=3)   # Creamos una matriz 3 x 2
> x                        # Se muestra la matriz x
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> length(x)                # Número de elementos de x
[1] 6
```

```
> mode(x)              # Tipo de datos de la matriz x
[1] "numeric"
> dim(x)               # Dimensiones de la matriz x
[1] 3 2
> dimnames(x)          # Nombre de las dimensiones de la matriz
NULL
> rownames(x)          # Nombre de las filas de la matriz
NULL
> colnames(x)          # Nombre de las columnas de la matriz
NULL
> is.matrix(x)         # El objeto x, ¿es una matriz?
[1] TRUE
> y<-c("blanco","negro")  # Creamos un vector de dos palabras
> is.matrix(y)         # El objeto y, ¿es una matriz?
[1] FALSE
> x[]                  # Se muestran todos los elementos de x
     [,1] [,2]
[1,]    1    4
[2,]    2    5
[3,]    3    6
> x[1,2]               # Se muestra el elemento 1,2 de x
[1] 4
> x[1,]                # Se muestra la primera fila de x
[1] 1 4
> x[,2]                # Se muestra la segunda columna de x
[1] 4 5 6

> cbind(x,c(0,0,0))    # Se añade una columna de ceros a x
     [,1] [,2] [,3]
[1,]    1    4    0
[2,]    2    5    0
[3,]    3    6    0
```

> rbind(x,c(0,0)) # Se añade una fila de ceros a x

[,1] [,2]

[1,] 1 4

[2,] 2 5

[3,] 3 6

[4,] 0 0

Asignando nombre a las filas y columnas de las matrices

Para ello utilizaremos las funciones **dimnames, colnames** y **rownames**, ya comentadas en el párrafo anterior. Por ejemplo, vamos a crear una matriz con tres personas y su edad, altura y peso como variables:

>**datos = matrix(c(20,65,174,22,70,180,19,68,170), nrow=3,byrow=T)**	Se crea una matrix 3x3
> datos **[,1] [,2] [,3]** **[1,] 20 65 174** **[2,] 22 70 180** **[3,] 19 68 170**	Se muestra la matriz
> colnames(datos) = c("edad","peso","altura")	Se asignan nombres a las columnas
> datos **edad peso altura** **[1,] 20 65 174** **[2,] 22 70 180** **[3,] 19 68 170**	Se vuelve a mostrar la matriz
> rownames(datos) = c("paco","pepe","kiko")	Se asignan a nombres a las columnas
> datos **edad peso altura** **paco 20 65 174** **pepe 22 70 180** **kiko 19 68 170**	Se vuelve a mostrar la matriz. Ya se ven los nombres asignados.

Alternativamente:

```
> datos = matrix(c(20,65,174,22,70,180,19,68,170),nrow=3,byrow=T)
> datos
    [,1] [,2] [,3]
[1,]   20   65  174
[2,]   22   70  180
[3,]   19   68  170
> dimnames(datos) = list(c("paco","pepe","kiko"), # Aqui esta el cambio
  c("edad","peso","altura"))
> datos
     edad peso altura
paco   20   65    174
pepe   22   70    180
kiko   19   68    170
```

La función **dimnames** funciona asignando a su argumento una lista de dos vectores de caracteres: los nombres de las filas y de las columnas de la matriz:

dimnames(objeto) = list(vector de nombres de las filas, vector de nombres de las columnas)

Ahora podemos acceder también a los elementos de la madrid usando los nombres:

```
> datos[,"edad"]          # Edades de todas las personas
paco pepe kiko
  20   22   19
> datos["pepe",]          # Variables del individuo "Pepe"
  edad   peso altura
    22     70    180
> datos[,c("edad","altura")] # Edad y altura de todas las personas
     edad altura
paco   20    174
pepe   22    180
kiko   19    170
```

```
> dimnames(datos)          # Muestra los nombres de filas y cols.
[[1]]
[1] "paco" "pepe" "kiko"

[[2]]
[1] "edad"   "peso"   "altura"

> apply(datos,2,mean)      # Hallamos la media de las variables
    edad      peso    altura # edad, peso y altura
 20.33333  67.66667 174.66667
```

Arrays

Un array es la generalización de una matriz de dos dimensiones al caso multidimensional. Su definición general es de la forma:

array(datos, dimensiones)

Los comandos para manejar arrays son similares a los que manejan matrices. Por ejemplo:

```
> array(1:12,c(2,3,2))
, , 1

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

, , 2
```

```
    [,1] [,2] [,3]
[1,]    7    9   11
[2,]    8   10   12
```

Un ejemplo más ilustrativo:

Vamos a crear un array con la edad media, el peso medio y la estatura media para hombres y mujeres de dos poblaciones: Villarriba y Villabajo:

```
> x<-array(c(45,46,65,55,170,167,48,49,68,56,169,165),c(2,3,2))
> dimnames(x) = list(c("hombres","mujeres"),c("edad","peso","altura"),
c("villarriba","villabajo"))
> x
, , villarriba

        edad peso altura
hombres   45   65    170
mujeres   46   55    167

, , villabajo

        edad peso altura
hombres   48   68    169
mujeres   49   56    165
```

Para acceder a los elementos del array:

```
> dimnames(x)               # Nombre de las dimensiones del array
[[1]]
[1] "hombres" "mujeres"

[[2]]
```

[1] "edad" "peso" "altura"

[[3]]

[1] "villarriba" "villabajo"

> x[,,"villarriba"] # Datos para la población "Villarriba"

** edad peso altura**

hombres 45 65 170

mujeres 46 55 167

> x["hombres",,] # Datos de todos los hombres

** villarriba villabajo**

edad 45 48

peso 65 68

altura 170 169

> x[,"edad",] # Edades de las personas

** villarriba villabajo**

hombres 45 48

mujeres 46 49

Vamos a aplicar ahora funciones a los elementos del array, utilizando la función apply, del mismo modo que lo hacíamos para matrices:

> apply(x,1,mean) # Media de las variables edad, peso y altura

hombres mujeres Para hombres y para mujeres, sin distinguir

94.16667 89.66667 población. Por ejemplo, la primera variable

se obtiene de calcular

(45+65+170+48+68+169)/6 = 94.16667

> apply(x,2,mean) # Media de la variable edad para toda la

edad peso altura población: (45+46+48+49)/4 = 47, y lo

47.00 61.00 167.75 mismo para las variables peso y altura

> apply(x,3,mean) # Media de todas las variables para las villarriba villabajo
poblaciones. En este ejemplo no tiene

91.33333 92.50000 mucho sentido este cálculo

Listas

Las listas sirven para concatenar objetos donde cada uno puede tener una estructura distinta. Esto no ocurre, por ejemplo, en los arrays, donde todos los elementos deben ser del mismo tipo (todos números, o todos carácter digamos).

Una lista tiene una serie de componentes, a los que deberemos asignar un nombre.
Para crear una lista podemos hacer algo como lo siguiente:

> familia = list(padre="juan",madre="maria",numero.hijos=3,

nombre.hijos=c("luis","carlos","eva"),edades.hijos=c(7,5,3),ciudad="lugo")

> familia

$padre

[1] "juan"

$madre

[1] "maria"

$numero.hijos

[1] 3

$nombre.hijos

[1] "luis" "carlos" "eva"

$edades.hijos

[1] 7 5 3

$ciudad

[1] "lugo"

Para ver los nombres de los objetos dentro de la lista:

> names(familia)

[1] "padre" "madre" "numero.hijos" "nombre.hijos" "edades.hijos" "ciudad"

Para acceder a componentes concretos se usa el operador $ seguido del nombre de la componente de la lista, o bien el número de la componente entre corchetes dobles [[]]:

> familia$padre

[1] "juan"

> familia$numero.hijos

[1] 3

Equivalentemente:

> familia[[1]]

[1] "juan"

> familia[[3]]

[1] 3

Muchas funciones devuelven como resultado una lista de objetos. Veremos ejemplos más adelante.

Data Frames

Los data frames son una estructura de datos que generaliza a las matrices, en el sentido en que las columnas (variables a menudo) pueden ser de diferente tipo entre sí (no todas numéricas, por ejemplo). Sin embargo, todos los elementos de una misma columna deben ser del mismo tipo. Al igual que las filas y columnas de una matriz, todos los elementos de un data frame deben ser de la misma longitud. De este modo, pueden usarse funciones tales como dimnames, dim, nrow sobre un data frame como si se tratara de una matriz. Los datos de un data frame pueden ser accedidos como elementos de una matriz o de una lista.

Para introducir los dataframes, partiremos de la matriz "datos", utilizada anteriormente:

```
> datos = matrix(c(20,65,174,22,70,180,19,68,170),nrow=3,byrow=T)
> dimnames(datos)<-list(c("paco","pepe","kiko"),
  c("edad","peso","altura"))
```

Vamos a añadir una columna a la matriz datos para que contenga la provincia de origen de cada persona:

```
> provincia = c("madrid","malaga","murcia")
> datos2 = cbind(datos,provincia)
> datos2
    edad peso altura
paco "20" "65" "174"  "madrid"
pepe "22" "70" "180"  "malaga"
kiko "19" "68" "170"  "murcia"
```

Como vemos, todas las variables han sido convertidas a tipo carácter, lo que no nos conviene, porque si intentamos hacer algún tipo de cálculo obtendremos un error:

```
> mean(datos[,"edad"])     # Calculamos la media de la variable edad
[1] 20.33333
> mean(datos2[,"edad"])    # Hacemos lo mismo, pero con datos2
Error in sum(..., na.rm = na.rm) : invalid "mode" of argument
```

Veamos qué tipo de datos hay en ambas matrices:

```
> mode(datos)
[1] "numeric"
> mode(datos2)          # Los datos han sido convertidos a carácter
[1] "character"
```

Para poder añadir la columna de tipo carácter sin causar problemas:

```
> datos2 = data.frame(datos,provincia)
> datos2
      edad peso altura provincia
paco   20   65    174    madrid
pepe   22   70    180    malaga
kiko   19   68    170    murcia

> mean(datos2[,"edad"])    # Ahora el cálculo no da problemas
[1] 20.33333
```

Sin embargo, a la hora de utilizar ciertas funciones hay que tener presente que no todas las variables son del mismo tipo:

```
> apply(datos2,2,mean)
Error in sum(..., na.rm = na.rm) : invalid "mode" of argument
```

No funciona porque hemos intentado hallar la media aritmética de cada columna de datos2. El error lo provoca la variable "provincia", que no es numérica.

```
> apply(datos2[,1:3],2,mean)
    edad     peso    altura
 20.33333  67.66667 174.66667
```

En este caso no ha habido error, porque la función apply sólo se ha aplicado sobre las variables numéricas.

Para acceder a los datos podemos proceder indistintamente como si de una matriz o de una lista se tratase:

> datos2[,2] # Acceso en modo matriz

paco pepe kiko

65 70 68

> datos2[,"edad"] # Acceso en modo matriz

paco pepe kiko

20 22 19

> datos2$edad # Acceso en modo lista

paco pepe kiko

20 22 19

Lo mismo para la variable "provincia":

> datos2[,4]

[1] madrid malaga murcia

Levels: madrid malaga murcia

> datos2[,"provincia"]

[1] madrid malaga murcia

Levels: madrid malaga murcia

> datos2$provincia

[1] madrid malaga murcia

Levels: madrid malaga murcia

Como vemos, las variables no numéricas se presentan como factores.

Si queremos utilizar las variables de un data frame por su nombre, sin hacer referencia a la matriz (por comodidad), utilizaremos la función **attach.**

> edad # intentamos acceder directamente a la variable edad

Error: Object "edad" not found

> datos2[,"edad"]# De este modo se puede acceder

paco pepe kiko

20 22 19

> attach(datos2)# Permite acceder a los nombres de datos2 directamente

> edad

paco pepe kiko

20 22 19

> detach(datos2)# Anula el acceso directo

> edad # Ya no se reconoce la variable directamente

Error: Object "edad" not found