Vectores y tipos de datos

Imanol

10/2/2021

Estructuras de datos

Basicos

Vectores

```
c(1,2,3) # Para definir un vector
```

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

```
scan() # Para definir un vector por consola a mano
```

```
## numeric(0)
```

```
##fix(c): # Para modificar visualmente un vector x
rep(1,5) # Para definir un vector que tiene el mismo dato a (1) repetido n (5) veces
```

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

Para especificar como colocar los decimales -> scan(dec = ",") Para especificar que tipo de datos seran -> scan(what = "character")

Ejercicio

Repetir mi año de nacimiento 10 veces

```
anno <- 1993
n <- 10
rep(anno,n)
```

```
##  [1] 1993 1993 1993 1993 1993 1993 1993 1993 1993 1993
```

Crear un vector que tenga como entradas 16,0,1,20,1,7,88,5,1,9, llamalo **vec** y modifica la cuarta entrada con la funcion **fix()**.

```r
vec <- c(16, 0, 1, 20, 1, 7, 88, 5, 1, 9)
vec
```

```
##  [1] 16  0  1 20  1  7 88  5  1  9
```

```r
fix(vec)
vec
```

```
##  [1] 16  0  1 20  1  7 88  5  1  9
```

Mas operaciones con vectores

Operaciones directas aplicadas a un vector

```r
x <- 1:10
x + pi # Suma pi a todos
```

```
##  [1]  4.141593  5.141593  6.141593  7.141593  8.141593  9.141593 10.141593
##  [8] 11.141593 12.141593 13.141593
```

```r
pi*x # Multiplica a todos por pi
```

```
##  [1]  3.141593  6.283185  9.424778 12.566371 15.707963 18.849556 21.991149
##  [8] 25.132741 28.274334 31.415927
```

```r
sqrt(x) # Raiz cuadrada de cada valor
```

```
##  [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427
##  [9] 3.000000 3.162278
```

```r
2^x # 2 elevado a cada valor
```

```
##  [1]    2    4    8   16   32   64  128  256  512 1024
```

```r
x^2 # cada valor elevado a dos
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

```r
1:10 + 1:10 #Se pueden sumar vectores de la misma longitud
```

```
##  [1]  2  4  6  8 10 12 14 16 18 20
```

```r
1:10 * 1:10 #Se pueden multiplicar vectores de la misma longitud
```

```
##  [1]   1   4   9  16  25  36  49  64  81 100
```

```r
n = 1:100
x = 2*3^(n/2)-15 # Esta formula calcularia los n elemntos de esta sucesion
x
```

```
##   [1] -1.153590e+01 -9.000000e+00 -4.607695e+00  3.000000e+00  1.617691e+01
##   [6]  3.900000e+01  7.853074e+01  1.470000e+02  2.655922e+02  4.710000e+02
##  [11]  8.267767e+02  1.443000e+03  2.510330e+03  4.359000e+03  7.560990e+03
##  [16]  1.310700e+04  2.271297e+04  3.935100e+04  6.816891e+04  1.180830e+05
##  [21]  2.045367e+05  3.542790e+05  6.136402e+05  1.062867e+06  1.840951e+06
##  [26]  3.188631e+06  5.522882e+06  9.565923e+06  1.656868e+07  2.869780e+07
##  [31]  4.970606e+07  8.609343e+07  1.491182e+08  2.582803e+08  4.473546e+08
##  [36]  7.748410e+08  1.342064e+09  2.324523e+09  4.026192e+09  6.973569e+09
##  [41]  1.207858e+10  2.092071e+10  3.623573e+10  6.276212e+10  1.087072e+11
##  [46]  1.882864e+11  3.261215e+11  5.648591e+11  9.783646e+11  1.694577e+12
##  [51]  2.935094e+12  5.083732e+12  8.805282e+12  1.525119e+13  2.641584e+13
##  [56]  4.575358e+13  7.924753e+13  1.372608e+14  2.377426e+14  4.117823e+14
##  [61]  7.132278e+14  1.235347e+15  2.139683e+15  3.706040e+15  6.419050e+15
##  [66]  1.111812e+16  1.925715e+16  3.335436e+16  5.777145e+16  1.000631e+17
##  [71]  1.733144e+17  3.001893e+17  5.199431e+17  9.005678e+17  1.559829e+18
##  [76]  2.701703e+18  4.679488e+18  8.105110e+18  1.403846e+19  2.431533e+19
##  [81]  4.211539e+19  7.294599e+19  1.263462e+20  2.188380e+20  3.790385e+20
##  [86]  6.565139e+20  1.137115e+21  1.969542e+21  3.411346e+21  5.908625e+21
##  [91]  1.023404e+22  1.772588e+22  3.070212e+22  5.317763e+22  9.210635e+22
##  [96]  1.595329e+23  2.763191e+23  4.785987e+23  8.289572e+23  1.435796e+24
```

Operaciones que no se pueden hacer directamente a un vector (uso de sapply)

Ejemplo básico:

```r
sapply(x, FUN = function(elemento){sqrt(elemento)}) # A x le aplicamos la función FUN
```

```
## Warning in sqrt(elemento): Se han producido NaNs

## Warning in sqrt(elemento): Se han producido NaNs

## Warning in sqrt(elemento): Se han producido NaNs
```

```
##   [1]          NaN          NaN          NaN 1.732051e+00 4.022054e+00
##   [6] 6.244998e+00 8.861757e+00 1.212436e+01 1.629700e+01 2.170253e+01
##  [11] 2.875372e+01 3.798684e+01 5.010319e+01 6.602272e+01 8.695395e+01
##  [16] 1.144858e+02 1.507082e+02 1.983709e+02 2.610918e+02 3.436321e+02
##  [21] 4.522574e+02 5.952134e+02 7.833519e+02 1.030954e+03 1.356816e+03
##  [26] 1.785674e+03 2.350081e+03 3.092883e+03 4.070464e+03 5.357033e+03
##  [31] 7.050252e+03 9.278654e+03 1.221140e+04 1.607110e+04 2.115076e+04
##  [36] 2.783597e+04 3.663419e+04 4.821331e+04 6.345228e+04 8.350790e+04
##  [41] 1.099026e+05 1.446399e+05 1.903568e+05 2.505237e+05 3.297077e+05
##  [46] 4.339198e+05 5.710705e+05 7.515711e+05 9.891232e+05 1.301759e+06
##  [51] 1.713212e+06 2.254713e+06 2.967369e+06 3.905278e+06 5.139635e+06
##  [56] 6.764140e+06 8.902108e+06 1.171583e+07 1.541890e+07 2.029242e+07
##  [61] 2.670633e+07 3.514750e+07 4.625671e+07 6.087726e+07 8.011898e+07
##  [66] 1.054425e+08 1.387701e+08 1.826318e+08 2.403569e+08 3.163275e+08
##  [71] 4.163104e+08 5.478953e+08 7.210708e+08 9.489825e+08 1.248931e+09
```

```
##  [76] 1.643686e+09 2.163212e+09 2.846948e+09 3.746794e+09 4.931058e+09
##  [81] 6.489637e+09 8.540843e+09 1.124038e+10 1.479317e+10 1.946891e+10
##  [86] 2.562253e+10 3.372114e+10 4.437952e+10 5.840673e+10 7.686758e+10
##  [91] 1.011634e+11 1.331386e+11 1.752202e+11 2.306028e+11 3.034903e+11
##  [96] 3.994157e+11 5.256606e+11 6.918083e+11 9.104709e+11 1.198247e+12
```

Ejemplo más complicado:

```
cd = function(x){summary(lm((1:4)~c(1:3,x)))}$r.squared # Coeficiente de determinación, muy avanzado po
cd(5)
```

```
## [1] 0.9657143
```

```
cd(6)
```

```
## [1] 0.9142857
```

```
cd(7)
```

```
## [1] 0.8698795
```

Más funciones

```
v = c(1,2,3,4,5,6)
mean(v) # media
```

```
## [1] 3.5
```

```
cumsum(v) # suma acumulada
```

```
## [1]  1  3  6 10 15 21
```

```
sort(v) # Ordena el vector de forma creciente
```

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

```
rev(v) # Invierte el vecto
```

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

```
length(v) # Longitud del vector
```

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

```r
max(v) # EL mas grande
```

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

```r
min(v) # El mas pequeño
```

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

```r
sum(v) # La suma de todos
```

```
## [1] 21
```

```r
prod(v) # El producto de todos
```

```
## [1] 720
```

```r
cumsum(v) # La suma acumulada
```

```
## [1]  1  3  6 10 15 21
```

```r
cummax(v) # EL valor mas grande encontrado hasta el momento
```

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

```r
cummin(v) # El mas pequeño encontrado hasta el momento
```

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

```r
cumprod(v) # El producto acumulado
```

```
## [1]   1   2   6  24 120 720
```

```r
diff(v) # La diferencia de uno a otro numero
```

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

Ejercicio

Combinad las dos funciones, **sort** y **rev** para crear una funcion que dado un vector x os lo devuelva ordenado en orden decreciente.

```r
x <- scan()
x <- sort(x)
x <- rev(x)
x
```

```
## numeric(0)
```

```r
# Otro metodo más corto
y <- c(2,4,3,1,7,8,5,9,6,0)
y <- sort(y, decreasing = TRUE) # Al decirle decreasing TRUE hago que lo ordene en orden decreciente
y
```

```
##  [1] 9 8 7 6 5 4 3 2 1 0
```

SUBVECTORES Y FILTROS

```r
x = seq(3, 50, by = 3.5)
x
```

```
##  [1]  3.0  6.5 10.0 13.5 17.0 20.5 24.0 27.5 31.0 34.5 38.0 41.5 45.0 48.5
```

```r
x[3] # Para sacar el dato de una posicion
```

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

```r
x[length(x)] # Para ir al ultimo valor
```

```
## [1] 48.5
```

```r
x[length(x)-1] # Para ir al penultimo valor
```

```
## [1] 45
```

```r
x[-3] # Para quitar el valor de esa posicion
```

```
##  [1]  3.0  6.5 13.5 17.0 20.5 24.0 27.5 31.0 34.5 38.0 41.5 45.0 48.5
```

```r
x[4:8] # Para quedarme solo en el rango que quiero de posiciones
```

```
## [1] 13.5 17.0 20.5 24.0 27.5
```

```r
x[8:4]
```

```
## [1] 27.5 24.0 20.5 17.0 13.5
```

```r
x[seq(2, length(x), by = 2)] # Para coger solo las posiciones pares
```

```
## [1]  6.5 13.5 20.5 27.5 34.5 41.5 48.5
```

```r
x[seq(1, length(x), by = 2)] # Para coger solo las posiciones impares
```

```
## [1]  3 10 17 24 31 38 45
```

```r
x[-seq(1, length(x), by = 2)] # Elimar posiciones impares
```

```
## [1]  6.5 13.5 20.5 27.5 34.5 41.5 48.5
```

```r
x[(length(x)-3):length(x)] # Para sacar solo las posiciones de antepenultima a ultima posicion
```

```
## [1] 38.0 41.5 45.0 48.5
```

```r
x[c(1,2,4)]
```

```
## [1]  3.0  6.5 13.5
```

```r
x[x>20 & x<40] # Escoger solo los numeros mayores de 20 y menores de 40
```

```
## [1] 20.5 24.0 27.5 31.0 34.5 38.0
```

```r
x[x%%2==0] #Sacar solo numeros pares
```

```
## [1] 10 24 38
```

```r
x>30 # Nos da true o false de los que cumplen o no
```

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

```r
x = c(1,7,4,2,4,8,9,2,0)
y = c(5,2,-3,-7,-1,4,-2,7,1)
x[y>0] # Ambos vectores deben ser del mismo tamaño, nos saca los valores en x
```

```
## [1] 1 7 8 2 0
```

```r
      # de las posiciones de los numeros de y que cumplen la condición

which(x>4) # me da la posicion de los que son mayores de 4
```

```
## [1] 2 6 7
```

```r
x[which(x>4)] # da los valores que cumplen la condicion
```

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

```r
which.min(x) # La posicion del valor mas pequeño
```

```
## [1] 9
```

```r
which.max(x)
```

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

```r
which(x == max(x)) # Posiciones de los valores iguales al valor mas grande
```

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

```r
x = c() # Vector vacio, NULL
x
```

```
## NULL
```

```r
y = NULL
y
```

```
## NULL
```

```r
z = c(x, 2, y, 7) # los null no salen
z
```

```
## [1] 2 7
```

LOS VALORES DE NA

PASARTE DEL RANGO DEL VECTOR

```r
x <- c(1:10)
x[length(x)+5] = 9 # Me pone un 9 en la ultima posicion mas cinco
                   # pero como solo hay 10 posiciones en el resto pone NA
x
```

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

OPERACIONES TENIENDO NA

```r
sum(x) # No se puede, da NA
```

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

```r
sum(x, na.rm = TRUE) # no tiene en cuenta los NA y hace la suma
```

```
## [1] 64
```

```r
mean(x, na.rm = TRUE)
```

```
## [1] 5.818182
```

```r
is.na(x) # Me dice si hay NA
```

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

```r
which(is.na(x)) # Posiciones en las que hay NA
```

```
## [1] 11 12 13 14
```

En estadistica descriptiva no se puede trabajar con NA asi que se sustituyen por la media

```r
y = x
y[is.na(y)]
```

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

```r
y[is.na(y)] = mean(y, na.rm = TRUE)
y
```

```
##  [1]  1.000000  2.000000  3.000000  4.000000  5.000000  6.000000  7.000000
##  [8]  8.000000  9.000000 10.000000  5.818182  5.818182  5.818182  5.818182
## [15]  9.000000
```

Cumsum con NA

```r
cumsum(x[!is.na(x)])
```

```
##  [1]  1  3  6 10 15 21 28 36 45 55 64
```

Atributos y su eliminación

```r
x_clean = na.omit(x) #Elimina los na
x_clean # Se puede ver que elimina los na pero se queda con el atributo "na.action"
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10  9
## attr(,"na.action")
## [1] 11 12 13 14
## attr(,"class")
## [1] "omit"
```

```r
attr(x_clean, "na.action") = NULL # Asi elimino el atributo
x_clean
```

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

FACTORES

Un factor es una lista que tiene copias de las etiquetas colocadas *(Levels)*

```r
nombres = c("Juan", "Antonio", "Maria", "Paco")
nombres
```

```
## [1] "Juan"    "Antonio" "Maria"   "Paco"
```

```r
nombres.factor = factor(nombres) # Convierto el vector enun factor
nombres.factor
```

```
## [1] Juan    Antonio Maria   Paco
## Levels: Antonio Juan Maria Paco
```

EJEMPLOS CON NIVELES

```r
gender = c("M", "H", "H", "M", "M", "M", "M", "H", "H")
gender.fact = factor(gender)
gender.fact
```

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

```r
gender.fact3 = factor(gender, levels = c("M", "H", "B")) # Si quiero darle mas etiquetas
gender.fact3
```

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

```r
gender.fact4 = factor(gender, levels = c("M", "H", "B"), labels = c("Mujer", "Hombre", "Hermafrodita"))
gender.fact4 # Cambia todas las etiquetas con su nombre completo para que sea mas facil entender
```

```
## [1] Mujer  Hombre Hombre Mujer  Mujer  Mujer  Mujer  Hombre Hombre
## Levels: Mujer Hombre Hermafrodita
```

```r
levels(gender.fact4) # Para ver sus niveles
```

```
## [1] "Mujer"        "Hombre"       "Hermafrodita"
```

```r
levels(gender.fact4) = c("Femenino", "Masculino", "Híbrido") # Para cambiar de nombre las etiquetas
gender.fact4
```

```
## [1] Femenino  Masculino Masculino Femenino  Femenino  Femenino  Femenino
## [8] Masculino Masculino
## Levels: Femenino Masculino Híbrido
```

```r
notas = c(1,4,3,2,3,2,4,3,1,2,3,4,2,3,4)
notas.factor = factor(notas)
notas.factor
```

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

```r
levels(notas.factor)
```

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

```r
levels(notas.factor) = c("Suspendido", "Suficiente", "Notable", "Excelente")
levels(notas.factor)
```

```
## [1] "Suspendido" "Suficiente" "Notable"    "Excelente"
```

UNIR NIVELES

```r
notas.factor = factor(notas)
levels(notas.factor) = c("Suspendido", "Aprobado", "Aprobado", "Aprobado")
notas.factor
```

```
##  [1] Suspendido Aprobado   Aprobado   Aprobado   Aprobado   Aprobado
##  [7] Aprobado   Aprobado   Suspendido Aprobado   Aprobado   Aprobado
## [13] Aprobado   Aprobado   Aprobado
## Levels: Suspendido Aprobado
```

FACTOR ORDENADO

Es un factor donde los niveles siguen un orden

```r
ordered(notas, labels = c("Sus", "Suf", "Not", "Exc")) # Hemos sustituido los valores por su etiqueta p
```

```
##  [1] Sus Exc Not Suf Not Suf Exc Not Sus Suf Not Exc Suf Not Exc
## Levels: Sus < Suf < Not < Exc
```

```r
                                                    # ahora los niveles estan ordenados de menor a m
```

LISTAS

Listas formadas por diferentes objetos, no necesariamente del mismo tipo, cada cual con su nombre interno

CREAR UNA LISTA

Asignamos a cada nombre que creemos lo que es.

```
x = c(1,5,-2,6,-7,8,-3,4,-9)
x
```

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

```
L = list(nombre = "temperaturas", datos = x, media = mean(x), sumas = cumsum(x))
L
```

```
## $nombre
## [1] "temperaturas"
##
## $datos
## [1]  1  5 -2  6 -7  8 -3  4 -9
##
## $media
## [1] 0.3333333
##
## $sumas
## [1]  1  6  4 10  3 11  8 12  3
```

ACCEDER A CADA NOMBRE DE LA LISTA

Mediante el nombre

```
L$nombre
```

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

```
L$datos
```

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

```
L$media
```

```
## [1] 0.3333333
```

```
L$sumas
```

```
## [1]  1  6  4 10  3 11  8 12  3
```

Mediante la posición

```
L[[1]]
```

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

```
L[[2]]
```

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

```
L[[3]]
```

```
## [1] 0.3333333
```

```
L[[4]]
```

```
## [1]  1  6  4 10  3 11  8 12  3
```

```
L[2] # Seria una lista no un vector, no se podria operar con la lista, para que sea un vector se usa [[
```

```
## $datos
## [1]  1  5 -2  6 -7  8 -3  4 -9
```

OBTENER INFORMACIÓN DE UNA LISTA (str y name)

str

```
str(L) # Me da las listas, que tipo de datos son y el numero de datos
```

```
## List of 4
##  $ nombre: chr "temperaturas"
##  $ datos : num [1:9] 1 5 -2 6 -7 8 -3 4 -9
##  $ media : num 0.333
##  $ sumas : num [1:9] 1 6 4 10 3 11 8 12 3
```

name

```
names(L) # Me da el nombre de las listas
```

```
## [1] "nombre" "datos"  "media"  "sumas"
```

MATRICES

DEFINIR MATRICES

$$matrix(vector, nrow = n, byrow = valor\_lógico)$$

nrow: Número filas **byrow**: True = se construye por filas (por defecto), False = se construye por columnas

```
M = matrix(1:12, nrow = 4)
M
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```
P = matrix(1:12, nrow = 5) # Si defines la matriz con número de filas que no son multiplos del numero d
```

```
## Warning in matrix(1:12, nrow = 5): la longitud de los datos [12] no es un
## submúltiplo o múltiplo del número de filas [5] en la matriz
```

```
P                          # te saldra un warning y creara la matriz rellenando los datos que faltan re
```

```
##      [,1] [,2] [,3]
## [1,]    1    6   11
## [2,]    2    7   12
## [3,]    3    8    1
## [4,]    4    9    2
## [5,]    5   10    3
```

```
                           # datos desde el principio

matrix(1, nrow = 4, ncol = 6) # Crea la matriz de un solo numero
```

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

EJERCICIOS

¿Como definirias una matriz constante? Es decir, ¿Cómo definirías una matriz A tal que $\forall = 1, \ldots, n; j = 1, \ldots, m, a_{ij} = k$ siendo $k \in \mathbb{R}$? Como R no admite incógnitas, prueba para el caso especifico $n = 3, m = 5, k = 0$

```
n <- 3
m <- 5
k <- 0
E1 = matrix(k, nrow = n, ncol = m)
E1
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    0    0    0    0    0
## [2,]    0    0    0    0    0
## [3,]    0    0    0    0    0
```

Con el vector vec = (1,2,3,4,5,6,7,8,9,10,11,12) crea la matriz *(Entradas por columna)*:

$$\begin{pmatrix} 1 & 4 & 7 & 10 \\ 2 & 5 & 8 & 11 \\ 3 & 6 & 9 & 12 \end{pmatrix}$$

```
vec = c(1,2,3,4,5,6,7,8,9,10,11,12)
E2 <- matrix(vec, ncol = 4, byrow = FALSE)
E2
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    4    7   10
## [2,]    2    5    8   11
## [3,]    3    6    9   12
```

CONSTRUIR MATRICES

rbind(vector1, vector2,..): Construye la matriz de filas vector1, vector2,..

cbind(vector1, vector2,..): Construye la matriz de columnas vector1, vector2,..

- Los vectores tienen que ser de la misma longitud
- Tambien sirve para añadir filas/columnas a una matriz o concatenar por filas/columnas matrices del mismo numero de filas/columnas

diag(vector): Construye una matriz diagonal con un vector dado

- Si aplicamos diag a un numero n, produce una matriz identidad de orden n.

```
rbind(c(1,2,3), c(-1,-2,-3))
```

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

```
cbind(c(1,2,3), c(-1,-2,-3))
```

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

```
diag(5, nrow = 3)
```

```
##      [,1] [,2] [,3]
## [1,]    5    0    0
## [2,]    0    5    0
## [3,]    0    0    5
```

SUBMATRICES

Para sacar datos especificos de la matriz

```
M = matrix(1:12, nrow = 4)
M
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```
M[1,2] # Un dato
```

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

```
M[2, ] # TOdos los datos de una fila
```

```
## [1]  2  6 10
```

```
M[,3]  # Todos los datos de una columna
```

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

```
M[c(1,3,4), 1:2] # Datos de filas concretas y que columnas
```

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

```
nrow(M) # Da numero filas
```

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

```
ncol(M) # Numero columnas
```

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

```
dim(M) # Dimension
```

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

```
sum(M)  # Suma de todos los datos de la matriz
```

```
## [1] 78
```

```r
prod(M) # producto de todos los datos de la matriz
```

```
## [1] 479001600
```

```r
mean(M) # media de todos los datos de la matriz
```

```
## [1] 6.5
```

```r
colSums(M) # suma de todas las columnas
```

```
## [1] 10 26 42
```

```r
rowSums(M)
```

```
## [1] 15 18 21 24
```

```r
colMeans(M)
```

```
## [1]  2.5  6.5 10.5
```

Aplicar una munción a una matriz, por filas:

```r
apply(M, MARGIN = 1, FUN = function(x){sqrt(sum(x^2))}) # MARGIN = 1 es para aplicar funcion por filas,
```

```
## [1] 10.34408 11.83216 13.37909 14.96663
```

```r
apply(M, MARGIN = 2, FUN = function(x){sqrt(sum(x^2))}) # MARGIN = 2 es para aplicar funcion por columna
```

```
## [1]  5.477226 13.190906 21.118712
```

REPASO ALGEBRA LINEAL

OPERACIONES

```r
M = matrix(1:12, nrow = 4)
M
```

```
##      [,1] [,2] [,3]
## [1,]    1    5    9
## [2,]    2    6   10
## [3,]    3    7   11
## [4,]    4    8   12
```

```r
t(M)      #Traspuesta
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    3    4
## [2,]    5    6    7    8
## [3,]    9   10   11   12
```

```r
M%*%t(M) # Multiplicacion de matrices
```

```
##      [,1] [,2] [,3] [,4]
## [1,]  107  122  137  152
## [2,]  122  140  158  176
## [3,]  137  158  179  200
## [4,]  152  176  200  224
```

```r
#mtx.exp(M,2) # Para elevar la matriz (paquete Biodem), no es exacto aproxima
M%%2 # Para elevar la matriz (Paquete expm), no es exacto aproxima
```

```
##      [,1] [,2] [,3]
## [1,]    1    1    1
## [2,]    0    0    0
## [3,]    1    1    1
## [4,]    0    0    0
```