Estructura de Datos

Angela Di Serio

Contents

1. Tipos de Datos **2**

Ejemplos 2

2. Estructuras de Datos **4**

2.1 Vectores 4

Creación de Vectores 4

Examinando Vectores 6

Agregar elementos 7

Missinga Data 7

Varios tipos en un vector 8

Subsettings 9

Operaciones con vectores 10

Práctica 1 **13**

2.2 Matrices 13

Extracción de elementos 16

Operaciones con Matrices 17

Operaciones matriz con escalar 17

Operaciones matriz con matriz 18

Funciones para matrices 19

2.3 Listas 21

Indexación 23

Manipulación de listas 25

Funciones que devuelven listas 27

Práctica 2 **30**

2.4 Data frame 30

Creación de data frame 30

Funciones para visualizar 31

Extracción de datos 33
Expansión de data frame 35
Agregar columnas 35
Añadir filas 36
2.5 Factores 37

Práctica.3 **39**

1. Tipos de Datos

Para aprovechar al máximo el lenguaje R, necesitamos una sólida comprensión de los tipos de datos básicos, las estructuras de datos y cómo operar con ellos.

En R todo es un objeto y se tienen 6 tipos de datos básicos:

- character
- numeric (real o decimal)
- integer
- logical
- complex
- raw

Elementos de estos tipos de datos se pueden combinar para formar estructuras de datos, como vectores, matrices, listas, etc.

Los vectores solo contienen datos de un mismo tipo.

- character: “a”, “swc”
- numeric: 2, 15.5
- integer: 2L (L indica que se almacene como un entero)
- logical: TRUE FALSE
- complex: 1+5i

R proporciona una serie de funciones para examinar características de vectores y otros objetos, por ejemplo:

- **class()** - ¿qué tipo de objeto es (alto nivel)?
- **typeof()** - ¿cuál es el tipo de datos del objeto (nivel bajo)?
- **length()** - ¿cuántos elementos tiene?
- **atributes()** - ¿tiene metadatos?

Ejemplos

```r
x="cadena de caracteres" # x<-"cadena de caracteres"
typeof(x)
```

```
## [1] "character"
```

```r
attributes(x)
```

```
## NULL
```

```r
x=1:10
x
```

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

```r
typeof(x)
```

```
## [1] "integer"
```

```r
length(x)
```

```
## [1] 10
```

```r
y = as.numeric(x)
y
```

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

```r
typeof(y)
```

```
## [1] "double"
```

2. Estructuras de Datos

R permite trabajar con diversas estructuras de datos:

- vectores
- matrices
- listas
- data frames
- factores

2.1 Vectores

Un vector es la estructura de datos más básica y común en R. Técnicamente pueden ser de dos tipos:

- vectores atómicos y matrices
- listas

Un vector es una colección de elementos del mismo tipo. Los vectores pueden ser caracteres, lógicos, enteros o numéricos.

Creación de Vectores

Se puede crear un vector vacío usando la función **vector()**. Por defecto su tipo será lógico. Se puede ser más específico durante su creación indicando el tipo de dato de los elementos del vector.

```
vector()
```

```
## logical(0)
```

```
vector("character",length = 4)
```

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

```
character(4)
```

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

```
numeric(4)
```

```
## [1] 0 0 0 0
```

```
logical(6)
```

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

También se pueden crear vectores indicando directamente cada uno de sus elementos.

```
x=c(1,2,3,4)
typeof(x)
```

```
## [1] "double"
```

Para crear un vector de enteros debemos indicar que los elementos son enteros agregando una L a cada uno de los elementos.

```
x=c(1L,2L,3L,4L)
typeof(x)
```

```
## [1] "integer"
```

Para crear un vector de tipo lógico usamos como valores TRUE y FALSE para especificar cada uno de los elementos.

```
y=c(TRUE,TRUE,FALSE)
typeof(y)
```

```
## [1] "logical"
```

Un vector de caracteres lo creamos indicando cada elemento como una cadena de caracteres entre comillas (simple o dobles).

```
nombres = c('Maria',"Pedro","Alfredo")
nombres
```

```
## [1] "Maria"   "Pedro"   "Alfredo"
```

```
typeof(nombres)
```

```
## [1] "character"
```

Otra forma de crear vectores es especificando una secuencia.

```
serie = 1:10
serie
```

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

```
seq(1:5)
```

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

```
seq(from=1, to=6, by=0.4)
```

```
##  [1] 1.0 1.4 1.8 2.2 2.6 3.0 3.4 3.8 4.2 4.6 5.0 5.4 5.8
```

Examinando Vectores

Las funciones **typeof()**, **length()**, **class()** y **str()** nos devuelven información útil sobre los vectores y sobre otros objetos en general.

```
typeof(nombres)
```

```
## [1] "character"
```

```
length(nombres)
```

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

```
class(nombres)
```

```
## [1] "character"
```

```
str(nombres)
```

```
##  chr [1:3] "Maria" "Pedro" "Alfredo"
```

Agregar elementos

La función **c()** (concatenar o combinar) puede ser usada para agregar elementos a un vector.

```
nombres = c(nombres,"Ana","Pablo")
nombres
```

```
## [1] "Maria"   "Pedro"   "Alfredo" "Ana"     "Pablo"
```

Missinga Data

R soporta **missing data** en vectores. Se representan con **NA** (Not Available).

```
x = c(1, NA, 5)
y = c(TRUE,FALSE,NA)
z = c("AB",NA,NA)
```

```
x
```

```
## [1]  1 NA  5
```

```
y
```

```
## [1]  TRUE FALSE    NA
```

```
z
```

```
## [1] "AB" NA   NA
```

La función **is.na()** indica los elementos que tienen valor **NA** en un vector, y la función **anyNA()** retorna TRUE si el vector contiene algún missing value.

```
is.na(x)
```

```
## [1] FALSE  TRUE FALSE
```

```
anyNA(x)
```

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

Varios tipos en un vector

Una característica de los vectores es que todos sus elementos son del mismo tipo. Si mezclamos tipos, R creará el vector con el tipo que mejor acomode a todos los elementos que contiene. Realiza automáticamente una conversión de tipos.

```
x = c(1, 1.5, "ABC")
y = c(TRUE, 5, FALSE, TRUE)
```

```
x
```

```
## [1] "1"   "1.5" "ABC"
```

```
y
```

```
## [1] 1 5 0 1
```

También podemos controlar la conversión indicando explícitamente el tipo deseado (**as.numeric()**, **as.character()**).

```
as.numeric(c("12","3"))
```

```
## [1] 12  3
```

```
as.character(c(1,2,3))
```

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

Subsettings

R nos permite extraer datos que cumplan con ciertas condiciones a partir de una estructura de datos.

```
valores=c(8150,100234,5040,9000)

valores > 8000              # devuelve un vector lógico
```

```
## [1]  TRUE  TRUE FALSE  TRUE
```

```
valores[valores>8000]       # devuelve el subconjunto que cumple con la condición
```

```
## [1]   8150 100234   9000
```

```
which(valores>8000)         # devuelve los índices o nombres de los elementos que cumplen con la condici
```

```
## [1] 1 2 4
```

```
valores[which(valores>8000)]# similar a valores[valores>8000]
```

```
## [1]   8150 100234   9000
```

```
valores[1]                  # devuelve el elemento en la posición 1
```

```
## [1] 8150
```

```
valores[1:2]                # devuelve los dos primeros valores
```

```
## [1]   8150 100234
```

```
valores[c(1,4)]             # devuelve el primero y cuarto elemento
```

```
## [1] 8150 9000
```

Operaciones con vectores

R nos permite realizar operaciones aritméticas con vectores

- suma (+)
- resta (-)
- producto de elementos (*)
- producto escalar (%*%)
- división (/)
- módulo (%)
- suma de los elementos (sum())

```
# creamos un vector
x = 1:5
x
```

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

```r
y = seq(from=1,to=20,length.out=5)
y
```

```
## [1]  1.00  5.75 10.50 15.25 20.00
```

```r
z = x + y
z
```

```
## [1]  2.00  7.75 13.50 19.25 25.00
```

```r
z= x * y
z
```

```
## [1]   1.0  11.5  31.5  61.0 100.0
```

```r
z= x%*%y
z
```

```
##      [,1]
## [1,]  205
```

```r
sum(x)
```

```
## [1] 15
```

También podemos realizar operaciones estadísticas sobre vectores como:

- media (mean())
- mediana (median())
- máximo (max())
- mínimo (min())
- cuantiles (quantile())
- coeficiente de correlación (cor())
- suma acumulada (cumsum())
- producto acumulado (cumprod())
- diferencias (diff())

```
mean(x)
```

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

```
median(y)
```

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

```
max(x)
```

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

```
quantile(y)
```

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

```
cumsum(y)
```

```
## [1]  1.00  6.75 17.25 32.50 52.50
```

```
cumprod(x)
```

```
## [1]   1   2   6  24 120
```

```
diff(y)
```

```
## [1] 4.75 4.75 4.75 4.75
```

```
cor(x,y)
```

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

Práctica 1

Realizar los ejercicios del archivo **02MBIF_07_Practica_1_Estructura_de_Datos.R**

2.2 Matrices

En R, las matrices son una extensión de los vectores numéricos o de caracteres. Se trata de vectores con dimensiones, es decir número de filas y de columnas. Al igual que en el caso de vectores, los elementos deben ser del mismo tipo.

```
m= matrix(nrow=2, ncol=3)
m
```

```
##      [,1] [,2] [,3]
## [1,]   NA   NA   NA
## [2,]   NA   NA   NA
```

```
dim(m)
```

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

```
class(m)
```

```
## [1] "matrix" "array"
```

```r
m= matrix(c(1:6), nrow=2, ncol=3)
m
```

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

```r
class(m)
```

```
## [1] "matrix" "array"
```

```r
typeof(m)
```

```
## [1] "integer"
```

```r
typeof(m[1])
```

```
## [1] "integer"
```

```r
m = 1:10
m
```

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

```r
dim(m) = c(2,5)
m
```

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

Podemos usar las funciones **cbind()** y **rbind()** que nos permiten unir vectores por columna o por fila respectivamente.

```r
x = 1:4
x
```

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

```r
y = 9:6
y
```

```
## [1] 9 8 7 6
```

```r
rbind(x,y)
```

```
##   [,1] [,2] [,3] [,4]
## x    1    2    3    4
## y    9    8    7    6
```

```r
cbind(x,y)
```

```
##      x y
## [1,] 1 9
## [2,] 2 8
## [3,] 3 7
## [4,] 4 6
```

Se puede usar el argumento **byrow** para especificar cómo se debe llenar la matriz con los valores que se especifican como elementos de la matriz.

```r
m = matrix(c(1:10), nrow=2, ncol=5, byrow=TRUE)
m
```

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

```r
m = matrix(c(1:10), nrow=2, ncol=5)
m
```

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

Podemos asignar nombres a las columnas y filas de una matriz usando las funciones **colnames()** y **rownames()** respectivamente.

```r
colnames(m) = c("M1","M2","M3","M4","M5")
m
```

```
##      M1 M2 M3 M4 M5
## [1,]  1  3  5  7  9
## [2,]  2  4  6  8 10
```

```r
rownames(m) = c("G1","G2")
m
```

```
##    M1 M2 M3 M4 M5
## G1  1  3  5  7  9
## G2  2  4  6  8 10
```

Extracción de elementos

Los elementos de una matriz se referencian especificando el índice/nombre a lo largo de cada una de las dimensiones entre corchetes.

```r
m[2,1]          # elemento en la fila 2, columna 1
```

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

```r
m["G2","M1"]          # elemento en la fila 2, columna 1
```

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

```r
m[2,]       # todos los elementos en la fila 2
```

```
## M1 M2 M3 M4 M5
##  2  4  6  8 10
```

```r
m["G2",]
```

```
## M1 M2 M3 M4 M5
##  2  4  6  8 10
```

```r
m[1,c(1,3,5)]  # elementos fila 1, columnas 1, 3 y 5
```

```
## M1 M3 M5
##  1  5  9
```

```r
m["G1",c("M1","M3","M5")]  # elementos fila 1, columnas 1, 3 y 5
```

```
## M1 M3 M5
##  1  5  9
```

```r
m[,c(1,3,5)]  # elementos columnas 1, 3 y 5
```

```
##    M1 M3 M5
## G1  1  5  9
## G2  2  6 10
```

Operaciones con Matrices

Operaciones matriz con escalar Cuando realizamos una operación aritmética de una matriz con un escalar, la operación indicada se lleva a cabo sobre cada uno de los elementos de la matriz.

```r
m = matrix(c(1:6), nrow=2, ncol=3, byrow=TRUE)
m
```

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

```
m + 5
```

```
##      [,1] [,2] [,3]
## [1,]    6    7    8
## [2,]    9   10   11
```

```
m / 2
```

```
##      [,1] [,2] [,3]
## [1,]  0.5  1.0  1.5
## [2,]  2.0  2.5  3.0
```

Operaciones matriz con matriz Las operaciones con matrices se pueden realizar de manera similar a cómo se realizan operaciones aritméticas con números. Sin embargo debemos tener cuidado con la dimensión de las matrices que usemos en las operaciones.

```
m = matrix(c(1:6), nrow=2, ncol=3, byrow=TRUE)
n = matrix(c(1,2,1,2,1,2), nrow=2, ncol=3)
m;n
```

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

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

```
# suma de matrices
m + n
```

```
##      [,1] [,2] [,3]
## [1,]    2    3    4
## [2,]    6    7    8
```

Cuando utilizamos el operador * para la multiplicación de matrices, la operación se realiza elemento a elemento. En este caso las matrices deben tener la misma dimensión.

```
# multiplicacion elemento a elemento
m * n
```

```
##      [,1] [,2] [,3]
## [1,]    1    2    3
## [2,]    8   10   12
```

Para realizar una multiplicación de matrices se coloca el operador %*%.

```
# multiplicacion de matrices
# debemos usar la transpuesta de n

m %*% t(n)
```

```
##      [,1] [,2]
## [1,]    6   12
## [2,]   15   30
```

Funciones para matrices Podemos utilizar algunas funciones que también se usan para vectores como sum, prod, mean, que se aplican sobre las filas o columnas de la matriz. Como por ejemplo:

- colSums: Vector de sumas de columnas.
- rowSums: Vector de sumas de filas.
- colMeans: Vector de medias de columnas.
- rowMeans: Vector de medias de filas.

```
m = matrix(c(1:6), nrow=2, ncol=3, byrow=TRUE)
m
```

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

```
colSums(m)
```

```
## [1] 5 7 9
```

```
rowSums(m)
```

```
## [1]  6 15
```

```
colMeans(m)
```

```
## [1] 2.5 3.5 4.5
```

```
rowMeans(m)
```

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

Se puede usar la función **apply( )** que permite aplicar funciones sobre filas o columnas de una matriz.

apply(X, MARGIN, FUN, ..., simplify = TRUE)

Argumentos:

- X: un vector incluyendo matrices
- MARGIN: un vector que proporciona los subíndices sobre los que se aplicará la función. Por ejemplo, para una matriz, 1 indica filas, 2 indica columnas, c(1, 2) indica filas y columnas.
- FUN: la función a ser aplicada
- simplify: argumento lógico que indica si los resultados deben simplificarse de ser posible.

```
m
```

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

```
apply(m,1,sum)
```

```
## [1]  6 15
```

```
m
```

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

```
apply(m,2,sum)
```

```
## [1] 5 7 9
```

```
m
```

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

```
apply(m,c(1,2),sum)
```

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

```
my= apply(m,1:2, function(x) x+3) # equivalente a my = m + 3
my
```

```
##      [,1] [,2] [,3]
## [1,]    4    5    6
## [2,]    7    8    9
```

2.3 Listas

En R, las listas actuan como contenedores. A diferencia de los vectores, no hay restricción en cuanto al tipo de los elementos.

Para crear una lista usamos la función **list()**, También podemos hacer una conversión de otro objeto a lista usando la función **as.list()**.

```
x = list("Maria", 19,"S", c(8.5,9.3,9.0))
x
```

```
## [[1]]
## [1] "Maria"
##
## [[2]]
## [1] 19
##
## [[3]]
## [1] "S"
##
## [[4]]
## [1] 8.5 9.3 9.0
```

En general, resulta más legible si los distintos componentes de la lista se identifican con un nombre. Esto se puede realizar directamente al crear la lista o, a posteriori, con la función **names()**.

```r
names(x) <- c("Nombre", "Edad", "Estado Civil", "Calificaciones")
x
```

```
## $Nombre
## [1] "Maria"
##
## $Edad
## [1] 19
##
## $`Estado Civil`
## [1] "S"
##
## $Calificaciones
## [1] 8.5 9.3 9.0
```

```r
y = list(Nombre="Maria", Edad=19,"Estado Civil"="S", Calificaciones=c(8.5,9.3,9.0))
y
```

```
## $Nombre
## [1] "Maria"
##
## $Edad
## [1] 19
##
## $`Estado Civil`
## [1] "S"
##
## $Calificaciones
## [1] 8.5 9.3 9.0
```

```r
x = vector("list", length=4) # Lista vacia
x
```

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

Las listas suelen contener componentes de distintos tipos y por tanto, no resulta fácil visualizarlas en la consola. La función **str()** produce una visualización más estructurada.

```
str(y)
```

```
## List of 4
##  $ Nombre        : chr "Maria"
##  $ Edad          : num 19
##  $ Estado Civil  : chr "S"
##  $ Calificaciones: num [1:3] 8.5 9.3 9
```

Esta función **str()** es de tipo genérica, por lo que puede aplicarse a cualquier objeto. Otra función útil para visualizar el contenido de objetos es **View()**, que abre un panel con información del objeto consultado en la zona del editor de texto de RStudio.

```
View(y)
```

Para obtener el número de componentes de una lista se usa la función **length()**.

```
length(y)
```

```
## [1] 4
```

Indexación

Existen varias formas de indexar una lista: **[], [[ ]] y $**.

La **indexación de listas con** [ ] admite las mismas posibilidades que con vectores.

```
y[c(1,4)]
```

```
## $Nombre
## [1] "Maria"
##
## $Calificaciones
## [1] 8.5 9.3 9.0
```

```
y[-c(1,3)]
```

```
## $Edad
## [1] 19
##
## $Calificaciones
## [1] 8.5 9.3 9.0
```

```
y[c("Nombre","Edad")]
```

```
## $Nombre
## [1] "Maria"
##
## $Edad
## [1] 19
```

```
y[c(TRUE,TRUE,FALSE,TRUE)]
```

```
## $Nombre
## [1] "Maria"
##
## $Edad
## [1] 19
##
## $Calificaciones
## [1] 8.5 9.3 9.0
```

Al indexar con [ ] se obtiene una lista con los componentes que hemos seleccionado.

Es importante notar que al indexar con [ ] se obtiene una lista aunque sólo accedamos a un componente de la lista.

Veamos que sucede si queremos sumarle 1 al componente Edad del objeto y:

```
y["Edad"]
```

```
## $Edad
## [1] 19
```

```
typeof(y["Edad"])
```

```
## [1] "list"
```

```
#  y["Edad"] + 1    PROBAR
```

No es posible aplicar este tipo de operaciones cuando usamos [ ].

Cuando queramos acceder al contenido de un único componente de la lista para trabajar con este, hay que usar el **operador [[ ]] o el operador $**.

```
y[["Edad"]]
```

```
## [1] 19
```

```
typeof(y[["Edad"]])
```

```
## [1] "double"
```

```
y$Edad
```

```
## [1] 19
```

```
typeof(y$Edad)
```

```
## [1] "double"
```

Cuando se usa el operador[[ ]] sólo se puede indicar un índice.

El **operador $** es cómodo para acceder al contenido de un componente por su nombre.

Manipulación de listas

Podemos agregar, eliminar y actualizar elementos de la lista como se muestra a continuación. Podemos agregar y eliminar elementos solo al final de una lista. Pero podemos actualizar cualquier elemento.

```
list_data = list(c("Ene","Feb","Mar"),
                 matrix(c(3,9,5,1,-2,8), nrow = 2),
                 list("verde",12.3))
```

```
# Asignar nombres a los elementos en la lista
names(list_data) = c("1er Trimestre", "Una_Matriz", "Una_lista_Interna")

# Show the list.
print(list_data)
```

```
## $`1er Trimestre`
## [1] "Ene" "Feb" "Mar"
##
## $Una_Matriz
##      [,1] [,2] [,3]
## [1,]    3    5   -2
## [2,]    9    1    8
##
```

```
## $Una_lista_Interna
## $Una_lista_Interna[[1]]
## [1] "verde"
##
## $Una_lista_Interna[[2]]
## [1] 12.3
```

```
str(list_data)
```

```
## List of 3
##  $ 1er Trimestre    : chr [1:3] "Ene" "Feb" "Mar"
##  $ Una_Matriz       : num [1:2, 1:3] 3 9 5 1 -2 8
##  $ Una_lista_Interna:List of 2
##   ..$ : chr "verde"
##   ..$ : num 12.3
```

```
length(list_data)
```

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

```
# Agregar un elemento al final de la lista
list_data[4] <- "Nuevo Elemento"
print(list_data)
```

```
## $`1er Trimestre`
## [1] "Ene" "Feb" "Mar"
##
## $Una_Matriz
##      [,1] [,2] [,3]
## [1,]    3    5   -2
## [2,]    9    1    8
##
## $Una_lista_Interna
## $Una_lista_Interna[[1]]
## [1] "verde"
##
## $Una_lista_Interna[[2]]
## [1] 12.3
##
##
## [[4]]
## [1] "Nuevo Elemento"
```

```r
print(list_data[4])
```

```
## [[1]]
## [1] "Nuevo Elemento"
```

```r
# Eliminar el último elemento
list_data[4] <- NULL

# Imprimir el 4to elemento
print(list_data[4])
```

```
## $<NA>
## NULL
```

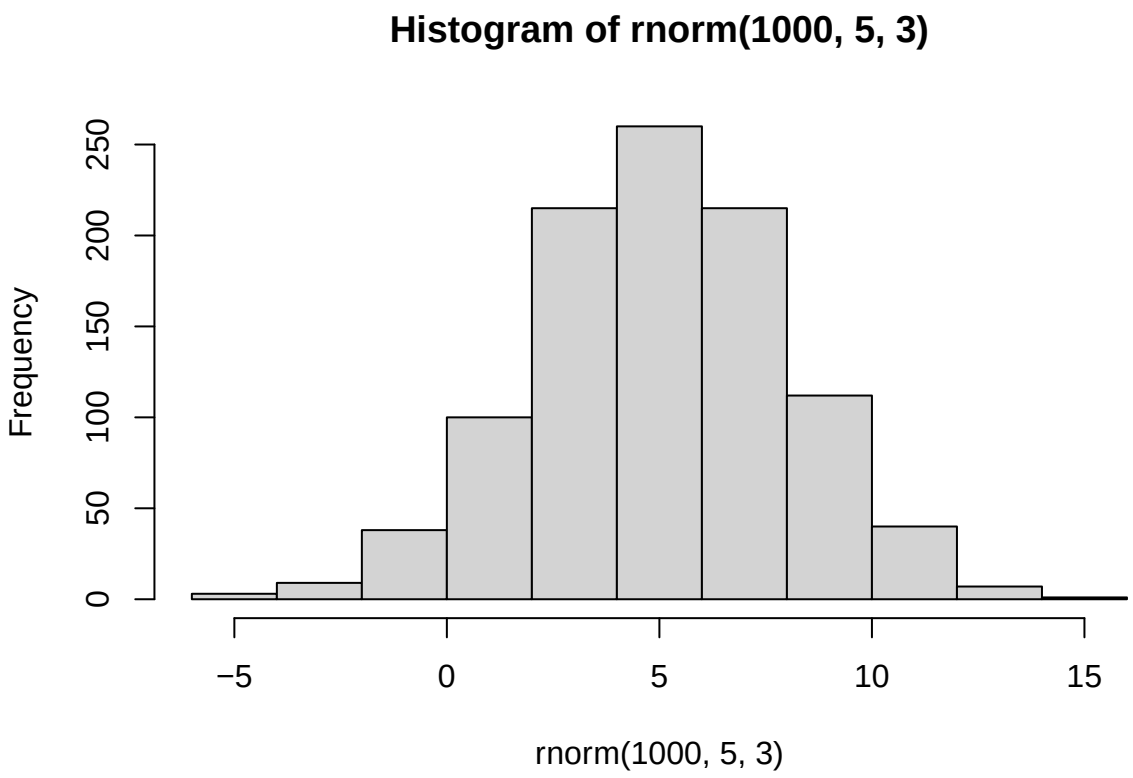
```r
# Modificar el tercer elemento
list_data[3] <- "elemento modificado"
print(list_data[3])
```

```
## $Una_lista_Interna
## [1] "elemento modificado"
```

Funciones que devuelven listas

Muchas funciones de R devuelven información diversa sobre los cálculos que realizan. Una función sólo puede devolver un objeto, asi que la forma usual de devolver esta información es una lista con distintos componentes. Cada componente almacena algún tipo de datos sobre el cómputo realizado por la función.

```r
set.seed(5)
grafica = hist(rnorm(1000,5,3))
```




```
typeof(grafica)
```

```
## [1] "list"
```

```
print(grafica)
```

```
## $breaks
##  [1] -6 -4 -2  0  2  4  6  8 10 12 14 16
##
## $counts
##  [1]   3   9  38 100 215 260 215 112  40   7   1
##
## $density
##  [1] 0.0015 0.0045 0.0190 0.0500 0.1075 0.1300 0.1075 0.0560 0.0200 0.0035
## [11] 0.0005
##
## $mids
##  [1] -5 -3 -1  1  3  5  7  9 11 13 15
##
## $xname
## [1] "rnorm(1000, 5, 3)"
##
```

```
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

```
str(grafica)
```

```
## List of 6
##  $ breaks  : num [1:12] -6 -4 -2 0 2 4 6 8 10 12 ...
##  $ counts  : int [1:11] 3 9 38 100 215 260 215 112 40 7 ...
##  $ density : num [1:11] 0.0015 0.0045 0.019 0.05 0.1075 ...
##  $ mids    : num [1:11] -5 -3 -1 1 3 5 7 9 11 13 ...
##  $ xname   : chr "rnorm(1000, 5, 3)"
##  $ equidist: logi TRUE
##  - attr(*, "class")= chr "histogram"
```

La función **lapply()** permite aplicar una función a los distintos componentes de una lista. La salida de esta función es una lista.

Para cada componente de la lista de entrada se genera un componente en la lista de salida con el resultado de aplicar la función al componente de la lista de entrada.

```
str(grafica)
```

```
## List of 6
##  $ breaks  : num [1:12] -6 -4 -2 0 2 4 6 8 10 12 ...
##  $ counts  : int [1:11] 3 9 38 100 215 260 215 112 40 7 ...
##  $ density : num [1:11] 0.0015 0.0045 0.019 0.05 0.1075 ...
##  $ mids    : num [1:11] -5 -3 -1 1 3 5 7 9 11 13 ...
##  $ xname   : chr "rnorm(1000, 5, 3)"
##  $ equidist: logi TRUE
##  - attr(*, "class")= chr "histogram"
```

```
lapply(grafica,length)
```

```
## $breaks
## [1] 12
##
## $counts
## [1] 11
##
## $density
## [1] 11
##
## $mids
## [1] 11
##
## $xname
```

```
## [1] 1
##
## $equidist
## [1] 1
```

Práctica 2

Realizar los ejercicios del archivo **02MBIF_07_Practica_2_Estructura_de_Datos.R**

2.4 Data frame

Los **data frames** son estructuras de datos de dos dimensiones que pueden contener datos de diferentes tipos, por lo tanto, son heterogéneas.

Es prácticamente la estructura de datos de facto para la mayoría de los datos tabulares.

Los **data frame** pueden tener atributos adicionales como **rownames()**, que pueden ser útiles para anotar datos, como subject_id o sample_id. Pero la mayoría de las veces no se utilizan.

Características:

- Generalmente son creados usando **read.csv()** y **read.table()**, es decir, al importar los datos a R.
- Si todas las columnas de un **data frame** son del mismo tipo, este se puede convertir en una matriz con **data.matrix()** o **as.matrix()**
- Un nuevo **data frame** se puede crear con la función **data.frame()**.
- El número de filas y columnas se puede obtener con **nrow()** y **ncol()** respectivamente.
- Los nombres de fila a menudo se generan automáticamente y se identifican con 1, 2,. . . , n. Es posible que no se respete la coherencia en la numeración de los nombres de filas cuando las filas se reorganizan o se extrae un subconjunto del **data frame**.

Creación de data frame

No es usual crear un data frame a mano pero lo haremos para entender en qué consiste esta estructura de datos.

```
datos <- data.frame(id = 1:10,
                    nombre=c("Luisa","Manuel","Rafael","Pedro","Carlos","Maria","Ana","Lourdes","Rosa",
                    edad = c(15,20,56,34,11,45,67,32,31,18),
                    sexo = c("M","H","H","H","H","M","M","M","M","H"))

datos
```

```
##    id  nombre edad sexo
## 1   1   Luisa   15    M
## 2   2  Manuel   20    H
## 3   3  Rafael   56    H
## 4   4   Pedro   34    H
## 5   5  Carlos   11    H
## 6   6   Maria   45    M
## 7   7     Ana   67    M
## 8   8 Lourdes   32    M
## 9   9    Rosa   31    M
## 10 10 Antonio   18    H
```

Funciones para visualizar

- head() - Retorna los primeros 6 elementos del dataframe
- tail() - Retorna los últimos 6 elementos del dataframe
- dim() - Retorna la dimensión del dataframe, es decir número de filas y columnas
- nrow() - Retorna el número de filas
- ncol() - Retorna el número de columnas
- str() - Estructura del dataframe - nombre, tipo y una previsualización de los datos de cada columna
- names() or colnames() - Muestran los atributos nombres del dataframe
- lapply(dataframe, class) - Retorna una lista con la clase de cada columna del dataframe
- sapply(dataframe, class) - Muestra la clase de cada columna del dataframe de una forma más amigable
- View() - Visualización de los datos en forma de tabla, se abre una pestaña
- summary() - Naturaleza de los datos y resumen estadístico

```
head(datos)
```

```
##   id nombre edad sexo
## 1  1  Luisa   15    M
## 2  2 Manuel   20    H
## 3  3 Rafael   56    H
## 4  4  Pedro   34    H
## 5  5 Carlos   11    H
## 6  6  Maria   45    M
```

```
tail(datos)
```

```
##    id  nombre edad sexo
## 5   5  Carlos   11    H
## 6   6   Maria   45    M
## 7   7     Ana   67    M
## 8   8 Lourdes   32    M
## 9   9    Rosa   31    M
## 10 10 Antonio   18    H
```

```
dim(datos)
```

```
## [1] 10  4
```

```
nrow(datos)
```

```
## [1] 10
```

```
ncol(datos)
```

```
## [1] 4
```

```
str(datos)
```

```
## 'data.frame':    10 obs. of  4 variables:
##  $ id    : int  1 2 3 4 5 6 7 8 9 10
##  $ nombre: chr  "Luisa" "Manuel" "Rafael" "Pedro" ...
##  $ edad  : num  15 20 56 34 11 45 67 32 31 18
##  $ sexo  : chr  "M" "H" "H" "H" ...
```

```
names(datos)
```

```
## [1] "id"     "nombre" "edad"   "sexo"
```

```
colnames(datos)
```

```
## [1] "id"     "nombre" "edad"   "sexo"
```

```
sapply(datos,class)
```

```
##          id      nombre        edad        sexo
##   "integer" "character"   "numeric" "character"
```

```
lapply(datos,class)
```

```
## $id
## [1] "integer"
##
## $nombre
## [1] "character"
##
## $edad
## [1] "numeric"
##
## $sexo
## [1] "character"
```

```
View(datos)
```

```
summary(datos)
```

```
##        id           nombre               edad           sexo
##  Min.   : 1.00   Length:10          Min.   :11.00   Length:10
##  1st Qu.: 3.25   Class :character   1st Qu.:18.50   Class :character
##  Median : 5.50   Mode  :character   Median :31.50   Mode  :character
##  Mean   : 5.50                      Mean   :32.90
##  3rd Qu.: 7.75                      3rd Qu.:42.25
##  Max.   :10.00                      Max.   :67.00
```

Extracción de datos

Podemos ver que un data frame es un tipo de lista usando la función **is.list()**.

Como los data frame son listas, es posible referenciar a las columnas usando la notación de lista, es decir usando los corchetes dobles o usando el $ antepuesto al nombre de la columna.

```
is.list(datos)
```

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

```r
class(datos)
```

```
## [1] "data.frame"
```

```r
datos[1,2]   # muestra el valor que se encuentra en la fila 1, columan 2
```

```
## [1] "Luisa"
```

```r
datos[[2]]    # dos formas de obtener los elementos de la segunda columna
```

```
##  [1] "Luisa"   "Manuel"  "Rafael"  "Pedro"   "Carlos"  "Maria"   "Ana"
##  [8] "Lourdes" "Rosa"    "Antonio"
```

```r
datos$nombre
```

```
##  [1] "Luisa"   "Manuel"  "Rafael"  "Pedro"   "Carlos"  "Maria"   "Ana"
##  [8] "Lourdes" "Rosa"    "Antonio"
```

```r
datos[1,]   # valores de la fila 1
```

```
##   id nombre edad sexo
## 1  1  Luisa   15    M
```

```r
datos[1:2,]   # muestra un rango de filas 1 a 2
```

```
##   id nombre edad sexo
## 1  1  Luisa   15    M
## 2  2 Manuel   20    H
```

```r
datos[c(1,3),]   # fila 1 y 3
```

```
##   id nombre edad sexo
## 1  1  Luisa   15    M
## 3  3 Rafael   56    H
```

Podemos extraer datos utilizando los nombres de las columnas del data frame

```r
result <- data.frame(datos$nombre,datos$edad)
print(result)
```

```
##    datos.nombre datos.edad
## 1         Luisa         15
## 2        Manuel         20
## 3        Rafael         56
## 4         Pedro         34
## 5        Carlos         11
## 6         Maria         45
## 7           Ana         67
## 8       Lourdes         32
## 9          Rosa         31
## 10      Antonio         18
```

Expansión de data frame

Agregar columnas Se pueden agregar nuevas columnas a un data frame existente.

```r
print(datos)
```

```
##    id  nombre edad sexo
## 1   1   Luisa   15    M
## 2   2  Manuel   20    H
## 3   3  Rafael   56    H
## 4   4   Pedro   34    H
## 5   5  Carlos   11    H
## 6   6   Maria   45    M
## 7   7     Ana   67    M
## 8   8 Lourdes   32    M
## 9   9    Rosa   31    M
## 10 10 Antonio   18    H
```

```r
datos$apellido = c("Martinez", "Gomez", "Perez", "Rodriguez","Martin", "Fernandez","Lopez", "Gomez","Ga

print(datos)
```

```
##    id  nombre edad sexo  apellido
## 1   1   Luisa   15    M  Martinez
## 2   2  Manuel   20    H     Gomez
```

```
## 3   3  Rafael   56    H     Perez
## 4   4   Pedro   34    H Rodriguez
## 5   5  Carlos   11    H    Martin
## 6   6   Maria   45    M Fernandez
## 7   7     Ana   67    M     Lopez
## 8   8 Lourdes   32    M     Gomez
## 9   9    Rosa   31    M    Garcia
## 10 10 Antonio   18    H   Alvarez
```

Añadir filas Para agregar más filas permanentemente a un data frame existente, necesitamos traer las nuevas filas en la misma estructura que el data frame existente y usar la función **rbind()**.

En el siguiente ejemplo, creamos un data frame con nuevas filas y lo fusionamos con el data frame existente para crear el data frame final.

```
datos.new = data.frame(
  id = c (11:13),
  nombre = c("Ramon","Pedro","Mario"),
  edad = c(20,25,56),
  sexo = c("H","H","H"),
  apellido=c(" Carrasco", "Marco", "Vazquez")
)
datos.new
```

```
##   id nombre edad sexo  apellido
## 1 11  Ramon   20    H  Carrasco
## 2 12  Pedro   25    H     Marco
## 3 13  Mario   56    H   Vazquez
```

```
# Bind the two data frames.
datos.final <- rbind(datos, datos.new)
datos.final
```

```
##    id  nombre edad sexo  apellido
## 1   1   Luisa   15    M  Martinez
## 2   2  Manuel   20    H     Gomez
## 3   3  Rafael   56    H     Perez
## 4   4   Pedro   34    H Rodriguez
## 5   5  Carlos   11    H    Martin
## 6   6   Maria   45    M Fernandez
## 7   7     Ana   67    M     Lopez
## 8   8 Lourdes   32    M     Gomez
## 9   9    Rosa   31    M    Garcia
## 10 10 Antonio   18    H   Alvarez
## 11 11   Ramon   20    H  Carrasco
## 12 12   Pedro   25    H     Marco
## 13 13   Mario   56    H   Vazquez
```

2.5 Factores

Los factores son los objetos de datos que se utilizan para categorizar los datos y almacenarlos como niveles. Pueden almacenar tanto cadenas como enteros. Son útiles en las columnas que tienen un número limitado de valores únicos. Como "Masculino," Femenino "y Verdadero, Falso, etc. Son útiles en el análisis de datos para el modelado estadístico.

Los factores se crean usando la función **factor()** tomando un vector como entrada.

```
# Crear un vector
data <- c("East","West","East","North","North","East","West","West","West","East","North")

print(data)
```

```
##  [1] "East"  "West"  "East"  "North" "North" "East"  "West"  "West"  "West"
## [10] "East"  "North"
```

```
print(is.factor(data))
```

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

```
# Aplicar la funcion factor
factor_data <- factor(data)

print(factor_data)
```

```
##  [1] East  West  East  North North East  West  West  West  East  North
## Levels: East North West
```

```
print(is.factor(factor_data))
```

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

Para ver los niveles que tiene un objeto de tipo factor podemos utilizar la función levels() y nlevels() que nos proporciona el número de niveles.

```
str(datos)
```

```
## 'data.frame':    10 obs. of  5 variables:
##  $ id      : int  1 2 3 4 5 6 7 8 9 10
##  $ nombre  : chr  "Luisa" "Manuel" "Rafael" "Pedro" ...
##  $ edad    : num  15 20 56 34 11 45 67 32 31 18
##  $ sexo    : chr  "M" "H" "H" "H" ...
##  $ apellido: chr  "Martinez" "Gomez" "Perez" "Rodriguez" ...
```

```
print(is.factor(datos$sexo))
```

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

```
# Cambiar el campo sexo a factor

datos$sexo = as.factor(datos$sexo)
print(is.factor(datos$sexo))
```

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

```
# Imprimir el campo sexo para ver los niveles
print(datos$sexo)
```

```
##  [1] M H H H H M M M M H
## Levels: H M
```

```
levels(datos$sexo)
```

```
## [1] "H" "M"
```

A veces, el orden de los factores no importa, otras veces es posible que se desee especificar el orden porque es significativo (por ejemplo, “bajo”, “medio”, “alto”) o es requerido por un tipo particular de análisis. Además, especificar el orden de los niveles nos permite comparar niveles:

```
food <- factor(c("low", "high", "medium", "high", "low", "medium", "high"))
levels(food)
```

```
## [1] "high"   "low"    "medium"
```

```r
food <- factor(food, levels = c("low", "medium", "high"))
levels(food)
```

```
## [1] "low"    "medium" "high"
```

¿Qué sucede si aplicamos ciertas funciones estadísticas sobre objetos de tipo factor?

```r
min(food)
```

Mediante el argumento ordered=TRUE de la función factor, le indicamos que deseamos usar el factor como valores con orden.

```r
food <- factor(food, levels = c("low", "medium", "high"), ordered = TRUE)
levels(food)
```

```
## [1] "low"    "medium" "high"
```

```r
str(food)
```

```
##  Ord.factor w/ 3 levels "low"<"medium"<..: 1 3 2 3 1 2 3
```

```r
min(food)
```

```
## [1] low
## Levels: low < medium < high
```

Práctica.3

Realizar los ejercicios del archivo **02MBIF_07_Practica_3_Estructura_de_Datos.R**