

Introducción al software estadístico

Módulo VI

Nicolás Schmidt

`nschmidt@cienciassociales.edu.uy`

Departamento de Ciencia Política
Unidad de Métodos y Acceso a Datos
Facultad de Ciencias Sociales
Universidad de la República

Estructura de la presentación

1 Listas

- Estructura y Atributos
- Creación
- Vectores recursivos
- Indexación

■ Operaciones con listas

- `unlist()`
- `do.call()`
- `lapply()` y `sapply()`
- `stack()`

Estructura de la presentación

1 Listas

■ Estructura y Atributos

■ Creación

■ Vectores recursivos

■ Indexación

■ Operaciones con listas

`unlist()`

`do.call()`

`lapply()` y `sapply()`

`stack()`

Estructura y atributos de una lista

Las listas son vectores genéricos. Se diferencian de los vectores atómicos en que las listas pueden contener datos de distinto tipo. Y se diferencia de un `data.frame` (que es una lista) en que la longitud y la dimensión de los elementos que componen la lista pueden ser distintos.

- Al igual que los vectores atómicos, las listas tienen dos atributos, el nombre (`names()`) y el largo (`length()`).

Estructura y atributos de una lista

Ejemplo:

```
lista1 <- list(vec = 1:10,  
              logico = c(TRUE, FALSE, TRUE),  
              mat = matrix(1:4, 2,2),  
              miss = NA); lista1
```

```
## $vec  
## [1] 1 2 3 4 5 6 7 8 9 10
```

```
##  
## $logico  
## [1] TRUE FALSE TRUE
```

```
##  
## $mat  
##      [,1] [,2]  
## [1,]    1    3  
## [2,]    2    4
```

```
##  
## $miss  
## [1] NA
```

```
attributes(lista1)
```

```
## $names  
## [1] "vec"      "logico" "mat"     "miss"
```

Estructura de la presentación

1 Listas

- Estructura y Atributos
- Creación
- Vectores recursivos
- Indexación

■ Operaciones con listas

- `unlist()`
- `do.call()`
- `lapply()` y `sapply()`
- `stack()`

Creación de una `list()`

Las listas se crean se pueden crear de dos maneras.

- con la función `list()`
- con la función `as.list()`

Cuando creamos una lista con la función `list()` la estructura es la siguiente:

```
list(nombre1 = contenido1, nombre2 = contenido2, ... )
```

Es opcional incluir los nombres. Si no se incluyen nombre las listas por defecto hacen una indexación numérica.

Ejemplo:

```
a <- list(vec = 1:10, logico = c(TRUE, FALSE, TRUE), miss = NA)
a
```

```
## $vec
## [1] 1 2 3 4 5 6 7 8 9 10
##
## $logico
## [1] TRUE FALSE TRUE
##
## $miss
## [1] NA
```

```
b <- list(1:10, c(TRUE, FALSE, TRUE), NA)
b
```

```
## [[1]]
## [1] 1 2 3 4 5 6 7 8 9 10
##
## [[2]]
## [1] TRUE FALSE TRUE
##
## [[3]]
## [1] NA
```


Ejemplo: estructura de las listas a y b

```
str(a)

## List of 3
## $ vec      : int [1:10] 1 2 3 4 5 6 7 8 9 10
## $ logico: logi [1:3] TRUE FALSE TRUE
## $ miss    : logi NA
```

```
str(b)

## List of 3
## $ : int [1:10] 1 2 3 4 5 6 7 8 9 10
## $ : logi [1:3] TRUE FALSE TRUE
## $ : logi NA
```

Ejemplo: as.list

```
vec <- 1:5
vec

## [1] 1 2 3 4 5

vec <- as.list(vec)
vec

## [[1]]
## [1] 1
##
## [[2]]
## [1] 2
##
## [[3]]
## [1] 3
##
## [[4]]
## [1] 4
##
## [[5]]
## [1] 5
```

Estructura de la presentación

1 Listas

- Estructura y Atributos
- Creación
- Vectores recursivos
- Indexación

■ Operaciones con listas

- `unlist()`
- `do.call()`
- `lapply()` y `sapply()`
- `stack()`

Vectores recursivos

Como las listas pueden contener listas es que comúnmente se las denomina como vectores recursivos. Esta es otra diferencia sustantiva con los vectores atómicos.

Ejemplo: estructura de las listas a y b

```
c <- list(list(list(list(list()))));c

## [[1]]
## [[1]][[1]]
## [[1]][[1]][[1]]
## [[1]][[1]][[1]][[1]]
## list()

str(c)

## List of 1
## $ :List of 1
## ..$ :List of 1
## .. ..$ :List of 1
## .. .. ..$ : list()

c(1, c(1, c(1, c(1)))) #vector atómico

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

Estructura de la presentación

1 Listas

- Estructura y Atributos
- Creación
- Vectores recursivos
- Indexación

■ Operaciones con listas

- `unlist()`
- `do.call()`
- `lapply()` y `sapply()`
- `stack()`

Indexación: '[', '[[', '\$'

Ejemplo:

```
# creamos una lista
lista2 <- list(vec = 1:10, mat = matrix(1:4, 2, 2), dframe = iris[1:5,])
lista2

## $vec
## [1] 1 2 3 4 5 6 7 8 9 10
##
## $mat
##      [,1] [,2]
## [1,]    1    3
## [2,]    2    4
##
## $dframe
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1           3.5           1.4           0.2  setosa
## 2           4.9           3.0           1.4           0.2  setosa
## 3           4.7           3.2           1.3           0.2  setosa
## 4           4.6           3.1           1.5           0.2  setosa
## 5           5.0           3.6           1.4           0.2  setosa
```

Indexación: '[', '[[', '\$'

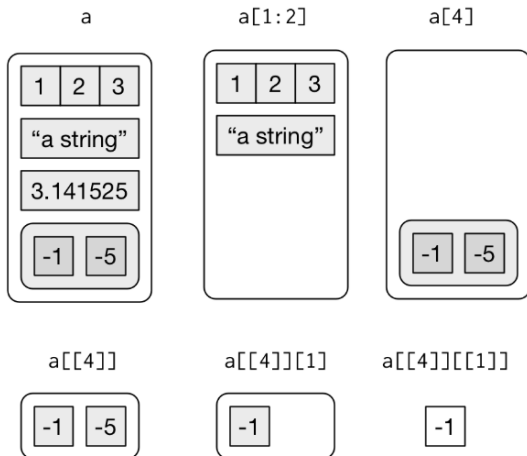
Ejemplo:

```
lista2[[1]]  
  
## [1] 1 2 3 4 5 6 7 8 9 10  
  
lista2$vec  
  
## [1] 1 2 3 4 5 6 7 8 9 10
```

Ejemplo: diferencia importante entre '[[' y '\$'

```
lista2[["vec"]]      # coincidencia exacta  
  
## [1] 1 2 3 4 5 6 7 8 9 10  
  
lista2$ve           # coincidencia parcial  
  
## [1] 1 2 3 4 5 6 7 8 9 10  
  
lista[["ve"]]        # coincidencia parcial que da error  
  
## Error in eval(expr, envir, enclos): object 'lista' not found
```

Diferencia en los usos de la indexación



Diferencia en los usos de la indexación



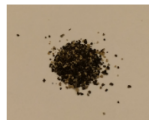
lista `x`



`x[1]`



`x[[1]]`



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

Fuente: <https://r4ds.had.co.nz>

Ejemplo: acceder y modificar objetos dentro de la lista

```
lista2[["mat"]][,2]           # segunda columna de la matriz

## [1] 3 4

lista2[["mat"]][,2] <- NA     # modifiko los valores de la segunda columna
lista2[[2]]                   # verifico el cambio

##           [,1] [,2]
## [1,]         1  NA
## [2,]         2  NA

lista2[[3]][1,]               # observo fila 1 del data.frame

##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1           5.1          3.5           1.4          0.2   setosa

colMeans(lista2[[3]][2])

## Sepal.Width
##           3.28

mean(lista2[[3]]$Sepal.Width)

## [1] 3.28
```

Estructura de la presentación

1 Listas

- Estructura y Atributos
- Creación
- Vectores recursivos
- Indexación

■ Operaciones con listas

- `unlist()`
- `do.call()`
- `lapply()` y `sapply()`
- `stack()`

unlist()

La función `unlist()` es útil para comprimir una lista, aplanarla.

Ejemplo:

```
mi.lista <- list(1:3, 9, NA, 78:89); mi.lista

## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] 9
##
## [[3]]
## [1] NA
##
## [[4]]
## [1] 78 79 80 81 82 83 84 85 86 87 88 89

unlist(mi.lista)

## [1] 1 2 3 9 NA 78 79 80 81 82 83 84 85 86 87 88 89
```

```
mi.lista2 <- list(5:6, list(1:2, "a")); mi.lista2
```

```
## [[1]]  
## [1] 5 6  
##  
## [[2]]  
## [[2]][[1]]  
## [1] 1 2  
##  
## [[2]][[2]]  
## [1] "a"
```

```
unlist(mi.lista2, recursive = FALSE)
```

```
## [[1]]  
## [1] 5  
##  
## [[2]]  
## [1] 6  
##  
## [[3]]  
## [1] 1 2  
##  
## [[4]]  
## [1] "a"
```

```
unlist(mi.lista2, recursive = TRUE)
```

```
## [1] "5" "6" "1" "2" "a"
```

do.call()

La función `do.call()` permite llamar a *una* función y ejecutarla.

Ejemplo:

```
lista5 <- list(sample(1:100, 10), sample(1:100, 10), sample(1:100, 10)); lista5

## [[1]]
##  [1] 92 42 95 66 89 68  3 57 75 74
##
## [[2]]
##  [1] 17 42 28  6 53  1 40 46 86 22
##
## [[3]]
##  [1] 40 10 74 30 24 25 97 86 80  8

do.call(rbind, lista5)

##           [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
## [1,]      92   42   95   66   89   68    3   57   75    74
## [2,]      17   42   28    6   53    1   40   46   86    22
## [3,]      40   10   74   30   24   25   97   86   80     8
```

lapply() y sapply()

Estas dos funciones permiten aplicar funciones a cada entrada de una lista. En el caso de `lapply()` el resultado va a ser en formato lista y en el caso de `sapply()` el resultado va a ser en un vector (la mayoría de las veces!).

```
lista6 <- list(1:20, 1:10); lista6
```

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

```
lapply(lista6, mean)
```

```
## [[1]]  
## [1] 10.5  
##  
## [[2]]  
## [1] 5.5
```

```
sapply(lista6, mean)
```

```
## [1] 10.5 5.5
```

lapply() vs. do.call()

```
lista <- list(10:12, 1:3)
```

```
lista
```

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

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

```
##
```

```
## [[2]]
```

```
## [1] 1 2 3
```

```
do.call(sum, lista)
```

```
## [1] 39
```

```
lapply(lista, sum)
```

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

```
## [1] 33
```

```
##
```

```
## [[2]]
```

```
## [1] 6
```


lapply() vs. do.call()

```
do.call(rbind,lista)
```

```
##      [,1] [,2] [,3]  
## [1,]  10  11  12  
## [2,]   1   2   3
```

```
lapply(lista, rbind)
```

```
## [[1]]  
##      [,1] [,2] [,3]  
## [1,]  10  11  12  
##  
## [[2]]  
##      [,1] [,2] [,3]  
## [1,]   1   2   3
```

stack()

Ejemplo:

```
df <- data.frame(var1 = 1:12, var2 = rep(letters[1:3], each=4))  
df
```

##	var1	var2
## 1	1	a
## 2	2	a
## 3	3	a
## 4	4	a
## 5	5	b
## 6	6	b
## 7	7	b
## 8	8	b
## 9	9	c
## 10	10	c
## 11	11	c
## 12	12	c

stack()

```
unstack(df)
```

```
##   a b  c
## 1 1 5  9
## 2 2 6 10
## 3 3 7 11
## 4 4 8 12
```

```
stack(unstack(df), select = -c)
```

```
##   values ind
## 1      1  a
## 2      2  a
## 3      3  a
## 4      4  a
## 5      5  b
## 6      6  b
## 7      7  b
## 8      8  b
```

stack()

Ejemplo:

```
lis <- list(a = 1:5, b = 11:15, c = 1)
lis

## $a
## [1] 1 2 3 4 5
##
## $b
## [1] 11 12 13 14 15
##
## $c
## [1] 1
```

stack()

```
df <- stack(lis)
df
```

```
##      values ind
## 1         1   a
## 2         2   a
## 3         3   a
## 4         4   a
## 5         5   a
## 6        11   b
## 7        12   b
## 8        13   b
## 9        14   b
## 10       15   b
## 11         1   c
```

```
unstack(df)
```

```
## $a
## [1] 1 2 3 4 5
##
## $b
## [1] 11 12 13 14 15
##
## $c
## [1] 1
```